НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»

УДК 004.415.25

До захисту допущено:

В. о. завідувача кафедри

_____ Едуард ЖАРІКОВ

«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»
спеціальність 121 «Інженерія програмного забезпечення»

на тему: «Архітектурне рішення програмного забезпечення системи технічної підтримки користувачів»

Виконав (-ла):
студент (-ка) ІІ курсу, групи ІТ-04мп
Петрова Марина Євгеніївна _____

Керівник:
Доцент, кандидат технічних наук,
Новінський Валерій Петрович _____

Консультант з нормоконтролю:
Доцент, кандидат технічних наук,
Ліщук Катерина Ігорівна _____

Рецензент:
Доцент, кандидат технічних наук,
Жданова Олена Григорівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність — 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Едуард ЖАРІКОВ

«___»_____2021 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Петровій Марині Євгеніївні

1. Тема проєкту «Архітектурне рішення програмного забезпечення системи технічної підтримки користувачів», керівник проєкту Новінський Валерій Петрович, кандидат технічних наук, доцент, затверджені наказом по університету від «27» жовтня 2021р. № 3587-с

2. Термін подання студентом дисертації «6» грудня 2021 р.

3. Об’єкт дослідження − програмне забезпечення у області Service Desk.

4. Предмет дослідження — удосконалення процесу класифікації заявок у системах технічної підтримки користувачів.

5. Перелік завдань, які потрібно розробити — аналіз існуючих рішень; проектування архітектури системи; розробка системи; оцінка ефективності запропонованого рішення.

6. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): Діаграма сутностей системи (А3), Схема архітектури (А3), Діаграма поведінки (А3).

7. Орієнтовний перелік публікацій — тези SoftTech-2021.

8. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

Перевірка на співпадіння			
Норм.контроль			

9. Дата видачі завдання «30» вересня 2021р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Аналіз предметної області	01.10.2021 — 08.10.2021 рр.	
2	Проектування архітектурного рішення	09.10.2021 — 16.10.2021 рр.	
3	Розробка архітектурного рішення	17.10.2021 — 24.10.2021 рр.	
4	Реалізація архітектурного рішення	25.10.2021 — 02.11.2021 рр.	
5	Розробка маркетингового стартап-проєкту	03.11.2021 — 10.11.2021 рр.	
6	Виконання експериментальних досліджень	11.11.2021 — 18.11.2021 рр	
7	Оформлення пояснювальної записки	19.11.2021 — 22.11.2021 рр	
8	Подання дисертації на попередній захист	22.11.2021	
9	Подання дисертації на захист	6.12.2021	

Студент _____ Марина ПЕТРОВА

Керівник _____ Валерій НОВІНСЬКИЙ

РЕФЕРАТ

Розмір пояснювальної записки — 103 аркуші, містить 11 ілюстрацій, 27 таблиць, 6 додатків.

Актуальність теми. Підприємства з великою чисельністю співробітників та користувачів потребують автоматизованих бізнес-процесів для роботи з інцидентами. Для підвищення ефективності процесів та швидкості реагування на інциденти, необхідно удосконалити процес технічної підтримки на підприємствах.

Мета проекту полягає в тому, щоб підвищити ефективність роботи підприємства і зменшити навантаження на інженерів, відповідальних за технічну підтримку, шляхом удосконалення архітектури програмного забезпечення системи технічної підтримки користувачів.

Задачі, які необхідно вирішити для досягнення описаної вище мети:

- проаналізувати існуючі рішення;
- спроектувати архітектуру системи;
- розробити систему;
- оцінити ефективність запропонованого рішення.

Об’єкт дослідження — програмне забезпечення у області Service Desk.

Предмет дослідження — удосконалення процесу класифікації заявок у системах технічної підтримки користувачів.

Наукова новизна даної роботи полягає в удосконаленні архітектурного рішення програмного забезпечення системи технічної підтримки користувачів з метою підвищення ефективності етапу класифікації запитів.

Практичне значення одержаних результатів. У результаті роботи над даним продуктом буде досягнуто вирішення проблеми надмірних заявок та питань,

що надходять у службу технічної підтримки. Завдяки цьому — підвищено ефективність роботи ліній технічної підтримки і, власне, підприємства.

Зв'язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Апробація. Наукові положення дисертації пройшли апробацію на Першій Всеукраїнській науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2021). Секція кафедри інформатики та програмної інженерії.

Публікації. Наукові положення дисертації опубліковані в публікації: Петрова М. Є. Архітектурне рішення програмного забезпечення системи технічної підтримки користувачів / Петрова М. Є. // Матеріали Першої Всеукраїнська науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2021). Секція кафедри інформатики та програмної інженерії. Матеріали конференції. – Київ. – 2021. 22–26 листопада 2021р. – С.XX.

Ключові слова: УПРАВЛІННЯ ІТ-ПОСЛУГАМИ, SERVICE DESK, ІНЦИДЕНТ, ПОСЛУГА, КЛАСИФІКАЦІЯ.

SUMMARY

Explanatory note size — 103 pages, contains 11 illustrations, 27 tables, 6 applications.

Actuality of theme. Enterprise companies, who have a lot of employees and users, should have automated business processes to work with incidents. To make processes in companies more effective and faster, customer technical support mechanisms should be improved.

Aim of research is to increase effectiveness of companies' work and lower the load of those engineers who are responsible for customer support. This should be made by improving the software architecture of customer technical support system.

The tasks which should be done to reach the goal:

- investigate current solution in Service Desk field;
- develop an architecture for system;
- develop the software solution itself;
- to test the effectiveness of the solution developed.

The object of the research is Service Desk software.

The subject of the research is to improve the process of ticket classification in customer technical support systems.

Scientific novelty of the research is to improve architectural solution for software of customer technical support to increase effectiveness of ticket classification step.

The practical value of the results is solved problem of huge number of tickets and requests which customer technical support receives. Also, the result is increased effectiveness of company and namely customer technical support.

Relationship with working with scientific programs, plans, topics. Work was performed at the Department of Informatics and Software Engineering of the National Technical University of Ukraine «Kyiv Polytechnic Institute. Igor Sikorsky».

Approbation. The scientific provisions of the dissertation were tested at the First All-Ukrainian Scientific and Practical Conference of Young Scientists and Students “Software engineering and information technologies” SoftTech-2021) - Kyiv. Section of informatics and software engineering.

Publications. The scientific provisions of the dissertation published in: Petrova Maryna Architectural solution of customer technical support system software / Petrova Maryna // Proceedings of First All-Ukrainian Scientific and Practical Conference of Young Scientists and Students “Software engineering and information technologies” SoftTech-2021) - Kyiv. Section of informatics and software engineering. — Kyiv. — 2021. November 22-26, 2020.

Keywords: INFORMATION TECHNOLOGY SERVICE MANAGEMENT, SERVICE DESK, REQUEST, SERVICE, CLASSIFICATION.

ЗМІСТ

ВСТУП

Протягом останніх років бізнес-процеси активно розвивалися під впливом інформаційних технологій. Епоха інформації, у свою чергу, спричинила розвиток програмного забезпечення. Завдяки цьому підприємства та організації почали швидше виводити свої додатки та сервіси на ринок. Відповідно, з підвищенням рівня компанії на ринку з'являється необхідність у збільшенні кількості співробітників. Як відомо, розширення штабу працівників потребує налагоджених процесів управління. Саме ці потреби і спричинили формування Процесів Управління IT-послугами. Проте, процеси потребують уніфікованого підходу та певних стандартів, які будуть незалежними відносно підприємств, компаній або постачальників послуг. Для того, щоб вирішити цю проблему, було розроблено єдину бібліотеку, що об'єднувала в собі найкращі підходи в сфері IT-послуг — ITIL (IT Infrastructure Library). Дана бібліотека містила описи, стандарти, задачі та процедури, які слугують основною для IT-сервісів. Базуючись на цій бібліотеці, передові компанії в сфері IT будували власні структуровані підходи до Управління IT-послугами. Це і стало поштовхом до створення IT Сервіс-менеджменту (IT Service Management — ITSM), якого потребувала індустрія для вирішення проблеми уніфікації в IT. Одним з ключових аспектів у IT-процесах є робота з запитами, що надходять у IT-організацію. Запити, що надходять на підприємство, потребують правильної реєстрації, відслідковування та вирішення. Саме ці задачі покликана вирішити служба Service Desk. Ця служба забезпечує контакт з IT-організацією і бере на себе розв'язання частини запитів до того, як вони потраплять до спеціалістів наступних рівнів підтримки. Завдяки цьому, шляхом фільтрації звернень, що надходять у Service Desk, зменшується навантаження на вищі лінії підтримки, що займаються вирішення проблем. У результаті, підвищується ефективність роботи компанії.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз об'єкта дослідження

Управління IT сервісами, тобто IT Сервіс Менеджмент випливає з таких понять як організація, якість та послуга. Ці поняття, об’єднуючись у єдину концепцію, створюють базу для ITSM. Оскільки ITSM є засобом для підвищення ефективності IT-послуг, то зазначені вище засади якості послуг мають бути подані на відповідному рівні. У даних системах якість послуги можна оцінити лише після того, як послугу надано замовнику. Тобто, якість визначається в основному очікуваннями замовника. Загалом же процес надання послуги утворюється двома складовими — виробництвом та споживанням, у яких беруть участь відповідно постачальник та замовник. Зазвичай можна наперед визначити, чи буде замовник задоволений послугами, якщо справджуються такі умови:

- послуга відповідає заздалегідь заданим очікуванням;
- послугу може бути надано повторно;
- послугу надано за адекватною (розумною) ціною.

Для якісного надання послуги і для можливості її своєчасного вдосконалення, важливо приділяти увагу такому важливому аспекту, як неперервний діалог із замовником. Завдяки цьому як постачальник послуги, так і її споживач, матимуть власне і взаємне бачення очікувань щодо наданої послуги. Коли послугу надано, настає момент, у який з’являється можливість оцінити якість послуги, про що було сказано вище. Тут важливу роль відіграє те, наскільки якісно було проведено роботу з очікуваннями. Якщо постачальник неперервно оцінював реакцію замовника на внесені в послугу зміни, то збільшується ймовірність того, що замовник буде задоволений результатом. У протилежному випадку, клієнт може звикнути до функціоналу, який спочатку здавався особливим та унікальним і у результаті надана послуга не справить такого враження, на яке очікував постачальник.

У будь-якому випадку, результати надання послуг можна використати з користю. Базуючись на результатах, можна спрогнозувати певні показники ефективності, а також визначити, які аспекти роботи варто вдосконалити, а якими можна знехтувати або зменшити їхній вплив.

Згадане вище поняття «адекватної (розумної) ціни» варто трактувати таким чином: це поняття є похідним і набуває сенсу після того, як сформовано так зване погодження щодо очікувань замовника. Щойно проведено роботу над таким погодженням, постачальнику необхідно оцінити власні витрати на сервіси, а також провести порівняльну статистику, базуючись на аналогічних продуктах на ринку.

Підсумовуючи тему якості послуг, варто підкреслити, що якщо постачальник в певний момент перевищує очікування замовника, а в інший момент не виправдовує їх, це викликатиме у замовника лише невдоволення. Найбільш надійною стратегією є стабільна відповідність очікуванням клієнта, без перекосу в бік перевищених або невиправданих очікувань.

Таким чином, робота з клієнтами та процес надання послуг є результатом рівномірного приділення уваги таким аспектам як якість, рівень послуг та їхня ціна. Для того, щоб забезпечити такий процес, необхідна зосередженість, уважність та координація усіх ланок процесу постачання.

Програмне забезпечення системи технічної підтримки користувачів є реалізацією функції першої лінії підтримки у організації. Така система дає можливість вирішувати значну частину запитів користувачів без необхідності у зверненні до спеціалістів вузького профілю. Дана система відповідає не лише за вирішення проблем зовнішніх користувачів підприємства, але й власне за запити співробітників компанії.

Служба Service Desk, що реалізує програмне забезпечення технічної підтримки користувачів, задіяна у широкому спектрі діяльності IT. Ось список базових процесів, які забезпечує Service Desk:

- управління інцидентами;
- управління релізами;
- управління конфігураціями;
- управління змінами;
- управління рівнем послуг.

Крім того, Service Desk може бути задіяна у інших процесах ITIL, наприклад в управлінні інфраструктурою, іншими словами — в операційній діяльності. Завдяки цьому програмне забезпечення системи технічної підтримки користувачів надає інформацію для замовників про сервіси, що обслуговуються. Окрім того, така система, забезпечуючи щоденний контакт користувачів, може надати інформацію про те, наскільки користувачі задоволені наданим підприємством сервісами.

Робота служби Service Desk, окрім програмного забезпечення, може бути реалізована за допомогою різних технічних засобів. Найбільш поширені варіанти — реалізація за допомогою телефону або електронної пошти. Менш поширені, але присутні на ринку варіанти — комунікація зі службою технічної підтримки за допомогою факсу, телефонів або чат-ботів.

Загалом, для забезпечення роботи служби технічної підтримки користувачів може бути використано такі засоби:

- телефонія — cti або voip;
- системи інтерактивного мовленнєвого меню;
- електронна пошта;
- факс;
- перенаправлення запитів на мобільні телефони, пейджери;
- інтранет та інтернет-портали.

Для того, аби користувачі не відчували дискомфорту при виникненні проблем і зверталися з цими проблемами до служби Service Desk, така служба має містити кваліфікованих співробітників, які працюватимуть послідовно, згідно з

усталеними процедурами. Такі процедури можуть бути побудовані на основі стандартного набору базових відповідей та стандартних анкетувань.

Запити, що надходять на Service Desk, можна класифікувати наступним чином:

- запити, викликані інцидентами у технічній інфраструктурі;
- питання, що виникли в процесі роботи з певними додатками;
- запити з проханням про уточнення статусу роботи над інцидентами або проблемами;
- запити про стандартні зміни;
- інші запити.

Служба Service Desk може бути організована безліччю різних варіантів. Залежно від способу організації, служба Service Desk може опрацьовувати лише частину запитів, а решту передавати на опрацювання вищим рівнями служби технічної підтримки. Також допускається варіант, при якому Service Desk опрацьовує всі запити, що надходять від користувачів.

Для того, щоб оцінити ефективність служби технічної підтримки користувачів, варто взяти до уваги такі параметри, як:

- швидкість реагування на запити;
- швидкість перенаправлення запитів на другу лінію підтримки;
- відновлення сервісу протягом допустимого часу;
- своєчасне інформування користувачів про можливі оновлення або недоступність сервісу.

Також є певні показники, ефективність яких визначається суб’єктивно, але які є не менш важливими у роботі технічної підтримки. Такими показниками є ввічливість співробітників служби Service Desk у процесі роботи над запитами користувачів, а також ефективність рекомендацій, наданих службою технічної підтримки.

Якщо зазначені вище параметри задовольняють потреби замовника та оговорені стандарти, можна вважати таку службу підтримки ефективною та успішною.

Взаємодія з замовниками, що споживають IT-послуги, це ключовий фокус, на якому має бути зосереджено увагу виробників IT-послуг. Так само важливо правильно взаємодіяти і з користувачами, що використовують IT-послуги у щоденній роботі або житті. Цей фокус має бути підкріплено правильно побудованими процесами взаємодії з замовниками та користувачами.

Базуючись на домовленостях замовника та постачальника послуг, укладається певний договір, що прийнято називати Погодженням Рівня Послуг (SLA — Service Level Agreement). Ці пропозиції фіксуються у рамках Процесу Управління Рівнем Послуг. Також у процесі взаємодії замовника з постачальником неодмінно виникає необхідність у внесенні змін до послуги, описаної в SLA. У такому разі створюється Запит на Зміни (Request for Change — RFC), який йде на обробку в рамках Процесу Управління Змінами (Change Management). Такі зміни, що знаходяться поза існуючими погодженнями, передаються Процесу Управління Рівнем Послуг. У той самий час, більшість питань, пов’язаних з роботою системи, може бути передано на Service Desk.

У даній роботі основною метою є вдосконалення процесу **класифікації заявок**, що надходять у службу технічної підтримки (Service Desk). Для того, аби поліпшити даний процес, варто заглибитися у його суть. Тому розглянемо процес класифікації інцидентів у діяльності Service Desk.

Мета класифікації інцидентів полягає в тому, щоб полегшити моніторинг та звітність шляхом визначення категорії інцидентів. У системі бажано мати якомога більше категорій, що будуть використані для класифікації, але для цього варто мати спеціалістів з високим рівнем кваліфікації. Також вдалим підходом для класифікації є висока гранулярність категорій. Мається на увазі те, що краще мати велику кількість підкатегорій, ніж багато категорій об’єднаних. Часто архітектори систем намагаються об’єднувати різні аспекти разом, наприклад тип, групу підтримки та

джерело укупі. Проте, варто розуміти, що подібний підхід з більшою ймовірністю створить плутанину, аніж допоможе у побудові якісної аналітики.

У класифікації ключовими є такі поняття:

- категорія;
- пріоритет;
- послуга;
- група підтримки;
- термін вирішення;
- ідентифікаційний номер інциденту;
- статус інциденту.

Ці поняття можна використати для класифікації інцидентів. Пропоную розглянути дані категорії детальніше.

Категорія — це ознака, яка присвоюється інцидентам у першу чергу. Категорія присвоюється базуючись або на джерелі інциденту, або на відповідній групі підтримки, якій може бути назначено даний інцидент. Ось найбільш поширені приклади базових категорій:

- основна система обробки. У ролі такої системи може виступати центр доступу, основний сервер або аплікація (додаток).
- мережа. Це може бути або певний hub, або сегмент. Так само це може бути адреса або маршрутизатор.
- робоча станція. Нею є клавіатура або дисковод, також це може бути мережева карта і навіть монітор.
- функціональність або спосіб використання. Це можуть бути різні процеси: починаючи з документації і закінчуючи можливостями системи. Наприклад, доступність або резервне копіювання. Також у ролі цієї категорії може виступати певна послуга або сервіс.

- процедури або спосіб організації. Тут допустимі різні варіанти способів комунікації: як нотифікації, так і певні запити, замовлення, або інші способи підтримки користувачів.
- запит про Обслуговування. Це певний загальний запит користувача в службу технічної підтримки. Користувач може здійснити запит на отримання певної документації або іншого виду інформації. Також він може попросити про консультацію, онлайн або оф лайн, письмову або в усному вигляді. Такий вид запиту може бути виокремлено в окрему процедуру та опрацьовано так, як реальний інцидент.

Окрім категорії, у деяких випадках інциденту може бути визначено і підкатегорію. У цьому може виникнути потреба, коли необхідно опрацьовувати великі кількості інцидентів, тобто тисячі та десятки тисяч запитів.

Пріоритет — показник, який прийнято визначати після того, як визначено категорію. Це роблять для того, щоб група, яка займається технічною підтримкою, знала, у якому порядку та у який термін необхідно виконувати певний інцидент. Пріоритет описується у вигляді номера, який розраховується за допомогою таких параметрів як терміновість розв’язання інциденту (тобто те, як швидко його необхідно вирішити) та величина впливу інциденту (які збитки буде нанесено системі, її замовникам або користувачам у випадку, коли даний інцидент не буде розв’язано швидко та якщо йому не буде приділено необхідної уваги). Пріоритет розраховується як добуток терміновості та величини впливу інциденту. Тобто, формула виглядатиме наступним чином: Пріоритет = Терміновість * Ступінь впливу.

Послуга — аспект, який можна визначити за допомогою переліку погоджень про рівень послуг (описаний у даному розділі Service Level Agreement — SLA). Цей перелік використовують для того, щоб визначити послуги, на які чинить вплив певний інцидент. Він дозволить визначити, який час є необхідним для того, щоб провести ескалацію послуг, визначених у Погодженні про Рівень Послуг.

Група підтримки — перелік осіб, відповідальних за розв’язання певного інциденту. Не завжди виходить так, що група Service Desk здатна вирішити інцидент, що надійшов у систему. У таких випадках група технічної підтримки займається делегуванням інциденту. В такому разі визначається група підтримки, що займатиметься вирішенням інциденту. Визначення цієї групи відбуватиметься на основі інформації щодо категорій інцидентів. Така інформація є базою для розподілу інцидентів або їхньої маршрутизації. У процесі розподілу категорій може знадобитися перегляд груп підтримки, а власне їхньої структури. Це важливо, адже такий розподіл чинить великий суттєвий вплив на Процес Управління Інцидентами, а саме його ефективність та продуктивність. Цю ефективність можна визначити за допомогою коефіцієнта, який являється одним з ключових параметрів ефективності (KPI — Key Performance Indicators). У Процесі Управління Інцидентами в якості такого показника може бути використано кількість звернень користувачів системи, які було неправильно визначено за категорією або за групою підтримки.

Термін вирішення — один з ключових показників, які фіксуються в системі. Служба технічної підтримки інформує користувача про те, за який максимальний термін буде вирішено його інцидент. Цей час розраховується за допомогою Погодження Про Рівень Послуг (Service Level Agreement — SLA) та пріоритету заявки.

Ідентифікаційний номер інциденту — запитувача інциденту інформують про те, який номер було призначено заявці або інциденту. За допомогою цього номера користувач матиме можливість відслідковувати роботу над інцидентом, його статус, а також визначити відповідальних осіб, з якими можна контактувати у разі виникнення додаткових прохань або питань з приводу інциденту. Тобто, за допомогою ідентифікаційного номера інциденту користувач матиме можливість при наступних зверненнях ідентифікувати інцидент, який він створив.

Статус інциденту — параметр, який присвоюється інциденту залежно від стану обробки інциденту. Тобто, статус інциденту вказує на те, який стан має

інцидент у процесі роботи над ним. Варіанти статусу можуть бути різноманітними. Зазвичай, найбільш наближеним до стандартного є такий набір статусів:

- новий — новий інцидент. Такий статус зазвичай автоматично призначається інцидентам, які щойно створено, тобто які тільки-но потрапили в систему;
- прийнято — інцидент прийнято в роботу. Такий статус свідчить про те, що служба технічної підтримки Service Desk отримала запит і вже проінформована про нього. На даному етапі служба технічної підтримки ознайомлюється з інформацією, зазначену в інциденті та обробляє її для того, щоб в подальшому запланувати виконання інциденту та визначити групу, яка займатиметься його розв'язанням;
- запланований — вирішення інциденту заплановано. На такому етапі визначається терміновість заявки та величину впливу такого інциденту. На основі визначених даних інциденту надається пріоритет. Більш детально про цей процес описано вище в роботі в контексті класифікації заявок за пріоритетом;
- призначений — інцидент надіслано відповідальній групі. В цей момент служба технічної підтримки Service Desk визначає, чи можливо їхніми власними силами вирішити даний інцидент. Якщо так, то такий інцидент залишається на обробці у служби Service Desk. Якщо ж група Service Desk не володіє компетенціями, необхідними для розв'язання даного інциденту, то такий запит передається в роботу службам підтримки вищих рівнів.
- активний — інцидент взято в роботу. Цей статус свідчить про те, що вже є певна група працівників, яка здатна вирішити даний інцидент і вона займається роботою над ним. Після цього статусу інцидент може бути або розв'язано або відкладено залежно від того, чи є блокуючі фактори, що утримують цей запит від того, щоб бути розв'язаним.
- відкладений — інцидент поставлено на паузу. Така ситуація може виникнути у тому випадку, коли є певні фактори, що не дають можливості

вирішити запит — тобто, блокують роботу групи, відповідальної за вирішення даного завдання. Наприклад, можуть виникати такі блокуючі фактори: у групі людей, відповідальних за вирішення заявки, є лише одна людина з необхідними компетенціями і на даний момент ця людина знаходиться у відпустці. В такій ситуації, інцидент переходить у статус Відкладений до певної дати, наприклад, дня виходу співробітника, що може вирішити задачу, з відпустки. Після того, як співробітник відходить з відпустки, статус інциденту знову переходить в Активний.

- вирішений (розв'язаний) — такий статус інцидент отримує, коли проведено роботу над вирішенням і інцидент можна передати на перевірку вирішення запитувачу, тобто користувачу. З цього статусу інцидент може перейти у статус Закритий, якщо запитувач підтверджує, що проблема, яку було описано в інциденті, є вирішеною. У протилежному випадку, якщо користувач надає інформацію про те, що проблема, з якою він звернувся, все ще відтворюється, статус інциденту знову змінюється на Активний, до тих пір, поки проблему не буде вирішено і не підтверджено цей факт запитувачем.
- закритий — фінальний статус інциденту, що свідчить про те, що інцидент вирішено і запитувач підтвердив, що його проблема, через яку було створено інцидент, більше не відтворюється.

1.2 Аналіз відомих продуктів для ServiceDesk

Дослідження успішних товарів досліджуваної предметної області на ринку це ключовий етап процесу створення архітектури комп'ютерного забезпечення. Це дозволяє досліджувати позитивні та негативні аспекти додатків, що існують. Не менш важливим є виокремлення основних напрямків розвитку у області. Завдяки цьому підвищується ймовірність задоволення потреб користувачів.

У даній роботі проведено порівняльний аналіз лідерів предметної області серед програмних засобів для відстеження інцидентів. Завдяки таким системам

служби ServiceDesk стають автоматичними. Програми для аналізу було обрано звертаючи увагу на їхню поширеність використання та відповідність сучасним задачам на ринку. Розглянемо такі системи як ServiceDesk (Jira), Mantis Bug Tracker та ZenDesk.

ZenDesk

ZenDesk – продукт, що допомагає у виставленні пріоритетів, відстеженні та обробці інцидентів, що запустила компанія Zendesk у Данії. На сьогодні головний офіс компанії знаходиться у Сан-Франциско. Підприємство займається створенням програмних продуктів, обслуговуванням користувачів різних сфер, таких як фінанси, ЗМІ і багато інших. Компанію ZenDesk запустили в якості стартапу, де управляли троє фрілансерів країн Скандинавії. Через кілька років кількість користувачів стартапу досягла відмітки у сто п’ятдесят тисяч клієнтів і всі вони користувалися платною підпискою.

Переваги Zendesk

Серед позитивних характеристик продукту Zendesk можна виділити низьку ймовірність втрати інформації, завдяки чому можна бути впевненим у цілісності даних та тому, що запити будуть доступні у будь-який час та з будь-якого девайсу. Крім того, вартість використання Zendesk нижча, ніж у популярних аналогів на ринку, що робить цю програму ще більш привабливою для вибору.

Недоліки Zendesk

У даному продукті, крім переваг є також і недоліки. Один із них це повільне завантаження програми у випадку, коли внесена велика кількість даних. В таких випадках можна зіштовхнутися з довгою відповіддю на запити до програми і виникає необхідність у повторному натисканні клавіш. Також є незручність у складному для сприйняття інтерфейсу програми, що не приносить бажаних результатів у користуванні.

Jira ServiceDesk

Jira ServiceDesk є програмою, яка допомагає у керуванні процесами ITSM і використовується для того, аби швидко надавати послуги і задовольняти потреби бізнесів. Цей додаток розробила команда компанії Atlassian, що займається розробками, які використовуються в управлінні проектів й інформаційних ресурсів. Цю організацію було засновано у Сіднеї. Вони стали відомими завдяки таким своїм продуктам як Jira (програма для автоматизації Service Desk) та Confluence (додаток для створення документації). Цю організацію також заснували студенти як стартап. Продукт ServiceDesk Jira спроектували як платформу, що є складником Jira і цим прагнули підвищити ефективність команди, що займається ServiceDesk процесами.

Переваги Jira ServiceDesk

Дане програмне забезпечення має велику кількість переваг. По-перше, даний застосунок має зручний користувацький інтерфейс. Крім того, швидкість роботи є вищою, ніж у конкурентних програм на ринку. Не менш важливим також є те, що Jira ServiceDesk є зручною у розгортанні і гнучкою, коли йдеться про конфігурацію типів заявок та інцидентів.

Недоліки Jira ServiceDesk

Говорячи про недоліки, що має даний продукт, варто звернути увагу на те, що Jira ServiceDesk є достатньо дорогою у використанні. Тобто для того, аби один користувач міг календарний місяць використовувати продукт, варто заплатити суму еквівалентну 20 доларам США. У той самий час можна знайти аналоги за більш дешевою ціною, наприклад 5 доларів. Більш того, можна знайти і такі додатки, що є безкоштовними у використанні.

