НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Геометричне моделювання в інформаційних системах»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «iOS-застосунок керування дипломним проектуванням»

Виконав:
студент IV курсу, групи ТР-72 _____
Пірко Дмитро Андрійович

Керівник:
асистент,
Тихоход Володимир Олександрович _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)

”___”_____2021р.

ЗАВДАННЯ
на дипломну роботу студенту
Пірку Дмитру Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи: iOS-застосунок керування дипломним проектуванням
Керівник роботи: Тихоход Володимир Олександрович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від травня 2021р. № **1168-с**

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи
мова програмування Swift, середовище розробки Xcode

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Розробити мобільний застосунок до Веб-сервісу системи керування дипломним проектуванням.

5. Перелік ілюстративного матеріалу

схеми архітектури програмного продукту, знімки інтерфейсу, скріншоти екранів застосунку тощо.

7. Дата видачі завдання ”___”_____201___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	9.10.20р.	
2.	Вивчення та аналіз задачі	12.04.21р. – 16.04.21р	
3.	Розробка архітектури та загальної структури системи	17.04.21р. – 22.04.21р.	
4.	Розробка структур окремих підсистем	23.04.21р – 25.04.21р.	
5.	Програмна реалізація системи	26.04.21р – 08.05.21р	
6.	Оформлення пояснювальної записки	09.05.21р – 20.05.2021	
7.	Захист програмного продукту	11.05.2021р.	
8.	Передзахист	25.05.2021р.	
9.	Захист	15.06.2021р.	

Студент _____ ___Пірко Д. А._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ __Тихоход В. О._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Пірко Дмитро Андрійович

Тема: iOS-застосунок керування дипломним проектуванням.

Спеціальність: 122 Комп'ютерні науки та інформаційні технології.

Установа: Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», 2021 р.

Бакалаврська робота: 69с., вступ,. 5 розділів, висновок, 27 джерел, 2 додатки, 29 рисунків, 7 таблиць.

Актуальність теми. На сьогоднішній день мобільні пристрої дуже розповсюджені. Люди користуються ними щодня, дехто майже не випускає його з рук. Використовують їх у найрізноманітніших задачах, наприклад щоб полегшити робочий процес. У зв'язку з цим виникає потреба у аналогах комп'юторних програм та Веб-сервісів для мобільних пристроїв. Керуючись цими потребами було поставлено задачу: розробити мобільний застосунок для операційної системи iOS.

Метою дослідження. Метою дослідження є розробка програмного продукту у вигляді iOS-застосунку, який буде надавати можливість проводити операції, аналогічні Веб-сервісу.

Об'єктом дослідження є iOS-застосунку системи керування дипломним проектуванням.

Предметом дослідження є Веб-система керування дипломним проектуванням.

Результати роботи: розроблено мобільний застосунок системи керування дипломним проектуванням.

Ключові слова: ДИПЛОМ; МОБІЛЬНИЙ ЗАСТОСУНОК, SWIFT; XCODE; UIKIT.

SUMMARY

Pirko Dmytro Andriyovich

Topic: iOS application of diploma design management system.

Specialty: 122 Computer Science and Information Technology.

Institution: National Technical University of Ukraine "Kyiv Polytechnic Institute. Igor Sikorsky ", 2021

Bachelor work: . s., introduction,. sections, conclusion, 21 source,. applications,. drawings,. tables.

Actuality of theme. Today, mobile devices are very common. People use them every day, some almost never let go. They are used in a variety of tasks, such as to facilitate the workflow. Therefore, there is a need for analogues of computer programs and Web services for mobile devices. Guided by these needs, the task was set: to develop a mobile application for the iOS operating system.

The purpose of the study. The purpose of the study is to develop a software product in the form of an iOS application, which will provide the ability to perform operations similar to the Web service.

The object of research is the iOS application of the diploma design management system.

The subject of research is the Web-based management system of diploma design.

Results of work: the mobile application of the diploma design management system is developed.

Keywords: DIPLOMA, MOBILE APP, SWIFT, XCODE, UIKIT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система.

HTTP – протокол прикладного рівня передачі даних.

Xcode – інтегроване середовище розробки.

Swift – мова програмування.

UIKit – бібліотека мови Swift.

URLSession– бібліотека мови Swift.

API – прикладний програмний інтерфейс.

ООП – об’єктно-орієнтоване програмування

Вступ

Дипломне проектування – це комплексна самостійна творча робота по вирішенню певної задачі, в процесі якої студент здобуває та поліпшує свої навички та вміння. Дипломне проектування має важливу роль у завершальному етапі навчання. Контролювання дипломним проектуванням здійснюють дипломні керівники. Процес дипломного проектування не легкий та має багато особливостей. Зазвичай такий процес супроводжується немалою кількістю файлів та документів, працювати з якими ефективніше та швидше у електронному вигляді.

Для полегшення цього процесу кафедрою АПЕПС було створено Web-систему керування дипломним проектуванням. Але Web-система доступна лише у браузері, що не зовсім зручно, наприклад, якщо користувачу потрібно переглянути інформацію з мобільного пристрою. У зв’язку з цим виникає потреба у аналогах комп’ютерних програм та Веб-сервісів для мобільних пристроїв.

1. ЗАДАЧА РОЗРОБКИ IOS-ЗАСТОСУНКУ СИСТЕМИ КЕРУВАННЯ ДИПЛОМНИМ ПРОЕКТУВАННЯМ

Головною метою роботи є розробка мобільного застосунку для операційної системи iOS, що буде допоміжним елементом у роботі з існуючим Web-порталом системи керування дипломним проектуванням. Реалізувати взаємодію мобільного застосунку з сервером. Створити сучасний, адаптивний під різні пристрої дизайн та реалізувати його у застосунку.

Застосунок повинен надавати можливість переглядати актуальну інформацію, що буде розміщена на web-порталі. Повинна бути присутня авторизація.

Застосунок буде розрахований на 3 категорії користувачів: студенти, викладачі та адміністратори. Застосунок матиме такі кабінети користувачів, як: кабінет студента, кабінет викладача та кабінет адміністратора. В кабінеті студента користувач зможе переглянути свою дипломну тему, графік виконання дипломної роботи, поточне навантаження викладачів, переглянути та редагувати список з файлами. Завантаження та видалення файлів буде можливе лише у повній Веб-версії системи. Роль студента буде заключною у системі. Студент матиме можливість переглядати список тем, подавати заявки на теми та слідкувати за статусами заявок. Коли студент отримає згоду від викладача щодо обраної теми, йому буде надана інформація про його тему. Викладач також матиме можливість переглядати список дипломних тем та навантаження. Також він зможе керувати власними темами, додавати чи видаляти студентів з певних тем.

Відповідно до категорій розподіляється відповідний функціонал. В кабінеті студента користувач зможе переглянути свою дипломну тему, графік виконання дипломної роботи, поточне навантаження викладачів, переглядати та редагувати список з файлами. Завантаження та видалення файлів буде можливе лише у повній Веб-версії системи. Роль студента буде заключною у системі. Студент матиме

можливість переглядати список тем, подавати заявки на теми, які сподобались та слідкувати за статусами заявок. Коли студент отримав згоду від викладача щодо обраної теми, йому буде надана інформація про його тему. Викладач також матиме можливість переглядати список дипломних тем та навантаження. Також він зможе керувати власними темами, додавати чи видаляти студентів з певних тем. Адміністратор зможе переглядати список тем, як і попередні ролі. Окрім того він матиме можливість переглядати список кафедр та їх завідувачів. За необхідністю редагувати ці дані.

Застосунок та сервер будуть взаємодіяти за допомогою HTTP-запитів. Вони конвертуються у рядковий формат JSON. Його використання обумовлено легкістю серіалізації та десеріалізації складних структур даних.

Використовуються GET та POST запити. GET – для отримання данних з серверу, POST – наприклад, для надсилання данних авторизації (логіну та пароля).

Сервер надає усі необхідні дані мобільному додатку. Він відповідальний за авторизацію користувача в системі. Сервер надає доступ до бази даних, що містять усі дані, інформацію про дипломні проекти тощо. API (Application Programming Interface) - це певне уявлення даних для взаємодії між додатками. В окремому випадку, в якості відповідного додатка, може виступати сервер. API - це описаний формат, якому повинні відповідати обидві сторони обміну даними.

Технологія різних API - це набір методів взаємодії. Система API в тому чи іншому вигляді представлена всюди [8].

Мобільний застосунок формуватиме запит до сервера. Запит формується в певному форматі. Сервер отримуватиме цей запит і програма розумітиме, що від неї потрібно. Сервер звертатиметься до бази даних на мові SQL, що теж є приватним поданням API.

База даних звертатиметься до файлу через API файлової системи. Файлова система звертатиметься до жорсткого диска - через його API-протокол. Можна продовжити поглиблення, але й так зрозуміло, що будь-яке завдання програмування - це вибудовування ієрархії API.

2. ПРОЦЕС КЕРУВАННЯ ДИПЛОМНИМ ПРОЕКТУВАННЯМ ТА ЗАСОБИ ЇЇ ПІДТРИМКИ

2.1 Організація процесу дипломного проектування

Дипломне проектування – це комплексна самостійна творча робота по вирішенню певної задачі, в процесі якої студент здобуває та поліпшує свої навички та вміння. Дипломне проектування має важливу роль у завершальному етапі навчання. Контролювання дипломним проектуванням здійснюють дипломні керівники. Дипломна робота обов’язково повинна мати певний рівень новизни, оригінальний характер і вміщувати результати наукових досліджень.

Тематика дипломних проектів повинна відповідати сучасним вимогам науки і техніки, вміщувати основні питання, з якими спеціалісти будуть зустрічатися на виробництві. Теми мають відповідати обсягу теоретичних знань і практичних навичок, одержаних студентами під час навчання.

Студент може обрати тему роботи з орієнтовної тематики кафедри або з проблем, які розробляються відповідно до комплексного плану наукових досліджень. Тема і план кваліфікаційної роботи можуть уточнюватися або змінюватися за поданням кафедри.

Рисунок 2.1. Схема процесу дипломного проектування

Обов’язками керівника кваліфікаційної роботи є:

- видати завдання дипломної роботи;
- допомогти в розробці календарного плану виконання;
- надати методичні рекомендації щодо виконання роботи;
- надавати допомогу у виконанні роботи;
- перевіряти виконання календарного роботи;
- підготувати відгук про роботу дипломника;

Після підписання роботи дипломником, керівником та консультантами вона подається завідувачу кафедри. Після оцінювання якості роботи завідувач кафедри допускає студента до її захисту.

Консультантами з окремих розділів кваліфікаційної роботи можуть призначатися професори і доценти, асистенти ВНЗ, а також висококваліфіковані фахівці, наукові співробітники інших установ, організацій і підприємств за профілем розділу.

Консультанти надають допомогу студенту в роботі над відповідним розділом, перевіряють якість виконання завдання, ставлять на титульному аркуші свій підпис.

Захист дипломного проекту проводиться на відкритому засіданні державної кваліфікаційної комісії. Після захисту роботи студент має здати матеріали роботи на кафедру.

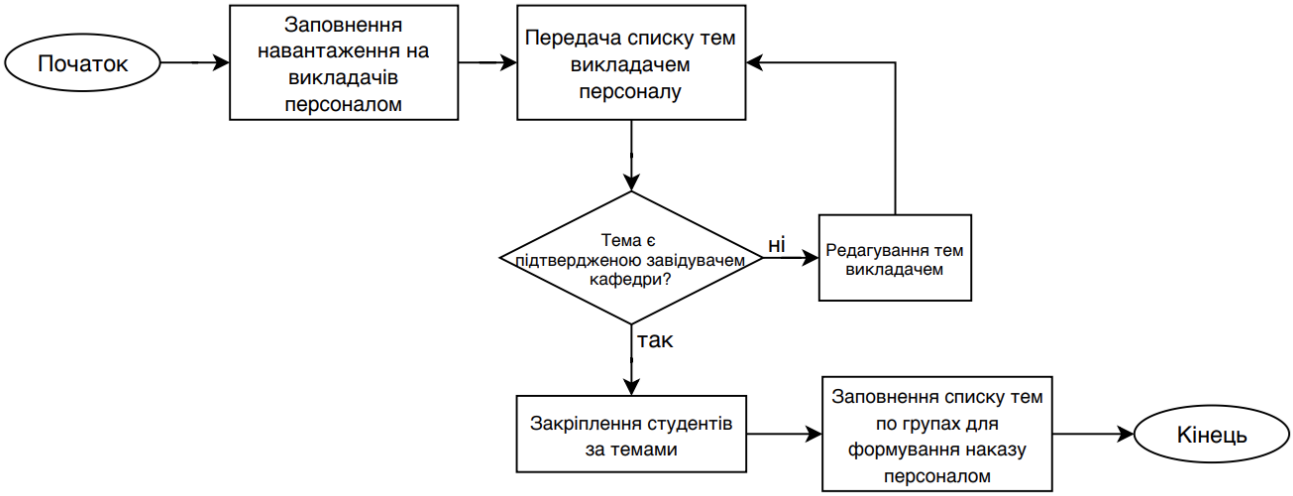


Рисунок 2.2. Етапи процесу керування дипломного проектування

Керування дипломним проектуванням у ВУЗ-і є складним процесом, оскільки він складається з багатьох етапів (рисунок 2.2) та процесів, що потребують контролю виконання. Тому було вирішено створити систему керування дипломним проектуванням, яка буде вирішувати проблеми автоматизації цих процесів.

Особливо функціонал належатиме навчальному персоналу. На рисунку 2.3 зображено основний алгоритм роботи системи керування дипломним проектуванням. На діаграмі активності перелічено основні дії користувачів системи. Окрім цього система повинна мати поштовий сервіс для відправки листів користувачам, сервіс аутентифікації та авторизації користувачів.

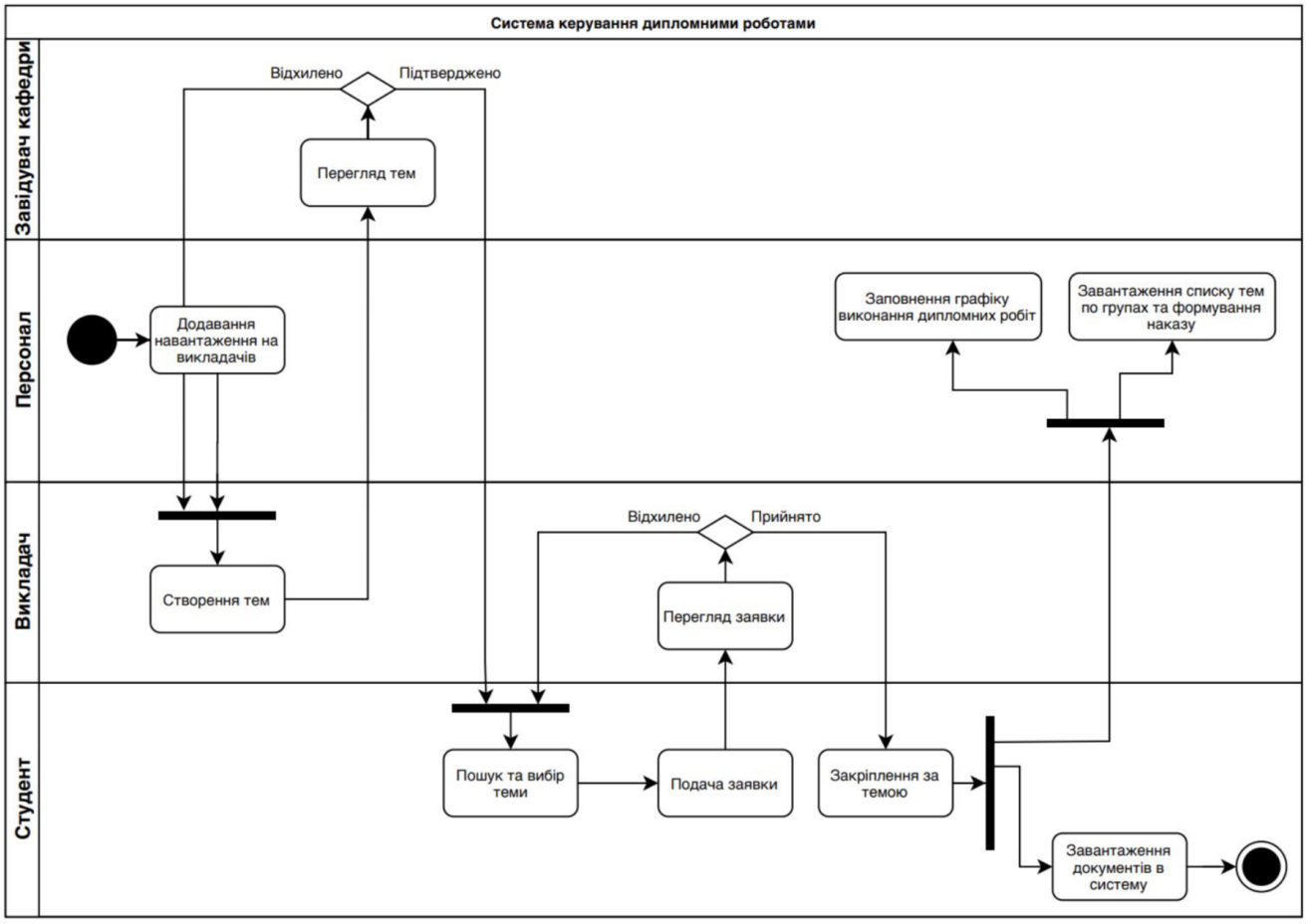


Рисунок 2.3. Алгоритм процесу керування дипломного проектування

Система є універсальною та, за потреби, здатною до безболісного розширення необхідного функціоналу. Завдяки системі усі учасники процесу дипломного проектування будуть почуватись впевненіше.

2.2 Програмна система підтримки процесу керування дипломним проектуванням

На сучасному етапі на кафедрі АПЕПС розроблена програмна система, що здійснює підтримку процесу керування дипломним проектуванням. Розглянемо особливості цієї системи.

2.2.1 Архітектура системи

Архітектура системи зображена на рисунку 4.2.

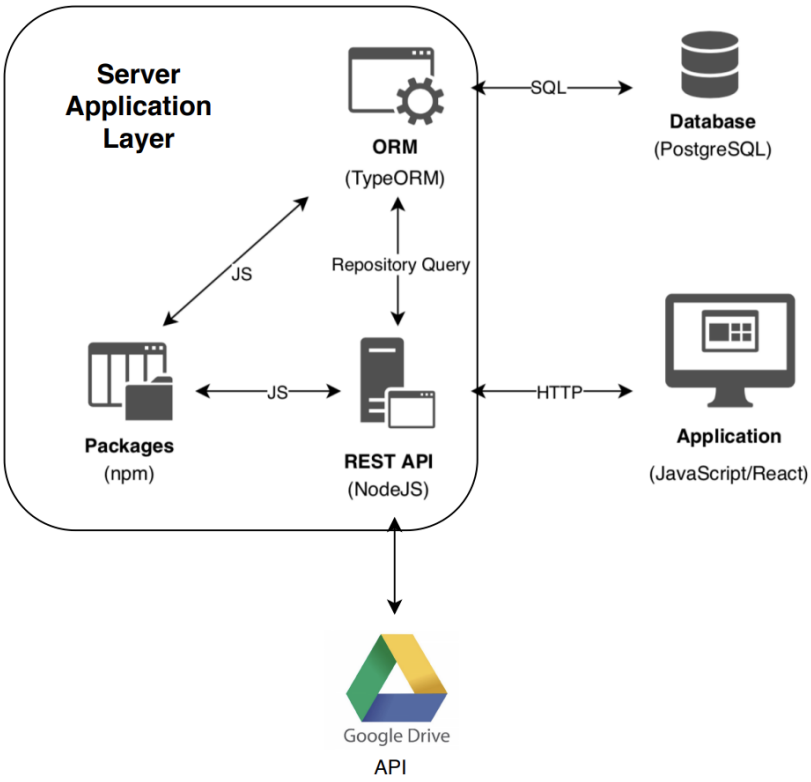


Рисунок 2.4 Схема архітектури системи

До головних елементів системи відносяться:

- REST API – набір веб-служб, що забезпечують доступ до даних системи;
- Application – веб-застосунок.

2.2.2 Автентифікація та авторизація

Система містить маршрутів по яким відбуваються запити. Але не всі користувачі мають доступ виконувати їх, необхідно бути авторизованим під певною роллю. Адміністратор в даній системі є окремим користувачем.

Ролей у системі кілька:

- Навчальний персонал;
- Викладач, Завідувач кафедри;
- Студент.

Ієрархія ролей показана на рисунку 4.7.

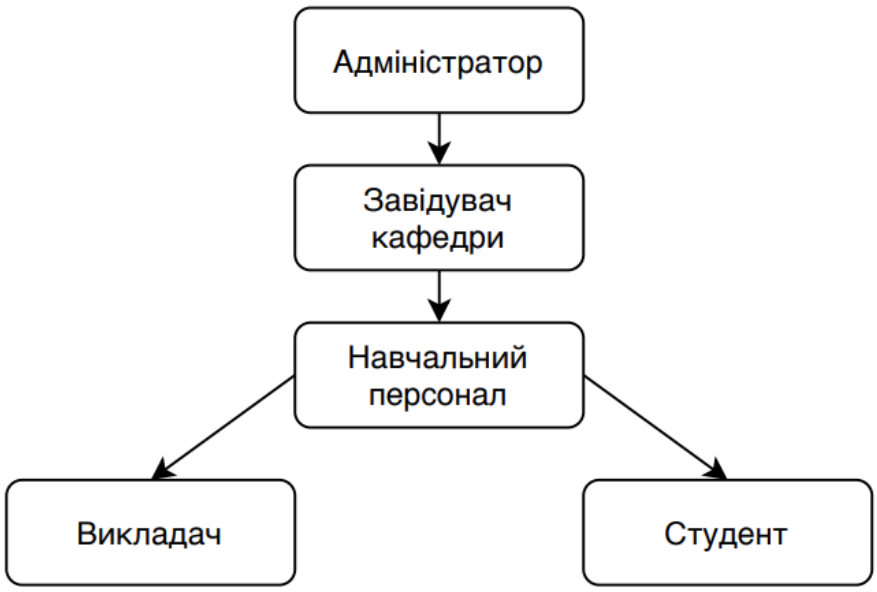


Рисунок 2.5 Ієрархія ролей системи

Аутентифікація є важливою частиною більшості систем. Існує багато різних підходів та стратегій для обробки аутентифікації. Підхід до будь-якого проекту залежить від його конкретних вимог до застосування.

У даній системі було вирішено використовувати метод аутентифікації за допомогою JWT (JSON Web Token). Для того, щоб кожен користувач мав змогу отримувати свою інформацію, щоб це було приватно, необхідно реалізувати функціонал логіну. Зазвичай при логіні користувач вводить свій username та пароль, у нашому випадку використовується електронна адреса користувача та пароль. Після отримання успішного результату, користувач отримує токен JWT, у токені зберігається певна інформація про користувача. Усі наступні запити до API необхідно виконувати разом з заголовком Authorization, значенням якого є токен (рисунок 4.5).

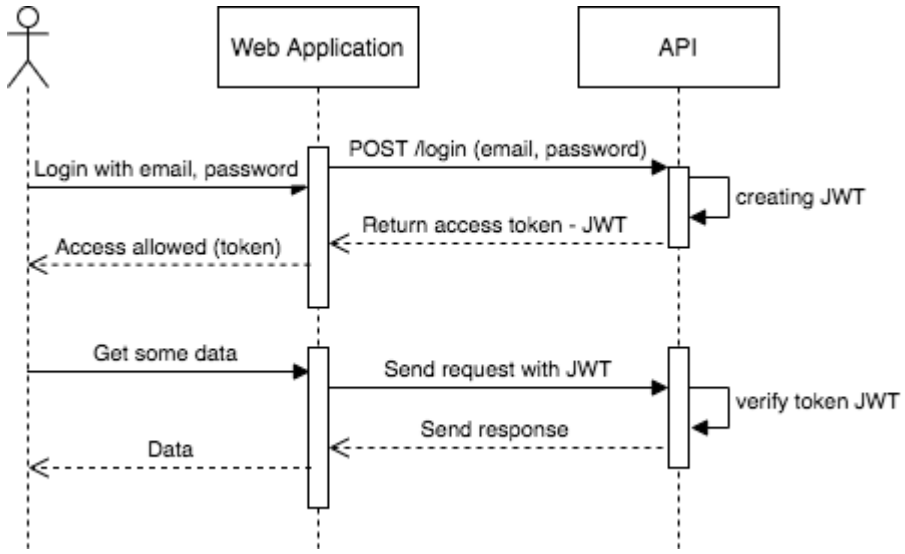


Рисунок 2.6 Діаграма послідовності процесу аутентифікації

У даній реалізації було використано бібліотеку Passport, яка і виконує за нас усі необхідні дії для виконання аутентифікації.

Passport — найпопулярніша бібліотека аутентифікації node.js, добре відома спільнотою та успішно використовується у багатьох виробничих додатках. Інтегрувати цю бібліотеку до нашого Nest застосунку можна за допомогою модуля @nestjs/passport. На високому рівні Passport виконує ряд кроків:

- аутентифікує користувача, перевіривши його "облікові дані" (наприклад, ім'я користувача / пароль, JSON Web Token (JWT) або токен посвідчення від постачальника ідентифікаційних даних);
- керує станом аутентифікації (видаючи портативний токен, такий як JWT, або створюючи сеанс Express);
- Додає інформацію про аутентифікованого користувача до об’єкта Request для подальшого використання в обробниках маршрутів.

Passport має багату екосистему стратегій, що реалізують різні механізми автентифікації. Незважаючи на простість у концепції, набір паспортних стратегій, з яких ви можете вибрати, великий і представляє багато різноманітності.

У нашому випадку ми використовуєм JWT стратегії для аутентифікації адміна і окремо користувача, оскільки вони мають різні доступи до системи.

Оскільки необхідно зберігати паролі користувачів безпечно, потрібно використати шифрування, щоб зберігати лише хеш паролю у базі даних. Для цього

було використано бібліотеку bcrypt, що надає функції по шифруванню. Момент шифрування паролів відбувається при реєстрації нових користувачів (рисунок 4.6).

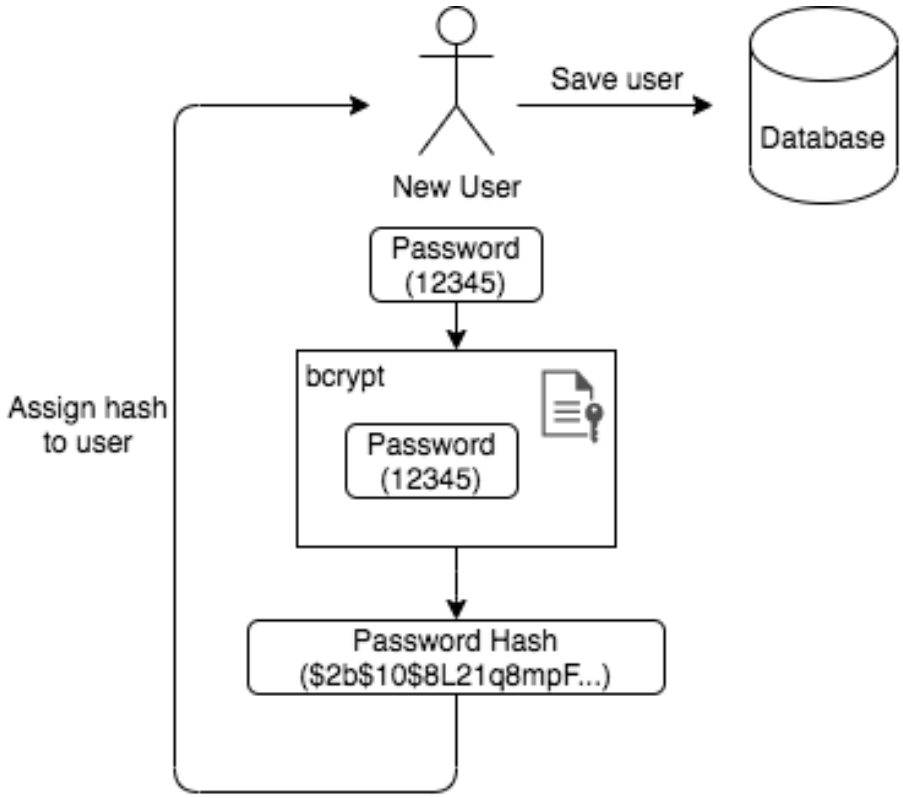


Рисунок 2.7 Процес реєстрації користувача

Тому, коли при вході в систему ми знаходимо в базі даних користувача по його електроній пошті, отримуємо об’єкт користувача, в якому міститься хеш паролю. Для того, щоб зрозуміти чи користувач ввів коректний пароль необхідно скористатись функцією compare з бібліотеки bcrypt. Таким чином, відбуваєтсья порівняння паролю в строковому вигляді та хешу. Після того як функція повернула успішний результат, відбувається генерація токену JWT і його повернення клієнту. В такому випадку аутентифікація відбулась.

2.2.3 REST API доступу до даних дипломних проектів

Оскільки мобільний застосунок для доступу до даних взаємодіятиме безпосередньо з REST API за допомогою ajax-запитів, тому розглянемо детальніше структуру прикладного програмного інтерфейсу.

До засобів підтримки процесу керування дипломним проектуванням можна віднести серверний API, за допомогою якого працює система керування дипломним

проектуванням. За допомогою API відбуваються усі комунікації системних модулей [7].

Щоб отримати список персоналу системи використовується API, що наведено у таблиці (Таблиця 2.1):

Таблиця 2.1 API списку персоналу

Запит	Параметри	Структура JSON-відповіді	URL-посилання
GET	No parameters	{ "id": 5, "role": "STUDENT", "firstName": "Віталій", "lastName": "Денисюк", "middleName": "Вікторович", "email": "dendoc97@gmail.com", "tmpPassword": null, "phone": null, "photo": null, "isActive": true, "isHead": false, "lastLogin": null, "departmentId": 1, "groupId": 2, "degreeId": null, "createdAt": "2020-11-11T16:11:28.364Z", "updatedAt": "2021-04-03T06:25:50.107Z", "group": { "id": 2, "name": "ТВ-91мп", "amountStudents": 11, "departmentId": 1, "academicYearId": 1, "academicDegreeId": 3,	https://diplomasystemapi.herokuapp.com/users

		"specialtyId": 1, "createdAt": "2020-11-11T15:17:18.855Z", "updatedAt": "2020-11-11T15:17:18.855Z" }, "degree": null }	

Список персоналу можна редагувати, наприклад видаляти користувачів. Для видалення чи редагування певного члена персоналу використовується наступний API (Таблиця 2.2):

Таблиця 2.2 API видалення та редагування персоналу

Запит	Параметри	Структура JSON-відповіді	URL-посилання
PUT	id	{ "id": 5, "role": "STUDENT", "firstName": "Віталій", "lastName": "Денисюк", "middleName": "Вікторович", "email": "dendoc97@gmail.com", "tmpPassword": null, "phone": null, "photo": null, "isActive": true, "isHead": false, "lastLogin": null, "departmentId": 1, "groupId": 2, "degreeId": null, "createdAt": "2020-11-11T16:11:28.364Z",	https://diplomasystemapi.herokuapp.com/users/personals

		"updatedAt": "2021-04-03T06:25:50.107Z",
		"group": {
		"id": 2,
		"name": "ТВ-91мп",
		"amountStudents": 11,
		"departmentId": 1,
		"academicYearId": 1,
		"academicDegreeId": 3,
		"specialtyId": 1,
		"createdAt": "2020-11-11T15:17:18.855Z",
		"updatedAt": "2020-11-11T15:17:18.855Z"
		},
		"degree": null
		}	

DELETE	id	{ "id": 5, "role": "STUDENT", "firstName": "Віталій", "lastName": "Денисюк", "middleName": "Вікторович", "email": "dendoc97@gmail.com", "tmpPassword": null, "phone": null, "photo": null, "isActive": true, "isHead": false, "lastLogin": null, "departmentId": 1, "groupId": 2, "degreeId": null,	https://diplomasystemapi.herokuapp.com/users/personal/

		"createdAt": "2020-11-11T16:11:28.364Z",
		"updatedAt": "2021-04-03T06:25:50.107Z",
		"group": {
		"id": 2,
		"name": "ТВ-91мп",
		"amountStudents": 11,
		"departmentId": 1,
		"academicYearId": 1,
		"academicDegreeId": 3,
		"specialtyId": 1,
		"createdAt": "2020-11-11T15:17:18.855Z",
		"updatedAt": "2020-11-11T15:17:18.855Z"
		},
		"degree": null
		}	

Викладачі можуть отримувати інформацію про студентів, які їх цікавлять. Для отримання інформації щодо ступеня освіти використовується API (Таблиця 2.3):

Таблиця 2.3 API ступеня освіти

Запит	Параметри	Структура JSON-відповіді	URL-посилання

GET	No parameters	"degree": {
		"id": 7,
		"name": "Професор",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/academic-degree

POST	No parameters	"degree": {
		"id": 7,
		"name": "Професор",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/academic-degree

PUT	id	"degree": {
		"id": 7,
		"name": "Професор",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/academic-degree/

DELETE	id	"degree": {
		"id": 7,
		"name": "Професор",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/academic-degree/

Також кожен користувач може дізнатися про рік надання освіти, для отримання цієї інформації використовується наступний API (Таблиця 2.4):

Таблиця 2.4 API року ступеня освіти

Запит	Параметри	Структура JSON-відповіді	URL-посилання

GET	No parameters	"year": {
		"name": "2020/2021",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/academic-year