Ще одна незручність виникла при використанні продукту Jira ServiceDesk — відсутня аналітика з інформації відвідувань додатку та статистика дій користувачів.

MantisBT

MantisBT — продукт, що допомагає відстежувати помилки і має відкритий вихідний код. Ця система дає можливість встановлювати зв'язок між тестувальниками та розробниками. Завдяки цьому можливо відслідкувати прогрес виконання робот над створеними інцидентами. Крім того, додаток має можливість налаштування конфігурації для того, щоб автоматизувати роботу технічної підтримки. Це дозволяє провести облік запитів по різним сферам. Продукт було започатковано програмним інженером Іто Кензабуро. Через певний час, завдяки тому, що продукт має вільний доступ до вихідного коду, функціонал поповнився вкладом спілки користувачів у мережі Інтернет.

Переваги MantisBT

З-поміж зазначених вище додатків, саме MantisBT має найбільшу кількість переваг. Дана система має зручний користувацький інтерфейс та високу швидкість роботи. Крім того, даний застосунок має не лише платну, але й безкоштовну версію. Проте, вартість платної версії MantisBT також не є надто високою. Також важливо зазначить, що і розгортання програми, і конфігурація типів інцидентів є гнучкою та зручною у використанні. І наостанок, у даній системі наявна функція збору аналітики по користувачам та їхнім діям.

Недоліки MantisBT

Незважаючи на те, що даний додаток є безкоштовним у використанні, у ньому присутні і деякі недоліки. Наприклад, відсутня можливість конфігурувати статуси інцидентів. Доступний лише статичний набір можливих типів. Є такі статуси як «очікує перевірки користувача», «призначено», «в роботі», «виконаний». Так само неможливо налаштовувати і послідовність переходу між статусами інцидентів. Ще одна незручність полягає у автоматичному оновленні сесії. Через це є висока ймовірність того, що інформацію буде втрачено.

У таблиці 1.1 можемо побачити інформацію зі зведеною порівняльною характеристикою даних рішень.

Таблиця 1.1 — Порівняльна характеристика існуючих рішень

	Zendesk	*Jira ServiceDesk*	*MantisBT*	*Дипломний проект*
Зручний UI	—	+	+	+
Висока швидкість роботи	—	+	+/—	+
Низька вартість використання	+/—	—	+	+
Наявність безкоштовної версії	—	—	+	+
Аналітика по користувачам	—	—	—	+
Висока ймовірність втрати інформації	+	—	+	—
Гнучка конфігурація типів заявок/інцидентів	—	+	—	+
Гнучке розгортання	—	+	—	+

1.3 Формулювання завдань

Одразу після того, як існуючі рішення програмного забезпечення служб Service Desk проаналізовано, можемо приступати до вдосконалення власної системи.

Бачимо, що всі зазначені системи, що використовуються для процесу Service Desk, дозволяють віднести інцидент до певного типу. Для того, щоб зробити процес визначення типу заявок більш ефективним, в даній роботі буде удосконалено процес класифікації заявок. Для цього в системі буде впроваджено розширені засоби для керування типами заявок.

Завдяки цьому матимемо можливість зменшити вплив недоліків продукту і підвищити вплив позитивних характеристик. Дане програмне забезпечення допомагає автоматизувати процеси ITSM у різних видах організацій.

Мета розробки

Дана система дозволяє керувати процесами ITSM, і конкретно Service Desk в контексті щоденних процесів надання IT-послуг.

Дане програмне забезпечення використовує поняття «групи». Варто виділити це поняття, оскільки воно є ключовим у даному контексті.

Продукт призначено для автоматизації роботи служби технічної підтримки у приватних або державних підприємствах під час виконання щоденних задач, пов’язаних з роботою над окремими проектами або з загальними корпоративними обов’язками.

Важливо звернути увагу на таке поняття, як «група», використане в контексті даного програмного забезпечення. У системі надано можливість виокремити групу користувачів, до якої належатиме певне коло людей, сконцентроване у роботі в окремій організації або над певним проектом, задачею, проблемою.

Основні ролі у системі такі:

- користувач (у групі)
- адміністратор (у групі)
- адміністратор (глобальний).

Попри перелічене вище, функціональність програми дає змогу конфігурувати ролі у рамках групи. В групах адміністратори призначають співробітників, що відповідають за розв’язання заявок, призначають модераторів груп та адміністраторів.

До продукту поставлено наступні вимоги:

- інструмент об’єднання користувачів у групи;
- інструмент додання користувачів у групи;
- інструмент формування користувацьких акаунтів й можливості входу в систему з їх допомогою;
- інструмент нотифікацій щодо реєстрації користувачів, приєднання у групи за посиланнями у електронних поштових листах;
- інструмент керування учасниками груп;
- інструмент для керування користувачами в системі:
- можливість заблокувати користувача;
- можливість надати роль користувачеві;
- інструмент статистики на основі інформації про відвідування додатку юзерами;
- інструмент створення інциденту на дошці групи;
- інструмент модифікації інциденту;
- інструмент для видалення інциденту;
- інструмент присвоєння статусів інцидентам;
- зручний користувацький інтерфейс;
- забезпечення цілісності інформації у випадку перебоїв браузера;
- зручна конфігурація параметрів системи.

План розробки

Оскільки метою розробки системи є підвищення ефективності продукту, було створено такий план імплементації архітектури програмного забезпечення служби Service Desk:

- зібрати вимоги системи;
- спроектувати систему;
- розробити програмний код програмного забезпечення;
- сформулювати документацію про користування продукту;
- протестувати продукт;
- сформулювати пояснювальну записку.

Висновки до розділу

В результаті написання першого розділу вдалося розглянути проблему архітектурного рішення для програмного забезпечення служб Service Desk. Також проаналізували представлені на ринку рішення і визначили переваги та недоліки популярних систем. На основі проведеного аналізу з’ясували, що необхідно створити власну систему, враховуючи незручності, що присутні у популярних аналогах систем технічної підтримки. Також провели підготовчі роботи для того, або почати проектувати власне рішення: поставили задачу, яку має виконувати додаток, описали мети створення продукту, описали головні ролі системи, що будуть задіяні у процесах даної системи. Порівняльний аналіз популярних систем Service Desk наведено у таблиці нижче:

2. РОЗРОБКА АРХІТЕКТУРНОГО РІШЕННЯ

2.1 Постановка завдань та вимог

Базуючись на потребах бізнесу, що виникають у зацікавлених сторін, архітектура програмного рішення має задовольняти такі вимоги:

- інженер має мати можливість класифікувати інцидент до певного типу, наявного в системі. а глобальний адміністратор має бути зданим додавати нові типи заявок залежно від потреб інженерів;
- має бути присутній дистанційний доступ до програмного забезпечення за допомогою мережі інтернет. також програмне рішення має бути доступне локально — розгорнуте на машині користувача;
- незалежно від пристрою, який використовує клієнт, графічний користувацький інтерфейс має бути доступним з однаковим набором базового функціоналу;
- дані, що вводить користувач, мають бути відповідним чином провалідовані, завдяки чому можна буде запобігти псуванню даних;
- для доступу до системи користувач обов’язково має бути авторизованим та автентифікованим, завдяки чому дані буде збережено і вони будуть у безпеці;
- конфігурація програмного рішення має забезпечувати персоналізацію даних та бути використаною для різних цілей, залежно від сценарію та групи людей, що продукт використовує;
- доступ до системи має бути стабільним у будь-який момент, щоб користувач міг прочитати, внести або змінити дані без ризику втратити інформацію, коли сесія браузера оновлюється.
- наведена нижче функціональність має бути забезпечена в додатку.

Базові функції ролей:

- роль матиме можливість автентифікуватися та авторизуватися;

- роль може долучитися до групи, використавши код з листа-нотифікації, який прийшов на пошту;
- роль може створювати групу і бути в ній адміністратором або модератором;
- користувач може переглядати список груп, у яких він виконує будь-яку роль;
- користувач може створити заявку і довільно вибрати тип даної заявки або інциденту;
- користувач може змінити або видалити інцидент;
- користувач може додати коментар до заявки.

Функції адміністраторів груп:

- адміністратор груп може запросити користувачів до своєї групи за допомогою електронного листа-запрошення;
- адміністратор групи може присвоїти ролі користувачам, які є членами цієї групи;
- адміністратор може присвоїти користувачам, що є членами групи, певні ролі;
- адміністратор може переглядати список користувачів у групі;
- адміністратор може видаляти користувачів з групи.

Функції адміністратора порталу (глобального):

- адміністратор може переглядати журнал відвідувань системи і бачити статистику по ним;
- адміністратор може очищувати записи в журналі статистики;
- адміністратор може призначати користувачам системні ролі (наприклад, роль звичайного користувача, адміністратора команди, глобального адміністратора);
- адміністратор може блокувати користувачів;
- адміністратор може створювати типи заявок;

- адміністратор може створювати ролі в групах, конфігурувати дозволи для ролей;
- адміністратор може змінювати імена, прізвища користувачів, а також прив'язану до них поштову адресу.

Опис MVC архітектури додатку

Одне з найбільш популярних архітектурних рішень для програмного забезпечення у вигляді веб-додатків є Model View Controller (MVC, перекладаючи з англійської — модель, представлення, контролер). Такий архітектурний патерн має основою поділ програмних компонентів на три складові. Модель — перша складова, що слугує ядром підходу MVC. Її основна функція — керувати даними системи, встановлювати програмну логіку і правила роботи системи.

Друга складова підходу MVC — представлення. Ним задається користувацький інтерфейс, базуючись на способі відображення даних. У представленні інформацію може бути подано у вигляді діграми, таблиць або завдяки побудові графіків. Варто звернути увагу і на те, що дані може бути подано у вигляді декількох представлень, зважаючи на вимоги бізнесу та його потреби.

Контролер — остання складова архітектурного підходу MVC, що використовується для того, щоб опрацьовувати вхідні дані, що генерує користувач за допомогою представлення. Контролер є інструментом взаємодії для користувача та інформації в моделі. Дана інформація проходить наступні етапи обробки: користувач вводить дані у браузері; ці вхідні дані потрапляють в модель. Потім зазвичай дані, що було введено, валідуються. Це робиться для того, щоб забезпечити цілісність продукту та гарантувати безпеку системи. Останній крок у етапах обробки це подання даних на модель. В моделі, базуючись на бізнес-логіці архітектурного рішення, дані проходять фінальну обробку.

Основна мета підходу MVC полягає в тому, щоб викорінити залежність програмних компонентів один від одного. Це забезпечується за допомогою розділення

сфер впливу для процесів представлення інформації, обробки даних та виведення результату обробки користувачеві за допомогою графічного користувацького інтерфейсу.

Переваги

Підхід MVC має наступні переваги. Перша перевага полягає у можливості використовувати код програми у різних місцях програмного рішення. Завдяки цьому є можливість покрити різні цілі бізнесу в архітектурному рішенні. Таку перевагу можна використати наступним чином: компонент представлення можна модифікувати, вносячи незначні зміни. Це адаптує представлення таким чином, щоб його можна було інтегрувати в інші системи або складові того самого продукту. З цього випливає факт, що розроблений на базі MVC програмний код можна легко модифікувати. Також бачимо ще одну перевагу — слабка зв'язаність (loose coupling) компонентів MVC. Як модель, так представлення і контролер може бути повторно використано або замінено іншою реалізацією, задіявши мінімальні зусилля.

Ще однією не менш важливою перевагою даного підходу є те, що використовуючи MVC можна збільшити ефективність тестування додатку, а також спростити цей процес завдяки можливості розподілу тестування моделі, представлення та контролера.

Сценарії роботи системи

Ключовою ланкою системи, навколо якої побудовано архітектурне рішення, це користувач. Для нього призначено даний сервіс. Щоб досягти позитивного відгуку від замовників та користувачів, варто враховувати їхні потреби при проектування архітектури системи. Для цього необхідно зібрати дані від зацікавлених сторін, провести їхній аналіз, і на основі інформації, отриманої в ході даного дослідження, можна формувати сценарії, які є можливими при використанні програми.

Як було описано в попередніх розділах, у архітектурному рішенні програмного забезпечення служби технічної підтримки ключовими є три основні ролі — адміністратор групи, користувач та глобальний адміністратор. Ці ролі мають різні сценарії використання програми, оскільки задачі у користувачів у цих ролях є різними. Нижче наведено потреби у функціоналі для даних ролей.

Базовий користувач

Інженер, що виконує роботу над інцидентами, виступає в ролі центрової фігури архітектурного рішення для програмного забезпечення служби технічної підтримки. Кожного дня такий користувач має певний спектр обов'язків:

- інженер консультує користувачів з приводу питань, що виникають з теми інформаційних технологій;
- інженер вирішує проблеми користувачів, допомагає у вирішенні інцидентів, що виникають в процесі експлуатації технічних продуктів в розпорядженні підприємства, в якому конкретний інженер-виконавець працює;
- інженер ідентифікує типи запитів;
- інженер визначає шляхи вирішення проблем, що зазначено в інциденті;
- інженер визначає пріоритети заявок;
- інженер відстежує етапи вирішення інцидентів;
- інженер контролює своєчасну обробку виконавцями взятих в роботу інцидентів.

Якщо враховувати задачі й обов'язки користувача програми, які виникають на щоденній основі, можна сформулювати сценарії роботи в програмі, які зазначено нижче.

Сценарій №1

В організації, що займається управлінням процесами інформаційних технологій, користувач «X» намагається здійснити вхід в систему, але бачить, що його акаунт заблоковано на інформаційному порталі корпорації. Співробітник не знає, з чим пов’язана проблема і як її вирішувати, тому він авторизується на порталі Service Desk своєї компанії, щоб повідомити інженерам про свою проблему. За допомогою посилання, яке користувач отримав на свою поштову адресу у листі запрошенні в той день, коли вийшов на роботу у цій корпорації, співробітник авторизується в системі. Після вдалої авторизації, виконаної за допомогою коду та посилання в листі, користувача буде направлено на сторінку «Технічні проблеми», яка знаходиться в групі, членом якої являється даний користувач. На цій сторінці можна знайти список заявок з проблемами апаратних та програмних засобів організації. В текстовому полі, де вноситься текст заявки, користувач описує свою проблему. Після цього він обирає тип запиту у випадаючому списку і створює заявку. В цей момент інженер «Y» служби Service Desk отримує доступ до щойно створеної заявки і має можливість долучитися до вирішення проблеми, описаної в даному інциденті. Даний сценарій можна описати за допомогою діаграми на рис. 2.1.

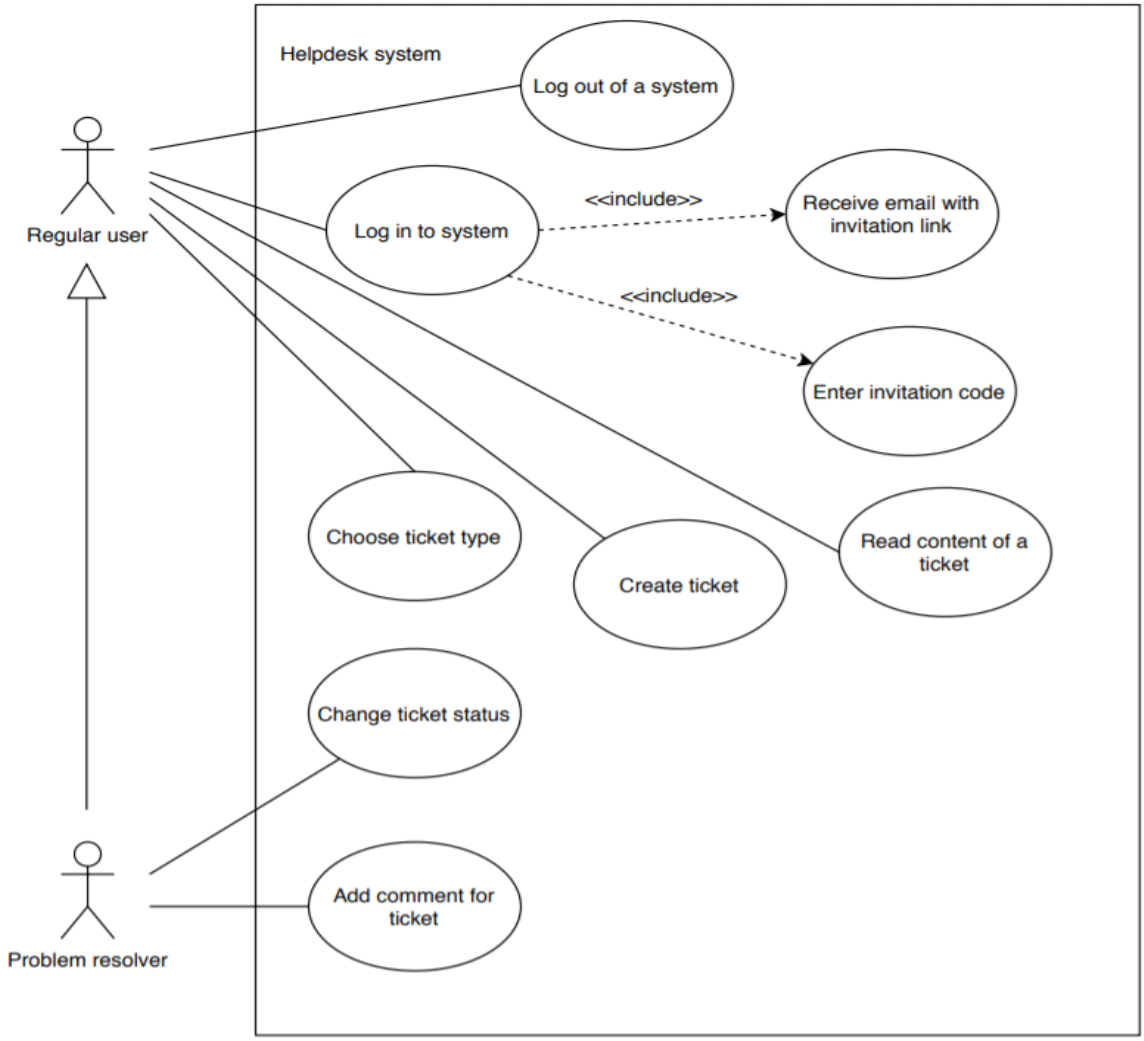


Рисунок 2.1 — Use-case діаграма сценарію використання №1

Адміністратор групи

У процесах керування IT послугами обробляється велика кількість різних інцидентів. Ці запити є різного типу, завдяки чому їх може обробляти інженер не лише з однієї сфери. Наприклад, інцидент може потрапити на опрацювання до спеціаліста з програмного забезпечення або до аналітика, до інженера з обладнання або до оператора кол-центру. Оскільки інженер певної спеціалізації має експертизу в обмеженому спектрі сфер, то необхідно цього інженера закріпити до певного інциденту, який підпадає під цю сферу. Здійснити закріплення інженера за певним типом заявок може адміністратор групи.

Сценарій №2

Перелік кроків, які необхідно виконати, аби створити заявку, описано у першому сценарії архітектури даного програмного рішення. Оскільки створений інцидент потребує вирішення, то він має надійти до інженерів, що можуть посприяти розв'язанню описаної проблеми. Як було сказано вище, розподілом заявок між спеціалістами займається адміністратор. Він здійснює авторизацію в системі та переходить в групу з інцидентами, пов'язаними з технікою, де відображено всі інциденти на цю тему. У списку інцидентів адміністратор бачить створену співробітником X заявку про блокування облікового запису, створення якої описано у сценарії використання №1. Таким типом заявок займаються інженери з інформаційної безпеки, до обов'язків яких і входить контроль облікових записів усіх співробітників підприємства. Проте, відвідавши сторінку списку учасників групи, яка займається технічними проблемами, адміністратор бачить, що серед учасників групи відсутній спеціаліст з інформаційної безпеки. Для того, щоб призначити такого спеціаліста відповідальним за виконання заявки, адміністратору необхідно додати його у групу з вирішення технічних проблем. Виконати це в системі можна за допомогою таких кроків: адміністратор має перейти у розділ управління членами групи з вирішення технічних проблем та натиснути кнопку додання користувача. Після того, як кнопку додання користувача натиснуто, адміністратор може ввести поштову адресу користувача, якого необхідно додати до цієї групи. Щойно адміністратор підтверджує запрошення цього користувача до групи, користувач отримує на електронну пошту листа-запрошення, в якому міститься посилання для долучення до групи з вирішення технічних проблем. Користувач переходить за цим посиланням і система перенаправляє його на сторінку групи з вирішення технічних проблем, де тепер він має всі необхідні доступи. Даний користувач, що є інженером з безпеки, переходить на сторінку інциденту про блокування облікового запису, переводить її у статус «В роботі», та лишає відповідний коментар до цієї заявки про те, що даний інцидент взято в роботу. По мірі того, як інженер працює над вирішенням проблеми, він має змогу тримати контакт з запитувачем, використовуючи функціонал

коментарів заявки. При цьому спеціаліст, що створив заявку, у коментарях має змогу надати інженеру всю інформацію, що може знадобитися. Даний процес відображено на Рис. 2.2.

Адміністратор системи (глобальний)

У ієрархії ролей архітектурного рішення для програмного забезпечення роботи служби технічної підтримки впроваджено роль, що має найбільшу кількість дозволів. Це роль глобального адміністратора, найвищої ланки у архітектурі даного програмного рішення. У цієї ролі є доступи до управління всім списком груп та користувачів системи. Також глобальний адміністратор може переглядати інформацію додатку щодо всіх дій, які було зроблено в ньому. Дана роль може оперувати всіма системними ресурсами: може редагувати заявки та їхні типи, призначати користувачам ролі, конфігурувати їхні дозволи. Тобто, роль глобального адміністратора є найбільш функціональною та має найбільше можливостей у архітектурному рішенні програмного забезпечення служби Service Desk.

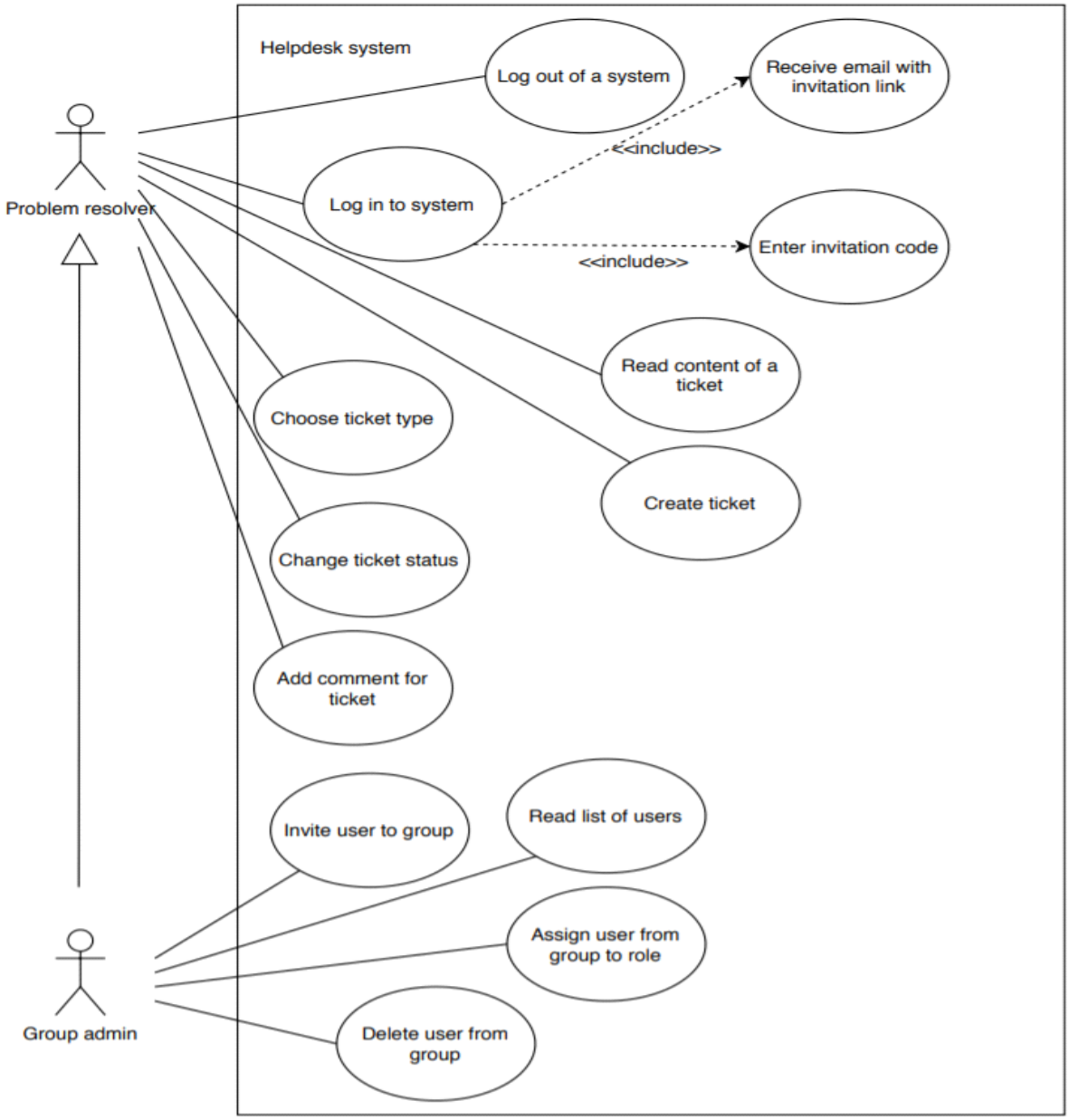


Рисунок 2.2 — Use-case діаграма сценарію адміністратора групи

Сценарій №3

Одним з основних обов’язків глобального адміністратора є керування типами заявок. Необхідність у такій дії може виникнути у випадку, описаному далі. Адміністратор групи проводить моніторинг заявок і бачить, що певний запит у програмному продукті потребує додаткової інформації для того, аби його було вирішено. Цією інформацією може не володіти ані автор інциденту, ані відповідальний за нього інженер. Наприклад, в ситуації, коли запитувач створює інцидент з проханням видати йому монітор, виникає необхідність у тому, щоб затвердити такий запит

керівником автора інциденту (для того, аби підрозділом було виділено бюджет на новий монітор). Проте, на даний момент керівник, що має узгодити інцидент, пішов у відпустку. Оскільки можливості отримати погодження від людини, що не має в певний момент доступу до системи, то і робота над такою заявкою призупиняється. Для того, аби позначити таку заявку як таку, роботу над якою призупинено, виникає потреба у новому статусі, що свідчитиме про те, що робота над інцидентом призупиняється на якийсь термін. У такому випадку адміністратор групи, що займається даним інцидентом, звертається до глобального адміністратора з тим, щоб внести в систему новий статус. Такий статус може мати назву «На утриманні». Щоб внести новий статус у систему, глобальний адміністратор здійснює вхід у систему. При авторизації користувача в ролі глобального адміністратора відбувається перенаправлення на сторінку, що містить навігаційну панель з точкою входу у адміністративну консоль додатку. Вхід у цю консоль можна здійснити за допомогою кнопки «Admin» на навігаційній панелі. При вході в адміністративну панель глобальному адміністратору доступне вікно типів заявок, де можна провести їхню модифікацію, створення, видалення. Глобальний адміністратор натискає кнопку «Створити» і таким чином додає новий тип заявки з назвою «На утриманні». Після того, як цей тип створено, адміністратор групи може призначити запиту про видачу монітора цей тип. На Рис. 2.3. зображено діаграму з алгоритмом роботи глобального адміністратора при описаному вище сценарії.

Сценарій №4

Контроль дій відвідувачів та їхньої активності є важливим у програмному забезпеченні, що використовує велика кількість користувачів. Щоб відслідкувати динамічну роботу користувачів у додатку, необхідно мати певний функціонал, що покриватиме такі потреби у архітектурному рішенні програмного забезпечення служби технічної підтримки. Для цього у даному рішенні для ролі глобального адміністратора спроектовано можливість переглядати журнал відвідувань системи користувачами. Також для того, аби проводити модерацію, в системі впроваджено

функціонал блокування користувачів. Необхідність у таких функціях може виникнути в описаному далі сценарії.