POST	No parameters	"year": {
		"name": "2020/2021",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/academic-year

PUT	id	"year": {
		"name": "2020/2021",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/academic-year/

DELETE	id	"year": {
		"name": "2020/2021",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/academic-year/

Користувачі мають змогу отримати інформацію щодо дипломних тем, для цього використовується такий API (Таблиця 2.5):

Таблиця 2.5 API дипломних тем

Запит	Параметри	Структура JSON-відповіді	URL-посилання

GET	No parameters	"theme": {
		"name": "Назва теми",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/theme

POST	No parameters	"theme": {
		"name": "Назва теми",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/theme

PUT	id	"theme": {
		"name": "Назва теми",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/theme//approve

PUT	id	"theme": {
		"name": "Назва теми",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/theme//decline

Для отримання інформації щодо ступіня освіти використовується наступний API (Таблиця 2.6):

Таблиця 2.6 API року ступіня освіти

Запит	Параметри	Структура JSON-відповіді	URL-посилання
GET	No parameters	"degree": { "id": 7, "name": "Професор", "departmentId": 1 }	https://diplomasystemapi.herokuapp.com/theme
POST	No parameters	"degree": { "id": 7, "name": "Професор", "departmentId": 1 }	https://diplomasystemapi.herokuapp.com/theme
PUT	id	"degree": { "id": 7, "name": "Професор", "departmentId": 1 }	https://diplomasystemapi.herokuapp.com/theme//approve
PUT	id	"degree": { "id": 7, "name": "Професор", "departmentId": 1 }	https://diplomasystemapi.herokuapp.com/theme//decline

Користувачі можуть переглядати розклад здачі дипломних робіт. Для отримання інформації щодо розкладу використовується наступний API (Таблиця 2.7):

Таблиця 2.7 API розкладу

Запит	Параметри	Структура JSON-відповіді	URL-посилання

GET	No parameters	"group": {
		"id": 2,
		"name": "ТВ-91мп",
		"amountStudents": 11,
		"departmentId": 1,
		"academicYearId": 1,
		"academicDegreeId": 3,
		"specialtyId": 1,
		"createdAt": "2020-11-11T15:17:18.855Z",
		"updatedAt": "2020-11-11T15:17:18.855Z"
		}
			https://diplomasystemapi.herokuapp.com/shedule

POST	No parameters	"group": {
		"id": 2,
		"name": "ТВ-91мп",
		"amountStudents": 11,
		"departmentId": 1,
		"academicYearId": 1,
		"academicDegreeId": 3,
		"specialtyId": 1,
		"createdAt": "2020-11-11T15:17:18.855Z",
		"updatedAt": "2020-11-11T15:17:18.855Z"
		}
			https://diplomasystemapi.herokuapp.com/shedule

PUT	id	"group": { "id": 2, "name": "ТВ-91мп", "amountStudents": 11, "departmentId": 1, "academicYearId": 1, "academicDegreeId": 3,	https://diplomasystemapi.herokuapp.com/theme//approve

		"specialtyId": 1, "createdAt": "2020-11-11T15:17:18.855Z", "updatedAt": "2020-11-11T15:17:18.855Z" }	

PUT	id	"group": {
		"id": 2,
		"name": "ТВ-91мп",
		"amountStudents": 11,
		"departmentId": 1,
		"academicYearId": 1,
		"academicDegreeId": 3,
		"specialtyId": 1,
		"createdAt": "2020-11-11T15:17:18.855Z",
		"updatedAt": "2020-11-11T15:17:18.855Z"
		}
			https://diplomasystemapi.herokuapp.com/shedule/

Для отримання інформації щодо спеціальності використовується наступний API (Таблиця 2.8):

Таблиця 2.8 API спеціальності

Запит	Параметри	Структура JSON-відповіді	URL-посилання

GET	No parameters	"specialty": {
		"id": 122,
		"name": "Комп Науки",
		"departmentId": 1
		}
			https://diplomasystemapi.herokuapp.com/specialty

POST	No parameters	"specialty": { "id": 122, "name": "Комп Науки", "departmentId": 1 }	https://diplomasystemapi.herokuapp.com/specialty
PUT	id	"specialty": { "id": 122, "name": "Комп Науки", "departmentId": 1 }	https://diplomasystemapi.herokuapp.com/theme//specialty
PUT	id	"specialty": { "id": 122, "name": "Комп Науки", "departmentId": 1 }	https://diplomasystemapi.herokuapp.com/specialty/

2.3 Висновки

Дипломне проектування – це комплексна самостійна творча робота по вирішенню певної задачі, в процесі якої студент здобуває та поліпшує свої навички та вміння. Дипломне проектування має важливу роль у завершальному етапі навчання. Контролювання дипломним проектуванням здійснюють дипломні керівники. Дипломна робота обов'язково повинна мати певний рівень новизни, оригінальний характер і вміщувати результати наукових досліджень.

Система містить маршрутів по яким відбуваються запити. Але не всі користувачі мають доступ виконувати їх, необхідно бути авторизованим під певною роллю. Адміністратор в даній системі є окремим користувачем.

Ролей у системі кілька:

- Навчальний персонал;

- Викладач, Завідувач кафедри;
- Студент.

Для коректної роботи системи окремі модулі повинні взаємодіяти як одне ціле. Щоб реалізувати таку роботу буде використаний серверний API. За його допомоги усі системні модулі будуть розумітись між собою, кожен буде знати, що від нього потрібно.

3. МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ IOS-ЗАСТОСУНКУ СИСТЕМИ КЕРУВАННЯМ ДИПЛОМНИМ ПРОЕКТУВАННЯМ

3.1 Сучасні тенденції розробки інтерфейсів користувача

Однією з основних рекомендацій було створити зручний та приємний інтерфейс для користувача. Інтерфейс - спільний кордон між двома функціональними об'єктами, вимоги до якої визначаються стандартом; сукупність засобів, методів і правил взаємодії (управління, контролю і т. д.) між елементами системи. А ось іншими словами, але цікавіше: призначений для користувача інтерфейс (UI) - це «спосіб, яким ви виконуєте будь-яку задачу за допомогою будь-якого продукту, а саме здійснюються вами дії і те, що ви отримуєте у відповідь» (джерело: Джеф Раскін, «Інтерфейс. Нові напрямки в проектуванні комп'ютерних систем»). У повсякденному житті ми постійно стикаємося з інтерфейсами. Це і сайти соцмереж, і елементи управління в салоні автомобіля, і пульт дистанційного керування для телевізора, і голосове управління розумним будинком, і панель кнопок у ліфті. Виходить, ми використовуємо один продукт для управління іншим продуктом. Але давайте не будемо перераховувати всі явища в нашому житті, а поговоримо безпосередньо про веб-сервісах і додатках і про те, як зробити їх використання зручним. Інтерфейс - це комплекс засобів, який призначений для взаємодії двох систем між собою. Як цих систем може бути все що завгодно, включаючи штучний інтелект і людей. Слово «інтерфейс» взято з англ. мови: interface - «місце зіткнення». Для реалізації інтерфейсу перш за все потрібно створити макет застосунку, на якому продумати карти екранів застосунку. Макет застосунку зображено на малюнку.

Приклад інтерфейсу користувача застосунку зображено на малюнку (рисунок 3.1). На рисунку зображено головне меню користувача застосунку. З головного меню можна потрапити на інші екрани застосунку. На верхньому меню присутні

дані користувача, такі як: ПІБ, електронна адреса, посада. Також на верхньому меню розташована кнопка «Налаштування». На боковому меню знизу знаходяться кнопки, за допомогою яких можна потрапити на інші екрани застосунку.

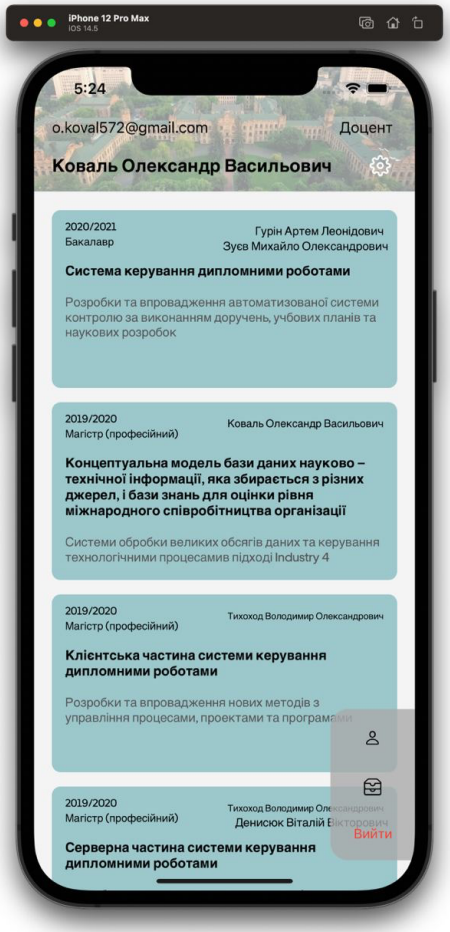


Рисунок 3.1 Приклад інтерфейсу користувача застосунку

У обчислювальної і комп'ютерної техніки під інтерфейсом найчастіше розуміються елементи, які забезпечують взаємодію програмних та апаратних засобів з людиною і між собою. В електронній комерції під ним мають на увазі методи взаємодії ПО і користувача. Такий вид інтерфейсу називають людино-машинним. Інтерфейс являє собою не більш ніж просто іменований набір абстрактних членів. Абстрактні методи є чистим протоколом, оскільки не мають ніякої стандартної реалізації. Конкретні члени, які визначаються інтерфейсом, залежать від того, яка поведінка моделюється за його допомогою. Це дійсно так. Інтерфейс висловлює поведінку, яке даний клас або структура може обрати для підтримки. Більш того, кожен клас (або структура) може підтримувати стільки інтерфейсів, скільки

необхідно, і, отже, тим самим підтримувати безліч поводжень. Правило «Зустрічають по одягу» активно використовується при взаємодії користувача з застосунком.

3.2 Особливості архітектурного стилю REST

Якщо програму розглядати як чорний ящик, то API — це багато важелів, які доступні користувачеві даного ящика, якими він може керувати.

Програмні компоненти взаємодіють один з одним за допомогою API. При цьому зазвичай компоненти утворюють ієрархію — високорівневі компоненти використовують API низькорівневих компонентів, а ті, в свою чергу, використовують API ще більш низькорівневих компонентів.

API бувають кількох видів:

- Web service APIs (XML-RPC and JSON-RPC, SOAP, REST)
- WebSockets APIs
- Library-based APIs
- Class-based APIs

Абревіатура REST розшифровується як representational state transfer - «передача стану представлення», тобто представлення даних в зручному для клієнта форматі. Автором ідеї та терміну "REST" є Рой Філдінг, винайшов у 2000 році. Основна ідея REST в тому, що кожне звернення до сервісу переводить клієнтську програму в новий стан. По суті, REST — не протокол і не стандарт, а підхід, архітектурний стиль проектування API.

Основні принцпип REST є такими:

- масштабованість взаємодії компонентів системи;
- спільність інтерфейсів;
- незалежне впровадження компонентів;
- проміжні компоненти, що знижують затримку, які посилюють безпеку;

Серед переваг даного архітектурного підходу можна перерахувати такі:

- Відсутність додаткових внутрішніх прошарків, що означає передачу даних в тому ж вигляді, якими є самі дані.
- Кожна одиниця інформації (ресурс) однозначно визначається URL — це означає, що URL по суті є первинним ключем для одиниці даних. Причому абсолютно не має значення, в якому форматі знаходяться дані за адресою - це може бути і HTML, і png, і документ Microsoft Word.
- управління інформацією ресурсу ґрунтується на протоколі передачі даних. Найбільш поширений протокол звичайно ж HTTP (Hyper Text Transfer Protocol) . Для HTTP дію над даними задається за допомогою методів: GET (отримати), PUT (додати, замінити), POST (додати, змінити, видалити), DELETE (видалити). Таким чином, дії CRUD (Create-Read-Update-Delete) можуть виконуватися як з усіма 4-ма методами, так і тільки за допомогою GET і POST.

Діаграма (рисунок 3.2) показує загальну модель REST API:

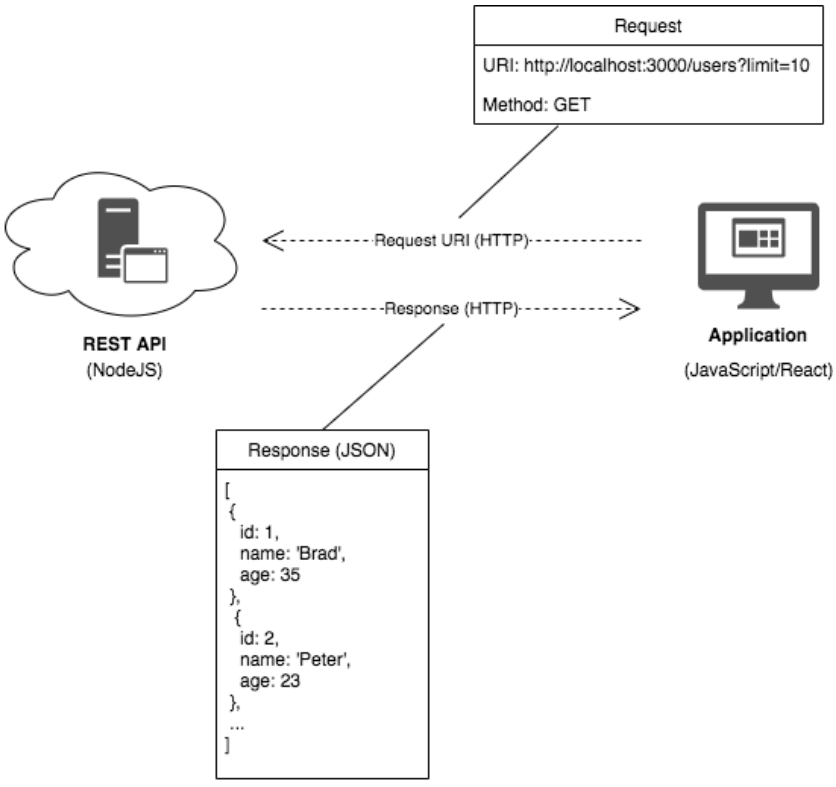


Рисунок 3.2 Модель REST API

Як видно, між клієнтом і сервером API існує запит і відповідь. Клієнт і сервер можуть бути засновані на будь-якій мові, але HTTP — це протокол, який

використовується для передачі повідомлення. Цей шаблон запиту і відповіді, по суті, є принципом роботи REST API.

Будь-яка мова програмування, що створює запит, буде по-своєму відправляти веб-запит і аналізувати відповідь на своїй мові. Ці специфічні для мови функції відправки запитів і аналізу відповідей не є частиною REST API. REST API не залежить від мови та обробляє вхідну і вихідну інформацію по HTTP.

REST API можуть використовувати будь-який формат повідомлень, який хочуть використовувати розробники API, включаючи XML, JSON, Atom, RSS, CSV, HTML і інші.

Незважаючи на різноманітність параметрів формату повідомлень, більшість API REST використовують JSON (нотацію об'єктів JavaScript) в якості формату повідомлень за замовчуванням. Вони використовують JSON, тому що він забезпечує легкий, простий і більш гнучкий формат повідомлень, який збільшує швидкість зв'язку. Полегшена природа JSON також дозволяє виконувати сценарії мобільного розробки і легкий для парсингу в Інтернеті за допомогою JavaScript.

REST API не зберігає стану і може кешуватися. Відсутність стану означає, що кожен раз, коли ви звертаєтеся до ресурсу через кінцеву точку, API надає одну і ту ж відповідь. Він не запам'ятовує ваш останній запит і не враховує його при наданні нового відповіді.

Відповіді можуть кешуватися для підвищення продуктивності. Якщо кеш браузера вже містить інформацію, запитувану в запиті, браузер може просто повернути інформацію з кеша замість того, щоб знову звертатися до серверу.

Кешування REST API аналогічно кешуванню веб-сторінок. Браузер використовує значення часу останньої зміни в заголовках HTTP, щоб визначити, чи потрібно йому знову отримувати ресурс. Якщо вміст не було змінено з моменту останнього вилучення, замість нього можна використовувати кешовану копію. Кешування збільшує швидкість відповіді.

З точки зору RESTful-сервісу, операція (або виклик сервісу) ідемпотентна тоді, коли клієнти можуть робити один і той же виклик неодноразово при одному і тому ж результаті на сервері. Іншими словами, створення великої кількості

ідентичних запитів має такий же ефект, як і один запит. В той час, як ідемпотентні операції виробляють один і той же результат на сервері, відповідь сама по собі може не бути тим же самим.

Методи PUT і DELETE за визначенням ідемпотентні. Проте, є один нюанс з методом DELETE. Проблема в тому, що успішний DELETE-запит повертає статус 200 (OK) або 204 (No Content), але для подальших запитів буде весь час повертати 404 (Not Found), стан на сервері після кожного виклику DELETE однаковий, але відповіді різні.