Під заявками в системі раптово почали з'являтися спам-коментарі з рекламою про спортивні ставки. Адміністратор групи, проводячи модерацію додатку, помітив одне з таких повідомлень і зробив припущення про те, що обліковий запис користувача, який є автором даних коментарів, було зламано. Найбільш ймовірно те, що даним акаунтом заволоділи несанкціоновано, а отже, такий обліковий запис варто якомога швидше заблокувати, щоб запобігти потенційним проблемам і ризикам з приводу безпеки і спаму в системі. Для того, аби заблокувати зламаний акаунт, адміністратор групи звертається до глобального адміністратора з запитом на блокування облікового запису користувача, який є автором спам-коментаря. Глобальному адміністратору надходить даний запит і наступною його дією буде пошук підтвердження тому, що активність користувача, вказаного в інциденті, дійсно була підозрілою. Глобальний адміністратор переглядає інформацію на вкладці журналу відвідувань системи і бачить, що дійсно на момент створення інциденту спостерігалася підозріла активність згаданого користувача. Далі для того, щоб заборонити підозрілому акаунту доступ до системи, глобальний адміністратор використовує функціонал, розташований у вкладці списку користувачів адміністративної консолі. У вікні пошуку глобальний адміністратор вводить ім'я акаунту, обирає його за допомогою кліку мишки і відкриває картку з інформацією про цього співробітника та функціями керування акаунтом. Навпроти імені користувача глобальний адміністратор бачить прапорець з підписом «Заблокувати» і робить цей прапорець активним. Тепер користувач, що є власником заблокованого акаунту, не зможе здійснити вхід у систему, оскільки при спробі здійснити логін за допомогою пошти та паролю він отримає повідомлення з інформацією про те, що даний акаунт заблоковано. Тому працювати в системі за допомогою цього акаунту користувач не зможе і необхідно буде створювати новий.

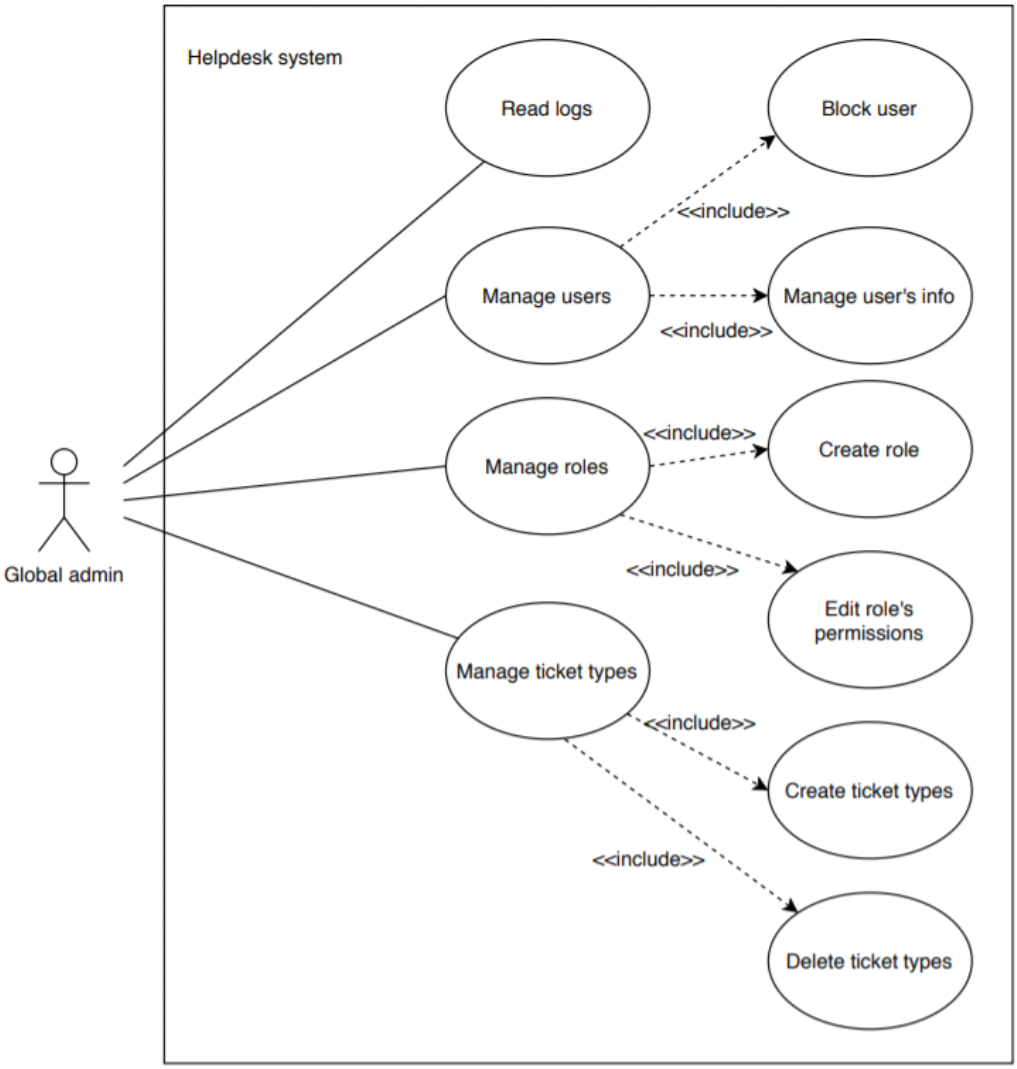


Рисунок 2.3 — Use-case діаграма сценарію глобального адміністратора

2.2 Розробка архітектурного рішення для інтерфейсу системи

Графічний інтерфейс це один з найбільш важливих компонентів системи. За допомогою графічного інтерфейсу користувачі мають можливість управляти даними системи, не маючи при цьому знань в області програмування. Не менш важливо і те, як користувач сприймає інформацію з графічного інтерфейсу на візуальному рівні. Саме тому графічний інтерфейс додатку має бути зручним у використанні та зрозумілим інтуїтивно. Для цього варто приділити велику кількість зусиль вигляду програми з боку користувача.

Архітектурне рішення для програмного забезпечення служби технічної підтримки містить у собі проект функціоналу для можливості створювати записи,

вводити дані в систему, авторизуватися та інше. На рівні графічного інтерфейсу задовільнити потребу у перелічених вище функціях можна за допомогою елементу, що має назву «текстове поле». Даний елемент зображено на рис. 2.4.

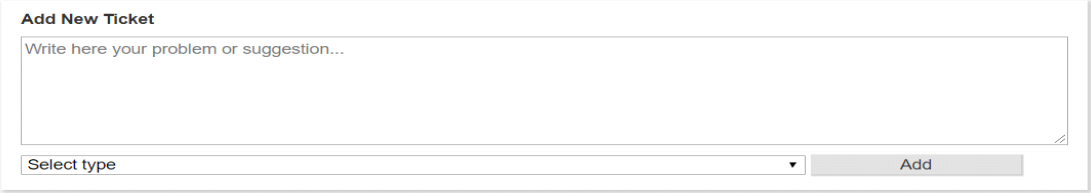


Рисунок 2.4 — Текстове поле в додатку

У програмному забезпеченні часто виникає необхідність у тому, щоб вибрати певні дані з доступних у даному контексті. Наприклад, коли користувач створює запит, він обирає з наявних типів той, що найкраще описує дану проблему. Необхідність у такому функціоналі реалізовано за допомогою випадаючого списку. Його зображено на рисунок 2.5.

Рисунок 2.5 — Приклад випадаючого списку

Також є певні дії в додатку, що потребують підтвердження з боку користувача. Дане підтвердження зазвичай реалізується за допомогою кнопок, на які користувач натискає за допомогою кліку мишки. Прикладами таких дій слугує робота з певними записами — їхнє створення, видалення і редагування. Зробивши певні дії над записом, користувач натискає кнопку «Зберегти». Також за допомогою кнопок користувач може переходити між модулями або вкладками. Приклад таких кнопок наведено на рис. 2.6.

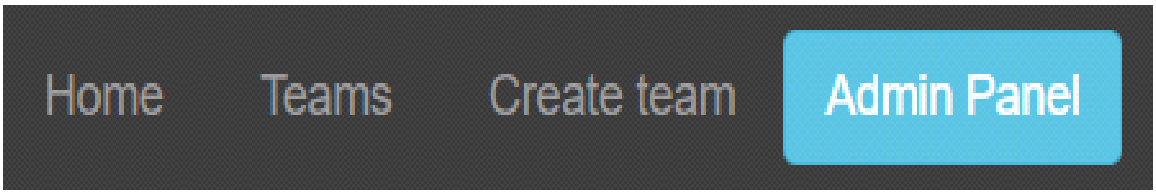


Рисунок 2.6 — Кнопки в системі

Говорячи про графічне відображення сутності «інцидент», важливим є об’єднання в єдине логічно-змістове ціле даних про опис інциденту, його автора, типу, дати створення та поточного статусу. Таке представлення інформації інкапсулюється у вигляді так званої «картки». Її вигляд в системі можна побачити на рис. 2.7.

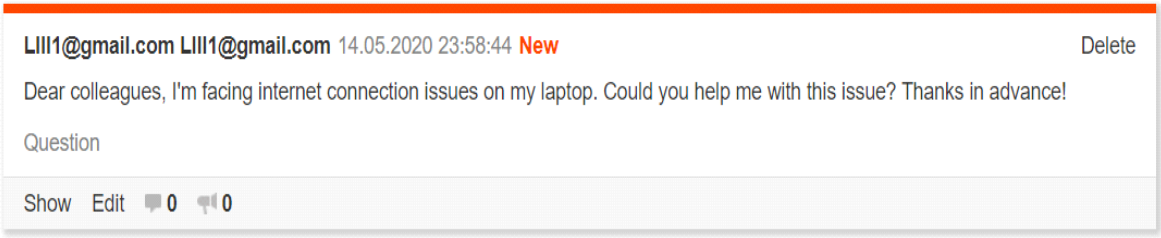


Рисунок 2.7 — Приклад заявки у вигляді картки

Висновки до розділу

Результатом роботи над даним розділом стало формулювання завдань і вимог архітектурного рішення програмного забезпечення для роботи Service Desk. У цьому розділі визначено інструментарій програми та описано її функціональність. Для проектування було вирішено обрати шаблон MVC, базуючись на проведеному аналізі поширених на ринку програмного забезпечення архітектурних підходів. Також було описано призначення даної архітектурної практики та її переваги. Як висновок, отримали чотири сценарії роботи програми з задіяними основними системними ролями. Завдяки цьому було уточнено мету створення даного архітектурного рішення системи та його призначення.

3 РЕАЛІЗАЦІЯ АРХІТЕКТУРНОГО РІШЕННЯ

3.1 Вибір платформи для програмного забезпечення

Для того, аби побудувати надійну та стійку базу для архітектурного рішення, варто вибрати таку платформу для розробки, яка враховуватиме всі сильні та слабкі сторони архітектури системи. Обираючи платформу, варто звернути увагу на всі ключові фактори. По-перше, важливо розуміти, скільки людей використовуватиме дане програмне забезпечення. По-друге, треба спрогнозувати, наскільки сильною буде залежність програмного забезпечення від операційної системи. Також варто проаналізувати, чи має дана програма бути доступної у мережі інтернет та інше.

Повертаючись до опису вимог, поставлених у другому розділі даної пояснювальної записки, можемо зробити наступні висновки: доступ до нашого програмного забезпечення має бути можливим за допомогою мережі інтернет. Також дана система за потреби має бути розгорнута на персональному комп’ютері користувача. Фокусуючись на останній вимозі, можна сказати, що її можна задовільнити і за допомогою десктопного додатку, і за допомогою платформи веб і при цьому не втратити якості. Щоправда, повертаючись до вимоги, пов’язаної з доступом до мережі інтернет, можемо зробити висновок, що тут більш доцільним є використання саме веб-додатку. Тому, підсумувавши описані вище пункти, зупиняємо свій вибір на тому, щоб використовувати платформу веб для розгортання даного програмного забезпечення.

Середовище, в якому буде виконуватися програма, є також важливим чинником, оскільки правильно вибране середовище забезпечуватиме те, як буде запущено програмний код і, відповідно, наскільки ефективно і на яких платформах працюватиме система. Одна з найбільш поширених платформ для розробки програмного забезпечення це .NET. Дана платформа, розроблена компанією Microsoft, має такі характеристики:

- підтримує декілька мов програмування. Загальномовне середовище виконання (CLR — Common Language Runtime), дозволяє платформі .NET запускати програми, написані на різних мовах програмування. Дана

платформа підтримує такі мови як C#, C++, .NET Visual Basic, F#, та інші мови, які є діалектами, прив'язаними до .NET, для прикладу, Delphi (.NET). Коли відбувається процес компіляції, написаний на якійсь із цих мов код компілюється в збірку, базою якої є спільна проміжна мова, що має назву Common Intermediate Language. Завдяки цій мові маємо можливість розробляти декілька модулів однієї й тої самої програми, використовуючи різні мови.

- кросплатформенність. Платформа .NET це така платформа, яку, з певними невеликими обмеженнями, можна переносити між різними середовищами. Версія даної платформи, що має назву .NET Core, сумісна з більшою частиною сучасних операційних систем, таких як Windows, Linux, MacOS. Тобто, на основі платформи .NET, можна використовувати різні платформи та технології, пишучи при цьому код, наприклад, на найбільш популярній мові даної платформи — C#. Написані цією мовою програми будуть доступні на великому різноманітті мобільних платформ Android, iOS і навіть Tizen. І на найбільш популярних платформах для персональних комп’ютерів Windows, Linux, MacOS.
- високофункціональна бібліотека класів. Платформа .NET має спільну для всіх мов програмування, які вона підтримує, бібліотеку класів. Не має значення, який додаток розробляється мовою, яку підтримує дана платформа (наприклад, C#) — чи то текстовий редактор, чи чат для веб-сайту, або навіть повноцінний веб-сайт, у будь-якому випадку при написанні програмного коду для цих додатків буде задіяно бібліотеку класів .NET.
- розмаїття технологій. При побудові додатків, з якими працюють .NET розробники, в якості бази вони використовують базову бібліотеку класів та загальномовне середовище виконання. Дана база містить різні технології для різних модулів системи. Наприклад, такі технології як ADO.NET і Entity Framework Core використовуються для того, аби працювати з базами даних. Технології WPF і UWP використовуються для розробки комплексних графічних додатків, що мають насичений інтерфейс. Windows Forms

— для базових графічних додатків з простим функціоналом. Також можна знайти бібліотеки класів, напрямлені на розробку веб-сайтів — ASP.NET або для мобільних додатків — це бібліотека класів Xamarin.

Остання, але не менш важлива особливість платформи .NET та мови C# — це можливість очищення ресурсів, що відбувається автоматично у системі. Завдяки цьому розробник виграє велику кількість часу, оскільки зникає необхідність піклуватися про те, яким чином і в який момент має бути звільнено пам'ять у програмі, в той час як у інших мовах програмування розробник самостійно має слідкувати за очищенням ресурсів. Фреймворк .NET полегшує роботу програмному інженеру завдяки тому, що платформа самостійно займається очисткою ресурсів і вивільненням пам'яті, використовуючи вбудований інструмент, що має назву «збирач сміття».

Вибір мов програмування

На сьогодні на ринку програмного забезпечення наявна велика кількість різноманітних мов програмування. Велике різноманіття мов ускладнює вибір мов для розробки кожного конкретного програмного забезпечення, тому у даному виборі варто бути сумлінним, адже правильно обрана мова програмування впливає на те, наскільки легко буде підтримуватися продукт, а також на те, на яких платформах можна буде запустити додаток. Також від мови програмування залежить і швидкість розробки, оскільки різні мови мають різний поріг входження й криву навчання. Вибір мов програмування значно впливає на те, чи будуть задоволені потреби бізнесу та очікування від продукту загалом. Також це важливо тому, що цілі програмного продукту різняться залежно від специфіки продукту і його термінів, а кожна з мов програмування має свої переваги та недоліки, що на цілі впливають.

У розділі проектування вимог та завдань ми визначили, що архітектурне рішення програмного забезпечення служби технічної підтримки варто реалізовувати в форматі веб-додатку. Однією з найбільш популярних мов для проектування серверної частини веб-додатків є мова C#. Вона є однією з найбільш потужних мов на сьогодні, швидко розвивається і користується високим попитом в галузі управління IT-послугами. Цією мовою на даний момент написано великий спектр програм: і

веб-сервіси, і десктопні додатки, і комплексні веб-сайти, і портали, якими користуються щосекунди мільйони користувачів — безліч таких продуктів написано мовою C#. Дана мова є достатньо молодою, порівнюючи її з аналогічними за функціональністю іншими популярними мовами. Але і мова C# має свою історію. Цю мову було випущено у 2002 році у лютому, разом з середовищем програмування компанії Microsoft, Microsoft Visual Studio .NET. Поточна версія C# 10.0 вийшла у поточному, 2021 році у листопаді.

C# є мовою, що базується на об'єктно-орієнтованому програмуванні, і за своїми особливостями схожа на мови, що були її попередниками — Java і C ++. C# є статично типізованою, підтримує такі бази об'єктно-орієнтованого програмування, як успадкування та поліморфізм, а також дозволяє використовувати перевантаження операторів. За допомогою цієї мови, а саме її властивості підтримувати об'єктно-орієнтоване програмування, стає можливим будувати комплексні додатки, які є масштабними і масштабованими, гнучкими і розширюваними. Мова C# і досі активно розвивається, а кожна її версія впроваджує все більше корисних можливостей.

Для того, аби зрозуміти, чи покриває дана мова вимоги, виставлені до функціоналу системи, необхідно розглянути мову C# в контексті її переваг та недоліків.

Переваги:

- мова C# повністю інтегрована з Windows. Не потрібно ніяким особливим чином конфігурувати програми, написані використовуючи мову C# для того, аби запустити її в середовищі Windows. Якщо сервер, на якому буде розгортатися програма, або робоча машина підтримує платформу .NET, перехід від розробки програми до запуску програмного продукту буде плавним — і декстопний додаток для персональних комп'ютерів, і Windows-сервіс, і веб-додаток можна буде без зайвих зусиль розгорнути в мережі підприємства;
- легко знайти спеціалістів, що володіють даною мовою. Оскільки C# це дуже поширена мова, то на ринку програмного забезпечення не виникає проблеми в тому, аби знайти нових розробників, якщо зростає потреба

бізнесу у додаткових ресурсах, а бізнес стрімко розвивається. Тому достатньо просто знайти розробників C# навіть на різний тип працевлаштування. І навіть якщо в певний момент розробників C# знайти не вдасться, то завжди можна буде знайти розробника, що володіє мовою Java, оскільки вона є дуже схожою за синтаксисом до мови C#. Такий розробник стане бізнесу в нагоді, якщо виникне потреба в тому, щоб запустити версію додатку під мобільні пристрої;

- C# це компільована мова. Тобто, код програми, написаний мовою C#, зберігатиметься на загальнодоступному сервері у двійковому форматі. Це буде корисним у випадку несанкціонованого доступу хакером до серверу. В такому випадку хакер, отримавши доступ до серверу, де зберігається програмне забезпечення, не зможе автоматично отримати доступ до вихідного коду програми. На відміну від C#, в деяких інших мовах програмування, що не є компільованими зловмисники отримують доступ до вихідного коду продукту щойно потрапляють на сервер. А це означає, що хакер зможе отримати доступ і до таких важливих даних, як паролі до баз даних. Якщо ж на сервері знаходиться код, що розроблявся мовою C#, хакеру для доступу до критичних модулів продукту необхідно буде попередньо декомпілювати програмний код, що зможе дати певний виграш у часі для того, аби відновити безпеку сервера у випадку зламу.
- мова C# підтримує об’єктно-орієнтовану парадигму програмування. Так само цю модель підтримує платформа .NET. Об’єктно-орієнтоване програмування засноване на тому, щоб розділяти програмне забезпечення на більш гранулярні фрагменти. Такими фрагментами в подальшому легше керувати та їх легше поєднувати між собою для створення нових сутностей або алгоритмів. У дані парадигмі прийнято розділяти дані на певні об’єкти, сутності. Ці сутності містять в собі і дані, реалізовані полями, і поведінку, впроваджену за допомогою методів. Структура таких сутностей описана за допомогою класів. Така модульність дозволяє окреслити поведінку сутностей та взаємодії об’єктів, не використовуючи при цьому внутрішні

атрибути об’єкта. Парадигма об’єктно-орієнтованого програмування спрощує розробку завдяки тому, що код стає більш керованим, його простіше тестувати, змінювати та використовувати повторно. Завдяки цьому зникає потреба у зайвому програмуванні, а коду для підтримки розробниками стає менше. Платформа .NET, у свою чергу, допомагає в тому, що написані під неї компоненти можна цілісно використовувати повторно, і цим економити час. А економія часу призводить до зменшення витрат на розробку, що є суттєвою перевагою з точки зору бізнесу;

- мова C# володіє простою та надійною системою кешування. При роботі з даними часто виникає можливість у тому, щоб тимчасово зберігати їх та повторно використовувати за потреби, коли необхідний швидкий доступ до них. Стандартний спосіб повторного запиту інформації з джерела, наприклад бази даних, може потребувати багато часу, що є ударом по продуктивності системи. Для цього дублікати інформації зберігаються у системах кешування. У платформі .NET надійна та проста у використанні. Крім того, система кешування тут спроектована таким чином, щоб її можна було розширювати та створювати власну реалізацію залежно від цілей продукту. Власна реалізація допомагає підвищити продуктивність додатку та забезпечити його масштабованість як для серверних додатків Windows, так і для клієнтських.

Недоліки:

- мова C# залежна від платформи .NET. Для того, аби розгорнути код, написаний мовою C#, сервер, на якому розгортатиметься програма має містити фреймворк .NET. А для того, щоб розгорнути даний фреймворк на сервері, сервер має бути Windows-сервером (це стосується старих версій фреймворка). Тобто, можемо зробити висновок, що для того, аби запустити будь-яку програму на .NET Framework, критично необхідно використовувати платформу Windows. Наразі на ринку є велика кількість нових компаній, що надають перевагу серверам Linux через їхню низьку ціну за підтримку та використання. Тобто, деякі нові підприємства не мають

хостингів Windows. Ще одна проблема полягає в тому, що Microsoft більше не підтримує старіші версії .NET Framework у останніх оновленнях операційних систем. Тобто, сервери, що є більш старими, наприклад Windows 2000, здатні підтримувати тільки такі програми, що написані на .NET 2.0. Проте, є підприємства і організації, що надають перевагу підтримці старіших операційних систем, хоча на перший погляд встановлення операційних систем попередніх поколінь не виглядає як доцільне рішення. Але таким чином підприємства потенційно уникають проблем, що можуть виникнути у випадку, коли необхідно оновити платформу. А оновлення архітектури та інфраструктури серверів потребує додаткового тестування та погодження перед тим, як бути розгорнутими і це впливає на час розробки і, відповідно, на витрати бізнесу;

- компільованість мови C#, яка вище була описана у розділі переваг. Дана властивість є двозначною, оскільки здатність мови до компіляції накладає певні обмеження. По-перше, код, що потребує компіляції, має проходити її кожного разу навіть при внесенні незначних змін. Через це з кодом стає складно працювати. Така потреба може призводити до великої кількості помилок у випадку, коли розробники ретельно не перевірили навіть незначну зміну в програмному коді;
- для мови C# обмежено об’єктно-реляційну підтримку. Оскільки платформа .NET базується на об’єктно-орієнтованій моделі програмування, основних фокус зосереджено не на діях і поведінці даних, а на самих об’єктах. Також увага зміщується з логіки на структури даних. Зв’язок між програмними сутностями та даними в середовищах .NET Framework та Core реалізовано за допомогою фреймворку Entity Framework, що виступає посередником у побудові об'єктно-реляційного зв’язку між .NET Framework та базами даних реляціного підходу. На думку деяких інженерів, що спеціалізуються на програмному забезпеченні, Entity Framework поступається у гнучкості іншим підходом та не є сумісним з великою кількістю рішень баз даних. Через це існує ризик того, що даний фреймворк

може виявитися таким, що не підтримує нові рішення баз даних. Також є одна додаткова проблема, суть якої в тому, що в певний момент корпорація Microsoft може припинити підтримувати даний фреймворк. Проте, розробник у праві обирати ORM на свій розсуд, а отже Entity Framework не можна цілком вважати таким, що можна віднести до недоліків.

Провівши аналіз мови C# на переваги та недоліки, можна заключити результат про те, що для архітектурного рішення програмного забезпечення служби технічної підтримки дана мова є оптимальним варіантом. Більш того, якщо організація, для якої буде впроваджено дане програмне забезпечення, надає перевагу робочим серверам та станціям Windows, то використання разом із ними платформи .NET — буде найбільш вдалою інтеграцією. Дана мова може бути використана для різних продуктів, і для Windows Services, і для веб-додатків. Вона буде найкращим варіантом для використання у середовищі Windows, і буде допомагати масштабованості в часи зростання та розвитку бізнесу. Мова C# безперечно має значно більше переваг, аніж недоліків, а ці недоліки не є складним усунути, правильно спроектувавши інфраструктуру та архітектуру середовищ.

Вибір бази даних

Проектуючи архітектурне рішення програмного забезпечення важливо визначити тип сховища даних, який буде використано для побудови системи. Згідно з вимогами до даного програмного продукту, однією з задач системи буде обслуговування великої кількості користувачів. Тому наш програмний продукт має бути розрахований на те, щоб зберігати велику кількість даних. В такому випадку найбільш вдалим варіантом сховища даних буде саме база даних. Наразі на ринку програмного забезпечення можемо обирати з двох типів — реляційних або нереляційних баз даних. Особливістю реляційних баз даних є те, що вони основою мають чітку структуру. Ця структура спрощує ідентифікацію блоків даних і дозволяє налаштувати зв’язки між цими блоками. В свою чергу, нереляційні бази даних основною метою мають зберігання даних, тому в такому типі баз даних нехтують структурою. У нереляційних базах даних відбувається оптимізація даних під конкретні вимоги, що залежать від типу даних, які зберігаються у сховищі. Часто можна

зустріти нереляційні бази даних, у яких зберігаються документи різних форматів (найчастіше JSON), інформація в парах ключ-значення, а також у виглядів певних графів з вершин та ребр.

Розглянемо детальніше переваги й недоліки обох типів баз даних для того, щоб визначитися з типом бази даних для нашого архітектурного рішення.

Переваги реляційних баз даних:

- в реляційних базах дані зберігаються у структурованому вигляді. Така архітектура буде корисною у випадку, коли система опрацьовує різні набори даних, які мають певні зв'язки одне з одним. В такому випадку виникає необхідність структурувати дані певним чином;
- підтримка транзакцій. Суть транзакцій та транзакційної системи полягає у певному наборі правил, націлених на збереження цілісності інформації. У випадку, коли відмовляє енергопостачання на серверах підприємства, дані все одно залишаються цілісними, якщо бази даних підтримують транзакційність. Більшість реляційних баз даних транзакційну систему підтримують;
- реляційні бази даних мають оператор для об'єднання даних. Такий оператор допомагає здійснювати пошук даних у ситуаціях, коли необхідно оперувати набором даних, що складається з колонок різних таблиць. Вся необхідна у такому випадку інформація може бути отримана з бази даних за допомогою оператора об'єднання даних. В нереляційних же базах даних здійснити це не вдасться, бо вони таким функціоналом не володіють. Тобто, якщо в нереляційній базі виникне необхідність отримати набір даних з різних блоків, то доведеться відповідно робити кілька запитів.

Недоліки реляційних баз даних:

- реляційні бази даних важко піддаються масштабуванню. Для того, аби масштабувати реляційну базу даних, розробникам необхідно мати на підприємствах потужні сервера. Підтримка таких серверів є фінансово складною. Для того, аби більш простим способом масштабувати базу даних не певній кількості серверів, необхідно розділити її на декілька таблиць. Проте,

підтримка таблиць однієї бази на різних джерелах є все ще достатньо складною;

- реляційна база накладає певні обмеження, спричинені структурою таблиці. Оскільки різні сутності не завжди підпадають під одну структуру таблиці, то, відповідно, для нової сутності необхідно буде створити нову таблицю в базі даних. А нові структури в базі даних потребують додаткової підтримки та обслуговування.