Методи GET, HEAD, OPTIONS і TRACE визначені як безпечні. Це означає, що вони призначені тільки для отримання інформації та не повинні змінювати стан сервера. Вони не повинні мати побічних ефектів, за винятком нешкідливих ефектів, таких як: логування, кешування.

HTTP метод GET використовується для отримання (або читання) ресурсу. У разі "вдалого" запиту, GET повертає ресурс у форматі XML або JSON в поєднанні з кодом стану HTTP 200 (OK). У разі наявності помилок зазвичай повертається код 404 (NOT FOUND) або 400 (BAD REQUEST).

HTTP метод PUT зазвичай використовується для надання можливості редагування ресурсу.

HTTP метод POST найчастіше використовується для створення нових ресурсів.

HTTP метод DELETE використовується для видалення ресурсу, ідентифікованого конкретним URI (ID).

3.3 Середовище розробки XCode

У кожного програміста є улюблене IDE (інтегроване середовище розробки), яке вони використовують для розробки своїх програм. Для розробників iOS – Xcode (рисунок 3.3) є найліпшим рішенням. Спочатку Xcode може здатися просто черговим текстовим редактором. Але це незамінний інструмент для компіляції та

налагодження коду, побудови користувальницьких інтерфейсів, читання документації, надсилання програм в App Store тощо [1].

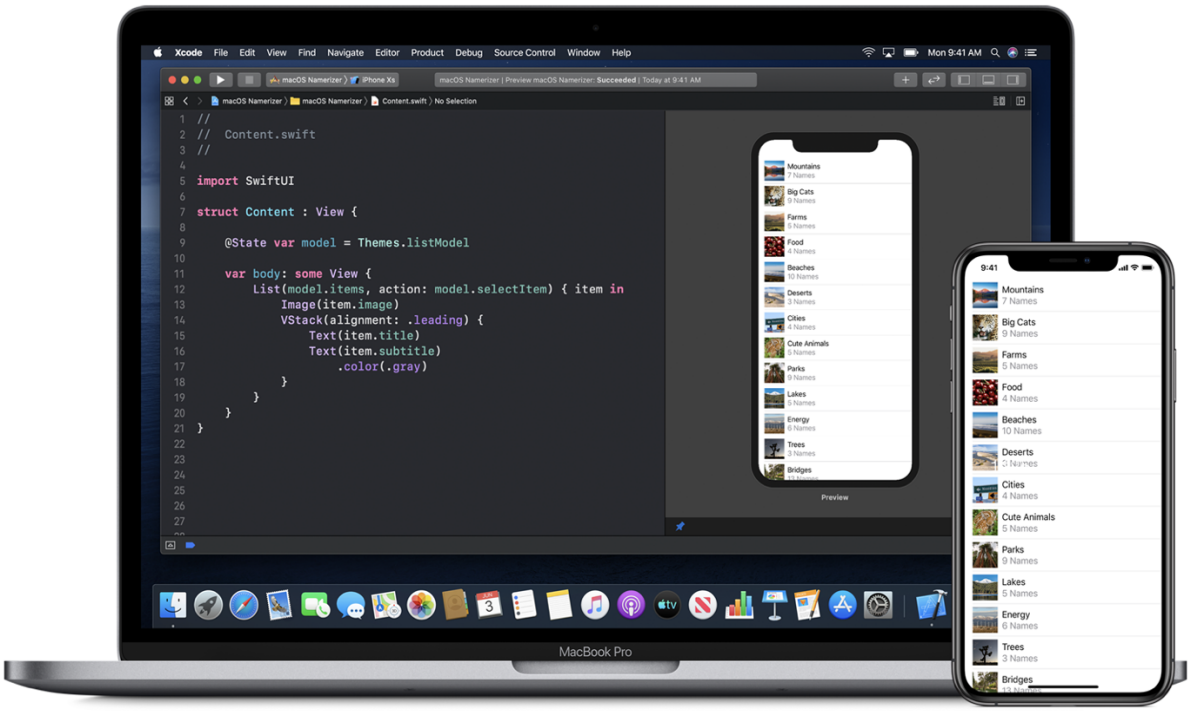


Рисунок 3.3 IDE XCode

Xcode знає, як працювати з різними файлами, які охоплюють не одну мову програмування. Він створений як універсальний застосунок, який на 100% працює на процесорах на базі Intel та Apple Silicon для чудової продуктивності та швидкого інтерфейсу. Він також включає SDK macOS з усіма фреймворками, компіляторами, налагоджувачами та іншими інструментами, які потрібні для створення програми для процесорів Apple Silicon та Intel x86_64.

Xcode складається з набору інструментів, які розробники використовують для створення додатків для платформ Apple. Він використовується для керування всіма процесами розробки - від створення програми, тестування та оптимізації і до надсилання її в App Store [15].

Для швидкого створення прототипів і тестування програми в імітованому середовищі є можливість використовувати симулятори пристроїв, коли реальний пристрій недоступний. Simulator надає середовища для пристроїв iPhone, iPad, Apple Watch та Apple TV з різними налаштуваннями, файлами та версіями операційної системи [3].

Для профілювання та аналізу вашого додатка, підвищення продуктивності та пошуку проблем із пам’яттю можна користуватись різними встроєними інструментами. Інструменти збирають дані та представляють результати за допомогою різних додатків.

Xcode надає шаблони для вихідних файлів Swift, Objective-C, C, C ++ та Metal, а також вихідні точки для загальних підкласів, модульних тестів та тестів користувальницького інтерфейсу. Можна обрати будь-який шаблон відповідно до операційної системи для вашого проекту та типу файлу, який ви хочете [16].

3.3.1 Мова програмування Swift

Мова Swift (рисунок 3.4) - це потужна та інтуїтивно зрозуміла мова програмування для macOS, iOS, watchOS, tvOS та інших операційних систем. Написання коду Swift є інтерактивним та цікавим, синтаксис стислий, але виразний, а Swift включає сучасні функції, за допомогою яких можна реалізувати свої задачі.

Рисунок 3.4 Приклад програмного коду мови Swift

Мова Swift досить молода, вона росте, розвивається і змінюється, хоча основні підходи до програмування і розробки вже сформувалися.

Swift - результат останніх досліджень мов програмування у поєднанні з багаторічним досвідом створення платформ Apple. Іменовані параметри виражаються в чистому синтаксисі, що робить API у Swift ще простішим для читання та обслуговування. Ще краще, вам навіть не потрібно вводити крапки з комою. Визначені типи роблять код чистішим і менш схильним до помилок, тоді як

модулі усувають заголовки та надають простори імен. Для найкращої підтримки міжнародних мов та смайлів, рядки мають правильність Unicode і використовують кодування на основі UTF-8 для оптимізації продуктивності для різноманітних випадків використання. Керування пам'яттю здійснюється автоматично за допомогою жорсткого, детермінованого підрахунку посилань, зводить використання пам’яті до мінімуму без накладних витрат на збір сміття.

Ряд вже згаданих функцій роблять Swift безпечною мовою, допомагаючи писати код, який з меншою ймовірністю призведе до збою програми. Комп’ютерні програми дуже буквальні, тому код, написаний для обробки однієї речі, може бути не здатний обробляти іншу. Безпека типу змушує вас бути чітко визначеною щодо «типу» кожного об’єкта, який ви створюєте, маніпулюєте та призначаєте, і дозволяє вам писати код, який може обробляти лише даний об’єкт. Це запобігає написанню коду, який може “впасти”, якщо він не призначений для роботи з “типами” об’єктів, на які ви посилаєтесь. Компілятор робить висновок про тип об'єкта, тим самим заощаджує. Це допомагає вам гарантувати, що ваш код може обробляти як сценарії, де існують значення, так і сценарії, коли вони відсутні. Swift також передбачає складну обробку помилок, яка дозволяє писати код, який може обробляти помилки швидко і просто [14].

З самого раннього задуму Swift був побудований швидко. Використовуючи неймовірно високопродуктивну технологію компілятора LLVM, код Swift перетворюється на оптимізований власний код, який отримує максимум від сучасного обладнання. Синтаксис та стандартна бібліотека також були налаштовані, щоб зробити найбільш очевидний спосіб написання коду найкращим, незалежно від того, працює він у годиннику на вашому зап'ясті або на сервері.

Свіфт є наступником мов C та Objective-C. Він включає примітиви низького рівня, такі як типи, управління потоком та оператори. Він також надає об'єктно-орієнтовані функції, такі як класи, протоколи та загальні засоби, надаючи розробникам Cocoa та Cocoa Touch необхідну продуктивність та потужність.

3.3.2 Бібліотека URLSession

Клас URLSession та пов’язані з ним класи надають API для завантаження даних до кінцевих точок, зазначених URL-адресами. Застосунок також може використовувати цей API для виконання фонових завантажень, коли програма не запущена або призупинена. Ви можете використовувати пов’язані URLSessionDelegate та URLSessionTaskDelegate для підтримки аутентифікації та отримання таких подій, як перенаправлення та завершення завдання.[13]

Фонові сеанси дозволяють виконувати завантаження та завантаження вмісту у фоновому режимі, поки програма не запущена.

3.3.3 Бібліотека UIKit

Фреймворк UIKit забезпечує необхідну інфраструктуру для програм iOS або tvOS. Він забезпечує архітектуру вікна та перегляду для реалізації інтерфейсу, інфраструктуру обробки подій для доставки Multi-Touch та інших типів вхідних даних у застосунок, а також основний цикл запуску, необхідний для управління взаємодіями між користувачем, системою та додатком. Інші функції, пропоновані фреймворком, включають підтримку анімації, підтримку документів, підтримку малювання та друку, інформацію про поточний пристрій, управління текстом та відображення, підтримку пошуку, підтримку доступності, підтримку розширень програми та управління ресурсами. UIKit керує взаємодією додатка з системою та забезпечує класи для керування даними та ресурсами додатка [10].

3.3.4 Формат JSON

При обміні даними між браузером та сервером дані можуть бути лише текстовими. JSON - це текст, і ми можемо перетворити будь-який об'єкт Swift у JSON і відправити JSON на сервер.

Ми також можемо перетворити будь-який JSON, отриманий від сервера, в об'єкти Swift. Таким чином, ми можемо працювати з даними як об’єктами Swift, без складного розбору та перекладу.

Оскільки формат JSON є лише текстовим, його можна легко надіслати на сервер і з нього, а також використовувати як формат даних будь-якою мовою програмування.

Swift має вбудовану функцію для перетворення рядка, записаного у форматі JSON, у власні об'єкти Swift [17].

Отже, якщо ви отримуєте дані із сервера у форматі JSON, ви можете використовувати їх як будь-який інший об’єкт Swift. Сам по собі JSON використовує розширення .json. Коли ж він визначається в інших файлових форматах, як .html, він з'являється в лапках як JSON рядок або може бути об'єктом, призначеним на змінну. Такий формат легко передавати між сервером і клієнтською частиною, ну або браузером [11].

Легкочитаємий і компактний, JSON являє собою гарну альтернативу XML і вимагає куди менше форматування контенту. Це інформативний посібник допоможе вам швидше розібратися з даними, які ви можете використовувати з JSON і основною структурою з синтаксисом цього ж формату.

Об'єкт JSON це формат даних - ключ-значення, який зазвичай рендерится в фігурних дужках. Коли ви працюєте з JSON, то ви швидше за все бачите JSON об'єкти в .json файлі, але вони також можуть бути і як JSON об'єкт або рядок вже в контексті самої програми [12].

3.3.5 Протоколи HTTP

HTTP розшифровується як HyperText Transfer Protocol, «протокол передачі гіпертексту». Спочатку цей протокол використовувався для передачі гіпертекстових документів в форматі HTML. Сьогодні він використовується для передачі довільних даних. В основі HTTP - клієнт-серверна структура передачі даних. Клієнт формує запит (request) і відправляє на сервер, на сервері запит обробляється, формується відповідь (response) і передається клієнтові. HTTP не шифрує передану інформацію. Для захисту переданих даних використовується розширення HTTPS (Hyper Text Transfer Protocol Secure), яке "упаковує" передані дані в криптографічний протокол SSL або TLS.

У випадку відправлення запитів GET відбувається динамічне визначення типів. Коли на клієнтську частину приходить відповідь, то також відбувається динамічна типізація завдяки стандартним методам мови Swift.

Протокол передачі гіпертексту (HTTP) призначений для забезпечення зв'язку між клієнтами та серверами [5].

HTTP працює як протокол відповіді на запит між клієнтом та сервером.

Наприклад: Клієнт (браузер) надсилає на сервер запит HTTP, тоді сервер повертає клієнту відповідь. Відповідь містить інформацію про статус запиту, а також може містити запитуваний вміст. Одні з найчастіше застосовуємих HTTP методів є:

Метод GET означає отримання будь-якої інформації (у формі сутності), визначеної Request-URI. Якщо Request-URI посилається на процес створення даних, саме отримані дані повинні бути повернуті як сутність у відповіді, а не вихідний текст процесу, якщо тільки цей текст не є результатом процесу.

Семантика методу GET змінюється на "умовний GET", якщо повідомлення запиту включає поле заголовка If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match або If-Range. Умовний метод GET вимагає перенесення сутності лише за обставин, описаних умовними полями заголовка. Умовний метод GET призначений для зменшення непотрібного використання мережі, дозволяючи оновити кешовані сутності, не вимагаючи численних запитів або передачі даних, які вже зберігаються у клієнта [9].

Семантика методу GET змінюється на "частковий GET", якщо повідомлення запиту містить поле заголовка Range. Частковий GET просить передати лише частину сутності, як описано в розділі 14.35. Метод часткового GET призначений для зменшення непотрібного використання мережі, дозволяючи частково отриманим об'єктам завершувати роботу без передачі даних, які вже зберігаються у клієнта.

Відповідь на запит GET можна кешувати тоді і лише тоді, коли він відповідає вимогам щодо кешування HTTP [6].

Метод POST використовується для запиту, щоб початковий сервер прийняв об'єкт, укладений у запит, як новий підлеглий ресурсу, ідентифікований Request-URI

у рядку запиту. POST призначений для того, щоб єдиний метод охоплював наступні функції:

- Анотація існуючих ресурсів;

- Розміщення повідомлення на дошці оголошень, групі новин, списку розсилки, або аналогічна група статей;

- Надання блоку даних, наприклад результату подання а форму, до процесу обробки даних;

- Розширення бази даних за допомогою операції додавання.

Фактична функція, що виконується методом POST, визначається сервером і зазвичай залежить від URI запиту. Опублікована сутність підпорядковується цьому URI так само, як файл підпорядковується каталогу, що містить його, стаття новин підпорядковується групі новин, до якої він розміщений, або запис підпорядковується базі даних.

Дія, виконана методом POST, може не призвести до ресурсу, який можна ідентифікувати за допомогою URI. У цьому випадку відповідним статусом відповіді є або 200 (OK), або 204 (Без вмісту), залежно від того, чи включає відповідь сутність, яка описує результат.

Якщо ресурс був створений на вихідному сервері, відповідь повинна бути 201 (Створений) і містити сутність, яка описує стан запиту та посилається на новий ресурс, та заголовок Location.

Відповіді на цей метод неможливо кешувати, якщо відповідь не містить відповідних полів заголовка Cache-Control або Expires.

3.4 Висновки

Для реалізації iOS-застосунку системи керування дипломним проектуванням будуть використані такі засоби розробки:

- IDE Xcode;

- Мова програмування Swift;

- Бібліотеки URLSession та UIKit;

- HTTP протоколи;

- Формат JSON;

Однією з основних рекомендацій було створити зручний та приємний інтерфейс для користувача. Інтерфейс - спільний кордон між двома функціональними об'єктами, вимоги до якої визначаються стандартом; сукупність засобів, методів і правил взаємодії (управління, контролю і т. д.) між елементами системи.

Програмні компоненти взаємодіють один з одним за допомогою API. При цьому зазвичай компоненти утворюють ієрархію — високорівневі компоненти використовують API низькорівневих компонентів, а ті, в свою чергу, використовують API ще більш низькорівневих компонентів.

API бувають кількох видів:

- Web service APIs (XML-RPC and JSON-RPC, SOAP, REST)

- WebSockets APIs

- Library-based APIs

- Class-based APIs

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ IOS-ЗАСТОСУНКУ СИСТЕМИ КЕРУВАННЯ ДИПЛОМНИМ ПРОЕКТУВАННЯМ

4.1 Функціональна модель користувачів системи

Система містить маршрутів по яким відбуваються запити. Але не всі користувачі мають доступ виконувати їх, необхідно бути авторизованим під певною роллю. Адміністратор в даній системі є окремим користувачем. Ролей у системі кілька:

- Навчальний персонал;
- Викладач, Завідувач кафедри;
- Студент.

Ієрархія ролей показана на рисунку 4.1.

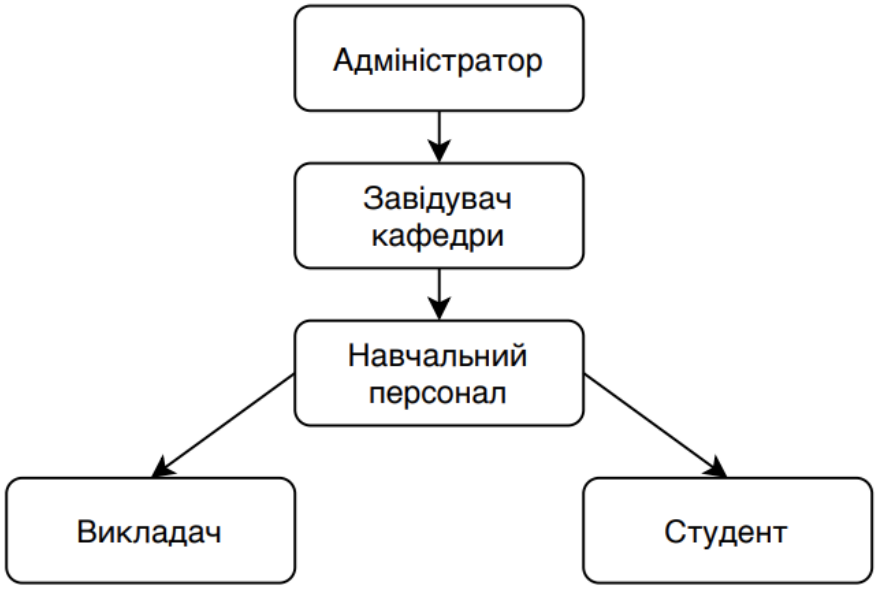


Рисунок 4.1. Ієрархія ролей системи

Перераховані ролі є користувачами системи, тому мають спільний функціонал, такий як вхід та вихід з системи, зміна або відновлення паролю тощо (рисунок 4.2). Кожна система не обходиться без сервісу авторизації та аутентифікації користувачів. Зміна та відновдення паролю грає не менш важливу

роль в системах керування. Додатковим функціоналом є активація або реєстрація користувачів. Про дані можливості користувачів описано у наступних розділах.

На рисунку зображено діаграму прецедентів системи для ролі студент (рисунок 4.2).

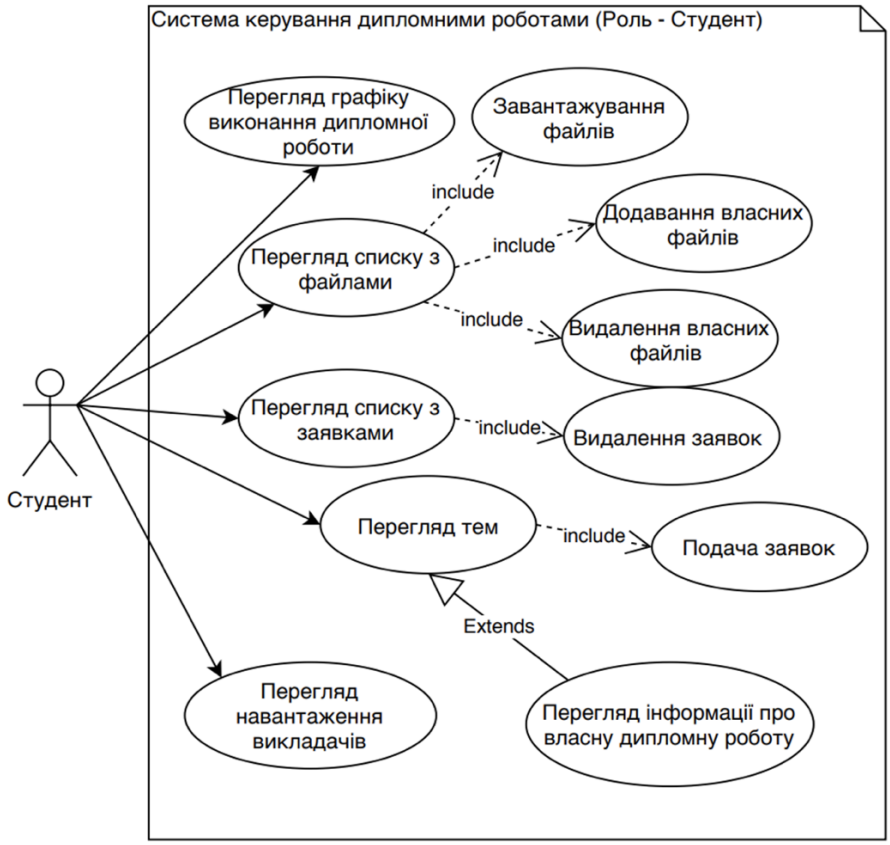


Рисунок 4.2 Діаграма прецедентів системи для ролі студент

Насамперед користувач дивиться, чи комфортно і приємно йому перебувати в додатку. Тільки потім він починає знайомитися з ним докладніше. UI-дизайн допомагає зробити продукт візуально зрозумілим і приємним. Це важлива складова, яка безпосередньо впливає на враження при користуванні продуктом.

UI-дизайн - User Interface, дослівно перекладається, як «Інтерфейс». Це певний процес візуалізації, який дозволяє реалізувати прототип сайту, програм й інших веб-ресурсу. В першу чергу, UI включає активну роботу над графічної складової інтерфейсу. Створюються анімації, ілюстрації, кнопки та інші елементи сайту, включаючи шрифти, кольори, форми.

Спочатку визначається призначений для користувача досвід і досліджується цільова аудиторія. Залежно від потреб відвідувачів визначається колірна палітра, форми, а також структура розміщення об'єктів. Інтерфейс користувача повинен бути зрозумілим, і одна з головних задач UI-дизайнера - визначити, чи зручний продукт візуально, чи легко потрапити по кнопках, чи добре читається текст.

Рисунок діаграми прецедентів для ролі викладач зображена на рисунку 4.3.

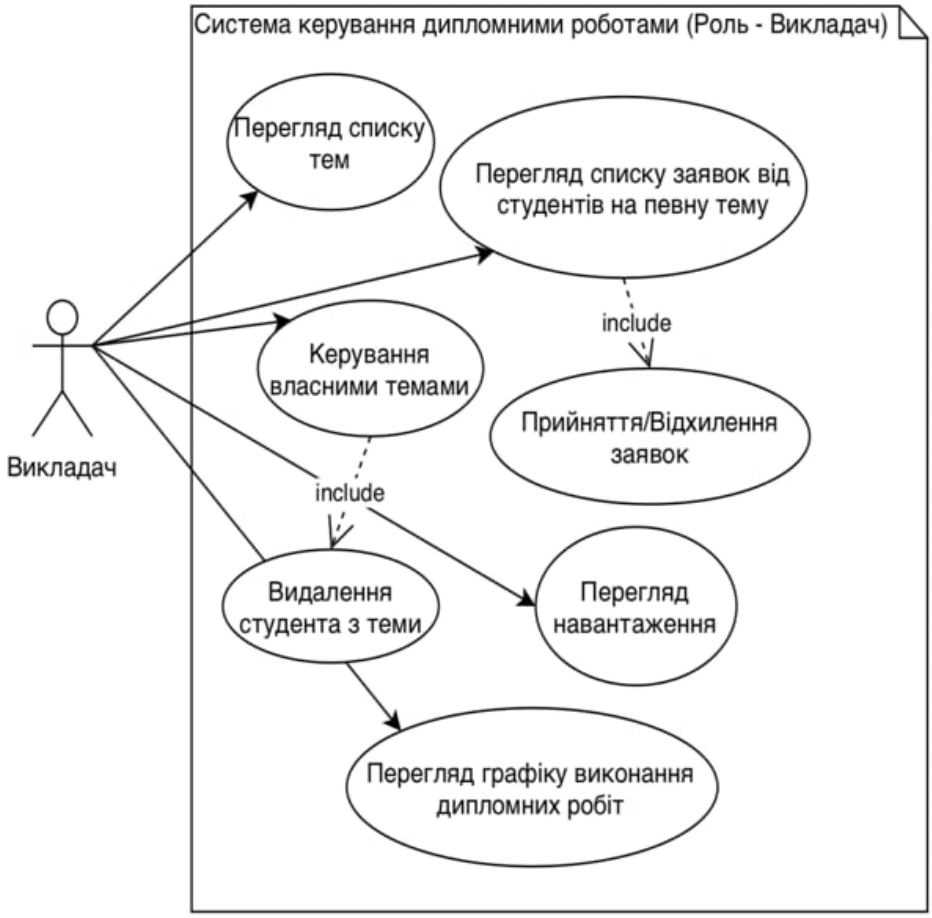


Рисунок 4.3 Діаграма прецедентів системи для ролі викладач

Також представлена діаграма прецедентів системи для ролі адміністратор (рисунок 4.4).

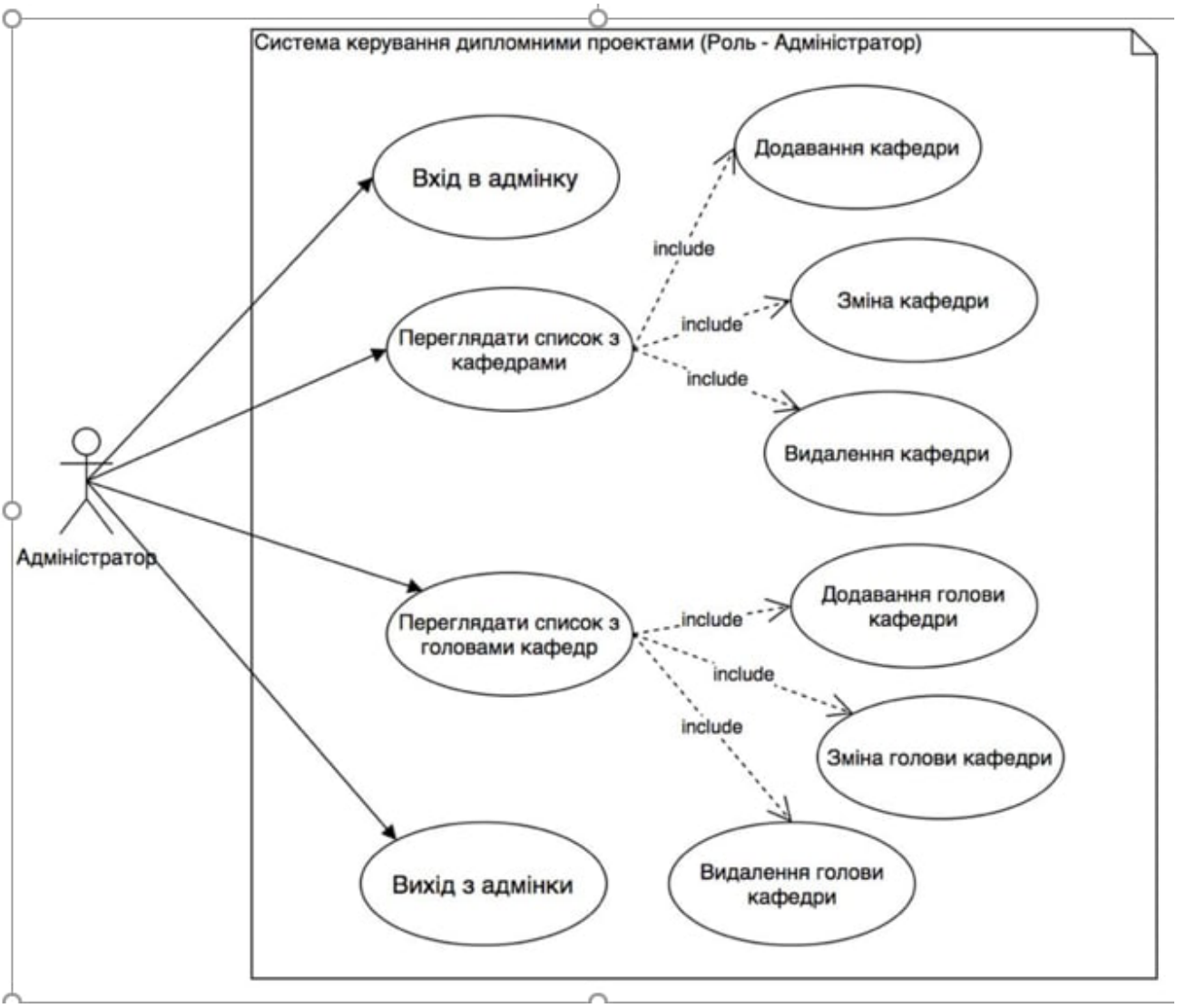


Рисунок 4.4 Діаграма прецедентів системи для ролі адміністратор

4.2 Карта екранів мобільного застосунку

Майже для кожного функціонального елементу додатку впроваджено певний екран чи елемент інтерфейсу. Кожен екран відповідає певному функціоналу системи. На рисунку можна побачити екран авторизації, головний екран застосунку, на який користувач потрапля після авторизації, також екрани перегляду навантаження викладачів та графіку виконання.

Кожен екран застосунку тісно пов’язаний з іншими, тому що переходи між екранами здійснюються з одного на другий безпосередньо. Структура нагадує деревовідну форму. Тому неможливо просто так видалити один з екрнів, якщо він не є крайнім.

Також на рисунку можна побачити детальну структуру кожного екрану застосунку, котрий містить в собі окремі елементи та реалізації окремих модулів. Присутня структура екрану налаштування. На екрані налаштувань можна побачити детальну інформацію про застосунок, також потрапити на екран магазину додатків “AppStore”. На малюнку зображено схему екранів застосунку (рисунок 4.5). Рисунок

4.5 Схема екранів застосунку

Головний екран поділяється на: основний список тем, бокове та верхнє меню. Основний список тем розташований посеред головного екрану, майже на всю його площу. В основному списку тем відображаються дипломні теми користувача. В списоку тем є можливість перейти на екран детального перегляду кожної теми та редагувати її.

Бокове меню розташоване внизу з правого боку екрану, воно займає місце поверх головного списку тем. На боковому меню розташовані такі елементи, як: кнопка переходу на екран графіку виконання, кнопка переходу на екран навантаження викладачів та кнопка виходу для повторної авторизації. На екрані графіку виконання можна переглянути календар та дипломні теми за певний період часу. На екрані навантаження викладачів є список з викладачами та кількість студентів, за яких вони відповідальні на даний момент.

Верхнє меню розташоване над основним списком дипломних тем, воно містить в собі таку інформацію: посада користувача, його електронна адреса та ПІБ. Також на верхньому меню розташована кнопка переходу на екран налаштувань. На екрані налаштувань можна переглянути правову інформацію про застосунок та перейти до магазину застосунків AppStore.

4.3 Логічна модель

Були реалізовані такі кабінети користувача як: кабінет студента, викладача, завідувача кафедрою та адміністратора. В кожного кабінету є свій функціонал в системі.

На діаграмі послідовності (рисунок 4.6) програмного продукту для ролей студента та викладача продемонстровано їх взаємодію з фрагментами програмного продукту. Після успішної авторизації вони мають можливість переглянути особисті дані, список дипломних тем, графік виконання, навантаження, файлів та заявок.

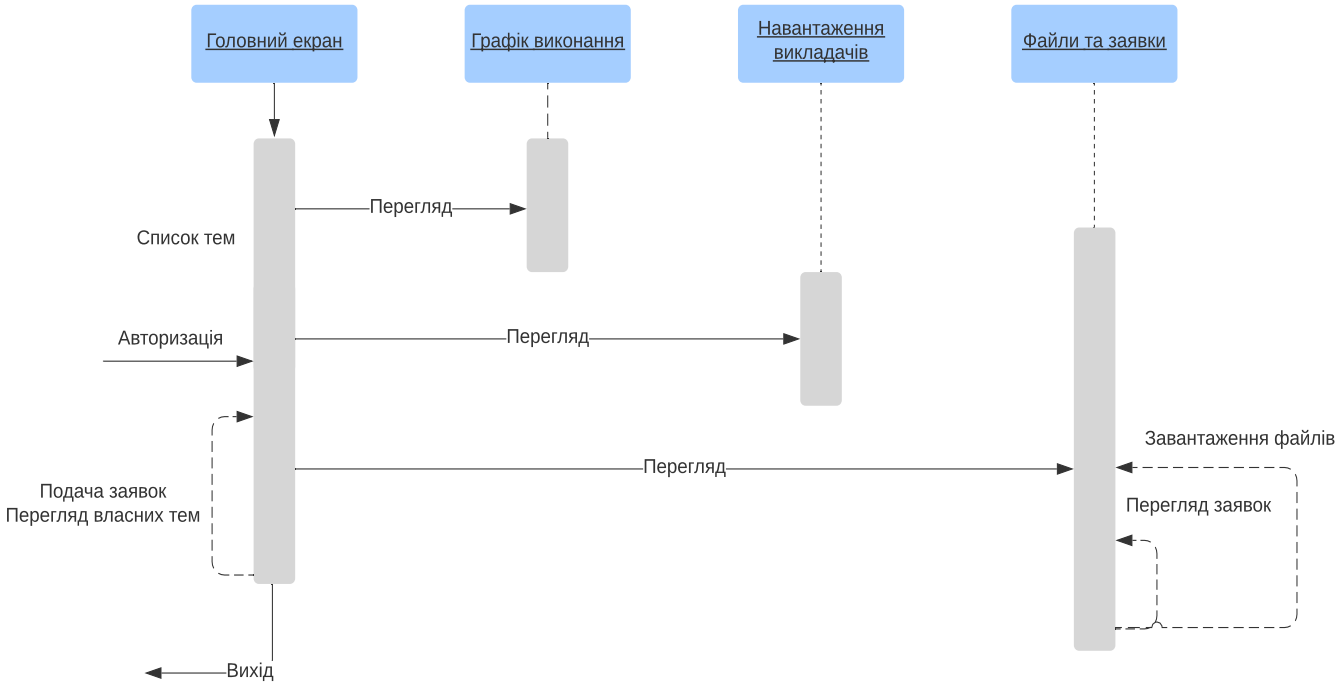


Рисунок 4.6 Діаграма послідовності дій студента та викладача

Застосунок розрахований на 3 категорії користувачів: студенти, викладачі та адміністратори.