Переваги нереляційних баз даних:

- нереляційні бази даних гнучкі. Для того, щоб використовувати нереляційні бази даних, немає необхідності у тому, аби підтримувати певну структуру даних. Для даних, які не потребують певної структури, а можуть зберігатися в невпорядкованому вигляді, можна використовувати нереляційні бази — наприклад, для даних, що містять багато вкладених об'єктів. Такий тип баз даних дозволяє знехтувати структуруванням даних і зосередитися лише на їхньому зберіганні. Перевагою є те, що зникає потреба дотримуватися певного формату для стовпців та рядків;
- нереляційні бази даних є більш швидкими. Оскільки в реляційних базах даних певні ресурси йдуть на те, щоб підтримувати відповідність властивостям ACID, то швидкість просідає. Хоча реляційні бази даних не можна назвати повільними, проте порівняно з нереляційними різниця відчувається у випадку, коли відбувається вибірка набору даних. Проте, в нереляційних базах даних є певні складнощі з тим, щоб об’єднувати дані з різних таблиць, тому варто зважати на цей момент при відмові від реляційних баз даних на користь нереляційних і їхньої швидкості, адже для того, аби вибрати дані з нереляційної бази даних, необхідно буде зробити декілька запитів, що може не дати вагомого виграшу у швидкості;
- нереляційні бази даних є простими у використанні. Дані у такому типі баз даних представлені у вигляді об’єктів, завдяки чому покращується сприйняття інформації, особливо у випадку, коли людина обізнана з тим, як дані з нереляційної бази даних співвідносяться з їхнім представленням у коді.

Тобто, ще одною перевагою таких баз даних є те, що навчитися їх використовувати простіше, ніж реляційним. Це може зіграти на користь у випадку, коли виникає необхідність у розробці певного продукту з нуля розробниками, що не мають навичок у роботі з базами даних.

Недоліки нереляційних баз даних:

- відкритий вихідний код. Найбільший недолік, котрий водночас є і перевагою даного типу баз даних, це можливість вільного доступу до їхнього вихідного коду. Через це кількість усталених стандартів для нереляційних баз даних є невеликою, а тому кожна з таких баз даних має власні шаблоні та правила, що не схожі між різними базами даних;
- у нереляційних базах даних не передбачено функціонал роботи зі збереженими процедурами;
- у нереляційних базах даних відсутній інструмент доступу та користування з графічним користувацьким інтерфейсом, що є зручним;
- важко знайти розробників експертного рівня, що мають досвід роботи з нереляційними базами даних. Дана технологія молодша за технологію реляційних баз даних і з’явилася на ринку програмного забезпечення недавно. Тому на ринку дуже мало експертів, що працюють з базами даних з моменту їхньої появи на ринку.

Базуючись на аналізі переваг і недоліків описаних вище баз даних можемо дійти висновку про те, що для архітектурного рішення програмного забезпечення роботи служби Service Desk буде доцільно використовувати реляційну базу даних. Такий вибір буде найбільш вдалим, враховуючи вимоги та завдання розроблюваної системи. Наступним кроком у проектуванні архітектурного рішення буде визначитися з конкретною реляційною базою даних. У попередньому розділі ми обрали мову програмування, якою буде написано програмний код системи, тому вибирати базу даних можемо відштовхуючись від того, яка буде сумісною з обраною нами мовою C#. Розглянемо переваги та недоліки найбільш популярної сумісної з C# реляційної бази даних Microsoft SQL Server.

Переваги Microsoft SQL Server:

- у Microsoft SQL Server високий рівень безпеки даних. Однією з основних цілей виробники Microsoft SQL Server вважають безпеку даних. Корисним для контролю безпеки баз даних цього типу є інструмент, за допомогою якого можна адмініструвати дані бази даних, що має назву Microsoft SQL Server Management Studio. Такий інструмент дозволяє вносити зміни в структури таблиць, а також працювати з елементами даних та їхніми функціями, щоб забезпечити захист даних, які зберігаються в даній базі. У базах даних, що містять конфіденційну інформацію, наприклад дані про користувача та його паролі, першим пріоритетом має бути саме безпека даних та їхня цілісність;
- Microsoft SQL Server є простою у конфігурації. Установка даної бази даних є достатньо простою, на відміну від деяких інших баз даних та програмного забезпечення. Для того, аби встановити Microsoft SQL Server, необхідно мати лише файл для установки і немає необхідності володіти окремим набором інструментів або глибокими знаннями у програмному забезпеченні. Крім того, Microsoft SQL Server оновлюється автоматично. При встановленні Microsoft SQL Server користувач за потреби встановлює додаткові компоненти даної бази даних, що йому необхідні, щоб задовольнити потреби бізнесу, не вдаючись до додаткових ускладнених процесів. Тому Microsoft SQL Server за параметром зручності є точно безпрограшним варіантом для того, аби управляти базами даних;
- у Microsoft SQL Server оптимізовано механізм збереження даних. У цій базі даних немає потреби володіти окремим сховищем даних того самого типу як і основна база даних при використанні іншого пристрою. У базі даних Microsoft SQL Server забезпечено можливість віддаленого доступу до таблиць, тому можна управляти своєю базою даних з різних пристроїв. Сервер, на якому зберігається база даних, також володіє можливістю віддаленого доступу. Завдяки цьому забезпечується ефективне та просте керування даними, при цьому значно знижується потреба у тому, аби усувати несправності і займатися зайвим технічним обслуговуванням. Тому

розробники баз даних отримують додатковий час для того, аби займатися більш важливими факторами для зростання бізнесу. А Microsoft SQL Server допоможе в організаційних питаннях та питаннях підтримки;

- у Microsoft SQL Server вбудовано функціонал для відновлення даних. У випадку, коли на підприємстві відбуваються перебої з електропостачанням або сервери випадково вимикаються, зростає ризик того, що дані буде втрачено чи пошкоджено. Це спричинить велику проблему тим підприємствам, які завчасно не подбали про збереження резервних копій даних. База даних Microsoft SQL Server має вбудовані функції для відновлення даних та попереднього створення їхніх резервних копій. У результаті, інженери та замовники впевнені у тому, що їхні дані в безпеці та можуть бути відновлені у будь-який момент. Основними інструментами для забезпечення такої підтримки даних є функціонал періодичного резервного копіювання даних, кешування даних та заповнення журналу дій, що відбуваються у базі даних.

Недоліки Microsoft SQL Server:

- у Microsoft SQL Server додатковий функціонал є платним. Хоча базова версія Microsoft SQL Server є безкоштовною та у вільному доступі, проте якщо розробнику необхідний функціонал, що не входить у базовий, йому необхідно буде купувати платну версію. Це буде додатковою інвестицією у вдосконалення додатків, на яку не кожне підприємство готове піти;
- Microsoft SQL Server має обмежену сумісність з іншими програмними засобами, окрім тих, що розроблені компанією Microsoft. Тому, коли підприємство вирішить відмовитися від використання інфраструктури Microsoft, йому необхідно буде інвестувати додаткові кошти у придбання нового програмного забезпечення, адже зі сторонньою інфраструктурою база даних Microsoft SQL Server може бути несумісною. Проте, такі вкладення коштів, зроблять використання продуктів більш гнучким;

- нові версії Microsoft SQL Server мають певні апаратні обмеження, оскільки для їхнього запуску необхідно мати новітні технології та прогресивне програмне забезпечення. Тому, якщо апаратна інфраструктура підприємства має здебільшого старе обладнання, то для того, аби проваджувати Microsoft SQL Server, підприємству необхідно буде поповнити свій арсенал, використавши обчислювальні машини нового покоління. Також у випадку, коли програмне забезпечення оперує великою кількістю даних, необхідно буде провести збільшення простору жорсткого диску сервера.

У даному розділі ми детально проаналізували базу даних Microsoft SQL Server і можемо дійти висновку, що такий тип бази даних це доцільний варіант для побудови архітектурного рішення програмного забезпечення служби технічної підтримки.

Вибір платформи для серверної частини

Для того, аби якісно спроектувати серверну частину архітектурного рішення, необхідно обрати, на базі якого набору бібліотек або фреймворку буде розроблено систему. У арсеналі фреймворку .NET є велика кількість бібліотек, орієнтованих на створення додатків під різні операційні системи. Наприклад, наявні інструменти бібліотек для десктопних аплікацій — Windows Presentation Foundation, Windows Forms; Unity — для побудови графічних ядер, Xamarin — для мобільних додатків і ASP.NET — для веб-додатків. У розділах, описаних вище, ми дійшли рішення, що наша система буде розроблена як веб-додаток, тому для розробки системи нам необхідно буде обрати таку бібліотеку, що забезпечить нам роботу з браузером та мережею інтернет. Для таких цілей доцільно буде використати фреймворк ASP.NET, абревіатура якого розшифровується як Active Server Pages .NET. Ця платформа, розроблена компанією Майкрософт для створення веб-додатків, містить в собі інструменти для створення веб-сервісів, моделей програмування та програмної інфраструктури. Дана технологія ASP.NET є складовою платформи .NET Framework та розвинулася з більш давньої платформи, що має назву Microsoft ASP. Модель на концепція ASP.NET має підґрунтя на базі протоколу HTTP. На правилах цього протоколу відбувається взаємодія між браузером та сервером. Основа всіх

додатків Microsoft .NET це загальномовне середовище виконання, на якому базуються всі додатки .NET. Тому, пишучи програмне забезпечення з використанням даної платформи, можна користуватися різними мовами програмування з арсеналу.NET Framework (з популярних це C# та Visual Basic.NET, з менш популярних — J# та JScript .NET). В механізмі формування сторінок основою є програмна абстрактна модель Web Forms. На цій програмній моделі засновано основну частину функціоналу розробки програмного коду. Відгалуженням такого фреймворку є набір програмних моделей, наприклад ASP.NET Web API та ASP.NET MVC, ASP.NET Web Forms та ASP.NET Web Pages. Також більш новими відгалуженнями є ASP.NET WebHooks та SignalR.

Перелічені вище моделі мають різне призначення:

ASP.NET Web Forms використовується для того, аби проектувати самостійні веб-сторінки. База для цього — компоненти, що дозволяють сприймати події користувача і обробляти їх на сервері:

- ASP.NET MVC базується на шаблоні проектування MVC. Він використовується для того, аби базуючись на цьому шаблоні розробляти веб-додатки;
- ASP.NET Web Pages також використовується для побудови веб-сторінок. Проте його синтаксис більш простий у використанні та до нього можна на льоту додавати код і мати доступ до інформації напряму зі сторінки розмітки;
- ASP.NET Web API використовується для побудови API;
- ASP.NET WebHooks дозволяє обробляти події за допомогою HTTP;
- SignalR використовується для того, аби клієнт та сервер могли спілкуватися використовуючи in real-time механізм.

Даний розділ було присвячено патерну розробки програмного забезпечення MVC. В результаті цього було прийнято рішення проектувати архітектурне рішення програмного забезпечення системи технічної підтримки з використанням даного шаблону. З усіх фреймворків MVC найбільше до цілей даного продукту

підходить бібліотека класів ASP.NET MVC. Ця бібліотека підтримує шаблон MVC та напрямлена на проектування додатків з використанням платформи веб.

Перед використанням даного набору бібліотек, варто ознайомитися з його позитивними та негативними сторонами:

Переваги ASP.NET MVC:

1) ASP.NET MVC використовує підхід “separation of concerns”. У архітектурі даного підходу застосовано даний принцип, згідно з яким всі частини шаблону — як модель, так контролер і представлення мають різні обов’язки, які не перетинаються один з одним. Через це маємо перевагу у тому, що різні частини даного шаблону можуть розробляти різні розробник, що надає виграш у швидкості;
2) ASP.NET MVC має багато готових шаблонів коду, який може бути повторно використано при початковій побудові програми і при наступній розробці додаткових модулів. Завдяки такому вбудованому функціоналу знижується ймовірність того, що розробник зробить помилку в коді під час розробки, що також надає перевагу у продуктивності;
3) даний фреймворк має багато вбудованих функцій, що допомагають пришвидшити розробку програмних продуктів. Завдяки ним можна розробляти програмне забезпечення базуючись на певних відомих даних, наприклад структурі моделі і т.д. Також дана платформа дозволяє зберігати в пам’яті певні дані, що можуть бути використані повторно на веб-сторінках;
4) великий асортимент засобів розробки. Оскільки програмне забезпечення, написане за допомогою цієї платформи найчастіше розробляється у стандартних платформах розробки компанії Майксрософт, то завдяки сумісності робота є максимально зручною. Вбудовані засоби розробки дозволяють використовувати великий арсенал гарячих клафіш, шорткатів та різні зручні функції, такі як drag-and-drop. Але навіть використовуючи звичайний редактор, розробляти програми на платформі ASP.NET можна достатньо швидко;

5) у даній платформі можна використовувати різні мови платформи .NET, оскільки середовище виконання є спільним для всіх мов;
6) легкість у розробці. Будь-яка задача виконується швидко завдяки вбудованим засобам, що полегшують роботу програми. Наприклад, вивільнення ресурсів і розділення відповідальностей програми;
7) простота у конфігурації та масштабуванні. Завдяки вдало спроектованій архітектурі платформи, більшість підкомпонентів можна замінити одне одним. Це стосується як компонентів компанії Майкрософт, так і компонентів, які розробники створюють власноруч;
8) у даній платформі впроваджено підвищену безпеку. Функціонал безпечного логіну та реєстрації дозволяє не перейматися про те, що дані користувачів будуть захищені;
9) зручне управління різними версіями програмного забезпечення. Завдяки тому, що у додатках, написаних на даній платформі, є гнучкий функціонал для підміни конфігураційних файлів, то розробник з легкістю може створювати артефакти одного і того самого веб-сайту під різні системи або середовища. Для цього буде використано одну і ту саму збірку програми, яку не треба буде робити в кількох екземплярах для різних середовищ;
10) автоматичне відстеження стану програми. Вбудовані засоби даної платформи автоматично відслідковують стан програмного забезпечення. Наприклад, вбудовано такі функції, як відстеження працездатності системи. Засоби платформи надсилають нотифікацію розробникам у випадку, якщо програмне забезпечення з якоїсь причини перестає працювати. Також система безпеки перевіряє додаток на предмет несанкціонованого доступу або несправностей в системі. Наприклад, якщо раптово в програму надходить тисяча реквестів на секунду, то нотифікація про це також прийде розробникам. До того, як розробники візьмуться за вирішення проблеми, програмне забезпечення спробує самостійно вирішити дану проблему.

11) просте перенесення між платформами. Завдяки універсальній архітектурі програми додаток можна легко переносити між різними серверами та простим способом налаштовувати його для повторного використання.

Недоліки ASP.NET MVC:

1) використання даної платформи має високу ціну. Дана ціна ж вищою, ніж в деяких аналогів, адже велика частина вартості йде на підтримку інтеграції в інфраструктуру Майкрософт. Тобто, певні кошти йдуть і на підтримку серверів, і на впровадження бази даних, і на ліцензію на засоби розробки. Також необхідно виділяти більші кошти, ніж у аналогів, на підтримку веб-сервера, а саме його потужностей;
2) недостатньо інформації в інтернеті стосовно даної платформи. На відміну від аналогів, дана платформа не може похвалитися вичерпною документацією. Тому інколи розробникам може бути важко знайти на інформаційному порталі відповіді на свої питання;
3) нестабільні шматки функціоналу. Оскільки дана платформа з’явилася достатньо нещодавно на ринку, то інколи розробники можуть стикнутися з проблемою, коли певні модулі перестають працювати в інтеграції. Наприклад, може збоїти модуль авторизації за певних умов;
4) нестабільна сумісність версій платформи. Розробник може стикнутися з проблемою, коли при оновленні версії фреймворку раптово деякі функції додатку стають неробочими. В таких випадках програмним інженерам доводиться додатково звертатися до інформаційних порталів фреймворку;
5) висока вартість міграції програми. Використовуючи дану платформу, при необхідності міграції програмного забезпечення між продуктами, можна наткнутися на проблему високовартісної міграції між серверами. Це може бути спричинено необхідністю використовувати додаткові інструменти для зміни низькорівневих налаштувань фреймворка залежно від сервера, на який програма мігрує;

6) проблемна сумісність зі старими версіями операційної системи. Якщо виникає необхідність розгорнути програму, написану на даній платформі, на більш старих версіях операційних систем, можливо, варто буде додатково звернутися до документації платформи, адже можуть знову виникнути певні труднощі.

У даному розділі ми ознайомилися з перевагами та недоліками фреймворку ASP.NET MVC. Визначили, що дана платформа цілком задовольняє вимоги та відповідає задачам, що було сформовано до архітектурного рішення програмного забезпечення системи технічної підтримки користувачів. Надалі для проектування такого програмного забезпечення і буде використано ASP.NET MVC як інструмент для бекенду продукту.

Вибір мови програмування для клієнтської частини

Не менш важливим кроком у створенні програмного забезпечення є вибір мови, якою буде написано фронтенд-частину додатку. Коли вибір щодо бази даних та мови для розробки серверної частини вже зроблено, необхідно зрозуміти, яке рішення покриє вимоги та завдання архітектурного рішення, і при цьому буде сумісним з технологіями, обраними для бази даних та серверу. Найчастіше у системах, подібних до даного програмного забезпечення, обирають мову JavaScript для розробки всієї частини, з якою комунікуватиме користувач, і яка надсилатиме запити на сервер. Дана мова є динамічно типізованою, також вона підтримує ООП. Крім того, ця мова найчастіше використовується для веб-сторінок, оскільки вона також підтримує функціональне програмування, різні парадигми. Також на базі цієї мови написано безліч інших фреймворків. Розглянемо переваги та недоліки даної мови та випадки, у яких ця мова буде найбільш вдалим вибором.

Переваги мови JavaScript:

1) мова JavaScript швидко відпрацьовує на клієнті. Завдяки тому, що ця мова не потребує компіляції, а лише інтерпретується, то виконання програми є дуже швидким. Завдяки цьому час відгуку програми при роботі

користувача з програмою є більш швидким, порівняно з іншими мовами програмування;

2) дана мова є простою у вивченні та використанні. З використанням цієї мови бізнес може зекономити велику кількість коштів на розробці програмного забезпечення, оскільки розробити на ній програму можна достатньо швидко. Крім того, на виході буде отримано якісний контент, який є динамічним;
3) дана мова є популярною. Багато відомих корпорацій, а також корпорацій середніх і малих розмірів використовують дану мову для побудови своїх продуктів. Дана мова користується попитом завдяки тому, що має високу сумісність майже з усіма браузерами;
4) дана мова має високу сумісність з різними мовами програмування. Її можна використовувати як самостійно при написанні додатків, так і у вигляді вбудованих скриптів;
5) оптимізоване завантаження сервера. У випадку, коли якась частина функціоналу, написаного за допомогою скриптів JavaScript перестає працювати коректно, не має необхідності перезавантажувати увесь додаток повністю. Користувач може перезавантажити лише той сегмент, який працює не так, як очікувалося. Завдяки цьому отримуємо певний виграш у продуктивності сервера під час завантаження;
6) мова надає інструменти для побудови динамічного та привабливого інтерфейсу. Завдяки цьому розробники можуть розробляти такі системи, які будуть зручними та приємними у використанні, що дозволить бізнесу поліпшити кількість позитивних відгуків про систему і цим збільшити продажі даного продукту;
7) великий асортимент вбудованих розширень. На ринку присутня велика кількість сторонніх додатків, за допомогою яких розробники можуть пришвидшити власну розробку. Дані додатки, при їх включені в код, додають певні готові фрагменти коду. Завдяки цьому розробка стає швидшою, і при цьому економляться кошти на розробку;

8) дану мову можна застосовувати до різних складових продукту. З використанням JavaScript можна писати не лише фронтенд-частину аплікацій, а і бекенд. Крім того, наразі існує велика кількість бібліотек як для першого, так і для другого виду розробки програмного забезпечення;
9) JavaScript допомагає скоротити об’єми програмного коду без втрат у якості. Завдяки тому, що JavaScript має велику кількість стандартних методів та бібліотек, код, написаний мовою JavaScript виглядає лаконічно і при цьому він є функціональним. Завдяки цій можливості даної мови веб-додатки також стають більш ефективними.

Недоліки мови JavaScript:

1) недостатня захищеність коду. Код JavaScript видимий у браузері. Через це шахраї мають можливість вставляти власний зловмисний код в браузері, чим можуть пошкодити безпеку системи;
2) мова JavaScript у різний спосіб інтерпретується браузерами. Це стосується не лише різних браузерів, але й різних версій одного й того самого браузера. Один і той самий функціонал, що працює в одному браузері, може не працювати в іншому. Тому перед тим, як передавати програмне забезпечення клієнту, варто протестувати цю систему у різних браузерах;
3) у коді, написаному даною мовою, важко відслідкувати помилки. Для JavaScript відсутні усталені засоби налагодження, тому покроково відслідкувати помилки у цьому коді достатньо важко. Через це розробка коду у такому браузері ускладнюється;
4) сутності, написані у мові JavaScript, можуть мати одну батьківську сутність. Це може стати обмеженням у певних випадках побудови архітектури програми;
5) у мові JavaScript об’єкти конвертуються з 64-бітової системи в 32-бітну і навпаки. Це виконується за допомогою побітової функції. Подібна конвертація займає певний час, тому скрипти, написані мовою JavaScript, виконуються недостатньо швидко;

6) помилка у коді, написаному мовою JavaScript, може спричинити до несправності усього сайту, написаного даною мовою. Хоча помилка може бути і незначною, проте браузер може сприйняти її так, наче розробник взагалі забув підключити JavaScript. Проте, наразі деякі браузери можуть виявляти такі помилки і нехтувати ними, щоб інша, робоча частина сайту працювала коректно і не блокувала роботу всіх користувачів.

3.2 Архітектура системи

Дане архітектурне рішення міститиме у собі базу даних, бекенд частину та фротенд частину. Такий підхід прийнято називати трирівневим. Крім того, основним елементом даного підходу буде архітектура MVC, що зробить проектування системи більш гнучким.

Зв’язок між користувацькими даними і сервером їхньої обробки буде впроваджено за допомогою HTTP. Цей протокол є дуже популярним для мережі інтернет. Його часто використовують тоді, коли необхідно передати зображення, аудіо і будь-які види файлів. При використанні браузера також використовується HTTP.

У класичних браузерах HTTP виконує роль зв’язку між користувачем та веб-сайтом. Коли користувач вносить у адресний рядок певну адресу, створюється веб-запит на сервер за допомогою HTTP. IP-адреса сервера отримує даний запит і генерує відповідь.

Загалом зв’язок через протокол HTTP при формуванні веб-сторінки можна описати наступним чином:

- генерується запит для того, щоб отримати HTML-сторінку на сервер. Сервер генерує файл відповідного типу і надсилає його у відповідь;
- генерується запит до сервера на отримання CSS. Сервер генерує таблиці стилів та повертає у відповідь;
- генерується запит для отримання зображення у конкретному форматі. Сервер генерує файл у відповідному форматі та повертає його;

- генерується запит на отримання скриптів. Сервер генерує файл скриптів і надсилає його у відповідь;
- генерується запит на отримання документа або файлу розмітки. Сервер генерує файл відповідного формату та повертає його у відповідь.

Для того, щоб не виникало необхідності при кожному запиті перезавантажувати сторінку і при цьому не виникало затримок у завантаженні сайту в браузерах користувачів, впроваджено технологію AJAX. Вона допомагає надсилати запити асинхронно на фоні і отримувати відповіді на ці запити таким чином, що відображення сторінки цілком не буде страждати. Такий метод розробки розділяє систему на шари. Перший шар це шар відображення. На ньому відбувається зображення даних. Другий шар, шар обміну даних, бере на себе відповідальність роботи з запитами, їхнього надсилання та обробки відповідей. Завдяки такій технології веб-сторінки стають більш динамічними, а веб-сайти — більш ефективними. JSON є форматом документів для передачі даних у таких технологіях. Скоріше, для групи технологій. Разом використовуються і засоби обробки запитів, і обробки стилів, і створення розмітки. Крім того, до цих засобів ще додаються скрипти. Вони допомагають у більш продуктивній взаємодії користувача з системою. Також за допомогою скриптів можна поліпшити спосіб відображення даних. Тобто, використовуючи наведені вище засоби, технології дійшли до того, що вони у поєднанні між собою надають можливість завантажувати дані на сторінці без необхідності у її оновленні.

Ajax це широка група технологій. Додаток, що їх використовує, може спілкуватися з сервером на фоні. При цьому не відбувається втручання у сторінку та її стан. У даній групі технологій наявні такі, що мають різне призначення:

- відображення даних;
- динамічна обробка даних та взаємодія з ними;
- обмін інформацією;
- асинхронність.

Оскільки технології постійно змінюються, відбуваються певні зміни і в форматах даних, якими оперують дані технології, і в бібліотеках, що керують цими технологіями. Наразі для роботи з технологією Ajax використовуються такі формати, як JSON та HTML. З їх допомогою Ajax може маніпулювати даними. Щодо бібліотек, які використовуються для роботи з Ajax, основною є JQuery. Вона інкапсулює інтерфейси, що спрощують надсилання запитів та їхню обробку.

Архітектурне рішення програмного забезпечення служби технічної підтримки складається з декількох модулів. Перший з них це власне модуль Service Desk. За допомогою нього користувачів здійснюють вхід у систему та працюють з інцидентами. Другий блок відповідає за адміністрування. У ньому надається можливість налаштовувати сутності системи та керувати користувачами. Останній, але не менш важливий блок, це блок збереження даних. Він містить різноманітні види сховищ для інформації додатку.

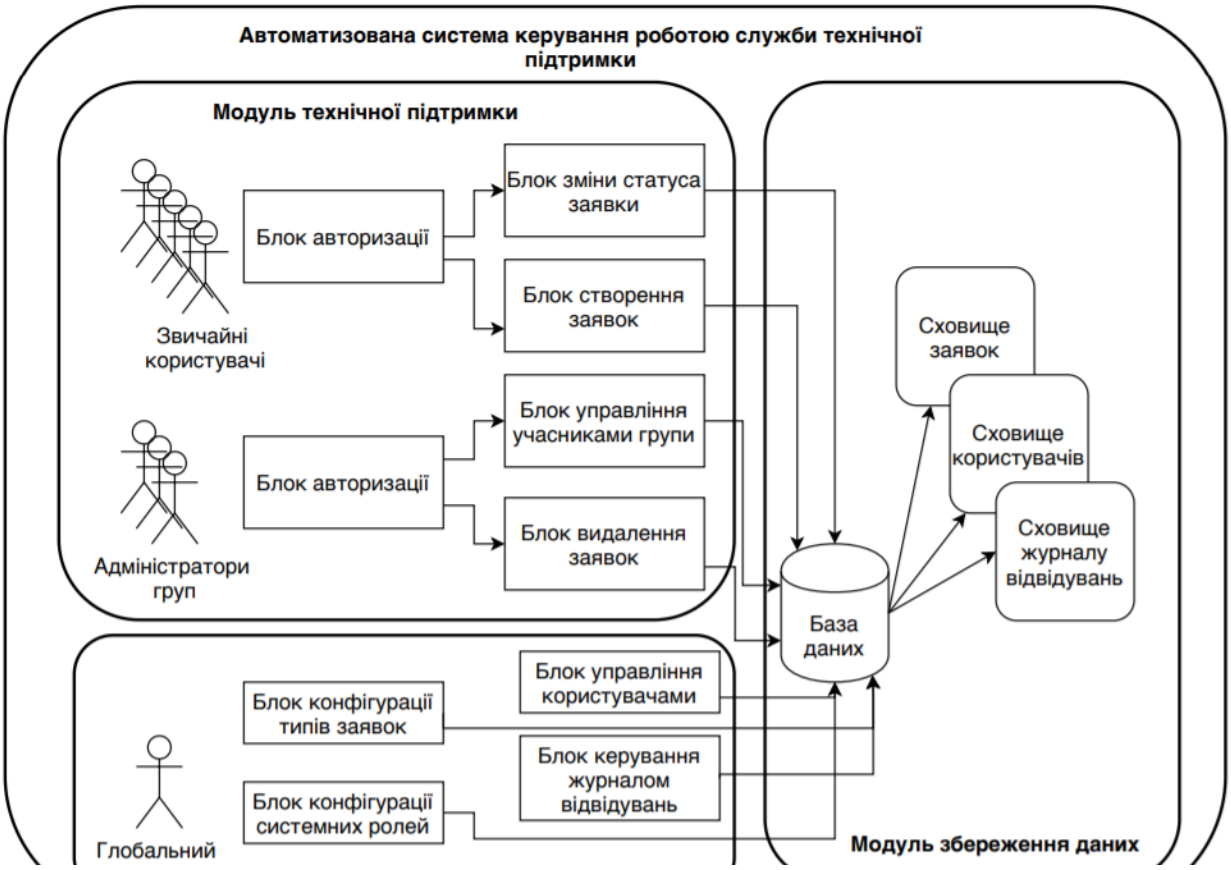


Рисунок 3.1 — Структурна схема системи

Архітектура бази даних

На рисунку 3.1 зображено діаграму з основними таблицями архітектурного рішення. З повною схемою бази даних можна ознайомитися, переглянувши додатки до даної роботи. На даній діаграмі відображено структури даних, у яких зберігається інформація про авторизованих користувачів. Назва кожної таблиці відповідає її призначенню.