Відповідно до категорій розподіляється відповідний функціонал. В кабінеті студента користувач може переглянути свою дипломну тему, графік виконання дипломної роботи, поточне навантаження викладачів, переглядати та редагувати список з файлами. Завантаження та видалення файлів можливе лише у повній Веб-версії системи. Роль студента є заключною у системі. Студент має можливість переглядати список тем, подавати заявки на теми, які сподобались та слідкувати за статусами заявок. Коли студент отримав згоду від викладача щодо обраної теми, йому буде надана інформація про його тему. Викладач також має можливість переглядати список дипломних тем та навантаження. Також він може керувати власними темами, додавати чи видаляти студентів з певних тем. Адміністратор може переглядати список тем, як і попередні ролі. Окрім того він має можливість переглядати список кафедр та їх завідувачів. За необхідністю редагувати ці дані. Діаграма послідовності дій адміністратора зображена на рисунку (рисунок 4.7).

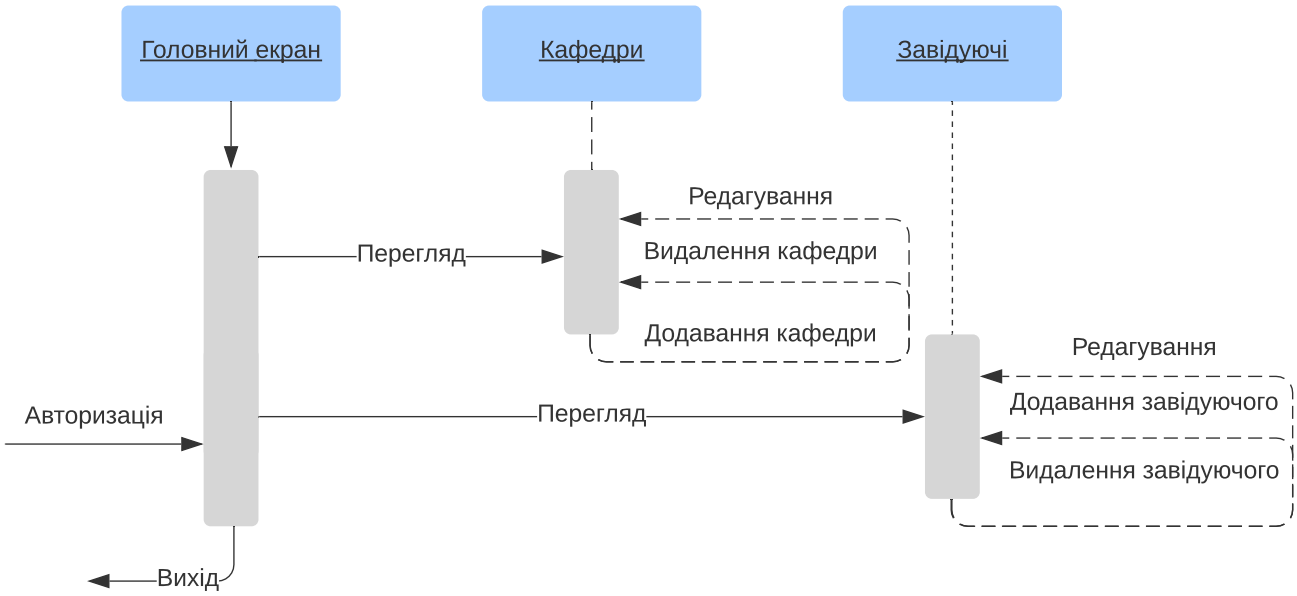


Рисунок 4.7 Діаграма послідовності дій адміністратора

4.4 Аутентифікація

Авторизація - це процедура перевірки прав користувача на виконання певних дій на веб-ресурсі або в комп'ютерній системі, результатом якої стає дозвіл або відмову в здійсненні потрібних операцій, а також надання відповідно до прав користувачеві можливостей, які гарантовані йому веб-ресурсом або комп'ютерною системою. Щоб отримати можливість авторизуватися на сайті, необхідно пройти процедуру реєстрації, вказавши e-mail, логін і пароль (останній може генеруватися автоматично і відправлятися на пошту).

Користувач має змогу авторизуватись та вийти зі свого акаунту. Реєстрація - це розширення авторизації, а не включення, тому що користувач, який має вже обліковий запис, не повинен мати змогу знову зареєструватись.

Для всіх користувачів процедура авторизації однакова. Після авторизації застосунок визначає тип користувача, та в залежності від цього користувач потряпляє на певний екран застосунку.

Процедура авторизації для адміністратора така сама як і для інших користувачів. Після авторизації він може переглянути список з кафедрами та їх завідуючими. Діаграма послідовності авторизації зображена на малюнку (рисунок

4.8). Для авторизації користувачу потрібно ввести логін та пароль, натиснути на кнопку «Увійти». Після чого він потрапить на головний екран застосунку.

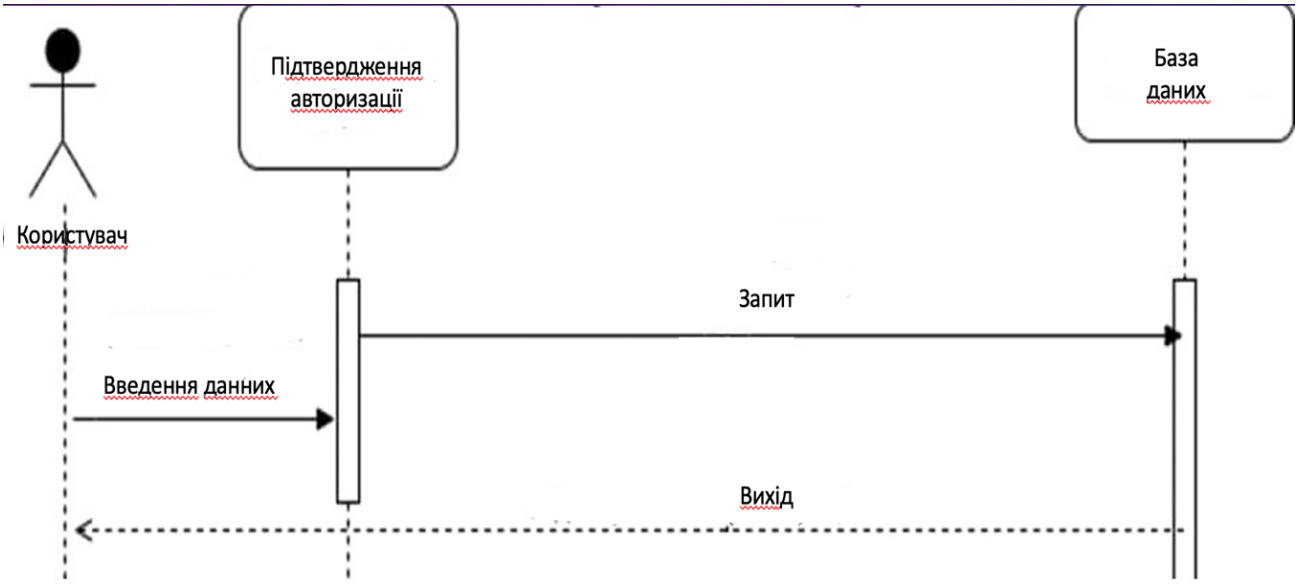


Рисунок 4.8 Діаграма послідовності авторизації

4.5 Архітектура

У застосунку реалізований архітектурний шаблон під назвою “Model-View-Controller”. Його головними перевагами є простота та швидкість реалізації. На малюнку зображено загальну архітектуру системи (рисунок 4.9). Детальну архітектуру iOS-застосунку можна побачити на малюнку (рисунок 4.10).

Шаблон MVC є звичним явищем у розробці iOS. Хоча це не складна модель для розуміння та ознайомлення, є речі, які слід врахувати, щоб уникнути загальних підводних каменів. Модельні об'єкти інкапсулюють дані, специфічні для програми, і визначають логіку та обчислення, які маніпулюють та обробляють ці дані.

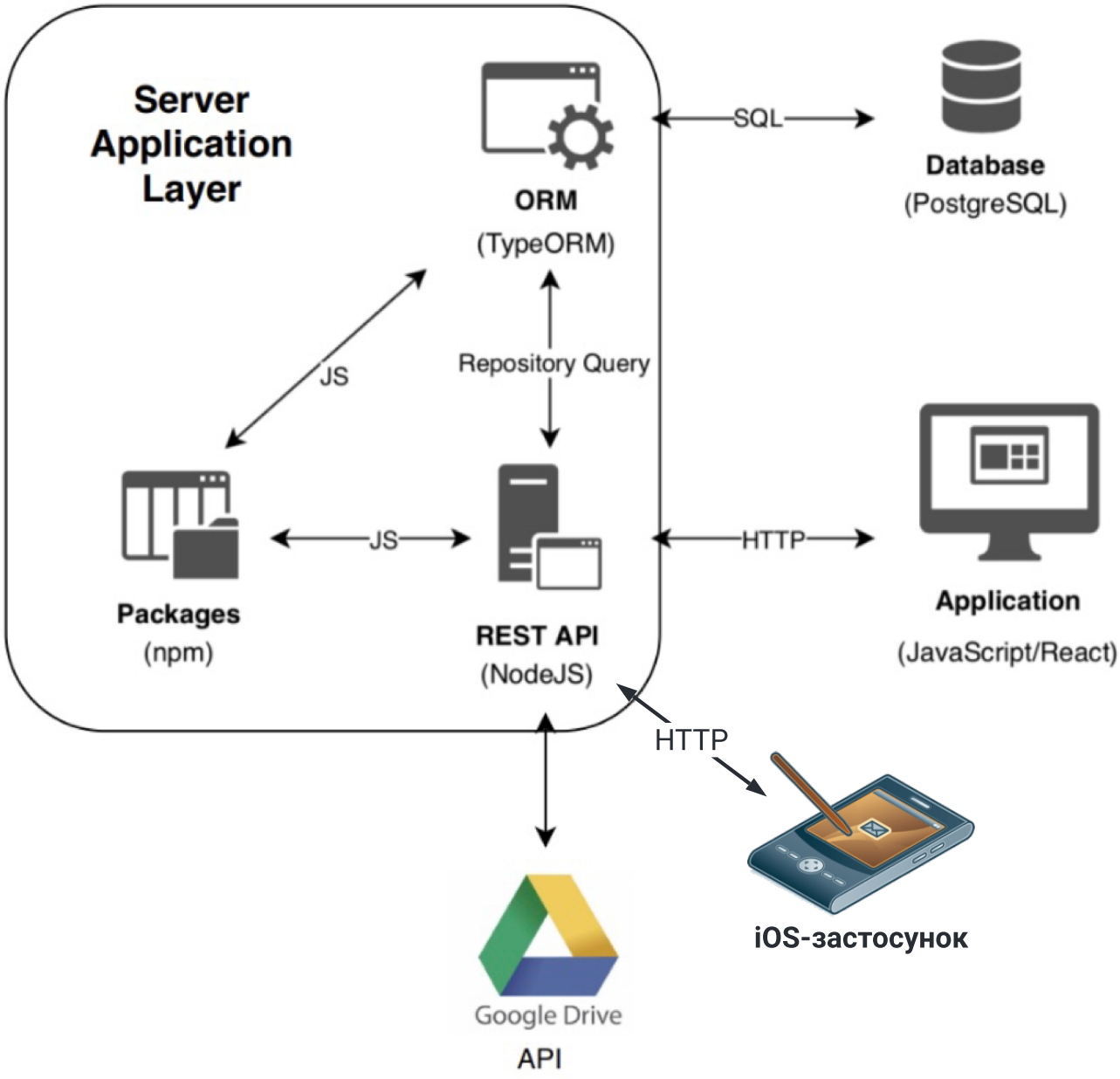


Рисунок 4.9 Схема архітектури системи

Наприклад, об'єкт моделі може представляти персонажа в грі або контакт в адресній книзі. Модельний об'єкт може мати взаємозв'язок "один-багато-багато" з іншими об'єктами моделі, тому інколи модельний рівень програми ефективно представляє один або кілька графіків об'єктів. Більша частина даних, що є частиною постійного стану програми (незалежно від того, зберігається цей постійний стан у файлах або базах даних), повинна міститися в об’єктах моделі після завантаження даних у програму. Оскільки об’єкти моделі представляють знання та знання, пов’язані з конкретним проблемним доменом, їх можна використовувати повторно в подібних

проблемних доменах. В ідеалі, об’єкт моделі не повинен мати явного зв’язку з об’єктами подання, які представляють його дані, і дозволяти користувачам редагувати ці дані - він не повинен стосуватися проблем інтерфейсу користувача та презентації.

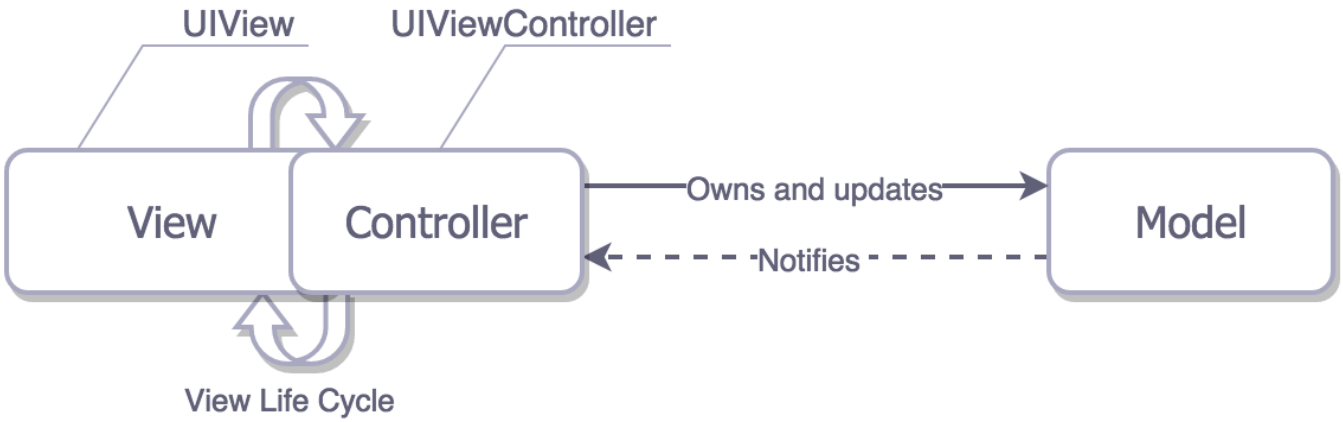


Рисунок 4.10 Схема архітектури iOS-застосунку

Об'єкт подання - це об'єкт у програмі, який користувачі можуть бачити. Об'єкт перегляду знає, як намалювати себе, і може реагувати на дії користувача. Основною метою об’єктів перегляду є відображення даних з об’єктів моделі програми та можливість редагування цих даних. Незважаючи на це, об'єкти подання зазвичай відокремлюються від модельних об'єктів у програмі MVC.

Оскільки ми, як правило, повторно використовуємо та переналаштовуємо їх, об’єкти подання забезпечують узгодженість між програмами. І фреймворки UIKit, і AppKit надають колекції класів перегляду, а Interface Builder пропонує десятки об'єктів перегляду у своїй бібліотеці.[2]

4.6 Модульна структура

Модульна структура застосунку зображена на малюнку (рисунок 4.11). Вона містить в собі представлену застосунком модель даних, основні контролери та функції На рисунку можна побачити такі функції, як: перегляд та редагування. Це основні функції роботи застосунку. Також зображено основні елементи застомунку, такі як: модель, контроллери та серверний API.

За структуру даних у застосунку відповідає модель даних, що зображена на малюнку (рисунок 4.12). Елемент Person відповідає структурі даних користувача. Department – структура даних департаменту освіти. Theme Element – структура даних елементу дипломної теми. Degree – структура даних ступеня освіти. LaboratoryDiraction - структура даних напрямку лабораторії. Profile - структура даних профіля користувача (особисті дані користувача). Teacher – структура даних користувача-вчителя (наприклад). До структури даних вчителя входять такі структури, як: Group – структура даних групи, AcademicDegree - структура даних академічного ступеня, AcademicYear - структура даних академічного року.