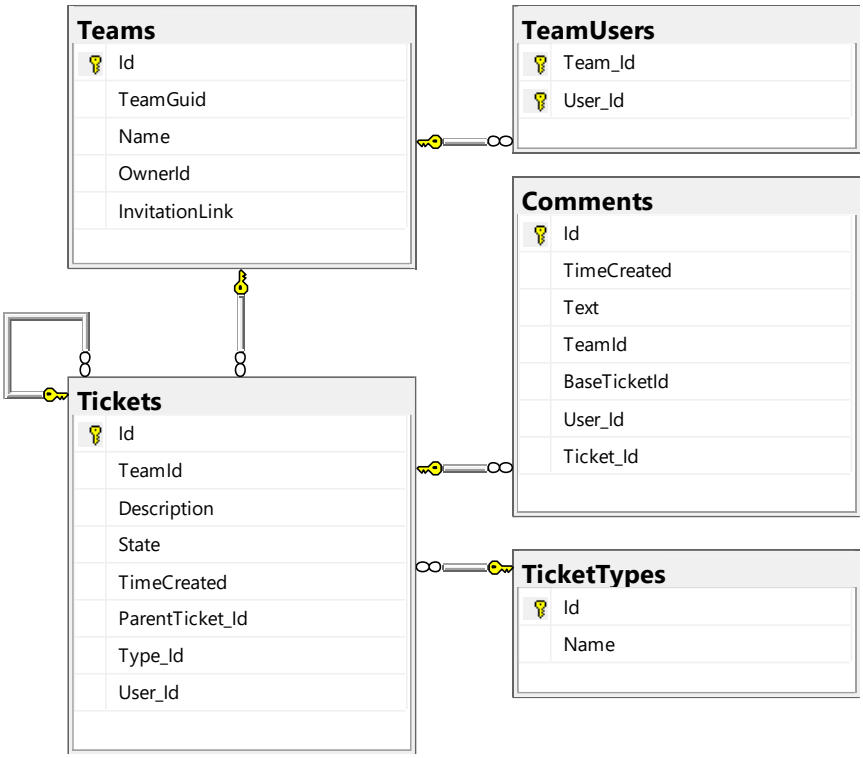


Рисунок 3.2 — Таблиці сутностей системи

Кожна таблиця зберігає відповідні дані:

- таблиця інцидентів містить ідентифікатор заявки, ідентифікатор команди, в рамках якої дану заявку створено, статус інциденту, час створення інциденту, ідентифікатор батьківського інциденту, якщо такий наявний у системі, тип інциденту та ідентифікатор користувача, який даний інцидент створив;

- таблиця команд зберігає ідентифікатори групи в форматах long та Guid, ім’я команди, ідентифікатор власника групи;
- таблиця учасників команд зберігає ідентифікатор групи та ідентифікатор учасника групи;
- таблиця коментарів зберігає ідентифікатор коментаря, час його створення, зміст коментаря, посилання на групу та батьківський інцидент даного коментаря, ідентифікатор власника коментаря;
- таблиця типів заявок містить ідентифікатор типу заявки та назву типу інциденту.

Рис. 3.2, зображений нижче, відображає частину бази даних, що зберігає таблиці з інформацією про авторизацію та рольову модель архітектурного рішення програмного забезпечення служби технічної підтримки. Призначення даних таблиць описано нижче у такому ж форматі, як і призначення таблиць попереднього блока.

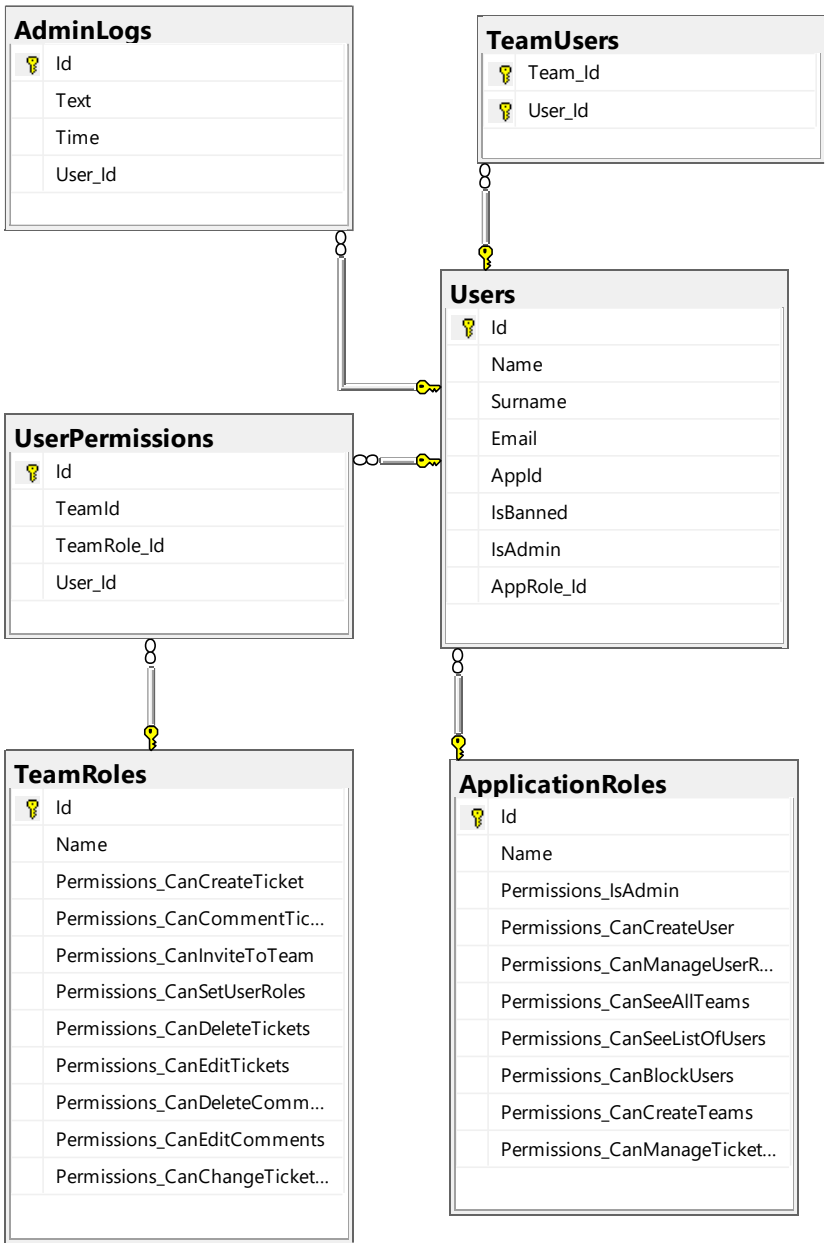


Рисунок 3.3 — Таблиці сутностей рольової моделі

- таблиця користувачів містить ідентифікатор користувача, його прізвище та ім’я, адресу електронної пошти, його роль, ознаку заблокованого користувача, ознаку адміністратора;
- таблиця ролей груп містить ідентифікатор ролі, назву ролі та перелік дозволів, що мають дані ролі у вигляді булевих ознак;
- таблиця ролі додатку зберігає ідентифікатор доступу, назву ролі та перелік дозволів у межах всієї системі (також булеві ознаки);

- таблиця дозволів користувачів зберігає ідентифікатор дозволу, ідентифікатор групи, ідентифікатор ролі у групі та ідентифікатор власне користувача;
- таблиця журналу дій містить ідентифікатор дії, зміст дії, час виконання дії та дані про користувача.

3.3 Програмна специфікація архітектурного рішення

Для програмної реалізації архітектурного рішення було використано таку специфікацію типів:

Для аутентифікації впроваджено тип UserService. Метод GetCurrent дозволяє ідентифікувати користувача за допомогою його сесії. Також одним з основних типів є TicketService, у якому відбуваються всі операції над інцидентами — їхнє створення, модифікація, видалення та вичитка. Також у цьому сервісі можна змінити статус заявки. За дані дії відповідають наступні методи з описаними нижче призначеннями:

- changeStatus — змінити стан інцидента.
- add — створити інцидент.
- edit — редагувати інцидент.
- delete — видалити інцидент зі сторінки групи.
- getByTeam — отримати повний список інцидентів, використовуючи ідентифікатор групи.

Ще одним важливим типом є UserTeamService. Він дозволяє управляти групами. В ньому наявні перелічені нижче функції для дій з користувачами в контексті груп:

- getByTeam — отримати список за ідентифікатором команди;
- invite — запросити.
- changeTeamRole — змінити роль у контексті групи
- deleteUser — видалити.
- isManager — ознака, що визначає чи є користувач адміністартором групи
- join — приєднатися.

3.4 Вимоги до версії платформ для використання

Дане архітектурне рішення потребує певного програмного забезпечення. Для комфортного користування програмою користувач має використовувати робочу машину такої конфігурації:

- персональний комп'ютер на основі Windows 7 або версії вище;
- процесор робочої машини Intel Core i3-5100 2,7 ГГц
- 4 ГБ оперативної пам’яті;
- Microsoft SQL Server СУБД;
- IIS веб-сервер.

Важливо також мати безперервний доступ в інтернет для того, аби дані зберігалися та була висока швидкість відгуку програми.

Також, щоб забезпечити безперервну роботу додатку, необхідно мати постійний доступ в інтернет на комп’ютері.

Якщо є можливість розгортати додаток у внутрішній мережі підприємства, то необхідно забезпечити інженерів, що користуватимуться даною системою програмними засобами, описаними вище. Також важливо впровадити безперервний доступ до мережі інтернет. Такі самі умови мають справджуватися для користувачів, які працюватимуть з системою індивідуально.

3.5 Інструкція по роботі з програмним забезпеченням

Після успішної авторизації у програмне забезпечення служби технічної підтримки перенаправлення користувача відбувається на головну сторінку програми. На цій сторінці доступний наступний функціонал: можливість побачити доступні команди, можливість ознайомитися з заявками в командах. Також користувач має можливість створити команду з головної сторінки. Крім того, глобальним користувачам доступна ще кнопка переходу на адміністративну панель.

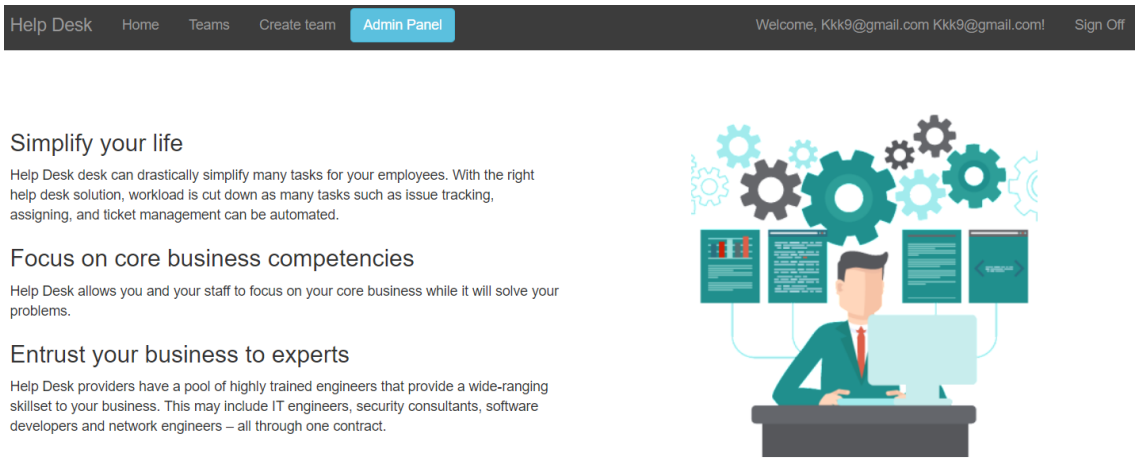


Рисунок 3.5 — Основна сторінка

Натиснувши на кнопку «Створити команду» користувач потрапляє на сторінку, на якій відбувається процес створення команди. Вигляд цієї сторінки зображено на рис. 3.6.

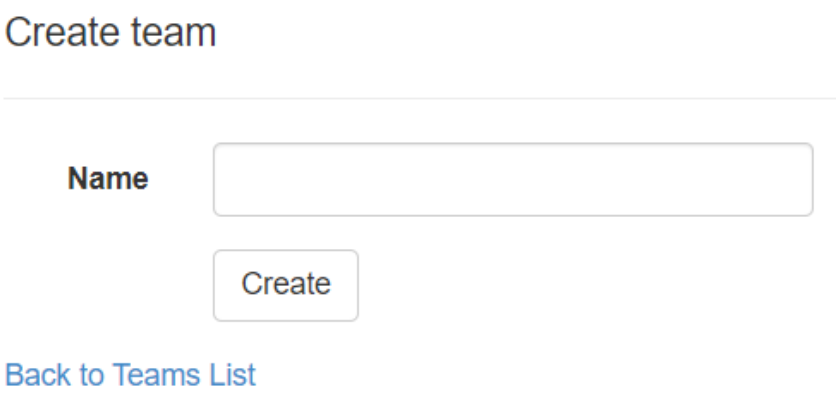


Рисунок 3.6 — Фрагмент сторінки створення групи

Коли користувач здійснює перехід на модуль з групами, він бачить список груп, у яких він є учасником. Також користувачеві доступні всі заявки, створені учасниками даної групи. Сторінка складається з таких елементів:

- область для створення заявки. Містить поле для введення тексту та список типів інциденту;
- відображення заявок даної групи з усією необхідною інформацією.
- бічна панель з командами конкретного інженера;
- для модераторів команди модуль управління командами та її користувачами.

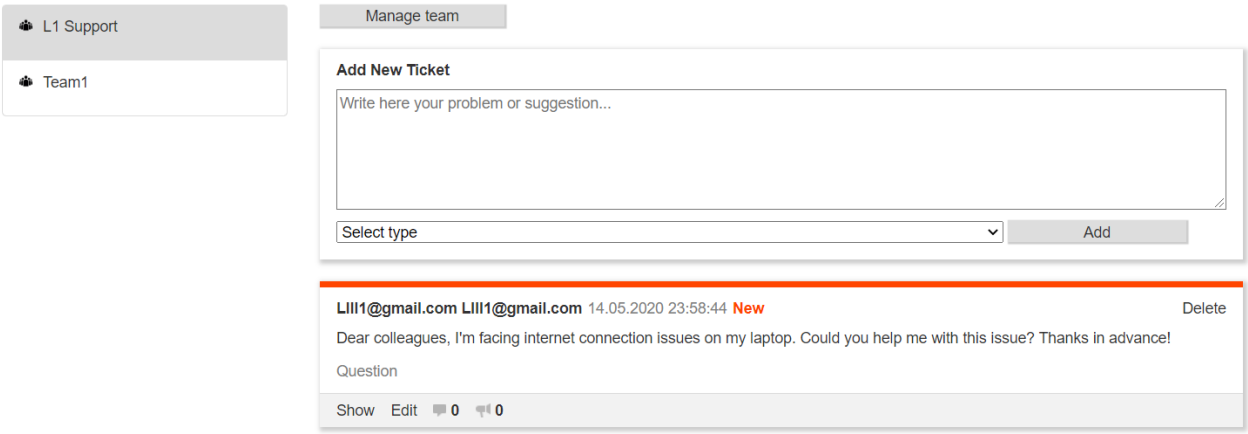


Рисунок 3.7 — Форма створення заявки

Глобальні адміністратори мають додаткові функції. Вони мають доступ до адміністративної панелі. На головній сторінці панелі адміністрування їм доступний список всіх інженерів, що користуються додатком. Адміністратор може бачити їхні дані та управляти ними. До таких даних належить інформація про користувача, його ролі, групу до якої він належить. На рисунку зображено приклад вигляду адміністративної панелі. У цій адміністративній панелі є функція конфігурації типів заявок. Саме ця функція дозволить вдосконалити процес класифікації заявок завдяки можливості конфігурації типів заявок.

На панелі адміністрування користувачу доступний наступний функціонал: сторінка журналу дій системи, список ролей і їхня конфігурація, типи інцидентів та їхня конфігурація, список користувачів та їхніх ролей, статистика системи.

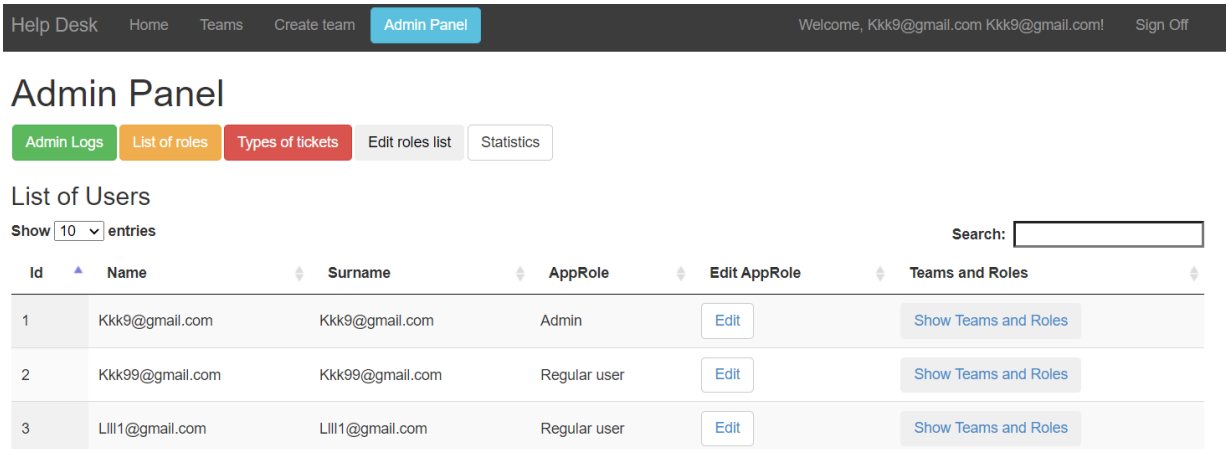


Рисунок 3.8 — Вигляд адміністративної панелі

Нижче бачимо як виглядає сторінка, на якій конфігуруються типи заявок.

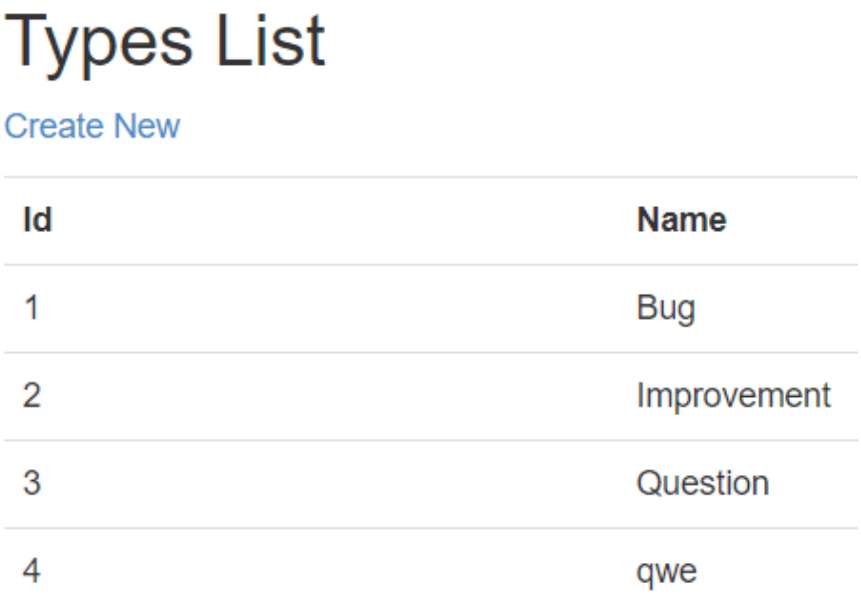


Рисунок 3.9 — Функція управління типами інцидентів

3.6 Експериментальні дослідження

Було проведено ряд експериментальних досліджень для даного архітектурного рішення. Було обчислено ключові метрики для аналізу архітектурного рішення програмного забезпечення служби технічної підтримки.

По-перше, визначено відсоток вирішених інцидентів. Сумарна кількість інцидентів становить 1500. Кількість невирішених інцидентів – 100. Для цього розрахували відношення всієї кількості інцидентів до кількості вирішених інцидентів (всі, крім статусу Unresolved):

$$Ticket\ resolution\ rate = \frac{(Total\ tickets\ quantity - unresolved\ tickets\ quantity)}{Total\ tickets\ quantity} \quad (3.1)$$

$$Ticket\ resolution\ rate = \frac{1500-100}{1500} = 0{,}93 \quad (3.2)$$

Визначили, що відсоток вирішених інцидентів становить 93%.

По-друге, розрахували відсоток невирішених інцидентів. Це буде рештою, що лишиться від вирішених інцидентів.

$$Unresolved\ tickets\ rate = 100\% - Ticket\ resolution\ rate \quad (3.3)$$

$$Unresolved\ tickets\ rate = 100\% - 93\% = 7\% \quad (3.4)$$

Таким чином визначили, що відсоток невирішених інцидентів для архітектурного рішення програмного забезпечення служби технічної підтримки становить 7%.

Також не менш важливо знати величину заборгованості по заявках. Ця величина визначатиме кількість невирішених заявок у певний період часу. Візьмемо за період часу тиждень. Якщо відомо, що за тиждень надійшло 60 заявок, а наше архітектурне рішення, враховуючи кількість відповідальних інженерів, дозволяє обробити 40 заявок, то величину заборгованості можна розрахувати таким чином:

$$Ticket\ backlog = Total\ incidents\ per\ week - resolved\ incidents\ per\ week$$

$$Ticket\ backlog = 60 - 40 = 20 \quad (3.5)$$

Бачимо, що заборгованість по інцидентах на тиждень становить 20 заявок.

Висновки до розділу

В результаті роботи над даним розділом було створено реалізацію бази даних, серверної частини, програмного інтерфейсу для архітектурного рішення програмного забезпечення служби технічної підтримки. Описано сутності системи, їхні залежності та реалізацію у вигляді програмного коду. Також до розділу надано документацію щодо використання даного програмного продукту. Детально описано кроки користувача в системі для різних ролей. Дане архітектурне рішення дозволить удосконалити механізм класифікації заявок у роботі служби технічної підтримки за рахунок того, що в даному програмному забезпеченні буде впроваджено механізм для розширеної конфігурації заявок та для збору статистики щодо створених інцидентів та присвоєних їм типів.

Даний розділ описує платформи, які було вирішено використовувати для побудови архітектурного рішення програмного забезпечення служби технічної підтримки. Також у цьому розділі було описано переваги та недоліки популярних підходів до проектування систем. У якості бази даних було обрано реляційну базу Microsoft SQL Server. Для побудови бекенд частини додатку було вирішено використовувати мову C#, адже після аналізу популярних на ринку мов виявилося, що ця мова є тою, що найкраще задовольняє вимоги до даного архітектурного рішення. Говорячи про мову для фронетнд частини додатку, найбільш вдалою виявилася мову JavaScript, оскільки вона забезпечить привабливий інтерфейс, який буде водночас зручним та технічно ефективним.

4 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ

4.1 Створення стартап-проекту

Партнерами даного стартап проекту є такі особи:

1) менеджер проекту — роль Організатор;
2) спеціаліст 1 (.NET) — роль Генератор ідеї;
3) спеціаліст 2 (JavaScript) — роль Генератор ідеї;
4) спеціаліст 3 (Тестувальник) — роль Генератор ідеї;
5) спеціаліст 4 (Системний адміністратор/DevOps) — роль Генератор ідеї;
6) спеціаліст 5 (Маркетолог) — роль Дипломат.

Вказані нижче етапи необхідно пройти у побудові стартап-проекту.

1) збір вимог до системи – 2;
2) виділення незалежних частин проекту – 1;
3) розподіл частин між спеціалістами – 0,5;
4) розробка (загальної структури продукту, інтерфейсу) - 3;
5) повне тестування продукту. - 2;
6) пошук інвесторів. - 2;
7) просування проекту (і реклама). – 1,5;
8) аналіз і планування потенційних поправок – 1.

Розрахунок завантаженості команди.

Завантаженість = Вхід/вихід.

IT-спеціаліст = 7/6 = 1,17.

Проектний менеджер = 7/9 = 0,78.

Маркетолог = 6/5 = 1,2.

Рисунок 4.1 — Розподіл етапів проекту за часом

4.2 Морфологічна карта продукту

Мова програмування – параметр, який визначає на якій мові буде написана основна частина ПЗ та які бібліотеки буде застосовано. Фреймворк WEB – параметр, який визначає, яка програма буде рендерити наші дані для користувача. Операційна система – параметр, який визначає на якій ОС будуть розробляти розробники. Додаткова мова програмування – параметр, який вказує на те яка мова буде додатковою для написання графічної частини.

Задум товару:

1) товар за задумом. Автоматизована система керування службою технічної підтримки вигідна для будь-яких компаній, що прагнуть підвищити ефективність процесу вирішення процесних запитів на підприємствах та організаціях з високою чисельністю співробітників; або у продуктах з десятками й сотнями користувачів;
2) товар у реальному виконанні. Товар є програмним продуктом, тому його можна встановлювати як на власній серверній інфраструктурі так і у хмарному сховищі, таким чином повністю зникає необхідність у нагляді та безперервному супроводі механізмів;

3) товар з підкріпленням. Розгортання, початкове налаштування та запуск є частиною контракту. Подальша підтримка може бути додатково узгоджена на підписній основі.

Таблиця 4.1 — Опрацювання питань для удосконалення продукту

№ з/п	**Запитання**	**Відповідь**
1	Частиною яких систем є продукт?	Інформаційна: 50%. Управління часом: 25%. Відстеження помилок: 25%.
2	Які функції надсистеми може виконувати продукт? Як їх з ним пов’язати?	Системи керування службою технічної підтримки переважно охоплюють функціональність керування людським ресурсом, часової звітності, розстановки пріоритетів і нагляду за робочим процесом в цілому. Але за відсутності комплексного підходу і застосування методологій управління командами програма може не принести бажаних результатів у контексті зменшення часових витрат на роботу. Дана проблема вирішується за умови зваженого підходу до побудови робочого процесу в продукті – базуючись на підходах управління командами та принципах тайм-менеджменту.
3	Чи можна розділити продукт на частини?	Програму можна поділити на три модулі: модуль адміністрування програми, модуль налаштування груп користувачів та їхньої «стрічки» задач і власне модуль управління заявками.

Продовження таблиці 4.1

4	Чи можна об’єднати (агрегувати) кілька елементів продукту в один?	Модуль управління заявками та модуль управління групами користувачів.
5	Чи можна нерухомі частини продукту зробити рухомими і навпаки?	Усі модулі не є ні рухомими, ні нерухомими, оскільки це програмна платформа.
6	Яким має бути ідеальний продукт?	Виконувати всі поставлені вимоги, легко масштабуватись та бути легким у супроводі.
7	Що відбудеться, якщо вилучити цей продукт? Чим його можна замінити?	Частину функціоналу можна замінити аналогами (Jira, HelpDesk), але повної альтернативи даному продукту наразі не існує.
8	Яким цей продукт був у минулому?	У минулому цей продукт не був поширений через слабкий розвиток технологій, тому керування службою технічної підтримки відбувалося в усній або письмовій формі.
9	На розвиток яких функцій було спрямоване удосконалення продукту?	Підвищення ефективності виконання процесних запитів на підприємствах.
10	Які функції залишилися «недорозвиненими»?	Система аналізу статистики.

1-й MVP: Керування службою технічної підтримки у аналоговому вигляді – за допомогою стікерних або маркерних дошок.

2-й MVP: Розрахунок завантаженості команди та розподіл задач за допомогою засобів MS Excel.

3-й MVP: Використання існуючих програм для керування службою технічної підтримки.

4-й MVP: Створення десктопного застосунку для керування службою технічної підтримки.

5-й MVP: Створення кросплатформенного веб-застосунку для керування службою технічної підтримки.

6-й MVP: Удосконалення застосунку. Реалізація функції конфігурування та впровадження user-friendly інтерфейсу.

Здійснюється відбір найцікавіших ідей і формується їх список.

Ефективне керування службою технічної підтримки потребує сучасних рішень та новітніх підходів. У нагоді стане розроблюваний застосунок. Розглянемо ідеї реалізації застосунку:

Ідея 1. Десктопний не кросплатформенний застосунок, який дозволяє лише вносити заявки користувачів у вигляді нотатків. Підтримується лише операційною системою Windows.

Ідея 2. Десктопний кросплатформенний застосунок, який дозволяє лише вносити заявки користувачів у вигляді нотатків. Підтримується різними операційними системами.

Ідея 3. Кросплатформенний веб-застосунок для ПК, який який дозволяє лише вносити заявки користувачів у вигляді нотатків.

Ідея 4. Кросплатформенний, адаптивний веб-застосунок, який дозволяє лише вносити заявки користувачів у вигляді нотатків.

Ідея 5. Кросплатформенний, адаптивний веб-застосунок, який дозволяє вносити заявки користувачів та призначати вище вказаному запиту відповідний тип.

Ідея 6. Кросплатформенний, адаптивний веб-застосунок, який дозволяє вносити заявки користувачів, призначати вище вказаному запиту відповідний тип та призначати на виконання заявки відповідальну особу.

Ідея 7. Кросплатформенний, адаптивний веб-застосунок, який дозволяє вносити заявки користувачів, призначати вище вказаному запиту відповідний тип, призначати на виконання заявки відповідальну особу, а також створювати нові типи заявок та об’єднувати користувачів у групи.

Наступним кроком є об’єднання знайдених ідей, їх агрегування або комбінування.

Агрегування 1. При об’єднанні ідеї 2 та ідеї 5, можна отримати десктопний застосунок, що дозволяє вносити заявки користувачів та призначати вище вказаному запиту відповідний тип.

Агрегування 2. При об’єднанні ідеї 2 та ідеї 6 отримаємо застосунок, що дозволяє вносити заявки користувачів, призначати вище вказаному запиту відповідний тип та призначати на виконання заявки відповідальну особу.

Агрегування 3. При об’єднанні ідеї 4 та 5 отримаємо веб-застосунок, що дозволяє вносити заявки користувачів та призначати вище вказаному запиту відповідний тип.

Таким чином отримано три додаткові ідеї. Кількість продуктивних та унікальних ідей збільшено до дев’яти. Навіть без їх ретельного аналізу, якщо скласти морфологічну таблицю, то можна отримати близько 30–40 якісно нових ідей.

Відбираємо найбільш працездатні ідеї, перевіряємо їх на своєчасність. Для цього необхідно за кожною з отриманих ідей відповісти на три запитання: що вийшло; де це можна використати; кому це потрібно.

Ідея 1. Десктопний не кросплатформенний застосунок, який дозволяє лише вносити заявки користувачів у вигляді нотатків. Підтримується лише операційною системою Windows:

- отримуємо робочий застосунок, який можна встановити за запустити лише в операційній системі Windows;
- можна використати при керуванні службою технічної підтримки в Україні;
- державним установам, що потребують єдиної системи фіксації процесних запитів.

Ідея 2. Десктопний кросплатформенний застосунок, який дозволяє лише вносити заявки користувачів у вигляді нотатків. Підтримується різними операційними системами.

- отримуємо робочий застосунок, який підтримується різними операційними системами;
- можна використати при керуванні службою технічної підтримки в Україні;
- державним установам, що потребують єдиної системи фіксації процесних запитів.

Ідея 3. Кросплатформенний веб-застосунок для ПК, який який дозволяє лише вносити заявки користувачів у вигляді нотатків.

- отримуємо робочий веб-застосунок, який коректно відображатиметься лише на комп’ютерах з визначеним розширенням екрану;
- можна використати при впровадженні систем єдиної фіксації процесних запитів в Україні;
- установам, що потребують керування службою технічної підтримки.

Ідея 4. Кросплатформенний, адаптивний веб-застосунок, який дозволяє лише вносити заявки користувачів у вигляді нотатків.

- отримуємо робочий адаптивний веб-застосунок, який доступний з будь-якого комп’ютеру, ноутбуку, планшету, телефону;
- можна використати керуванні службою технічної підтримки в Україні;
- установам, що потребують керування службою технічної підтримки.

Ідея 5. Кросплатформенний, адаптивний веб-застосунок, який дозволяє вносити заявки користувачів та призначати вище вказаному запиту відповідний тип.

- отримуємо більш функціональний веб-застосунок ніж в попередньому варіанті, який доступний з будь-якого комп'ютеру, ноутбуку, планшету, телефону;
- можна використати при керуванні службою технічної підтримки в Україні;
- установам, що потребують керування службою технічної підтримки або будь-яка приватна особа.

Ідея 6. Кросплатформенний, адаптивний веб-застосунок, який дозволяє вносити заявки користувачів, призначати вище вказаному запиту відповідний тип та призначати на виконання заявки відповідальну особу.

- отримуємо веб-застосунок, який доступний з будь-якого комп'ютеру, ноутбуку, планшету, телефону та призначати вище вказаному запиту відповідний тип та призначати на виконання заявки відповідальну особу;
- можна використати при керуванні службою технічної підтримки в Україні.Установам, потребують керування службою технічної підтримки або будь-яка приватна особа.

Агрегування 1.

Отримаємо десктопний застосунок, що дозволяє лише вносити заявки користувачів у вигляді нотатків:

- можна використати при керуванні службою технічної підтримки в Україні;
- установам, що потребують керування службою технічної підтримки.

Агрегування 2.

Отримаємо десктопний застосунок, що дозволяє вносити заявки користувачів у вигляді нотатків та призначати заявкам типи:

- можна використати при керуванні службою технічної підтримки в Україні;
- установам, що потребують керуванням службою технічної підтримки.

Агрегування 3.

Отримаємо веб-застосунок, що дозволяє лише вносити заявки користувачів у вигляді нотатків, дозволяє призначати заявкам певні типи та створювати групи користувачів:

- можна використати при керуванні службою технічної підтримки в Україні;
- установам, що потребують єдиної системи фіксації процесних запитів або будь-яка приватна особа.

4.3 Розроблення ринкової стратегії проекту

Таблиця 4.2 — Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1	Підприємства з великою кількістю співробітників або розподілені, що потребують автоматизації у вирішенні питань керування службою технічної підтримки та процесними запитами.	Готові використовувати	60%	Середня – бо подібні системи потрібні, але кількість конкурентів мала.	Помірно складно(присутній великий попит, але потрібно вигравати конкуренцію)

Продовження таблиці 4.2

2	Персональні особи, що зацікавлені в плануванні власного робочого навантаження	Подекуди використовуються альтернативи	10%	Низька – бо цей сегмент не потребує подібних систем	Складно (необхідно переконати кожного особисто, інколи навіть змінити аналог на дане рішення)
3	Компанії, що зацікавлені в єдиних системах збереження процесних запитів	Готові використовувати	30%	Середня – бо подібні системи потрібні, але кількість конкурентів мала.	Помірно складно (даний функціонал не є першочерговою ціллю але в цілому компанії будуть зацікавлені в даному рішенні)
Які цільові групи обрано: 1 та 3.					

Таблиця 4.3 — Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспроможні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку**
1	Створення систем із підтримкою заявок лише одного типу.	Реалізація одного типу заявок.	Виробники можуть мати свої вподобання, відмовитись від яких вони не захочуть	Спеціалізація

Продовження таблиці 4.3

2	Створення систем із підтримкою заявок зі заздалегідь визначеного списку типів.	Реалізація кількох типів заявок.	Затратна але більш затребувана реалізація	Спеціалізація
3	Створення систем з можливістю створювати власні типи заявок і застосовувати ці типи до заявок	Розробка модуля адміністрування, завдяки якому можна налаштовувати типи заявок.	Є кращі спеціалізовані аналоги	Диференціація

Таблиця 4.4 — Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки**
1	Система не є новою на ринку, але описуваний продукт зможе об’єднати функціонал різних систем в одному	Як шукати нових, так і забирати у конкурентів	Так, компанія буде копіювати базові функції у конкурентів, але з значними змінами. Копіюватиме насамперед функціонал.	Імітація

Таблиця 4.5 — Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегія розвитку*	*Ключові конкурентоспроможні позиції власного стартап-проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)*
1	Простота використання та функціональність	Залежно від відгуків споживачів	Об'єднує увесь функціонал подібних систем в одній.	Швидкий розвиток, застосування новітніх технологій, повноцінна підтримка користувачів.

Таблиця 4.6 — Визначення ключових переваг концепції потенційного товару

№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити*
1	Надійність продукту	Головна перевага	Основний фокус на безпеці даних та їх максимальному захисті.
2	Легкість використання	Одна з головних переваг	Простота в налаштуванні та підтримці.
3	Якість продукту	Одна з головних переваг	Нові технології забезпечують відмінний функціонал та не меншу відмовостійкість

Таблиця 4.7 — Опис трьох рівнів моделі товару

Рівні товару	*Сутність та складові*
I. Товар за задумом	Автоматизована система керування роботою служби технічної підтримки.

Продовження таблиці 4.7

II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Автоматизація розподілу заявок між спеціалістами. 2. Конфігурація заявок та груп користувачів.	Нм Нм	Тх Тх
	Якість: програмна документація.		
	Пакування: Завантажувальний програмний пакет.		
	Марка: HelpDesk		
III. Товар із підкріпленням	До продажу: Розроблена архітектура програмного забезпечення.		
	Після продажу: Підтримка інфраструктури.		
За рахунок чого потенційний товар буде захищено від копіювання: Ліцензування вихідного коду з використанням комерційних майнових ліцензій та (можливих) патентів на більшу частину продукту, щоб уникнути порушення авторських прав.			

Таблиця 4.8 — Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники*	*Рівень цін на товари-аналоги*	*Рівень доходів цільової групи споживачів*	*Верхня та нижня межі встановлення ціни на товар/послугу*
1	JIRA - $20+ на місяць Zendesk - $5+ на місяць	5-30$/місяць	10000-100000 $/рік	$5 - $300.

Таблиця 4.9 — Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1	Одноразова покупка	Первинне налаштування інфраструктури	Прямий контакт із компаніями-замовниками.	Всі компанії що працюють з чутливої інформацією

Таблиця 4.10 — Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуються цільові клієнти*	*Ключові позиції, обрані для позиціонування*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
1	Місячна підписка	Підтримка користувачів	Новий функціонал	Показати переваги продукту над конкурентами	Використання новітніх технологій

Таблиця 4.11 — Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1	Кількість головних гравців, од	4(JIRA, ServiceDesk, Trello, ZenDesk)

Продовження таблиці 4.11

2	Загальний обсяг продаж, тис. грн/ум.од	10 000
3	Динаміка ринку (якісна оцінка)	Швидко зростає.
4	Наявність обмежень для входу (вказати характер обмежень)	Немає.
5	Специфічні вимоги до стандартизації та сертифікації	Немає, стандартне патентування програмного забезпечення. Заява про видачу патенту на винахід чи декларaційного патенту на винахід (корисну модель); (Абзац другий частини п’ятої статті 12 із змінами, внесеними згідно із Законом N 850-IV (850-15) від 22.05.2003)
6	Середня норма рентабельності в галузі (або по ринку), %	30%

Таблиця 4.12 — Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1	Надійна система керування службами технічної підтримки.	Великі та/або розподілені корпорації	Підприємства зацікавлені у автоматизації керування роботою служби технічної підтримки та впровадженні єдиного реєстру для зберігання процесних запитів.	Отримання повноцінного функціоналу.

Таблиця 4.13 — Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Вхід дуже сильного конкурента на ринок.	Спроба одного із гігантів ринку автоматизованих систем зайняти цю нішу ринку.	Більш гнучке підлаштування до потреб кожного клієнта.
2	Поява більш інноваційного продукту	Різке впровадження нових рішень у керуванні службами технічної підтримки.	Постійне вкладання коштів у вдосконалення свого продукту, розробка нових функцій та впровадження нових.

Продовження таблиці 4.13

3	Стихійні лиха або епідемії	Епідемії та стихійні лиха негативно вплинуть на сплатоспроможність деяких сфер бізнесу	Зниження сплатоспроможності бізнесу сповільнить перехід до інновацій і автоматизації
4	Поява продукту зі схожими концепціями та нижчою ціною	Спроба конкурентів створити продукт з такою самою концепцією але нижчою ціною	Дешевший продукт на ринку привабить потенційних клієнтів
5	Поява продукту зі схожими концепціями та кращим маркетингом	Спроба конкурентів створити продукт з такою самою концепцією але кращим просуванням	Схожий продукт на ринку але більш просунутий привабить більше потенційних клієнтів

Таблиця 4.14 — Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Обвал/проблеми у одного із основних конкурентів	Виникнення проблем із законодавством (або довірою клієнтів) у одного із конкурентів.	Можливе захоплення його бази користувачів даного конкурента за рахунок особливих пропозицій/знижок компаніям, що перейдуть від нього.

Продовження таблиці 4.14

2	Рішення великих корпорацій на використання систем керування роботою служби технічної підтримки	Виникнення нових клієнтів, які раніше не використовували системи керування роботою служби технічної підтримки	У випадку такої активності, важливо акцентувати увагу на захопленні нових клієнтів перед конкурентами за рахунок знижок/пропозицій/агресивної реклами
3	Перехід державних установ на електронні реєстри процесних запитів	Значне розширення бази клієнтів,	У випадку такої активності важливо заключити договір с державою, для отримання великої кількості нових клієнтів
4	Проблеми у одного з основних постачальників хмарних сховищ даних	Розширення бази клієнтів	У такому випадку можлива поява клієнтів що зможуть використати даний продукт для зберігання даних власного користування

Таблиця 4.15 — Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Тип конкуренції — Oligopsony (Олігопсонія)	Кількість клієнтів порівняно невелика а кількість конкурентів значна.	Агресивна реклама, направлена на цільову аудиторію

Продовження таблиці 4.15

2. За рівнем конкурентної боротьби — міжнародний	Боротьба ведеться за усі компанії, незалежно від розміщення.	Варто пропонувати продукт із максимально можливим набором локалізацій.
3. За галузевою ознакою — внутрішньогалузева	Галузь - сфера програмного забезпечення.	Компанія будується в цій галузі і з акцентом виключно на ній.
4. Конкуренція за видами товарів: — Нішевий товар.		
5. За характером конкурентних переваг — цінова	Гнучкий вибір необхідного набору функціоналу	Компанії хочуть отримати максимальний функціонал за свої гроші.
6. За інтенсивністю — не марочна		

Таблиця 4.16 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	*Прямі конкуренти в галузі*	*Потенційні конкуренти*	*Постачальники*	*Клієнти*	*Товари-замінники*
	JIRA, Zendesk	*Бар'єри входження в ринок*	*Визначити фактори сили постачальників*	*Визначити фактори сили споживачів*	*Фактори загроз з боку замінників*

Продовження таблиці 4.16

Висновки:	Інтенсивність боротьби висока, потрібно прикладати максимально зусиль для боротьби з конкурентами	Бар'єрів входження в ринок немає.	Постачальники ніяк не впливають на ринок.	В першу чергу цінова пропозиція.	Товари замінники і є основними конкурентами продукту.

Таблиця 4.17 — Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1	Підтримка великої кількості типів заявок.	Можливість конфігурування типів заявок дозволяє підвищити конкурентоспроможність продукту
2	Особиста підтримка	Кожен клієнт отримує 24/7 зв'язок з службою підтримки.
3	Акції	Активна робота над залученням нових клієнтів.
4	Реклама.	
5	Додаткова розробка	Клієнти мають змогу замовити функціонал для доробки лише у їхній версії продукту

Продовження таблиці 4.17

6	Доступна версія з обмеженим функціоналом	Клієнти з низькою платоспроможністю мають змогу придбати за низькою ціною версію з обмеженим функціоналом
7	Висока продуктивність	Система не обтяжена зайвим функціоналом, за рахунок чого продукт працює швидко

Таблиця 4.18 — Порівняльний аналіз сильних та слабких сторін «ServiceDesk»

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні із системою*						
			–3	*–2*	*–1*	*0*	*+1*	*+2*	*+3*
1	Підтримка великої кількості типів заявок	20				*			
2	Особиста підтримка	10					*		
3	Акції	6			*				
4	Реклама	15						*	
5	Ціна	15		*					
6	Локалізація	8			*				
7	Конкуренти	10					*		

Таблиця 4.19 — SWOT- аналіз стартап-проекту

Сильні сторони: Вища, ніж у конкурентів, надійність продукту. Дуже агресивна тактика рекламування. Доступна ціна у використанні. Невелика кількість конкурентів.	Слабкі сторони: Дуже складний етап розробки системи. Хоча конкурентів на ринку мало, але вони є впливовими у своїй ніші. Відсутня локалізація на даному етапі розробки.

Продовження таблиці 4.19

Можливості: Конкуренція на ринку хоч і виглядає серйозною, але всі вони не мають такого функціоналу як розглянений продукт. Відсутність у інших продуктів конфігурування власних типів та груп. Рішення великих корпорацій на використання систем керування роботою служби технічної підтримки	Загрози: Існує ймовірність входу на ринок IT-гіганта, конкуренцію якому практично неможливо реалізувати. Стихійні лиха або епідемії. Поява продукту зі схожими концепціями та нижчою ціною. Поява продукту зі схожими концепціями та кращим маркетингом

Таблиця 4.20 — Альтернативи ринкового впровадження стартап-проекту

№ п/п	*Альтернатива (орієнтовний комплекс заходів) ринкової поведінки*	*Ймовірність отримання ресурсів*	*Строки реалізації*
1	Агресивна рекламна кампанія.	Висока.	2-3 роки.
2	Пасивна реклама, спроба набору клієнтів за рахунок вищої якості послуг.	Середня.	4-5 роки.

Таблиця 4.21 — Календарний план-графік реалізації стартап-проекту

№ з/п	*Етапи реалізації*	*Період реалізації проекту*						
		0-й рік				*1-й рік*	*2-й рік*	*3-й рік*
		1-й кв.	*2-й кв.*	*3-й кв.*	*4-й кв.*			
1	Проведення НДДКР	+	+	+				

Продовження таблиці 4.21

2	Розробка проектних матеріалів і ТЕО			+	+			
3	Робоче проектування і прив'язка проекту			+	+	+		
4	Створення компанії				+			
5	Придбання нематеріальних активів, отримання дозвільних документів тощо					+		
6	Придбання й оренда земельних ділянок, будівель, приміщень, споруд							
7	Придбання обладнання, устаткування та пристроїв							
8	Передвиробничі маркетингові дослідження		+					
9	Приймально-здавальні випробування							
10	Пусконалагоджувальні роботи							
11	Освоєння проектних потужностей							
12	Придбання матеріальних ресурсів							
13	Запуск виробництва							
14	Продаж продукції					+	+	+

Таблиця 4.22 — Планова потреба у виробничих площах

№ з/п	*Тип приміщення (будівлі, ділянки, споруди)*	*Кількість одиниць*	*Площа, кв. м*	*Вимоги до приміщення (будівлі, ділянки, споруди)*	*Умови надання*	*Вартість, тис. грн.*
1	Офіс	1	90	Вентиляція, освітлення, конкурентне розташування	Оренда	35
Разом		1	90	—	—	35

Таблиця 4.23 — Планова потреба у виробничому обладнанні та устаткуванні

№ з/п	*Вид обладнання (устаткування, пристрою)*	*Тип (модель)*	*Виробник обладнання (устаткування, пристрою)*	*Терміни постачання*	*Вартість, тис. грн.*
1	*Ноутбук*	*MacBook Pro 15*	*Apple*	*1 тиждень*	*70*
2	*Ноутбук*	*MacBook Pro 15*	*Apple*	*1 тиждень*	*70*
3	*Ноутбук*	*MacBook Pro 15*	*Apple*	*1 тиждень*	*70*
4	*Стіл 3шт.*	-	*IKEA*	*2-3 дні*	*3*
5	*Стілець 3шт.*	-	*IKEA*	*2-3 дні*	*1,5*
Разом:		—	—	—	*223,5*

Таблиця 4.24 — Планова вартість нематеріальних активів

№ з/п	*Вид активів*	*Активи, що можуть бути віднесені до даного виду*	*Вартість, тис. грн.*
1	Права користування майном	Право на оренду приміщень	10
2	Права на комерційні позначення	Право на торговельну марку “HelpDesk”	3

Таблиця 4.25 — Плановий обсяг проданих продуктів стартап-проекту

Вид продукції	*Одиниця виміру*	*Обсяги продажу за період*		
		1-й рік	*2-й рік*	*3-й рік*
$100 (базовий набір)	од.	10 тис.	50 тис.	200 тис.
$500 (повний функціонал)		1 тис.	10 тис.	50 тис.

Таблиця 4.26 — Планова потреба у матеріальних ресурсах та комплектуючих

№ з/п	*Вид ресурсу*	*Одиниця виміру*	*Витрати на одиницю продукції в натуральних показниках*	*Вартість на одиницю продукції, тис. грн.*	*Вартість за плановим обсягом виробництва за період, тис грн.*		
					1-й рік	*2-й рік*	*3-й рік*
1	Матеріали	—	0	0	0	0	0
Всього матеріалів		—	0	0	0	0	0

Продовження таблиці 4.26

2	Комплекту-ючі	—	0	0	0	0	0
Всього комплекту-ючих		—	0	0	0	0	0
3	Сировина	—	0	0	0	0	0
Всього сировини		—	0	0	0	0	0
Разом:		—	—	—	0	0	0

Таблиця 4.27 — Планова потреба та витрати на персонал

№ з/п	*Категорія персоналу*	*Чи-сель-ність*	*Заробі-тна плата, тис грн. на місяць*	*Відраху-вання на со-ціальні за-ходи, тис грн. на мі-сяць*	*Витрати на оплату праці за період, тис. грн.*		
					1-й рік	*2-й рік*	*3-й рік*
1	Адміністративний персонал (праців-ники апарату управ-ління)	2					
1.1	директор	1	9	4	96	192	288
1.2	бухгалтер	1	3	0,8	54	108	162
1.3	маркетолог						

Продовження таблиці 4.27

Всього витрати на оплату праці адміністративного персоналу		2			150	300	450
2	Промислово-виробничий персонал						
2.1	R&D спеціаліст.	1	5,6	1	54	108	162
2.2	Розробники.	3	9,1	1,8	96	192	288
2.3	Працівники служби підтримки	1-4	1,15	1,5	54	108	162
Всього витрати на оплату праці промислово-виробни-чого персоналу					450	900	1350
Разом:		6-13			600	1200	1800

Таблиця 4.28 — Загальні початкові витрати проекту

№ з/п	*Стаття витрат*	*Обсяги витрат в 0-й рік, тис. грн.*
1	Проведення НДДКР	31 (4 х 18)
2	Розробка проектних матеріалів і ТЕО	40 (4 х 5)
3	Робоче проектування і прив'язка проекту	5 (4 х 7)
4	Витрати на придбання й оренду земельних ділянок, будівель, приміщень, споруд	98

Продовження таблиці 4.28

5	Витрати на придбання обладнання та устаткування та пристроїв	213,5
6	Витрати на приймально-здавальні випробування	0
7	Витрати на пусконалагоджувальні роботи	0
8	Комплексне освоєння проектних потужностей	0
9	Витрати на придбання нематеріальних активів	40
10	Одноразові виплати, зокрема гарантуючим і страховим організаціям	0
11	Витрати на створення оборотного капіталу, необхідного для початку операційної діяльності (створення виробничих запасів, передоплата сировини, матеріалів і комплектуючих виробів, які мають бути поставлені на початку операційної діяльності)	0
12	Податкові платежі (земельний, комунальний та інші), здійснені до початку операційної діяльності	0
13	Оплата юридичних послуг	0
14	Витрати на передвиробничі маркетингові дослідження і створення збутової мережі	10
15	Витрати, пов'язані з діяльністю персоналу	1800
Разом		*2272,5*

Таблиця 4.29 — Планові загальногосподарські витрати

№ з/п	*Стаття витрат*	*Витрати за період, тис. грн.*		
		1-й рік	*2-й рік*	*3-й рік*
1	Витрати на оренду земельних ділянок, будівель, приміщень, споруд	35	40	45
2	Витрати на обладнання, устаткування та пристрої	100	120	15
3	Витрати на придбання нематеріальних активів	40	0	0
4	Витрати на персонал (на відрядження, соціальні заходи тощо)	20	30	40
5	Витрати на зв'язок	2	2,5	3
6	Витрати на паливо та електроенергію	7	8	9
7	Витрати на водопостачання	10	12	15
8	Витрати на утримання обладнання та приміщень	15	15	17
9	Витрати на збут	30	32	330
10	Витрати на просування та рекламу	500	200	200
11	Оплата юридичних послуг	70	10	10
12	Податкові платежі (земельний, комунальний податки, інші)	10	12	15
Разом:		839	481,5	402

Організаційно-правовою формою було вибрано акціонерне товариство.

Акціонерне товариство (АТ) є одним з різновидів організаційно-правової форми господарювання. У цій статті ми систематизували та проаналізували основні відмінності та особливості підприємств, створених у формі АТ. Крім цього, у статті розглянуто особливості ведення обліку АТ, а також оподаткування особливих операцій, що виникають у процесі діяльності саме АТ.

Висновки до розділу

В результаті роботи над даним розділом було розроблено план стартап-проєкту, описано його складові, ролі та етапи розвитку. Також було описано морфологічну карту продукту. Крім того, в даному розділі розроблено ринкову стратегію продукту, в результаті чого описано основні кроки розвитку даного рішення.

ВИСНОВКИ

Проектуючи архітектурне рішення програмного забезпечення було вивчено сферу управління IT послугами. Проаналізовано існуючі бізнес-процеси в цій сфері. Також було визначено роль Service Desk служб у сфері ITSM. Було визначено архітектуру програмного забезпечення для систем технічної підтримки, яка допоможе підвищити ефективність процесів на підприємствах. Особливу увагу приділено процесу класифікації заявок у системах Service Desk.

У процесі розробки системи було проаналізовано популярні рішення програмного забезпечення для Service Desk та їхню архітектуру. Завдяки цьому було визначено вимоги до додатку та можливості удосконалення даних систем. У даній системі вдосконалено спосіб класифікації інцидентів за рахунок того, що розроблене архітектурне рішення містить модуль для конфігурування типів заявок. У даному модулі впроваджено можливість створювати власні типи інцидентів, завдяки чому інженери технічної підтримки можуть обирати з широкого спектру типів інцидентів. А за необхідності введення нових типів заявок адміністратором може бути розширено даний список типів. Завдяки цьому процес класифікації заявок буде більш ефективним та зручним.

Дана архітектура програмного рішення пройшла кілька кроків створення. По-перше, було розроблено варіанти використання системи. Дані сценарії було розроблено для різних типів користувачів у системі. По-друге, для даного архітектурного рішення було розроблено графічний інтерфейс програмного забезпечення, спроектовано структуру бази даних та реалізовано серверну частину, яка займається обробкою даних, що надходять у систему.