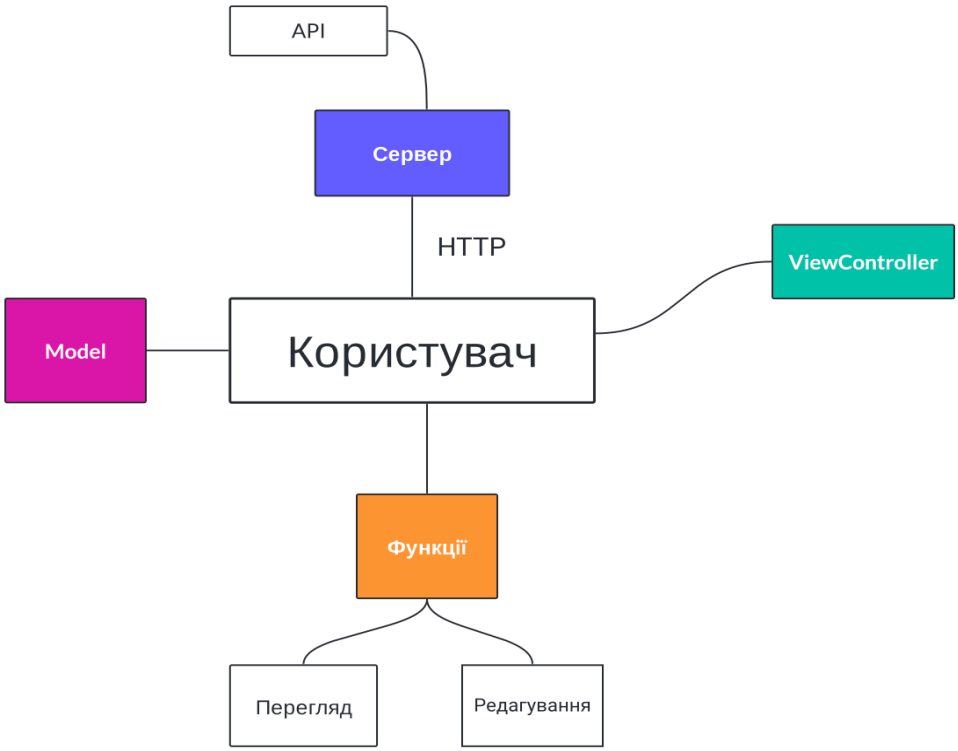


Рисунок 4.11 Модульна структура

На малюнку (рисунок 4.13) зображено карту екранів застосунку, кожен екран відповідає певному функціоналу системи. Кожний екран застосунку містить в собі окремі елементи реалізації різних модулів застосунку. Можна побачити такі контроллери, як: ViewController – основний контроллер зостосунку, на нього користувач потрапляє відразу після авторизації.

Контроллер авторизації – стартовий екран застосунку, на ньому відбувається визначення користувача та перевірка авторізації у системі. Якщо користувач вже авторизований – він портавляє на головний екран застосунку. Якщо ні – він обов’язково повинен пройти авторизацію - LogicViewController.

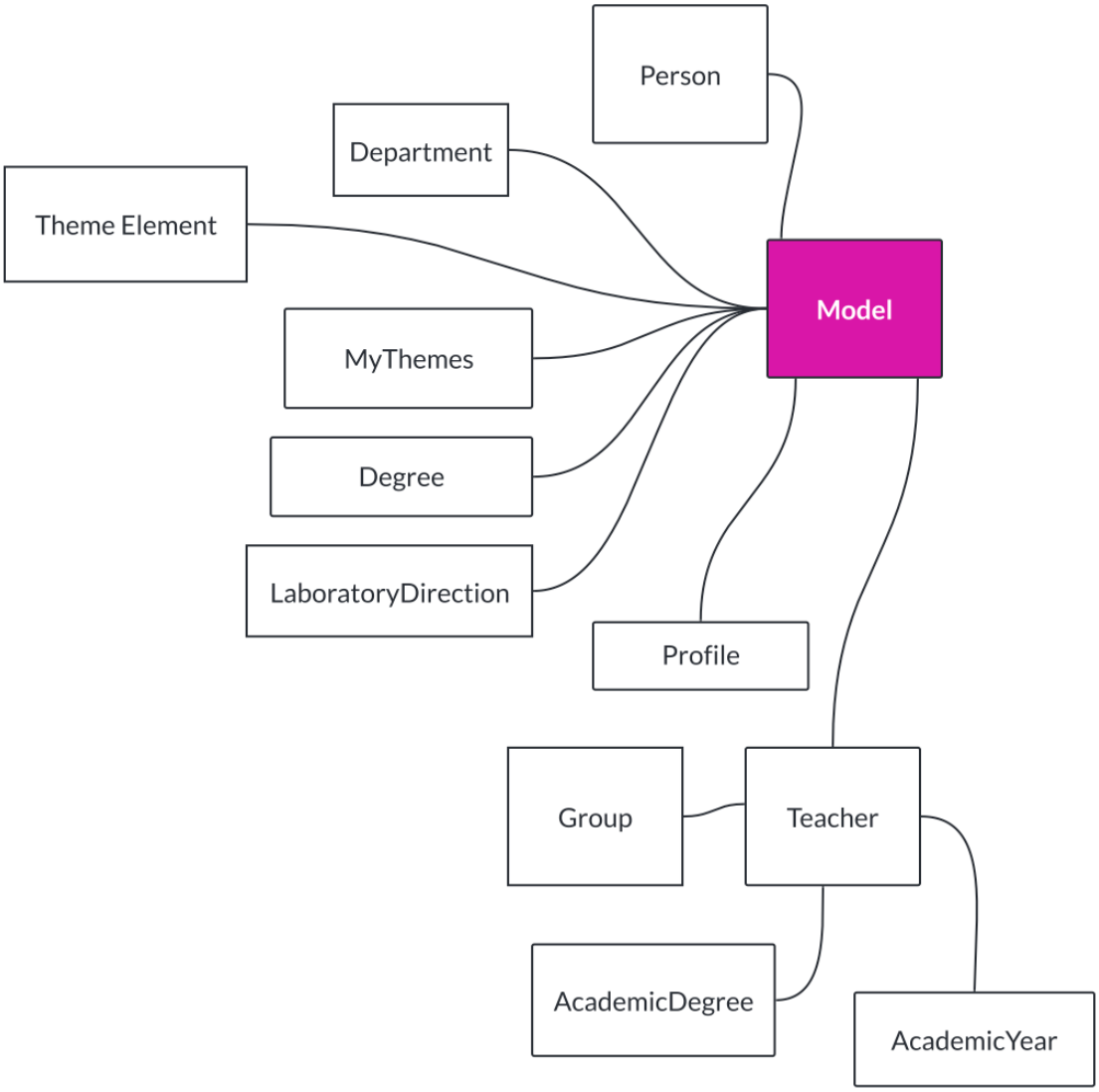
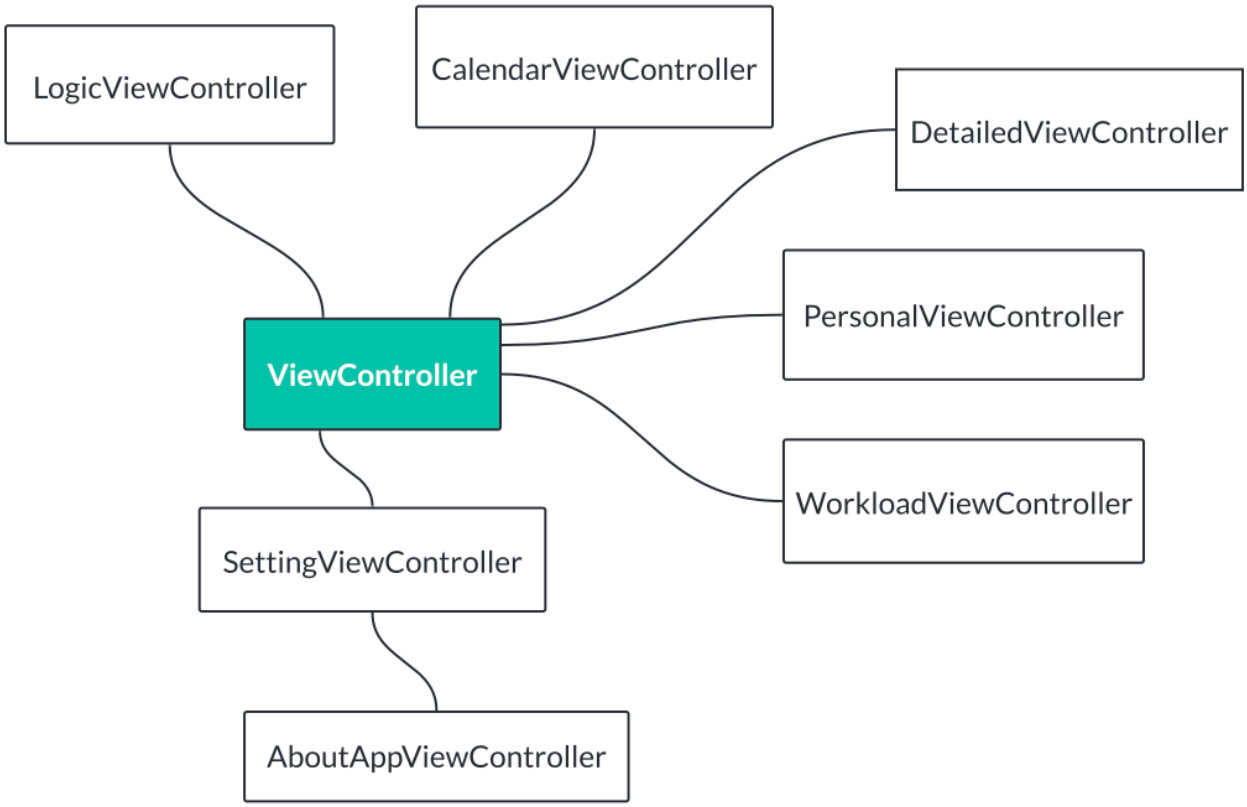


Рисунок 4.12 Структура моделі даних

З основного контроллеру користувач може перейти на інші. CalendarViewController – контроллер, що відповідає за екран графіку виконання діпломної роботи. DetailedViewController - контроллер, що відповідає за екран перегляду додаткової інформації щодо дипломної теми. PersonalViewController та WorkloadViewController - контроллери, що відповідають за екрани перегляду списку персоналу та його навантаження відповідно. За екран налаштувань відповідає SettingViewController.



AboutAppViewController – екран з інформацією про застосунок, правилами користування та правами.

Рисунок 4.13 Структура екранів застосунку

4.7 Розгортання та конфігурування системи

Оскільки система розроблена для мобільних пристроїв, що працюють під керуванням операційної системи iOS, для використання мобільного застосунку потрібен пристрій з даною операційною сиистемою. Встановлення застосунків на пристрої iOS здійснюється з магазину застосунків під назвою “AppStore”. Але можливий інших шлях встановлення – IDE XCode. За його допомогою можна протестувати застосунок на своєму пристрої чи на симуляторі пристрою, який надається стандартними засобами інтегрованої середи розробки.

4.8 Висновки

Відповідно до категорій розподіляється відповідний функціонал. В кабінеті студента користувач може переглянути свою дипломну тему, графік виконання дипломної роботи, поточне навантаження викладачів, переглядати та редагувати список з файлами. Роль студента є заключною у системі.

Студент має можливість переглядати список тем, подавати заявки на теми, які сподобались та слідкувати за статусами заявок. Коли студент отримав згоду від викладача щодо обраної теми, йому буде надана інформація про його тему. Викладач також має можливість переглядати список дипломних тем та навантаження. Також він може керувати власними темами, додавати чи видаляти студентів з певних тем. Адміністратор може переглядати список тем, як і попередні ролі. Окрім того він має можливість переглядати список кафедр та їх завідувачів. За необхідністю редагувати ці дані.

Для реалізації iOS-застосунку системи керування дипломним проектуванням було створено функціональну модель, розроблено архітектуру та модульну структуру. Для роботи окремих модулей системи використано серверні API.

Були реалізовані такі кабінети користувача як: кабінет студента, викладача, завідувача кафедрою та адміністратора. В кожного кабінету є свій функціонал в системі.

Застосунок реалізований за допомогою архітектурного шаблону під назвою “Model-View-Controller”. Його головними перевагами є простота та швидкість реалізації.

Кожний екран застосунку містить в собі окремі елементи реалізації різних модулів застосунку.

Для того, щоб користуватись iOS-застосунком системи керування дипломним проектуванням потрібно мати пристрій iPhone, на який, за допомогою IDE Xcode встановлюються програмні файли застосунку, але можливий інших шлях встановлення – IDE XCode.

5. МЕТОДИКА РОБОТИ З СИСТЕМОЮ

5.1 Автентифікація

Для всіх користувачів процедура авторизації однакова. Після авторизації (рисунок 5.1) застосунок визначає тип користувача, та в залежності від цього користувач потрапляє на певний екран застосунку.

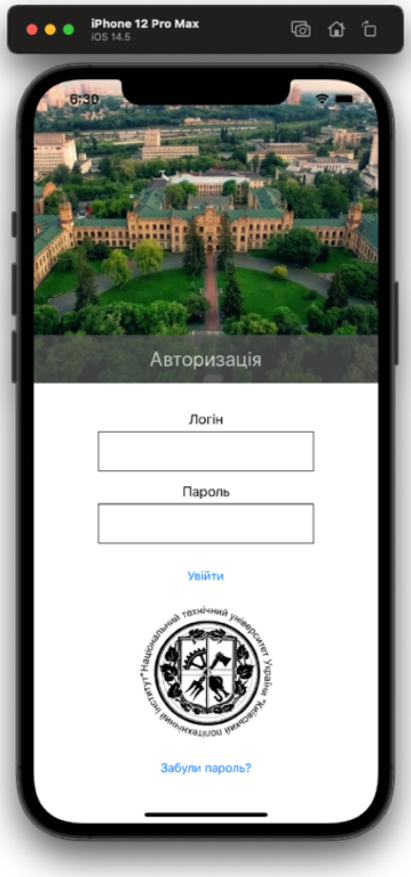


Рисунок 5.1 Приклад авторизації

5.2 Кабінет студента

В кабінеті студента користувач може переглянути свою дипломну тему, графік виконання дипломної роботи, поточне навантаження викладачів, переглядати та редагувати список з файлами. Завантаження та видалення файлів можливе лише у повній Веб-версії системи. Роль студента є заключною у системі. Студент має

можливість переглядати список тем, подавати заявки на теми, які сподобались та слідкувати за статусами заявок. Коли студент отримав згоду від викладача щодо обраної теми, йому буде надана інформація про його тему (Рисунок 5.1).

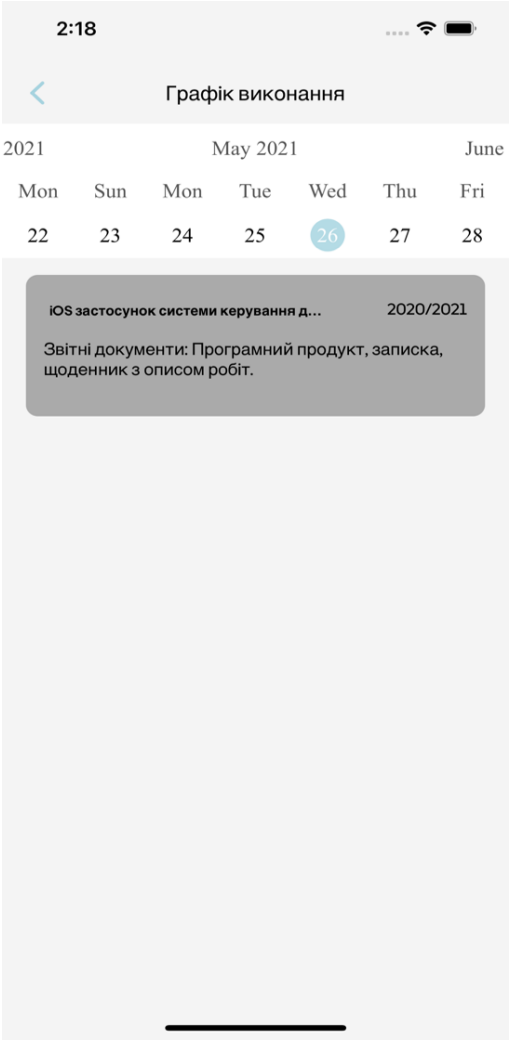


Рисунок 5.2 Сторінка студента

Також він може переглядати графік виконання дипломної роботи, приклад зображено на рисунку (рисунок 5.3):

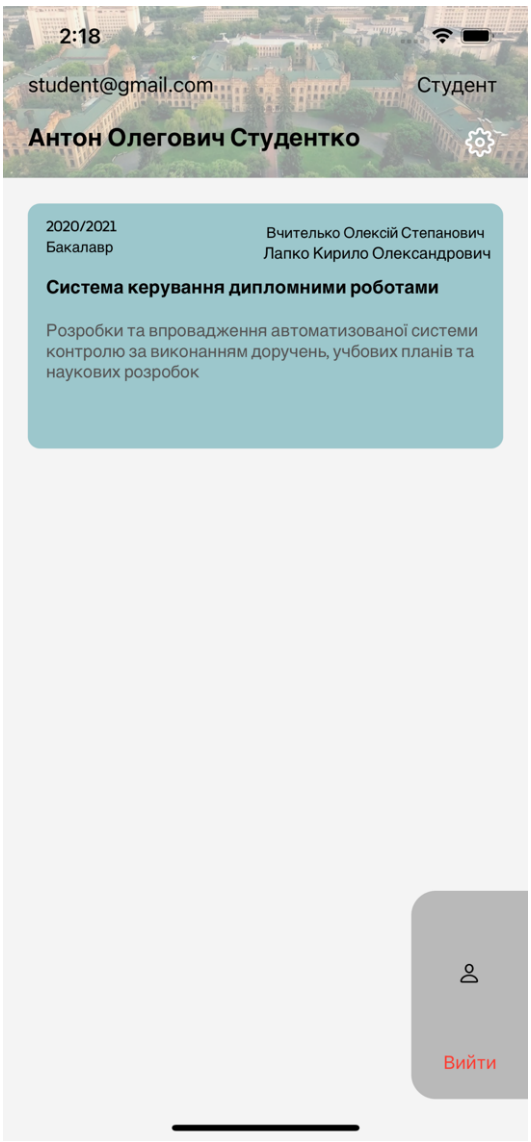


Рисунок 5.3 Графік виконання

5.3 Кабінет викладача

Викладач також має можливість переглядати список дипломних тем (рисунок 5.4) та навантаження інших викладачів (рисунок 5.5). Керувати власними темами, добавляти та видаляти студентів з теми. Також є можливість прийняття та відхилення заявок від студентів. Викладач може повністю керувати процесом дипломної роботи студента. Коли студент отримав згоду від викладача щодо обраної теми, йому буде надана інформація про його тему.

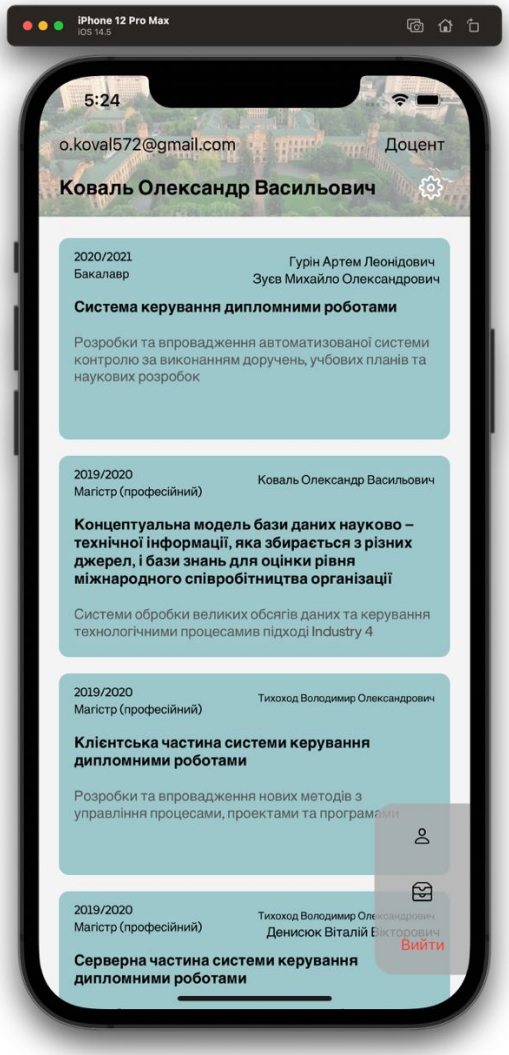


Рисунок 5.4 Сторінка викладача

Також він може керувати власними темами, додавати чи видаляти студентів з певних тем.

Викладач одна з основних ролей системи. Без дій викладача студент не зможе отримати дипломну тему, щоб почати працювати. Тобто без викладача функціонування та взагалі існування системи неможливе. Застосунок розрахований на необмежену кількість викладачів.

У розділі навантаження викладач може побачити інших викладачів та кількість студентів, з якими вони працюють на данний момент, щоб мати уявлення про загальне становище роботи кафедри.

Авторизація для викладача не відрізняється від авторизації студента або адміністратора.

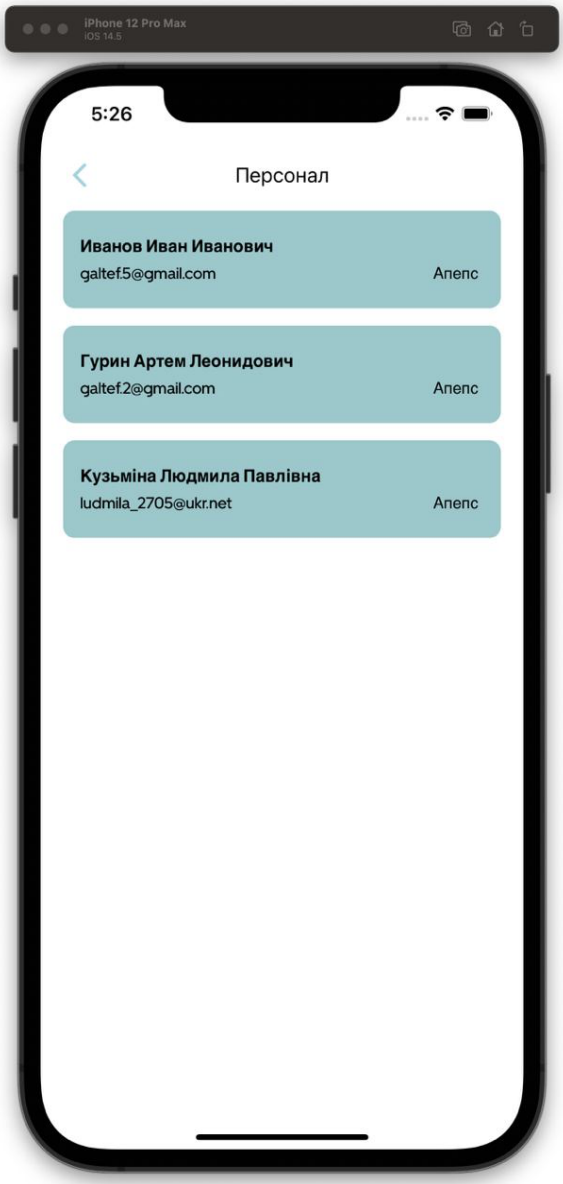


Рисунок 5.5 Навантаження викладачів

5.4 Кабінет адміністратора

Адміністратор може переглядати список тем, як і попередні ролі. Окрім того він має можливість переглядати список кафедр та їх завідувачів. За необхідністю редагувати ці дані (Рисунок 5.6). Адміністратор потрібен для встановлення завідувачів кафедрами.

Адміністратор може з легкістю визначити, що він ввійшов як адміністратор, звернувши увагу на меню, що знаходиться у верхній частині екрану. Якщо авторизація пройшла успішно, адміністратор побачить червоний текст з написом

«Ви увійшли як адміністратор». Також у верхній частині екрану можна побачити власну інформацію: ПІБ, адресу електронної пошти та статус користувача в системі (роль користувача системи).

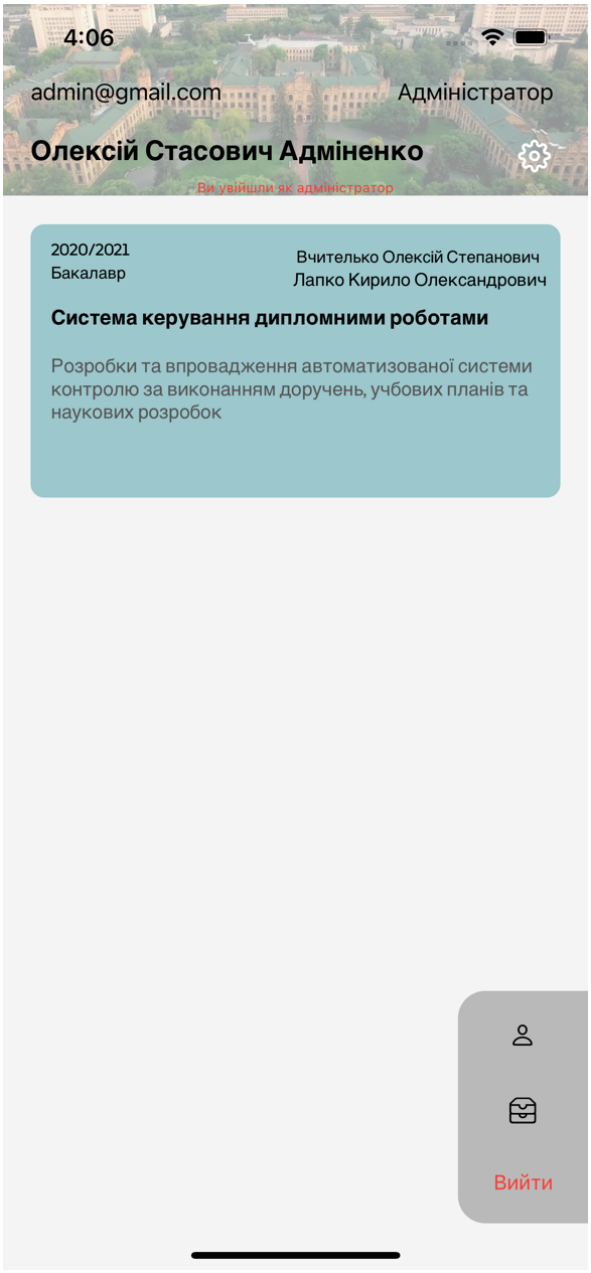


Рисунок 5.6 Сторінка адміністратора

5.5 Висновки

Застосунок розрахований на 3 категорії користувачів: студенти, викладачі та адміністратори.

Відповідно до категорій розподіляється відповідний функціонал. В кабінеті студента користувач може переглянути свою дипломну тему, графік виконання дипломної роботи, поточне навантаження викладачів, переглядати та редагувати список з файлами. Завантаження та видалення файлів буде можливе лише у повній Веб-версії системи. Роль студента є заключною у системі. Студент має можливість переглядати список тем, подавати заявки на теми, які сподобались та слідкувати за статусами заявок. Коли студент отримує згоду від викладача щодо обраної теми, йому буде надана інформація про його тему. Викладач також має можливість переглядати список дипломних тем та навантаження. Також він може керувати власними темами, додавати чи видаляти студентів з певних тем. Адміністратор може переглядати список тем, як і попередні ролі. Окрім того він має можливість переглядати список кафедр та їх завідувачів. За необхідністю редагувати ці дані.

ВИСНОВКИ

Дипломне проектування – це комплексна самостійна творча робота по вирішенню певної задачі, в процесі якої студент здобуває та поліпшує свої навички та вміння. Дипломне проектування має важливу роль у завершальному етапі навчання. Контролювання дипломного проектування – це досить складний процес, тому що він складається з багатьох етапів та підпроцесів. Здебільшого цей процес відбувається у паперовому вигляді, що відображається на прозорості доступу. Працювати таким чином не зовсім зручно та швидко.

Для полегшення процесу контролю було створено Web-систему керування дипломним проектуванням. Але Web-система доступна лише у браузері, що не зовсім зручно, наприклад, якщо користувачу потрібно переглянути інформацію з мобільного пристрою. У зв’язку з цим виникла потреба у аналогу системи для мобільного пристрою.

Було розлоблено iOS-застосунок системи керування дипломним проектуванням. Розробка даного програмного продукту забезпечила користувачам мобільних пристроїв під керуванням операційної системи iOS наступні можливості:

- зручний інтерактивний сервіс вибору студентами тем дипломних проектів;
- інтерфейс перегляду викладачами списку тем дипломів та розподілення студентів за темами.

Застосунок розрахований на 3 категорії користувачів: студенти, викладачі та адміністратори. Відповідно до категорій розподіляється відповідний функціонал. Застосунок став добрим помічником користувачам Web-сервісу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Develop in Swift Fundamentals : Розробка на мові Swift / Apple Inc – Education, 2020. – 603с.

2. Apple Developer Documentation [Електронний ресурс] : Документація розробника програмного продукту від компанії Apple / Apple Inc, 2020. – Режим доступу: https://developer.apple.com/documentation/.

3. Усов В. Swift. Основы разработки приложений под iOS, iPadOS и macOS. 6-е изд. дополненное и переработанное / Василий Усов. – Питер Пресс, 2021. – 544с.

4. Кришнамурти Б. Web-протоколы. Теория и практика. HTTP/1.1, взаимодействие протоколов, кэширование, измерение трафика / Кришнамурти Б. – Бином, 2002. – 592с.

5. Раскін Д. Інтерфейс. Нові напрямки в проектуванні комп'ютерних систем / Раскін Джеф. – Символ-Плюс, 2007. – 272с.

6. Груміт Г. iOS Development with Swift in Motion : Розробка на мові Swift / Груміт Грег. – 2018.

7. Барлетт Д. Swift Programming in Easy Steps : Програмування на мові Swift в легких кроках / Баретт Деріл. - In Easy Steps Limited, 2019. – 192с.

8. Ейдхоф Е. Functional Swift, Updated for Swift 4 : Функціональний Swift, оновлено для Swift 4 / Е. Ейдхоф, Ф. Куглер, В. Свірстра. – Florian Kugler, 2015. – 236с.

9. Нейборг М. iOS 10 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics 1st Edition : Фундаментальне програмування на мові Swift / Метт Нейборг. – O'Reilly Media, Inc, USA, 2016. – 620с.

10. Инт Вейн Ч. Swift детально / Инт Вейн Чейрд. – ДМК Прес, 2019. – 422с.

11. Граді Б. Object Oriented Analysis and Design with Applications : ООА та ООД у мобільних застосунках / Б. Граді, М. Роберт А., Б. Д. Йанг та інші. – Addison Wesley, 1990. – 694с.

12. Бек К. Рефакторинг. Поліпшення існуючого коду / К. Бек, М. Фаулер. – Диалектика, 1999. – 448с.

13. Сінтес Е. Sams Teach Yourself Object Oriented Programming in 21 Days : Самонавчання ООП за 21 день / Ентоні Сінтес. - Sams, Imprint of Simon asn Schuster 201W. 103 St. Indianapolis, IN, United States, 1997. – 698с.

14. Пйерс Б. Types and Programming Languages : Типи та мови програмування / Бенджамін Пйерс. – Mit Press, 2002. – 648с.

15. Мартін Р. Clean Code : Чистий програмний код / Роберт Мартін. – Pearson, 2008. – 464с.

16. Фріман Е. Head First Design Patterns : Головні шаблони дизайну / Е. Фріман, К. Сієра. – O'Reilly Media, Inc, 2004. – 694с.

17. Макконел С. Досконалий код / Стів Макконел. – Издательско-торговий дом «Русская Редакция», Питер, 2013. – 896с.

18. Кормен Т. Вступ до алгоритмів (Алгоритми: побудова та аналіз) / Т. Кормен, Ч. Лейзерсон, Р. Рівест та інші. - К.І.С., 1989. – 1312 с.

19. Кнут Д. Мистецтво програмування / Дональд Кнут. – Вильямс, 1968. – 720с.

20. Гант Е. The Pragmatic Programmer : Прагматичний програміст / Е. Гант, Т. Дейв. – Addison-Wesley Professional, 1999. – 352с.

21. Брукс Ф. Міфічний людино-місяць / Фред Брукс. – Символ-Плюс, 1975. – 304с.

22. Вірт Н. Algorithms + Data Structures = Programs : Алгоритми + Структури даних = Програма / Ніклаус Вірт. – Prentice Hall, 1976. – 366с.

23. Лаакман Г. Cracking the Coding Interview : Співбесіда з програмування / Гейл Макдавелл Лаакман. – Createspace, 2008. – 500с.

24. Маєрс Г. The Art of Software Testing : Искусство тестування / Гленфорд Маєрс. – John Wiley s Sons, 1979. – 800с.

25. Спольськи Д. Joel on Software : Джоел про програмне забезпечення / Джоел Спольськи. – Apress, 2004. – 384с.

26. Хліпала А. Certified Programming with Dependent Type : Сертифіковане програмування з окремими типами / Адам Джеймс Хліпала. – The MIT Press, 2013. – 424с.

27. Петзольд Ч. Code: The Hidden Language of Computer Hardware : Код: приховані мови програмного забезпечення / Чарльз Петзольд. – Microsoft Press, 2000. – 393 с.