Також було розроблено інструкцію для користувачів по роботі з системою, у якій описано такі кроки, як створення інцидентів з вибором їхнього типу, управління типами заявок і їхнє конфігурування.

Результатом роботи над дисертацією стала розробка архітектурного рішення програмного забезпечення системи технічної підтримки користувачів з

удосконаленою обробкою процесів IT послуг, а саме процесу класифікації інцидентів, чим підвищено ефективність роботи системи технічної підтримки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Ян Ван Бон., «ИТ Сервис Менеджмент. Введение» / Ян Ван Бон; — 1-е видання. — К.: Van Haren Publishing, 1999 — 524р.

2) Model–view–controller [Електронний ресурс] // Режим доступу: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

3) Help desk [Електронний ресурс] // Режим доступу: https://en.wikipedia.org/wiki/Help_desk

4) Troelsen Andrew, Japikse Philip. Pro C# 7: With .NET and .NET Core – Apress (2017) — 1595 p.

5) Martin Fowler, Patterns of Enterprise Application Architecture — Addison-Wesley Professional (2012) — 560 p.

6) JSON Web Token [Електронний ресурс] // Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JSON_Web_Token.

7) Douglas Crockford, JavaScript: The Good Parts — O'Reilly Media (2008) — 176 p.

8) Microsoft SQL Server [Електронний ресурс] // Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Microsoft_SQL_Server.

9) HTTP [Електронний ресурс] // Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/HTTP.

10) .NET Framework [Електронний ресурс] // Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/.NET_Framework.

11) Bug tracking system [Електронний ресурс] // Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Bug_tracking_system.

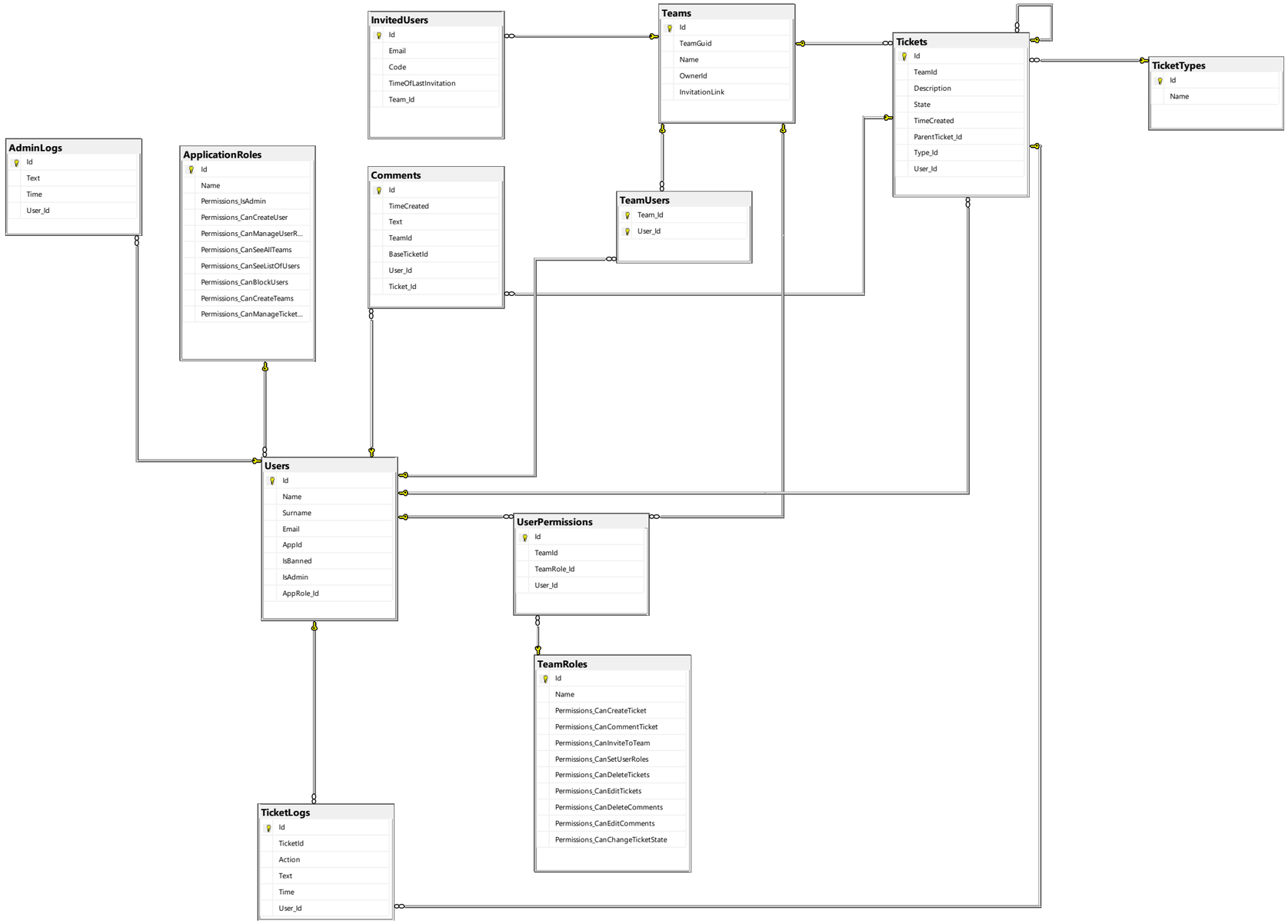
12) Issue tracking system [Електронний ресурс] // Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Issue_tracking_system.

13) Customer support [Електронний ресурс] // Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Customer_support.

ДОДАТОК А

ДІАГРАМА СУТНОСТЕЙ СИСТЕМИ

Діаграма сутностей



Демонстраційний плакат до магістерської дисертації
"Архітектурне рішення програмного забезпечення системи технічної підтримки"

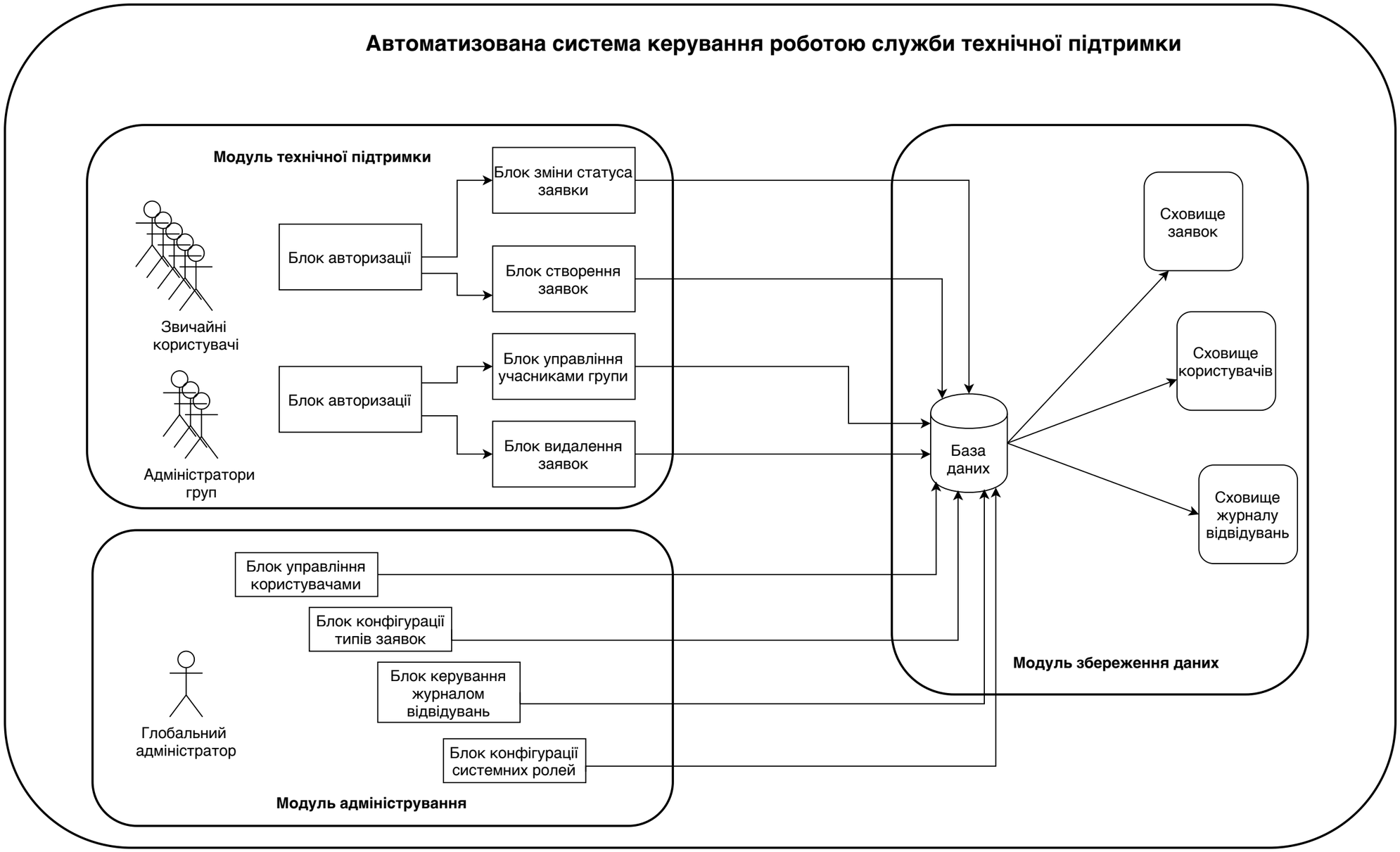
Виконала студентка гр. ІТ-04мп Петрова М.Є.

Керівник Новінський В. П.

ДОДАТОК Б

АРХІТЕКТУРА СИСТЕМИ

Архітектура системи



Демонстраційний плакат до магістерської дисертації
"Архітектурне рішення програмного забезпечення системи технічної підтримки"

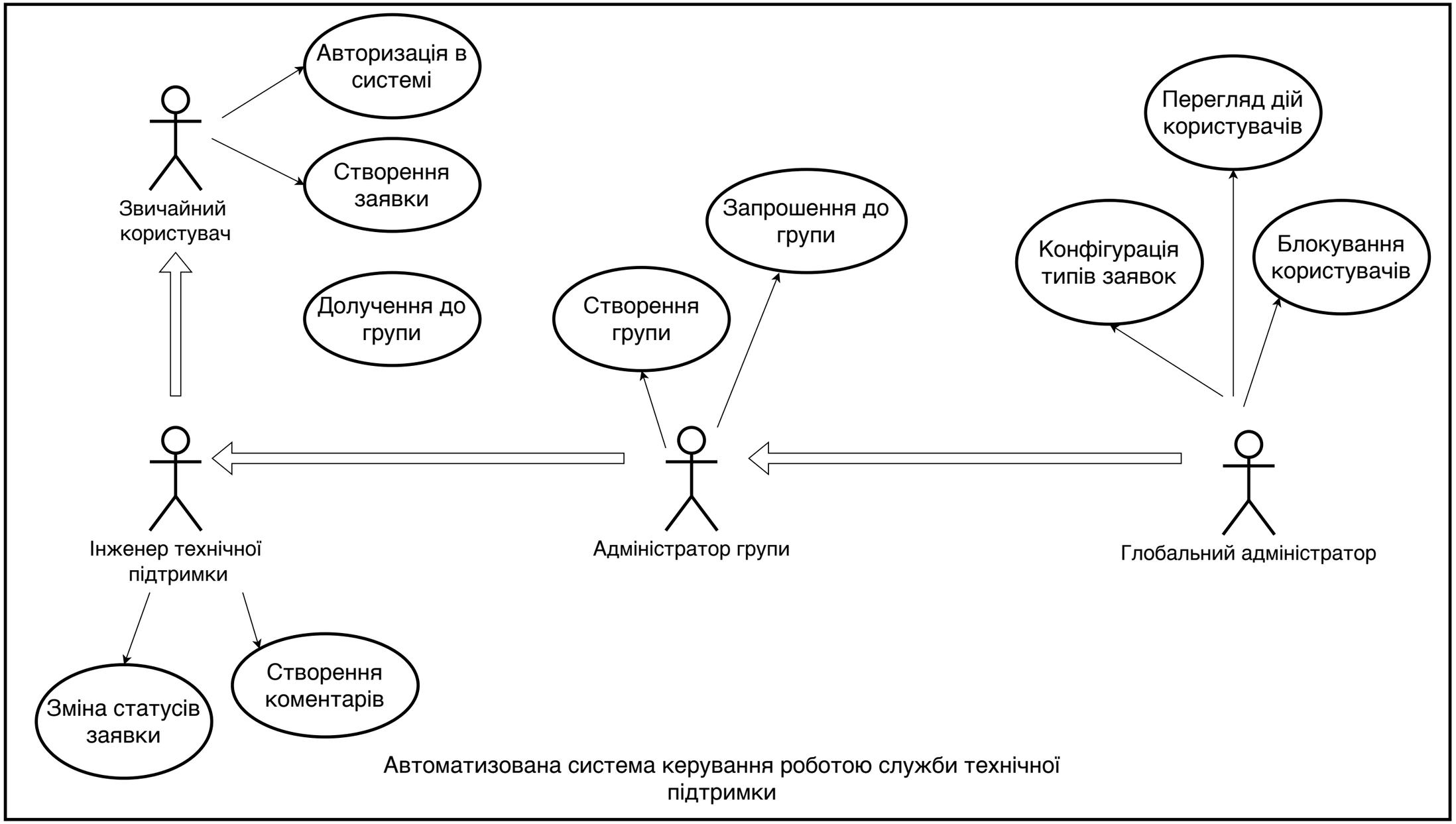
Виконала студентка гр. ІТ-04мп Петрова М.Є.

Керівник Новінський В. П.

ДОДАТОК В

ДІАГРАМА ПОВЕДІНКИ СИСТЕМИ

Діаграма поведінки



Демонстраційний плакат до магістерської дисертації
"Архітектурне рішення програмного забезпечення системи технічної підтримки"

Виконала студентка гр. ІТ-04мп Петрова М.Є.

Керівник Новінський В. П.

ДОДАТОК Г

ПРИКЛАД КОДУ ПРОГРАМИ

```
using
System;
        using System.Collections.Generic;
        using System.Data.Entity;
        using System.Linq;
        using System.Threading.Tasks;
        using System.Web;
        using HelpDeskTeamProject.Context;
        using HelpDeskTeamProject.DataModels;

        namespace HelpDeskTeamProject.Services
        {
            public class TicketManagerService : ITicketManager
            {
                IAppContext db;
                ITicketLogger ticketLogger;

                public TicketManagerService(IAppContext context, ITicketLogger tikLog)
                {
                    db = context;
                    ticketLogger = tikLog;
                }

                public async Task<Ticket> Add(TicketBase newTicket, User user)
                {
                    Ticket baseTicket = null;
                    if (newTicket.BaseTicketId != null && newTicket.Description != null &&
        newTicket.BaseTicketId > 0 && newTicket.BaseTeamId > 0)
                    {
                        baseTicket = await db.Tickets.SingleOrDefaultAsync(x => x.Id ==
        newTicket.BaseTicketId);
                    }
                    if (newTicket.Description != null && newTicket.BaseTeamId > 0 && user !=
        null && newTicket.TypeId > 0)
                    {
```

```
                TicketType ticketType = await db.TicketTypes.SingleOrDefaultAsync(x =>
x.Id == newTicket.TypeId);

                if (ticketType != null)

                {

                    Ticket ticket = new Ticket(newTicket.BaseTeamId, user,
newTicket.Description, ticketType, DateTime.Now, TicketState.New, baseTicket);

                    Ticket ticketFromDb = db.Tickets.Add(ticket);

                    ticketLogger.WriteTicketLog(user, TicketAction.CreateTicket,
baseTicket);

                    await db.SaveChangesAsync();

                    return ticketFromDb;

                }

            }

            throw new ArgumentNullException();

        }


        public async Task<bool> ChangeState(int? ticketId, int? state, User user)

        {

            if (ticketId != null && state != null && ticketId > 0 && state >= 0 &&
state <= 3 && user != null)

            {

                Ticket ticket = await db.Tickets.SingleOrDefaultAsync(x => x.Id ==
ticketId);

                if (ticket != null)

                {

                    ticket.State = (TicketState)state;

                    ticketLogger.WriteTicketLog(user, TicketAction.StateChange,
ticket);

                    await db.SaveChangesAsync();

                    return true;

                }

            }

            throw new ArgumentNullException();

        }


        public async Task<bool> Delete(int? id, User user)

        {

            if (id != null && user != null)
```

```
            {
                Ticket ticket = await db.Tickets.Include(z =>
z.ParentTicket).SingleOrDefaultAsync(x => x.Id == id);
                if (ticket != null)
                {
                    if (ticket.Comments.Count > 0)
                    {
                        db.Comments.RemoveRange(ticket.Comments);
                    }
                    if (ticket.ChildTickets.Count > 0)
                    {
                        db.Tickets.RemoveRange(ticket.ChildTickets);
                    }
                    if (ticket.Logs.Count > 0)
                    {
                        db.TicketLogs.RemoveRange(ticket.Logs);
                    }
                    ticketLogger.WriteTicketLog(user, TicketAction.DeleteTicket,
ticket.ParentTicket);
                    db.Tickets.Remove(ticket);
                    await db.SaveChangesAsync();
                    return true;
                }
            }
            throw new ArgumentNullException();
        }

        public async Task<Ticket> Edit(int? id, string description, int? type, User
user)
        {
            if (id != null && description != null && type != null && description != ""
&& user != null)
            {
                Ticket ticket = await db.Tickets.Include(y =>
y.User).SingleOrDefaultAsync(x => x.Id == id);
                TicketType newType = await db.TicketTypes.SingleOrDefaultAsync(x =>
x.Id == type);
                if (ticket != null && newType != null)
```

```
            {
                ticket.Description = description;
                ticket.Type = newType;
                ticketLogger.WriteTicketLog(user, TicketAction.EditTicket, ticket);
                await db.SaveChangesAsync();
                return ticket;
            }
        }
        throw new ArgumentNullException();
    }

    public async Task<Ticket> GetTicketById(int? id)
    {
        if (id != null)
        {
            Ticket ticket = await db.Tickets.Include(y => y.User)
                .Include(s => s.ChildTickets)
                .SingleOrDefaultAsync(x => x.Id == id);
            if (ticket != null)
            {
                ticket.ChildTickets = await db.Tickets.Include(z => z.User)
                    .Include(e => e.Type)
                    .Include(y => y.ChildTickets)
                    .Include(w => w.Comments)
                    .Where(x => x.ParentTicket.Id == ticket.Id)
                    .ToListAsync();
                return ticket;
            }
        }
        throw new ArgumentNullException();
    }

    public async Task<Ticket> GetTicketNoInclude(int? id)
    {
        if (id != null)
```

```
            {
                Ticket ticket = await db.Tickets.Include(t =>
t.User).SingleOrDefaultAsync(x => x.Id == id);
                return ticket;
            }
            throw new ArgumentNullException();
        }

        public async Task<List<Ticket>> GetTicketsByTeam(int? teamId)
        {
            if (teamId != null)
            {
                Team curTeam = await db.Teams.Include(x =>
x.Tickets).SingleOrDefaultAsync(y => y.Id == teamId);
                if (curTeam != null)
                {
                    List<Ticket> curTickets = await db.Tickets.Include(x =>
x.ChildTickets).Include(y => y.Comments).Include(z => z.User)
                        .Where(s => s.ParentTicket == null).Where(q => q.TeamId ==
curTeam.Id).ToListAsync();
                    if (curTickets != null)
                    {
                        return curTickets;
                    }
                }
            }
            throw new ArgumentNullException();
        }

        public async Task<List<Ticket>> GetTicketsByTeamAndType(int? teamId, int?
typeId)
        {
            if (teamId != null && typeId != null && teamId > 0 && typeId > 0)
            {
                TicketType curType = await db.TicketTypes.SingleOrDefaultAsync(x =>
x.Id == typeId);
                Team curTeam = await db.Teams.SingleOrDefaultAsync(x => x.Id ==
teamId);
```

```
                        if (curTeam != null && curType != null)
                        {
                            List<Ticket> ticketsList = await db.Tickets.Where(x => x.TeamId ==
        curTeam.Id).Where(y => y.Type.Id == typeId).ToListAsync();
                            if (ticketsList != null)
                            {
                                return ticketsList;
                            }
                        }
                    }
                    throw new ArgumentNullException();
                }
            }
        }
namespace HelpDeskTeamProject.Services
{
    public static class AdminService
    {
        public static bool CreateAdminButton(string userName)
        {
            AppContext dbContext = new AppContext();

            var currentUser = dbContext.Users.Where(u => u.Email ==
userName).FirstOrDefault();
            if (currentUser != null)
            {
                if (currentUser.IsAdmin)
                {
                    return true;
                }
            }
            return false;
        }
    }
        }
namespace HelpDeskTeamProject.Services
{
    public class CommentService : ICommentManager
    {
        IAppContext db;
        ITicketLogger ticketLogger;

        public CommentService(IAppContext context, ITicketLogger tikLog)
        {
            db = context;
            ticketLogger = tikLog;
        }
```

```
        public async Task<Comment> Add(User user, Ticket baseTicket, string text)
        {
            if (user != null && baseTicket != null && text != null && text != "")
            {
                Comment newComment = new Comment(text, user, DateTime.Now, baseTicket.TeamId,
baseTicket.Id);
                baseTicket.Comments.Add(newComment);
                ticketLogger.WriteTicketLog(user, TicketAction.CreateComment, baseTicket);
                await db.SaveChangesAsync();
                return newComment;
            }
            else
            {
                throw new ArgumentNullException();
            }
        }

        public async Task<bool> Delete(int? id, User user, Ticket ticket)
        {
            if (id != null && user != null && ticket != null)
            {
                Comment delComment = await db.Comments.SingleOrDefaultAsync(x => x.Id == id);
                db.Comments.Remove(delComment);
                ticketLogger.WriteTicketLog(user, TicketAction.DeleteComment, ticket);
                await db.SaveChangesAsync();
                return true;
            }
            else
            {
                throw new ArgumentNullException();
            }
        }
    }
}namespace HelpDeskTeamProject.Services
{
    public class DTOConverterService : IDtoConverter
    {
        public CommentDTO ConvertComment(Comment comment, User user, TeamPermissions
permissions)
        {
            if (comment != null && user != null && permissions != null)
            {
                CommentDTO dto = new CommentDTO(comment);
                if (comment.User.Id == user.Id || user.AppRole.Permissions.IsAdmin ||
permissions.CanDeleteComments)
                    dto.CanDelete = true;
                else
                    dto.CanDelete = false;
                return dto;
            }
            throw new ArgumentNullException();
        }

        public List<CommentDTO> ConvertCommentList(List<Comment> comments, User user,
TeamPermissions permissions)
        {
```

```
            if (comments != null && user != null && permissions != null)
            {
                List<CommentDTO> returnList = new List<CommentDTO>();
                foreach (Comment value in comments)
                {
                    returnList.Add(ConvertComment(value, user, permissions));
                }
                return returnList;
            }
            throw new ArgumentNullException();
        }

        public TicketDTO ConvertTicket(Ticket ticket, User user, TeamPermissions permissions)
        {
            if (ticket != null && user != null && permissions != null)
            {
                TicketDTO dto = new TicketDTO(ticket);
                if (ticket.User.Id == user.Id || user.AppRole.Permissions.IsAdmin)
                {
                    dto.CanDelete = true;
                    dto.CanEdit = true;
                }
                else
                {
                    dto.CanDelete = permissions.CanDeleteTickets;
                    dto.CanEdit = permissions.CanEditTickets;
                }
                return dto;
            }
            throw new ArgumentNullException();
        }

        public List<TicketDTO> ConvertTicketList(List<Ticket> tickets, User user,
TeamPermissions permissions)
        {
            if (tickets != null && user != null && permissions != null)
            {
                List<TicketDTO> returnList = new List<TicketDTO>();
                foreach (Ticket value in tickets)
                {
                    returnList.Add(ConvertTicket(value, user, permissions));
                }
                return returnList;
            }
            throw new ArgumentNullException();
        }

        public TicketLogDTO ConvertTicketLog(TicketLog log)
        {
            if (log != null)
            {
                TicketLogDTO dto = new TicketLogDTO(log);
                return dto;
            }
            throw new ArgumentNullException();
        }
```

```
        public List<TicketLogDTO> ConvertTicketLogList(List<TicketLog> logs)
        {
            if (logs != null)
            {
                List<TicketLogDTO> returnList = new List<TicketLogDTO>();
                foreach (TicketLog value in logs)
                {
                    returnList.Add(ConvertTicketLog(value));
                }
                return returnList;
            }


throw new ArgumentNullException();
        }
    }
using HelpDeskTeamProject.Classes;
using HelpDeskTeamProject.Context;
using HelpDeskTeamProject.DataModels;
using HelpDeskTeamProject.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Mail;
using System.Text;
using System.Web;
using System.Web.Configuration;
using System.Web.Mvc;

namespace HelpDeskTeamProject.Services
{
    public class TeamService : ITeamService
    {
        private IAppContext db;
        private string TEAM_OWNER_ROLE_NAME =
WebConfigurationManager.AppSettings["TeamOwnerRoleName"];
        private string DEFAULT_TEAM_ROLE_NAME =
WebConfigurationManager.AppSettings["DefaultTeamRoleName"];

        public TeamService(IAppContext context)
        {
            db = context;
        }

        public List<TeamMenuItem> GetUserTeamsList(User user)
        {
            var userTeamsList = db.Teams
               .Where(t => t.Users.Select(u => u.Id).Contains(user.Id))
               .ToList();

            var teamMenu = new List<TeamMenuItem>();

            foreach(var team in userTeamsList)
            {
```

```
        var teamMenuItem = new TeamMenuItem()
        {
            TeamId = team.Id,
            TeamName = team.Name
        };

        teamMenu.Add(teamMenuItem);
    }

    return teamMenu;
}


public List<TeamMenuItem> GetTeamsList()
{
    var teamsList = db.Teams
       .ToList();

    var teamMenu = new List<TeamMenuItem>();

    foreach (var team in teamsList)
    {
        var teamMenuItem = new TeamMenuItem()
        {
            TeamId = team.Id,
            TeamName = team.Name
        };

        teamMenu.Add(teamMenuItem);
    }

    return teamMenu;
}

public Team CreateTeamWithOwner(string teamName, User ownerOfTeam)
{
    Team createdTeam = CreateTeam(teamName, ownerOfTeam.Id);
    createdTeam.Users.Add(ownerOfTeam);
    db.Teams.Add(createdTeam);
    db.SaveChanges();

    return createdTeam;
}

private Team CreateTeam(string teamName, int ownerId)
{
    Guid teamGuid = Guid.NewGuid();

    var owner = db.Users.Find(ownerId);

    var request = HttpContext.Current.Request;
    string siteUrl = $"{request.Url.Scheme}://{request.Url.Authority}";

    Team team = new Team()
    {
        Name = teamName,
```

```
        OwnerId = ownerId,
        TeamGuid = teamGuid,
        InvitationLink = siteUrl + "/teams/jointeam/" + teamGuid.ToString() + "/",
        InvitedUsers = new List<InvitedUser>(),
        UserPermissions = new List<UserPermission>(),
        Tickets = new List<Ticket>(),
        Users = new List<User>()
    };


    var ownerRole = db.TeamRoles
       .Where(tr => tr.Name == TEAM_OWNER_ROLE_NAME)
       .FirstOrDefault();

    if (ownerRole == null)
        ownerRole = CreateTeamOwnerRole();

    var ownerPermission = new UserPermission()
    {
        TeamRole = ownerRole,
        User = owner,
        TeamId = team.Id
    };

    team.UserPermissions.Add(ownerPermission);

    return team;
}

private TeamRole CreateTeamOwnerRole()
{
    return new TeamRole()
    {
        Name = TEAM_OWNER_ROLE_NAME,
        Permissions = new TeamPermissions()
        {
            CanInviteToTeam = true,
            CanChangeTicketState = true,
            CanCommentTicket = true,
            CanCreateTicket = true,
            CanDeleteComments = true,
            CanDeleteTickets = true,
            CanEditComments = true,
            CanEditTickets = true,
            CanSetUserRoles = true
        }
    };
}

public int GetTeamOwnerId(int teamId)
{
    var team = db.Teams.Find(teamId);
    return team.OwnerId;
}

public ManageTeamViewModel CreateManageTeamViewModel(int teamId)
```

```
        {
            var team = db.Teams.Find(teamId);

            var teamMembers = team.Users;

            var viewModel = new ManageTeamViewModel()
            {
                Team = team,
                TeamMembers = new List<TeamMemberInfo>()
            };

            foreach (var teamMember in teamMembers)
            {
                if (teamMember.Id == team.OwnerId)
                    continue;

                var teamRole = team.UserPermissions
                    .Where(permission => permission.User == teamMember && permission.TeamId ==
team.Id)
                    .FirstOrDefault().TeamRole;

                var availableTeamRoles = db.TeamRoles
                    .Where(role => role.Name != TEAM_OWNER_ROLE_NAME).
                    ToList();

                var selectListForAvailableTeamRoles
                    = new SelectList(availableTeamRoles, "Id", "Name", teamRole.Id);

                var memberInfo = new TeamMemberInfo()
                {
                    TeamMember = teamMember,
                    TeamRole = teamRole,
                    AvailableTeamRoles = selectListForAvailableTeamRoles
                };

                viewModel.TeamMembers.Add(memberInfo);
            }
            return viewModel;
        }

        public bool ChangeUserRoleInTeam(int userId, int teamId, int newRoleId, User
currentUser)
        {
            var team = db.Teams.Find(teamId);

            var user = db.Users.Find(userId);

            var role = db.TeamRoles.Find(newRoleId);

            if (team == null || user == null || role == null)
                return false;

            var currentUserRole = team.UserPermissions
                .Where(perm => perm.User == currentUser && perm.TeamId == team.Id)
                .FirstOrDefault()
                .TeamRole;
```

```
        if (!currentUserRole.Permissions.CanSetUserRoles)
            return false;

        var userPermission = team.UserPermissions
            .Where(up => up.User == user && up.TeamId == team.Id)
            .FirstOrDefault();

        if (userPermission == null)
            return false;

        if (userPermission.TeamRole != role)
            userPermission.TeamRole = role;

        db.SaveChanges();

        return true;
    }

    public bool DeleteUserFromTeam(int userId, int teamId)
    {
        var team = db.Teams.Find(teamId);

        var user = db.Users.Find(userId);

        if (team == null || user == null)
            return false;

        var userPermission = team.UserPermissions
            .Where(up => up.User == user && up.TeamId == team.Id)
            .FirstOrDefault();

        var deletedUser = GetDeletedUser();

        var userTicketsInTeam = db.Tickets
            .Where(ticket => ticket.User.Id == user.Id && ticket.TeamId == teamId)
            .ToList();

        foreach(var ticket in userTicketsInTeam)
        {
            ticket.User = deletedUser;
            var logs = ticket.Logs;
            foreach(var log in logs)
            {
                log.User = deletedUser;
            }
        }

        var userCommentsInTeam = db.Comments
            .Where(comment => comment.TeamId == teamId && comment.User.Id == user.Id)
            .ToList();

        foreach(var comment in userCommentsInTeam)
        {
            comment.User = deletedUser;
        }
```

```
            team.UserPermissions.Remove(userPermission);
            user.Teams.Remove(team);
            team.Users.Remove(user);
            db.Permissions.Remove(userPermission);
            db.SaveChanges();

            return true;
        }

        private User GetDeletedUser()
        {
            var deletedUser = db.Users
                .Where(user => user.Name == "Deleted" && user.Surname == "User" && user.AppId
== null)
                .FirstOrDefault();

            if (deletedUser == null)
                deletedUser = CreateDeletedUser();

            return deletedUser;
        }

        private User CreateDeletedUser()
        {
            ApplicationRole defaultRole = db.AppRoles.SingleOrDefault(x =>
x.Name.Equals("default-user"));
            if (defaultRole == null)
            {
                defaultRole = new ApplicationRole("default-user", new
ApplicationPermissions(false, false, false, false, false, false, true, false));
            }

            User deletedUser = new User()
            {
                AppId = null,
                Email = "deleted@gmail.com",
                IsAdmin = false,
                IsBanned = false,
                Name = "Deleted",
                Surname = "User",
                Teams = new List<Team>(),
                AppRole = defaultRole
            };

            db.Users.Add(deletedUser);
            db.SaveChanges();

            return deletedUser;
        }

        public bool IsUserAbleToManageTeam(int teamId, User user)
        {
            var team = db.Teams.Find(teamId);

            if (team == null)
```

```
                return false;

            if (team.OwnerId != user.Id)
                return false;

            return true;
        }

        public string InviteUserToTeam(int teamId, string email)
        {
            var team = db.Teams.Find(teamId);

            bool isUserAlreadyInvited = team.InvitedUsers
                .Where(user => user.Email.ToLower() == email.ToLower()).
                Count() > 0;

            bool isUserAlreadyTeamMember = team.Users
                .Where(user => user.Email.ToLower() == email.ToLower())
                .Count() > 0;

            if (!isUserAlreadyInvited && !isUserAlreadyTeamMember)
            {
                var userCode = GenerateInvitationCode();

                var invitedUser = new InvitedUser()
                {
                    Email = email,
                    Code = userCode,
                    TimeOfLastInvitation = DateTime.Now
                };

                team.InvitedUsers.Add(invitedUser);
                db.SaveChanges();

                string invitationLink = team.InvitationLink + invitedUser.Id.ToString();
                string emailMessage = CreateInvitationEmailMessage(team.Name, invitationLink,
userCode);

                SendInvitationEmail(email, emailMessage);

                return "";
            }

            if (isUserAlreadyInvited)
                return "User with this email address is already invited to team.";

            return "User with this email address is already in team.";
        }

        public bool ReinviteUserToTeam(int invUserId, int teamId, DateTime currentTime)
        {
            var team = db.Teams.Find(teamId);
            var invitedUser = team.InvitedUsers.Find(user => user.Id == invUserId);

            if (invitedUser == null)
                return false;
```

```
            var invitedEarlierThanDayAgo = currentTime - invitedUser.TimeOfLastInvitation >
TimeSpan.FromHours(24);


            if (invitedEarlierThanDayAgo)
            {
                string invitationLink = team.InvitationLink + invitedUser.Id.ToString();
                string emailMessage = CreateInvitationEmailMessage(team.Name, invitationLink,
invitedUser.Code);

                SendInvitationEmail(invitedUser.Email, emailMessage);
                invitedUser.TimeOfLastInvitation = currentTime;
                db.SaveChanges();

                return true;
            }

            return false;
        }

        private void SendInvitationEmail(string emailTo, string emailText)
        {
            string emailServiceLogin =
WebConfigurationManager.AppSettings["EmailServiceLogin"];
            string emailServicePassword =
WebConfigurationManager.AppSettings["EmailServicePassword"];

            var client = new SmtpClient("smtp.gmail.com", 587)
            {
                Credentials = new NetworkCredential(emailServiceLogin, emailServicePassword),
                EnableSsl = true
            };

            client.Send(emailServiceLogin, emailTo, "Help Desk Invitation To The Team",
emailText);
        }

        private string CreateInvitationEmailMessage(string teamName, string invitationLink,
int code)
        {
            StringBuilder message = new StringBuilder();
            message.AppendLine("Dear Sir/Madam,");
            message.AppendLine();
            message.AppendLine($"You were invited to the team \"{teamName}\".");
            message.AppendLine($"To join the team follow the link: {invitationLink}");
            message.AppendLine($"Your code: {code}");
            message.AppendLine("If you are not interested in our service, delete this mail.");
            message.AppendLine();
            message.AppendLine("Yours faithfully,");
            message.AppendLine("Help Desk Service");
            return message.ToString();
        }

        private int GenerateInvitationCode()
        {
```

```
            Random random = new Random();
            return random.Next(100000, 999999);
        }

        public InviteUserToTeamViewModel CreateInviteUserToTeamViewModel(int teamId, User
currentUser)
        {
            var team = db.Teams.Find(teamId);

            if (team == null || team.OwnerId != currentUser.Id)
                return null;

            var currentUserRole = team.UserPermissions
                .Where(perm => perm.User == currentUser && perm.TeamId == teamId)
                .FirstOrDefault()
                .TeamRole;

            if (!currentUserRole.Permissions.CanInviteToTeam)
                return null;

            var viewModel = new InviteUserToTeamViewModel()
            {
                TeamToInvite = team
            };

            return viewModel;
        }

        public JoinTeamViewModel CreateJoinTeamViewModel(Guid teamGuid, int invitedUserId)
        {
            var team = db.Teams
               .Where(t => t.TeamGuid == teamGuid)
               .FirstOrDefault();

            if (team == null)
                return null;

            var invitedUser = team.InvitedUsers.Find(iu => iu.Id == invitedUserId);

            if (invitedUser == null)
                return null;

            invitedUser.Code = 0;

            var viewModel = new JoinTeamViewModel()
            {
                InvitedUser = invitedUser,
                TeamId = team.Id,
                TeamName = team.Name
            };

            return viewModel;
        }

        public List<Team> GetAllTeams()
        {
```

```
            return db.Teams.ToList();
        }

        public string JoinTeam(int teamId, int invitedUserId, int enteredCode, User
currentUser)
        {
            var team = db.Teams.Find(teamId);

            var invitedUser = team.InvitedUsers
                    .Find(iu => iu.Id == invitedUserId);

            if (invitedUser == null)
                return "User is not invited to team.";

            if (invitedUser.Code == enteredCode
                && currentUser.Email.ToLower() == invitedUser.Email.ToLower()
                && !currentUser.Teams.Contains(team) && !team.Users.Contains(currentUser))
            {
                team.Users.Add(currentUser);
                team.InvitedUsers.Remove(invitedUser);

                var defaultTeamRole = GetDefaultTeamRole();

                var defaultUserPermission = new UserPermission()
                {
                    TeamId = team.Id,
                    User = currentUser,
                    TeamRole = defaultTeamRole
                };

                team.UserPermissions.Add(defaultUserPermission);
                currentUser.Teams.Add(team);

                db.SaveChanges();

                return "";
            }

            if (invitedUser.Code != enteredCode)
                return "Code does not match.";

            if (currentUser.Email.ToLower() != invitedUser.Email.ToLower())
                return "Your current email does not match email which refer to this
invitation.";

            if (currentUser.Teams.Contains(team) || team.Users.Contains(currentUser))
                return "You are already in team.";

            return "Unknown error";

        }

        private TeamRole GetDefaultTeamRole()
        {
            var defaultTeamRole = db.TeamRoles
                        .Where(role => role.Name == DEFAULT_TEAM_ROLE_NAME)
```

```
                    .FirstOrDefault();

            if (defaultTeamRole == null)
                defaultTeamRole = CreateDefaultTeamRole();

            return defaultTeamRole;
        }

        private TeamRole CreateDefaultTeamRole()
        {
            return new TeamRole()
            {
                Name = DEFAULT_TEAM_ROLE_NAME,
                Permissions = new TeamPermissions()
                { CanCommentTicket = true, CanCreateTicket = true }
            };
        }

        public User GetCurrentUser(string currentAppUserId)
        {
            var currentUser = db.Users
                .Where(user => user.AppId == currentAppUserId)
                .FirstOrDefault();

            return currentUser;
        }

        public bool TeamExists(int teamId)
        {
            var team = db.Teams.Find(teamId);

            return team != null;
        }


    }

}
        }
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using HelpDeskTeamProject.DataModels;

namespace HelpDeskTeamProject.Services
{
    public class TicketLoggerService : ITicketLogger
    {
        public void WriteTicketLog(User user, TicketAction action, Ticket onTicket)
        {
            if (user != null && onTicket != null)
            {
                TicketLog curLog = new TicketLog()
                {
```

```
                    Action = action,
                    TicketId = onTicket.Id,
                    Time = DateTime.Now,
                    User = user
                };
                switch (action)
                {
                    case TicketAction.CreateComment:
                        {
                            curLog.Text = string.Format("{0} {1} left a comment.", user.Name,
user.Surname);
                            break;
                        }
                    case TicketAction.DeleteComment:
                        {
                            curLog.Text = string.Format("{0} {1} deleted a comment.",
user.Name, user.Surname);
                            break;
                        }
                    case TicketAction.CreateTicket:
                        {
                            curLog.Text = string.Format("{0} {1} created a new ticket.",
user.Name, user.Surname);
                            break;
                        }
                    case TicketAction.DeleteTicket:
                        {
                            curLog.Text = string.Format("{0} {1} deleted a ticket.",
user.Name, user.Surname);
                            break;
                        }
                    case TicketAction.EditTicket:
                        {
                            curLog.Text = string.Format("{0} {1} edited a ticket.", user.Name,
user.Surname);
                            break;
                        }
                    case TicketAction.StateChange:
                        {
                            curLog.Text = string.Format("{0} {1} changed ticket's state.",
user.Name, user.Surname);
                            break;
                        }
                }
                onTicket.Logs.Add(curLog);
            }
        }
    }
}

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
```

```
using HelpDeskTeamProject.Context;
using HelpDeskTeamProject.DataModels;

namespace HelpDeskTeamProject.Services
{
    public class TicketManagerService : ITicketManager
    {
        IAppContext db;
        ITicketLogger ticketLogger;

        public TicketManagerService(IAppContext context, ITicketLogger tikLog)
        {
            db = context;
            ticketLogger = tikLog;
        }

        public async Task<Ticket> Add(TicketBase newTicket, User user)
        {
            Ticket baseTicket = null;
            if (newTicket.BaseTicketId != null && newTicket.Description != null &&
newTicket.BaseTicketId > 0 && newTicket.BaseTeamId > 0)
            {
                baseTicket = await db.Tickets.SingleOrDefaultAsync(x => x.Id ==
newTicket.BaseTicketId);
            }
            if (newTicket.Description != null && newTicket.BaseTeamId > 0 && user != null &&
newTicket.TypeId > 0)
            {
                TicketType ticketType = await db.TicketTypes.SingleOrDefaultAsync(x => x.Id ==
newTicket.TypeId);
                if (ticketType != null)
                {
                    Ticket ticket = new Ticket(newTicket.BaseTeamId, user,
newTicket.Description, ticketType, DateTime.Now, TicketState.New, baseTicket);
                    Ticket ticketFromDb = db.Tickets.Add(ticket);
                    ticketLogger.WriteTicketLog(user, TicketAction.CreateTicket, baseTicket);
                    await db.SaveChangesAsync();
                    return ticketFromDb;
                }
            }
            throw new ArgumentNullException();
        }

        public async Task<bool> ChangeState(int? ticketId, int? state, User user)
        {
            if (ticketId != null && state != null && ticketId > 0 && state >= 0 && state <= 3
&& user != null)
            {
                Ticket ticket = await db.Tickets.SingleOrDefaultAsync(x => x.Id == ticketId);
                if (ticket != null)
                {
                    ticket.State = (TicketState)state;
                    ticketLogger.WriteTicketLog(user, TicketAction.StateChange, ticket);
                    await db.SaveChangesAsync();
                    return true;
                }
```

```
            }
            throw new ArgumentNullException();
        }

        public async Task<bool> Delete(int? id, User user)
        {
            if (id != null && user != null)
            {
                Ticket ticket = await db.Tickets.Include(z =>
z.ParentTicket).SingleOrDefaultAsync(x => x.Id == id);
                if (ticket != null)
                {
                    if (ticket.Comments.Count > 0)
                    {
                        db.Comments.RemoveRange(ticket.Comments);
                    }
                    if (ticket.ChildTickets.Count > 0)
                    {
                        db.Tickets.RemoveRange(ticket.ChildTickets);
                    }
                    if (ticket.Logs.Count > 0)
                    {
                        db.TicketLogs.RemoveRange(ticket.Logs);
                    }
                    ticketLogger.WriteTicketLog(user, TicketAction.DeleteTicket,
ticket.ParentTicket);
                    db.Tickets.Remove(ticket);
                    await db.SaveChangesAsync();
                    return true;
                }
            }
            throw new ArgumentNullException();
        }

        public async Task<Ticket> Edit(int? id, string description, int? type, User user)
        {
            if (id != null && description != null && type != null && description != "" && user
!= null)
            {
                Ticket ticket = await db.Tickets.Include(y => y.User).SingleOrDefaultAsync(x
=> x.Id == id);
                TicketType newType = await db.TicketTypes.SingleOrDefaultAsync(x => x.Id ==
type);
                if (ticket != null && newType != null)
                {
                    ticket.Description = description;
                    ticket.Type = newType;
                    ticketLogger.WriteTicketLog(user, TicketAction.EditTicket, ticket);
                    await db.SaveChangesAsync();
                    return ticket;
                }
            }
            throw new ArgumentNullException();
        }

        public async Task<Ticket> GetTicketById(int? id)
```

```
        {
            if (id != null)
            {
                Ticket ticket = await db.Tickets.Include(y => y.User)
                    .Include(s => s.ChildTickets)
                    .SingleOrDefaultAsync(x => x.Id == id);
                if (ticket != null)
                {
                    ticket.ChildTickets = await db.Tickets.Include(z => z.User)
                       .Include(e => e.Type)
                       .Include(y => y.ChildTickets)
                       .Include(w => w.Comments)
                       .Where(x => x.ParentTicket.Id == ticket.Id)
                       .ToListAsync();
                    return ticket;
                }
            }
            throw new ArgumentNullException();
        }

        public async Task<Ticket> GetTicketNoInclude(int? id)
        {
            if (id != null)
            {
                Ticket ticket = await db.Tickets.Include(t => t.User).SingleOrDefaultAsync(x
=> x.Id == id);
                return ticket;
            }
            throw new ArgumentNullException();
        }

        public async Task<List<Ticket>> GetTicketsByTeam(int? teamId)
        {
            if (teamId != null)
            {
                Team curTeam = await db.Teams.Include(x => x.Tickets).SingleOrDefaultAsync(y
=> y.Id == teamId);
                if (curTeam != null)
                {
                    List<Ticket> curTickets = await db.Tickets.Include(x =>
x.ChildTickets).Include(y => y.Comments).Include(z => z.User)
                        .Where(s => s.ParentTicket == null).Where(q => q.TeamId ==
curTeam.Id).ToListAsync();
                    if (curTickets != null)
                    {
                        return curTickets;
                    }
                }
            }
            throw new ArgumentNullException();
        }

        public async Task<List<Ticket>> GetTicketsByTeamAndType(int? teamId, int? typeId)
        {
            if (teamId != null && typeId != null && teamId > 0 && typeId > 0)
            {
```

```
                TicketType curType = await db.TicketTypes.SingleOrDefaultAsync(x => x.Id ==
typeId);
                Team curTeam = await db.Teams.SingleOrDefaultAsync(x => x.Id == teamId);
                if (curTeam != null && curType != null)
                {
                    List<Ticket> ticketsList = await db.Tickets.Where(x => x.TeamId ==
curTeam.Id).Where(y => y.Type.Id == typeId).ToListAsync();
                    if (ticketsList != null)
                    {
                        return ticketsList;
                    }
                }
            }
            throw new ArgumentNullException();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using HelpDeskTeamProject.Context;
using HelpDeskTeamProject.DataModels;

namespace HelpDeskTeamProject.Services
{
    public class TicketTypeService : ITicketTypeManager
    {
        IAppContext db;

        public TicketTypeService(IAppContext context)
        {
            db = context;
        }

        public async Task<bool> CreateNew(TicketType newTicketType)
        {
            TicketType type = await db.TicketTypes.SingleOrDefaultAsync(x =>
x.Name.Equals(newTicketType.Name));
            if (type == null)
            {
                db.TicketTypes.Add(newTicketType);
                await db.SaveChangesAsync();
                return true;
            }
            else
            {
                return false;
            }
        }

        public async Task<List<TicketType>> GetAllTypes()
        {
```

```
            List<TicketType> ticketTypes = await db.TicketTypes.ToListAsync();
            return ticketTypes;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using HelpDeskTeamProject.Context;
using HelpDeskTeamProject.DataModels;
using Microsoft.AspNet.Identity;

namespace HelpDeskTeamProject.Services
{
    public class UserManagerService : IUserManager
    {
        IAppContext db;

        public UserManagerService(IAppContext context)
        {
            db = context;
        }

        public async Task<User> GetCurrentUser()
        {
            string userAppId = System.Web.HttpContext.Current.User.Identity.GetUserId();
            User curUser = await db.Users.SingleOrDefaultAsync(x =>
x.AppId.Equals(userAppId));
            if (curUser != null)
            {
                return curUser;
            }
            throw new ArgumentNullException();
        }
    }
}
```

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace HelpDeskTeamProject.Models
{
    public class ExternalLoginConfirmationViewModel
    {
        [Required]
        [Display(Name = "Адрес электронной почты")]
        public string Email { get; set; }
    }

    public class ExternalLoginListViewModel
    {
        public string ReturnUrl { get; set; }
    }
```

```
    public class SendCodeViewModel
    {
        public string SelectedProvider { get; set; }
        public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }
        public string ReturnUrl { get; set; }
        public bool RememberMe { get; set; }
    }

    public class VerifyCodeViewModel
    {
        [Required]
        public string Provider { get; set; }

        [Required]
        [Display(Name = "Код")]
        public string Code { get; set; }
        public string ReturnUrl { get; set; }

        [Display(Name = "Запомнить браузер?")]
        public bool RememberBrowser { get; set; }

        public bool RememberMe { get; set; }
    }

    public class ForgotViewModel
    {
        [Required]
        [Display(Name = "Email")]
        public string Email { get; set; }
    }

    public class LoginViewModel
    {
        [Required]
        [Display(Name = "Email")]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [Display(Name = "Remember me")]
        public bool RememberMe { get; set; }
    }

    public class RegisterViewModel
    {
        [Required]
        [Display(Name = "Your name")]
        [StringLength(200, ErrorMessage = "Name cannot have more that 200 characters or less
than 4.", MinimumLength = 4)]
        public string Name { get; set; }
```

```
        [Required]
        [Display(Name = "Your surname")]
        [StringLength(200, ErrorMessage = "Surname cannot have more that 200 characters or
less than 4.", MinimumLength = 4)]
        public string Surname { get; set; }

        [Required]
        [EmailAddress]
        [Display(Name = "Email")]
        public string Email { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "{0} length has to be at least {2} characters.",
MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password", ErrorMessage = "Password and confirmation does not match.")]
        public string ConfirmPassword { get; set; }
    }

    public class ResetPasswordViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email")]
        public string Email { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "{0} length has to be at least {2} characters.",
MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password", ErrorMessage = "Password and confirmation does not match.")]
        public string ConfirmPassword { get; set; }

        public string Code { get; set; }
    }

    public class ForgotPasswordViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email")]
        public string Email { get; set; }
    }
}
using HelpDeskTeamProject.DataModels;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HelpDeskTeamProject.Models
{
    public class AppAndTeamRolesViewModel
    {
        public List<ApplicationRole> ApplicationRoles { get; set; }

        public List<TeamRole> TeamRoles { get; set; }
    }
}
```

```
using System.Data.Entity;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;

namespace HelpDeskTeamProject.Models
{
    // В профиль пользователя можно добавить дополнительные данные, если указать больше
свойств для класса ApplicationUser. Подробности см. на странице
https://go.microsoft.com/fwlink/?LinkID=317594.
    public class ApplicationUser : IdentityUser
    {
        public async Task<ClaimsIdentity>
GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)
        {
            // Обратите внимание, что authenticationType должен совпадать с типом, определен-
ным в CookieAuthenticationOptions.AuthenticationType
            var userIdentity = await manager.CreateIdentityAsync(this,
DefaultAuthenticationTypes.ApplicationCookie);
            // Здесь добавьте утверждения пользователя
            return userIdentity;
        }
    }

    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext()
            : base("DefaultConnection", throwIfV1Schema: false)
        {
        }

        public static ApplicationDbContext Create()
        {
            return new ApplicationDbContext();
        }
    }
}
```

```
using HelpDeskTeamProject.DataModels;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
```

```
using System.Linq;
using System.Web;

namespace HelpDeskTeamProject.Models
{
    public class InviteUserToTeamViewModel
    {
        public Team TeamToInvite { get; set; }

        [EmailAddress]
        public string EmailOfInvitedUser { get; set; }
    }
}
```

```
using HelpDeskTeamProject.DataModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HelpDeskTeamProject.Models
{
    public class JoinTeamViewModel
    {
        public int TeamId { get; set; }

        public string TeamName { get; set; }

        public InvitedUser InvitedUser { get; set; }
    }
}
```

```
using HelpDeskTeamProject.DataModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace HelpDeskTeamProject.Models
{
    public class ManageTeamViewModel
    {
        public Team Team { get; set; }

        public List<TeamMemberInfo> TeamMembers { get; set; }

    }

    public class TeamMemberInfo
    {
        public User TeamMember { get; set; }

        public TeamRole TeamRole { get; set; }
```

```
        public SelectList AvailableTeamRoles { get; set; }
    }
}

using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNet.Identity;
using Microsoft.Owin.Security;

namespace HelpDeskTeamProject.Models
{
    public class IndexViewModel
    {
        public bool HasPassword { get; set; }
        public IList<UserLoginInfo> Logins { get; set; }
        public string PhoneNumber { get; set; }
        public bool TwoFactor { get; set; }
        public bool BrowserRemembered { get; set; }
    }

    public class ManageLoginsViewModel
    {
        public IList<UserLoginInfo> CurrentLogins { get; set; }
        public IList<AuthenticationDescription> OtherLogins { get; set; }
    }

    public class FactorViewModel
    {
        public string Purpose { get; set; }
    }

    public class SetPasswordViewModel
    {
        [Required]
        [StringLength(100, ErrorMessage = "Значение {0} должно содержать символов не менее:
{2}.", MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Новый пароль")]
        public string NewPassword { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Подтверждение нового пароля")]
        [Compare("NewPassword", ErrorMessage = "Новый пароль и его подтверждение не совпа-
дают.")]
        public string ConfirmPassword { get; set; }
    }

    public class ChangePasswordViewModel
    {
        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Текущий пароль")]
        public string OldPassword { get; set; }

        [Required]
```

```
        [StringLength(100, ErrorMessage = "Значение {0} должно содержать символов не менее:
{2}.", MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Новый пароль")]
        public string NewPassword { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Подтверждение нового пароля")]
        [Compare("NewPassword", ErrorMessage = "Новый пароль и его подтверждение не совпа-
дают.")]
        public string ConfirmPassword { get; set; }
    }

    public class AddPhoneNumberViewModel
    {
        [Required]
        [Phone]
        [Display(Name = "Номер телефона")]
        public string Number { get; set; }
    }

    public class VerifyPhoneNumberViewModel
    {
        [Required]
        [Display(Name = "Код")]
        public string Code { get; set; }

        [Required]
        [Phone]
        [Display(Name = "Номер телефона")]
        public string PhoneNumber { get; set; }
    }

    public class ConfigureTwoFactorViewModel
    {
        public string SelectedProvider { get; set; }
        public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HelpDeskTeamProject.Models
{
    public class TeamMenuItem
    {
        public int TeamId { get; set; }

        public string TeamName { get; set; }
    }
}
```

```
using HelpDeskTeamProject.DataModels;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HelpDeskTeamProject.Models
{
    public class TeamWithLastChangesViewModel
    {
        public Team Team { get; set; }

        public string LastTicketText { get; set; }

        public string LastTicketAuthor { get; set; }

        public string LastTicketTime { get; set; }

    }
}
```

ДОДАТОК Ґ

РЕЗУЛЬТАТ ПЕРЕВІРКИ РОБОТИ НА СПІВПАДІННЯ

UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1009329489

Дата проверки:
24.11.2021 12:33:36 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
24.11.2021 12:35:12 EET

ID пользователя:
76913

Название файла: **ПЗ Петрова ІТ-04мп Перевірка на співпадіння**

Количество страниц: 65 **Количество слов:** 13910 **Количество символов:** 103866 **Размер файла:** 116.33 KB **ID файла:** 1009353205

0.79% Совпадения

Наибольшее совпадение: 0.59% с **источником из Библиотеки** (ID **файла:** 1003905430)

Не найдено источников из Интернета

0.79% Источники из Библиотеки | **14** ..... **Страница** 67

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников