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

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ *Сергій СТІРЕНКО*
(підпис)

“___”_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп’ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Система контролю бронювання заходів спортивного комплексу»

Виконав:
студент _IV_курсу, групи__ІП-64__
Зайченко Василь Володимирович _____
(підпис)

Керівник:
Асистент
Аленін Олег Ігорович _____
(підпис)

Консультант з нормоконтролю:
Професор, доктор технічних наук
Симоненко Валерій Павлович _____
(підпис)

Рецензент

_____
(підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
"___"_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Зайченку Василю Володимировичу

1. Тема проєкту «*Система контролю бронювання заходів спортивного комплексу»*

керівник проєкту Аленін Олег Ігорович, асистент, затверджені наказом по університету від *«__07__» __травня__* _____ 2020р. №__1081-с__

2. Термін здачі студентом закінченого роботи_____ 2020р.

3. Вихідні дані до проєкту *технічне завдання, теоретичні дані.*

4. Зміст пояснювальної записки: *Дослідження існуючих рішень. Обгрунтування та вибір технологій розробки. Проектування архітектури програми. Проектування сховища даних. Реалізація логіки програми. Інструкція з користування. Ілюстрація розробленого додатку та шляхи вдосконалення.*

5. Консультант роботи, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Симоненко В. П.		

6. Дата видачі завдання *01.09 .2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1.	*Затвердження теми роботи*	01.08.2019	
2.	*Вивчення та аналіз завдання*	15.09.2019-15.03.2020	
3.	*Розробка архітектури та загальної структури систем*	15.03.2020-25.03.2020	
4.	*Розробка структур окремих підсистем*	25.03.2020-05.04.2020	
5.	*Програмна реалізація системи*	05.04.2020-15.04.2020	
6.	*Оформлення пояснювальної записки*	15.04.2020-20.05.2020	
7.	*Передзахист*	26.05.2020	
8.	*Захист*	25.06.2020	

Студент Василь ЗАЙЧЕНКО _____

(підпис)

Керівник Олег АЛЕНІН _____

(підпис)

Анотація

У даній бакалаврській дипломній роботі розроблений веб-сервіс системи контролю бронювання заходів спортивного комплексу. Додаток призначений для комфортного регулювання подій даного комплексу.

Функціональність програми надає можливість створення та редагування бронювання спортивних заходів.

Серверна частина програмного продукту була реалізована на мові програмування Java, використовуючи бібліотеку Spring. Для написання графічного інтерфейсу були використані JavaScript, JQuery, Thymeleaf, CSS та HTML.

Annotation

The web service of the sports center booking management system is developed in this bachelor's work. The application is designed for comfortable regulation of events in this complex.

The functionality of the program provides the ability to create and edit sports bookings.

The server part of the software product was implemented in the Java programming language using the Spring library. JavaScript, JQuery, Thymeleaf, CSS and HTML were used to write the graphical interface.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ.467100.001 ВП	Відомість проєкту	1	
3	А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	70	
5	А4	ІАЛЦ.467100.004 Д1	Алгоритм програмного забезпечення	1	
6	А3	ІАЛЦ.467100.005 Д2	Функціональна схема (діаграма класів)	1	
7	А3	ІАЛЦ.467100.006 Д3	Структурна схема веб-застосунку	1	
8	А3	ІАЛЦ.467100.007 Д4	Текст програми	10	

Зм.	Арк.	№ документна	Підпис	Дата	ІАЛЦ.467100.001 ВП		
Розроб.		Зайченко В.В.			Система контролю бронювання заходів спортивного комплексу *Відомість дипломного проєкту*	Літ.	Аркуш 1 / Акрушів 1
Перевір.		Аленін О.І.					
Н. Контр.		Сімоненко В. П.				НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, гр.ІП-64	
Затверд.							

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня__бакалавр__

на тему: “Система контролю бронювання заходів спортивного комплексу”

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Бакалварська робота поширюється на дослідження та створення «Системи контролю бронювання заходів спортивного комплексу».

Область застосування: використання студентами, які проводять свій активний відпочинок займаючись футболом на території стадіону НТУУ «КПІ», щоб з легкістю резервувати поле.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр програмної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою бакалаврської роботи є створення веб-додатку, щоб полегшити життя студентам, що хочуть займатися футболом на території університету.

Розробка призначена для студентів та заслужених тренерів, які регулюють бронюванням футбольного поля.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з програмування, архітектури веб-додатків та ефективного використання баз даних, публікації в Інтернеті та в періодичних виданнях, що висвітлюють дані питання

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту, що розробляється

- Розробка архітектури за вимогами світових сучасних тенденцій
- Розробка серверної частини дотримуючись шаблонів проектування
- Створення інтуїтивно-зрозумілого та привабливого графічного користувацького інтерфейсу.
- Керування бронювання стадіону користувачами
- Контроль збоку адміністрації

5.2. Вимоги до програмного забезпечення

- Операційна система Windows або Linux
- Java 8, та новіші версії
- PostgreSQL 12 та новіші версії
- Intellij Idea 2019 та новіші версії

5.3. Вимоги до апаратної частини

- Процесор Intel та кращі.
- Оперативна пам’ять 4 Гб+.
- Вільне місце на жорсткому диску 20Гб+.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	20.12.2019
Складання і узгодження технічного завдання	20.12.2019
Створення модулів системи, що розробляється	12.02.2020
Тестування окремих модулів системи	20.03.2020
Допрацювання, налагодження і виправлення помилок	02.04.2020
Оформлення документації дипломної роботи	17.04.2020

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

на тему: “ Система контролю бронювання заходів спортивного комплексу”

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTTP – це протокол передачі гіпертекстових документів, який широко використовується в інтернет мережах.

ACID – це набір принципів, які надають транзакції бази даних коректну роботу. Основні принципи:, узгодженість, довговічність, ізольованість, атомарність.

OAuth – це стандарт авторизації у відкритому доступі, що надає змогу користувачам надавати доступ до приватних даних, інформації, документів (для прикладу, фотографії, контакти, відео та інші персональні дані), які зберігаються на веб-сайті, при цьому немає потреби вводити дані

Фреймворк – інфраструктура програмного коду уже з готовити рішенями поширених завдань, що спрощує розробку складних додатків та систем. Для легшого розуміння, дану інфраструктуру можна назвати своєрідною певною бібліотекою.

API – це прикладний програмний інтерфейс, який надає набір визначень для взаємодії програмного забезпечення один з одним.

Back end – серверна частина додатку, який зазвичай складається з контролерів, сервісів, репозиторіїв та моделю.

Front end – це графічний інтерфейс який надає змогу взаємодії між користувачем та серверною частиною програми. Front end та back end розподілені між собою однією або кількома системами, які також взаємодіють між собою.

REST – це підхід до архітектури протоколів, які надають доступ до ресурсів інформації.

JSON – це спеціальний формат, що надає змогу обмінюватися даними між користувачами, сервісами.

URI – це уніфікований ідентифікатор ресурсу, за допомогою якого ми можемо звертатися до певного ресурсу.

СУБД – це система управління базами даних.

HTML – це мова розмітки гіпертексту, для створення веб-сторінок в мережі інтернет.

CSS – це каскадна таблиця стилів, що забезпечує привабливий дизайн продукту.

Веб-сервер – це сервер, який приймає запити від користувачів від браузерів та повертає відповіді за допомогою протоколу HTTP, надаючи користувачеві зазвичай або HTML-сторінку, зображення, документ або інші дані.

Бібліотека – це система підпрограм або об'єктів, що надають розробці додатквого програмного рішення певних задач.

Веб-застосунок – це додаток, сервером стає веб-сервер, а клієнт – веб-браузер. Бізнес-логіка системи виконується на сервері, а веб-браузер надає користувачеві інформацію, яка завантажується мережею з сервера, що згодом передається назад користувачеві.

ВСТУП

На сьогоднішній день збільшується попит на навчання на технічних спеціальностях, особливо тих, де готують ІТ-спеціалістів. Популярність цього підтверджує не лише підвищений конкурс абітурієнтів на дані спеціалізації, а й швидке зростання кількості комерційних курсів, які навчають мовам програмування.

Проте незважаючи на численні переваги навчання на технічних спеціальностей університету, студентам у сфері інформаційних технологій притаманний малорухливий спосіб життя. Він є причиною цілої низки захворювань таких як:

- сколіоз;
- шийний остеохондроз;
- остеохондроз поперекового відділу хребта;
- протрузія міжхребцевих дисків;
- грижа поперекового відділу хребта;

та інших.

Найчастіше люди, які ведуть сидячий спосіб життя відчувають проблеми у поперековому відділу хребта. Для забезпечення правильної осанки та запобіганню болі в спині існує декілька підходів боротьби з чинниками, що на них впливають. Правильна організація робочого місця - перший крок до відсутності проблем зі здоров'ям. Важливо налаштувати не лише висоту стільця під власний ріст, а й не забувати за такі деталі як: розташування монітора, відстань та кут екрану до очей, положення тіла та зручність стільця.

Проте організація робочого місця — це лише початок, адже сидіти безперервно на одному місця теж не корисно. Для підтримання тіла у нормі важливим чинником є фізичні навантаження. За недавніми дослідженнями, сидячий спосіб життя неможливо компенсувати лише фізичними вправами під час невеликих перерв між сидінням на одному місці. Тому спорт — це не рекомендація, а необхідність.

Київський Політехнічний інститут надає широкий спектр можливостей для дозвілля власних студентів. Спортивний комплекс університету має велике розмаїття: від спортивних майданчиків на території студентського містечка до басейну та двох професійних футбольних стадіонів. Крім того, дисципліна фізичного виховання є обов'язковою для студентів першого та другого курсу. Університет надає довгий перелік секцій для відпрацювання даної дисципліни. Більше 20 видів секцій допоможуть підвищити рівень спортивної підготовки та зменшити ризик на захворювання, які зв'язані з болем у спині. Спортивний комплекс університету доступний і для всіх бажаючих.

У вік технологій та прогресу найкращий технічний університет країни використовує паперовий варіант бронювання футбольного поля. Більше того, ця можливість доступна лише в певному місці та певний час. Надається лише одна година у тиждень на території стадіону, що біля 7 корпусу. Це викликає складнощі для бронювання студентами футбольного поля, особливо для тих студентів, які не проживають в студмістечку НТУУ КПІ ім. І. Сікорського. Для студентів, які проводять більшість часу в сидячому положенню необхідні спортивні навантаження, а веб-додаток "Ядро" стане інструментом в цьому.

Веб-додаток призначений для бронювання та контролювання замовлення футбольних стадіонів НТУУ КПІ ім. І. Сікорського. Наразі у додатку присутні два стадіона НТУУ КПІ: "Ядро" - це стадіон, який розташований біля 7 корпусу університету та стадіон, який підпорядкований 24 корпусу. Стадіон "Ядро" має два футбольний поля: ліве пів-поле (left pitch) і праве пів-поле (right pitch). Стадіон 24 корпусу не поділяється на окремі частини.

Для кожного футбольного поля та пів-поля розроблений власний календар з бронюванням часу та таблиці з головними його характеристиками, такими як:

- кількість м'ячів;

• кількість манішок;

• наявність газону;

• прожектор;

• роздягальня та душ.

На календарі футбольного поля є можливість додати, змінити та видалити власне замовлення. Користувачам наявна можливість переглядати календар футбольних полів із замовленнями інших користувачів. Наявна адміністративна частина для контролювання та зміною сайту. Реалізовані всі вимоги до адміністратора та користувача. Дотримано всі внутрішні правила та рекомендації для бронювання футбольного поля.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1.Опис завдання

У даній роботі буде розроблений сервіс системи контролю резервації заходів спортивного комплексу. Дане програмне забезпечення призначене для зручного регулювання подій студентами даного комплексу.

Функціональність програми надає змогу користувачам створювати та керувати власними бронюваннями спортивними заходами.

Окрім звичайних користувачів наявні адміністратори, які регулюють веб-додаток аби забезпечити його правильне функціювання та коректну роботу. Звісно, що адміністратор щоб досягти даної мети йому надано більше прав та можливостей, зокрема управління користувачами, футбольними полями та іншими важливими частинами системи. Більш детально про функціональність програми та права і можливості користувача і адміністратора буде висвітлено в наступному розділі.

1.1.1. Перелік вимог до програмного забезпечення

Щоб забезпечити коректність вибору характеристик існуючих аналогів, на які буде звернута увага при дослідженні, і для формування критеріїв їх роботи, вкажемо основну ціль розробки даного веб-додатку. Основна ціль програмного забезпечення – це створення ефективної системи регулювання заходів.

На сьогодні, існуючі різноманітні «бронювальники» не мають достатнього функціоналу для якісного та ефективного забезпечення резервації футбольного поля для університетів, також мають непривабливий користувацький графічний інтерфейс, що є з одною з причин неіснування даного проекту в нашому університеті. Також не слід забувати про апаратну складову програмного забезпечення, чи є можливість використання додатку з усіх пристроїв користувача, не тільки персональний комп’ютер, а навіть з мобільного телефону.

Отже, основними характеристиками, які будуть оцінені існуючих аналогів, є:

• графічний користувацький інтерфейс;

• можливість створення, редагування, видалення подій;

• універсальність використання;

• контроль збоку адміністрації.

• автономність.

• безпечність.

• надійність.

Розглянемо детальніше критерії інтерфейсу:

- Кількість елементів повинна бути мінімальною, для того щоб надати користувачеві лише справді необхідну функціональність;
- Привабливість дизайну, оскільки це розширює аудиторію та робить процес використання даного веб-додатку приємним та ефективним

Для розширення цільової аудиторії необхідно, щоб студенти могли легко та ефективно використовувати даний продукт, тобто він має бути простим, зручним, але повнофункціональним, має забезпечувати всі примхи та потреби користувачів. Але при цьому, має бути жорсткий контроль та нагляд збоку адміністрації футбольного поля для забезпечення правомірної та коректної діяльності системи бронювання заходів на спортивному комплексі. Автономність означає, що студенти без особистої присутності на футбольному полі, а також тренерів можуть легко забронювати стадіон, щоб з задоволенням активно провести свій вільний від навчання час. Надійність, означає, що попри будь-які проблеми та навантаження, веб-додаток буде стабільно працювати, або надавати конкретні помилки при їх існуванні.

Отже, зробимо дослідження найпопулярніших та найсучасніших аналогів, щоб зрозуміти найкращі особливості даних проектів та втілити у життя деякі з них.

1.2. Аналіз існуючих рішень

1.2.1 Gnom Guru

Gnom.guru – це додаток для операційної системи Android, що забезпечує запис та облік клієнтів[1]. Гном Гуру – це мобільний додаток, який надає змогу легко та зручно користувачам записувати власних клієнтів бізнесу.

Провівши дослідження з фахівцями сфери послуг, власниками невеликого підприємства/бізнесу, салонів краси, медичних лікарнь та клінік, автомийок та автосервісів, приватних стилістів/парихмахерів та фітнес-інструкторів, дана компанія, зробила висновок, що незручно зберігати всю потрібну інформацію про їхніх клієнтів в особистому записнику або на комп'ютері, адже це не є ефективним, комфортним та зручним у використанні кожного дня.

Була замічена тенденція, що чим більше клієнтів у компанії, тим складніше виявити постійних, тих, що витрачають найбільше коштів, а також тих, хто відвідує заклад з самого початку існування. Ці дані необхідні для того, щоб наприклад запропонувати їм спеціальні умови бізнесу, такі як купони, знижки, подарунки, а також мати можливість нагадати користувачам про майбутній візит.

В даної програми великий спектр сфер аудиторії, які використовують даний продукт:

- Краса: спеціалісти по манікюру, стилісти, візажисти, парихмахері, косметологи та інші
- Спорт: фітнес-тренери та тренери йоги
- Здоров’я: реабілітологи, логопеди, масажисти, стоматологи, дієтологи, психологи та інші лікарі
- Сервіс: ткачі, юристи, автомеханіки, сантехніки, фотографи, електрики, грумери, няні, репетитори
- Продажі: бізнес-консультанти, страхові агенти, ріелтори, коучери

Даний додаток має привабливий графічний користувацький інтерфейс, який робить роботу користувачів приємною та легкою. Система інтуїтивно-зрозуміла та проста у використанні навіть для тих, хто має недостатню кваліфікацію з роботою мобільних додатків. Дане мобільне програне забезпечення містить календар, робота якого надає всю необхідну функціональність для потреб користувачів. Даний календар, зображений на рис 1.1.

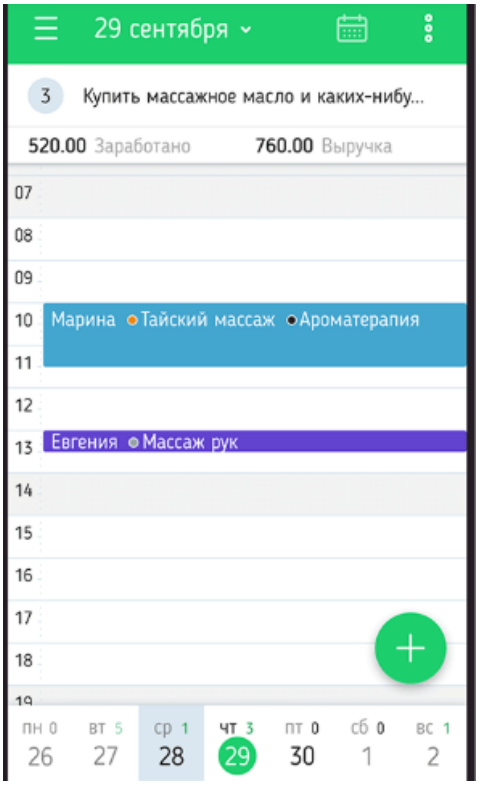


Рис. 1.1. Календар подій Gnom guru

Основні характеристики та можливості даної системи[2]:

- Швидко і зрозуміло. Основне завдання мобільного додатку зробити роботу користувачів приємною, простою і швидкою. Зберігаючи при цьому, сучасні методи розробки в керуванні розкладом, обліком клієнтів та лояльною політикою до користувачів. ГномГуру поєднує великий функціонал та інтуїтивно-зрозумілий графічний інтерфейсом.
- Простий запис ділових зустрічей і контактів зі споживачами. Всього три кроки необхідні для призначення зустрічі з клієнтом

- Наявність підказок на початку використання додатку розкривають весь функціонал програми.
- Настройку персоналізації займає лише п’ять хвилин
- Автоматична відправка персональних SMS клієнтові. Можливість визначити унікальний текст для кожного типу повідомлень.
- Розроблена зручна модель обліку і статистики. Облік клієнтів, статистика послуг, використаних матеріалів, оплати і відвідуваності.
- Перегляд розкладу, запису клієнтів можливо і в режимі відсутності інтернету
- Безпечний доступ до особистого розкладу з будь-якого девайсу.
- Інформація по клієнту до прийняття виклику

Щоб встановити певну подію потрібно натиснути на кнопку «Добавити», а потім у формочці заповнити всі дані:

- Назва події
- Дата події
- Час події
- Бізнес-партнер користувача, з ким призначена дана подія
- Тип події
- Колонка для додавання певних матеріалів, інструкцій, для прикладу, можна добавити зображення, відео або інший необхідний документ
- Товари для події
- Загальна сума
- Коментар при необхідності користувачі

Дана форма додання події зображена на рис. 1.2

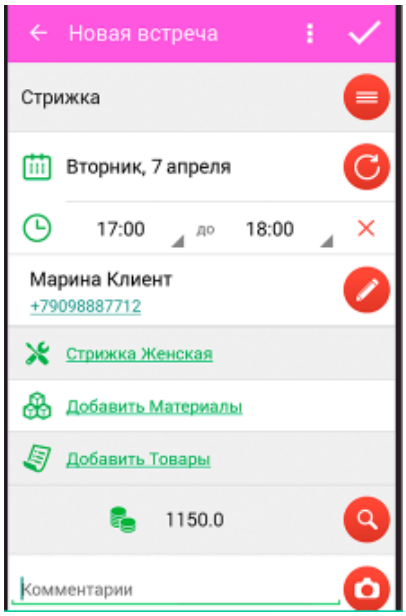


Рис. 1.2. Додання нової події у Gnom guru

Дана система має автоматичні SMS-нагадування, що вирішує проблему того, що багато клієнтів, незважаючи на наявний попередній запис, не приходять на вказану зустріч, маючи різні на це вагомі та невагомі причини: деякі клієнти втратили відлік часу і не прийшли, а деяким було незручно скасувати зустріч по телефону з користувачем. У підсумку, користувач втрачає кошти. Тому провівши дані дослідження, Gnom.guru добавив функцію автоматичного нагадування кожному клієнту про майбутній візит, що знизить неявку мінімум на 30 відсотків, що дасть користувачеві додатковий прибуток та причину рекомендацій та подальшого використання продукту.

Також присутня офлайн-версія даної системи, що надає багато переваг у його використанні. Gnom.guru забезпечує непотрібність сидіти на робочому місці в очікуванні дзвінка клієнта. Запис клієнтів, складання розкладу, планування зустрічей за допомогою телефона або планшета відбувається буквально в три кроки. При цьому немає залежності даного додатку від наявності доступу до Інтернету. Gnom.guru надає можливість переглядати розклад та записувати клієнтів в режимі оффлайн. Приклад роботи системи в режимі офлайн зображено на рис 1.3.

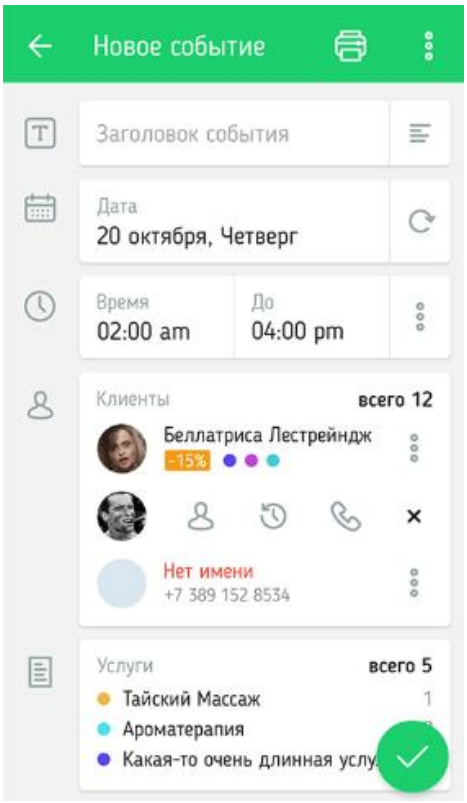


Рис. 1.3. Приклад роботи Gnom Guru в режимі офлайн

Дослідивши даний продукт, необхідно підкреслити наступні особливості продукту:

- Мобільна версія
- Привабливий і зручний графічний інтерфейс
- Наявна вся функціональність календаря
- Керування власними подіями
- СМС-нагадування
- Історія клієнтів, статистика
- Можлива робота в режимі офлайн
- Надійність та стабільність
- Безпечність використання клієнтами
- Відсутня можливість одночасного використання
- Програма не є безкоштовною (від 50грн/місяць за використання на єдиному пристрої)

1.2.2. ЗаписОнлайн

ЗаписОнлайн – це сервіс онлайн записів клієнтів користувача, що забезпечує надійна та автоматизовану роботу з заходами[3]. Дана система має зручний та простий сервіс переваги якого збільшать дохід користувачеві та розширюють цільову аудиторію користувача.

Використовує вбудований модуль CRM, що надає змогу перетворювати разові відвідування в постійних та лояльних клієнтів. Наявна функція, яка спрощує процес запису, адже кнопку «записатися» можливо встановити на власний веб-сайт, в Інстаграм і інші соціальні мережі. Смарт-віджет допомагає адміністратору в один крок зробити запис клієнта при вхідному дзвінку з ним.

Форма запису для клієнта зображена на рис 1.4 .

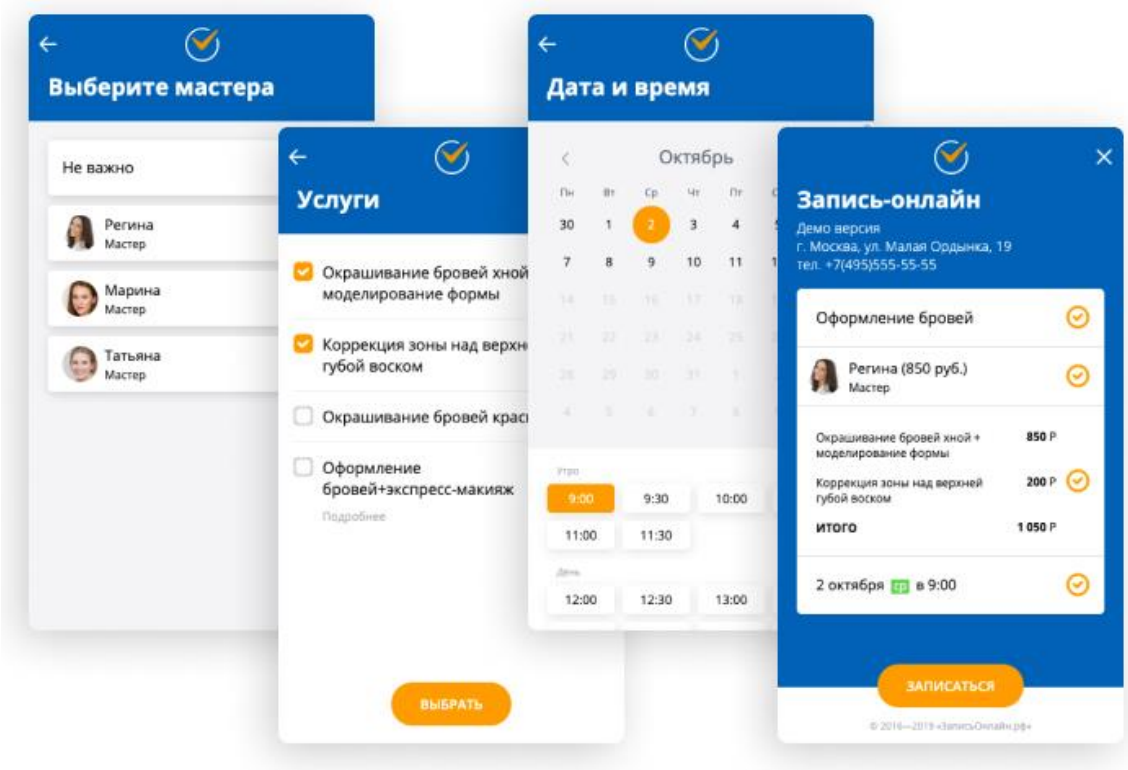


Рис 1.4. Форма запису для клієнта ЗаписОнлайн

Щоб зробити запис клієнту, необхідно зробити наступні кроки, що надає функціональність:

- Клієнт обирає зручний час для візиту до спеціаліста, при цьому не потрібно проходити довгі черги та вести телефонні переговори з оператором.
- Запис підтверджується за допомогою СМС-повідомлення
- Можлива передплата VISA
- Автоматично добавляється подія у особистий додаток Google календар
- Адаптована для планшетів і смартфонів
- Легко встановлюється на мобільний телефон
- Має зручний користувацький інтерфейс, який зображено на рис 1.5.

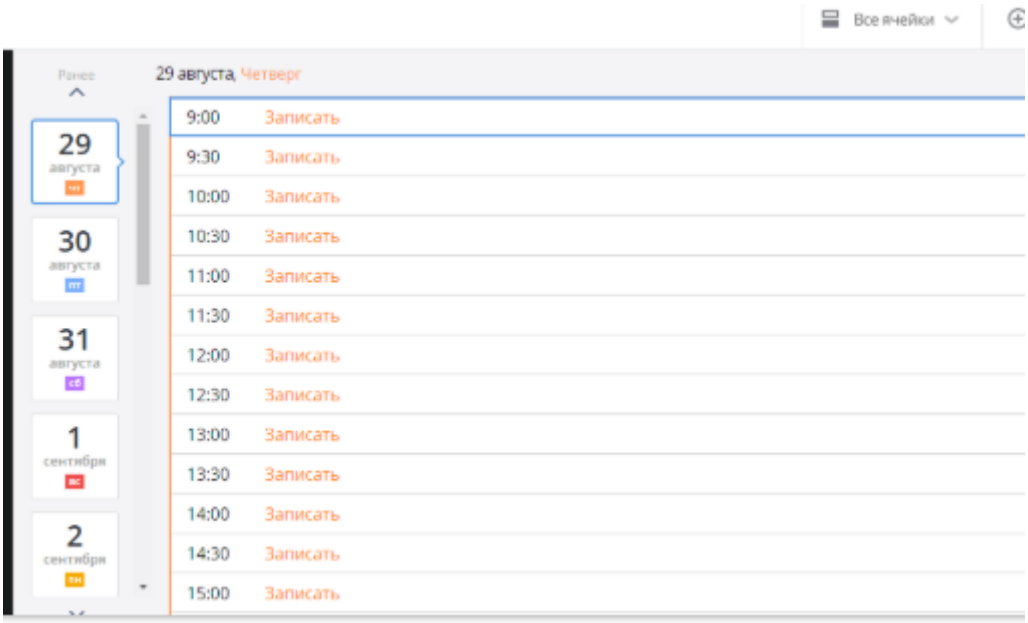


Рис. 1.5. Приклад роботи з системою ЗаписОнлайн

Стандартна сфера застосування даної системи:

- Салон краси, парихмахерські
- Клініки, лікарні, оздоровчі центри, санаторії
- Автосервіси, СТО
- Організація різноманітних засобів
- Навчання в університеті

Особливості ЗаписОнлайн для користувача:

- Детальний опис послуг/подій та переваг
- Можливість додавання фотографії спеціалістів та їх досягнення
- Оригінальна схема проїзду до місцезнаходження
- Розклад роботи спеціалістів та будівель
- Керування записами за допомогою панелі адміністратора
- Доступність і контроль з будь-якої частини світу
- Можливість надсилання СМС-нагадувань
- Аналіз на основі статистики, які надають клієнти

Схему календаря подій можна побачити на рис 1.6.

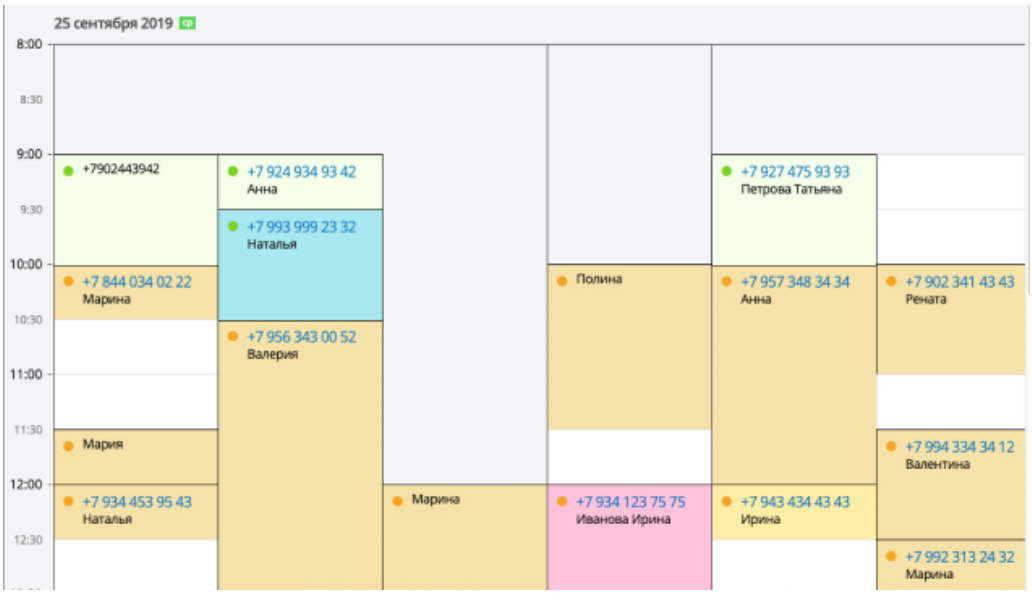


Рис. 1.6. Приклад роботи календаря ЗаписОнлайн

Дослідивши дану систему ЗаписОнлайн, необхідно підкреслити головні особливості[4]:

- Мобільна версія для клієнтів та веб-версія для користувачів
- Зручний користувацький графічний інтерфейс
- Управління подіями – створення та редагування
- СМС-повідомлення для нагадування
- Аналіз за статистикою

- Стабільна робота системи
- Безпечне використання користувачами і клієнтами
- Відсутня можливість додання запису клієнтами
- Програма небезкоштовна (від 800грн/місяць за використання організацією)

1.2.3. SimplyBook.me

SimplyBook.me - це система онлайн-запису для різних галузей сфери послуг. Головна сторінка SimplyBook.me зображено на рис. 1.7.

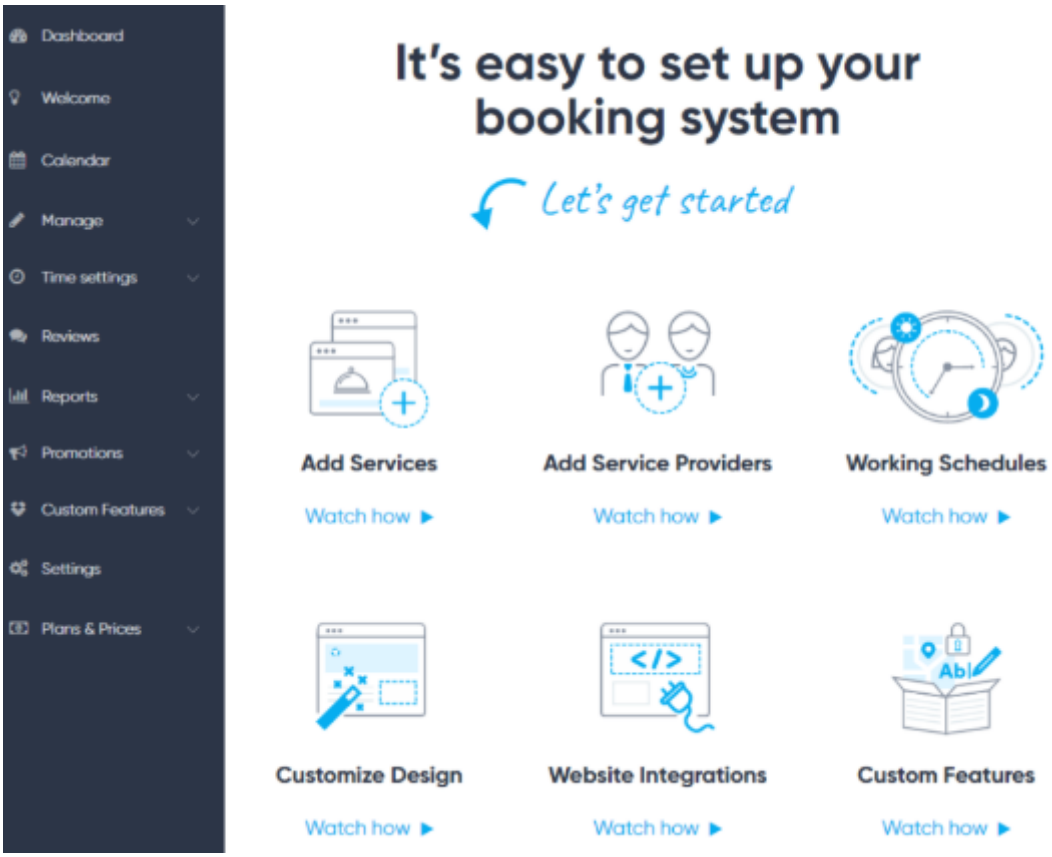


Рис. 1.7. Головний екран SimplyBook.me

Основні переваги SimplyBook.me[6]:

- Онлайн-запис. Розроблений веб-сайт для бронювання, який можна самостійно налаштувати.
- Оптимізований для мобільних пристроїв
- Можливість розміщення віджету записів на власному веб-сайті або соціальних мережах Facebook і Instagram

- Повідомлення по SMS/Email. Відправка повідомлень співробітникам та клієнтам при створенні, видаленні або редагуванні запису.
- Безпечність додатку. Шифрування даних при передачі, а також їх резервне копіювання щодня.
- Підтримка платежів. Можливість прийняття онлайн-переводів використовуючи Stripe, PayPal та інших систем оплати або ж за допомогою вбудованої системи POS, яка приймає і готівку і картку.
- Інтеграція & API. Вбудована інтеграція з соціальними мережами Facebook та Instagram. Легка інтешрація з Wordpress та різноманітними CMS-системами, а також додатково надається API, при необхідності розробити рішення розширюючи даний продукт.
- Колоритний спектр доповнень. Широкий вибір доповнень включають в собі: акційні купони та карти подарунків, схожі товари, абонементи для постійних користувачів, промо-коди, високий рівень безпеки особистих даних HIPAA, присутні SOAP-коментарі, додаткові методи і поля, пакети розробки, система оплати POS, та інші доповнення, що полегшують життя і користувачам і розробникам.

Розглянемо, що надає SimplyBook.me для бізнесу:

- Можливість розширення. Необмежена кількість клієнтів, сервісів, співробітників та користувачів даної системи без додаткової плати.
- Легке налаштування шаблону повідомлень, нагадувань, запиту відкликання.
- Можливість кастомізації під свій бренд

- Бізнес-аналітика. Можливість відстеження кількості бронювання, відвідувань веб-сайту, найпопулярніші сервіси та співробітників на всіх пристроях та у будь-який час дня.
- Синхронізація веб-сайту бронювання з особистим Outlook або Google календарем, для того щоб уникнути недостовірності розкладу
- Наявна можливість створення декількох адрес робіт, вказати для співробітника його місцезнаходження роботи, для того щоб забезпечити клієнтів вибором найпідходящого варіант.
- Наявність знижок і супер-пропозицій
- Штучний інтелект надає схожі товари для обраного клієнтом
- Забезпечення кожного співробітника універсальним календарем для бронювання подій, приклад якого зображений на рис 1.8

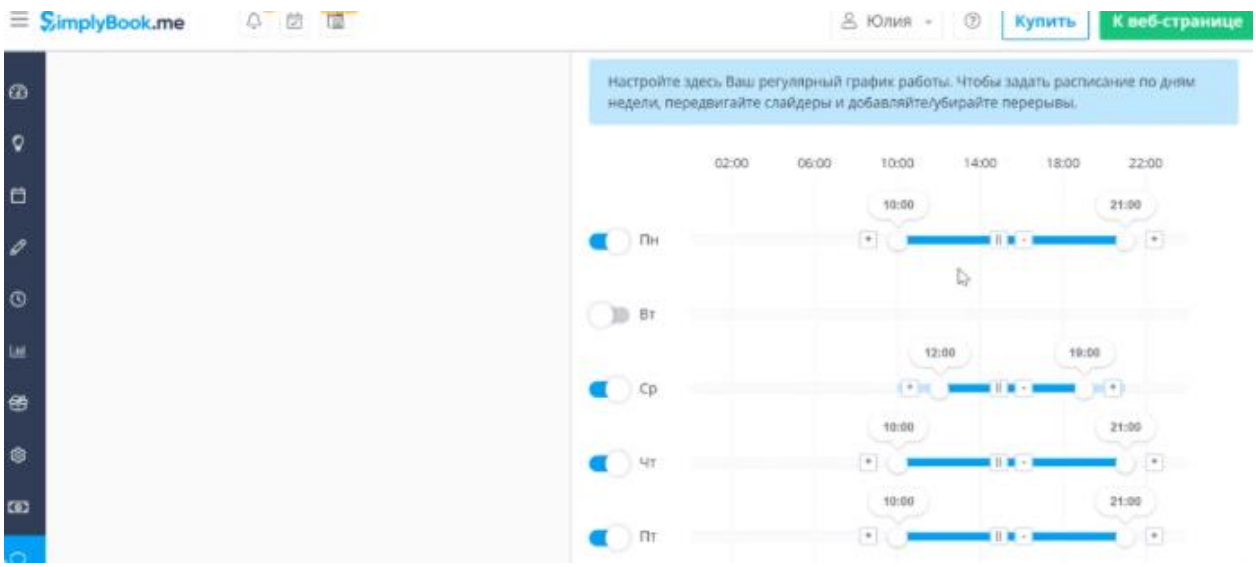


Рис. 1.8. Особистий календар для співробітника в SimpleBook.me

Найпопулярніші переваги для клієнтів SimpleBook.me, які варто відзначити:

- Клієнти можуть бронювати заходи онлайн використовуючи кілька джерел: особистий веб-сайт бронювання, особистий сайт-

візитка, Booking.page, аккаунт Facebook та профіль в соціальній мережі Instagram.

- Email та SMS нагадування, а також автоматичні підтвердження бронювання, для того щоб клієнти не забули про зустріч.
- Система автоматично надсилає запит відгуку після зустрічі, надаючи клієнтам можливість опублікувати їхні враження. Клієнти мають змогу ознайомитися з відгуками інших відвідувачів.
- Присутні замовлення одразу кількох заходів, а також постійні нагадування перед кожною забронюваною зустріччю.
- Простота скасування запису. Клієнти мають можливість скасування запису за необхідності. Можна налаштувати умови скасування запису.
- Запис для групи відбувається в один крок, а при необхідності наявна можливість оплачувати онлайн бронювання.

Безпека для сервісу SimplyBook.me - це безумовно пріоритет. Захист особистих даних має дві мети: перше, заборонити доступ неавторизованому користувачу до конфіденційної інформації при усіх подіях; друге, мінімізувати шанс втрати особистих даних. Всі особисті паролі зашифровані та захищені.

SimplyBook.me щодня робить резервування даних, інформації на власних дисках та центральних серверах. Запропоновано додаткові функції для захисту особистих даних, такі як SSL-шифрування передачі особистої даних між веб-сайтом бронювання та веб-системою додатку, а також захист даних використовуючи HIPAA, що забезпечує всі відповідні вимоги з безпеки. SSL-шифрування передачі особистих даних між графічним інтерфейсом адміна та системою є за замовчуванням.

Доповнення «Щоденний звіт» щоранку надсилає звіт з актуальними статусами всіх бронювань клієнтів на наступний день та на наступний робочий тиждень.

Відстеження активності веб-сайту за допомогою Google аналітики та результати статистики в особистому Google Adword. Приклад використання даної статистики та аналізу приведені на рис 1.9.

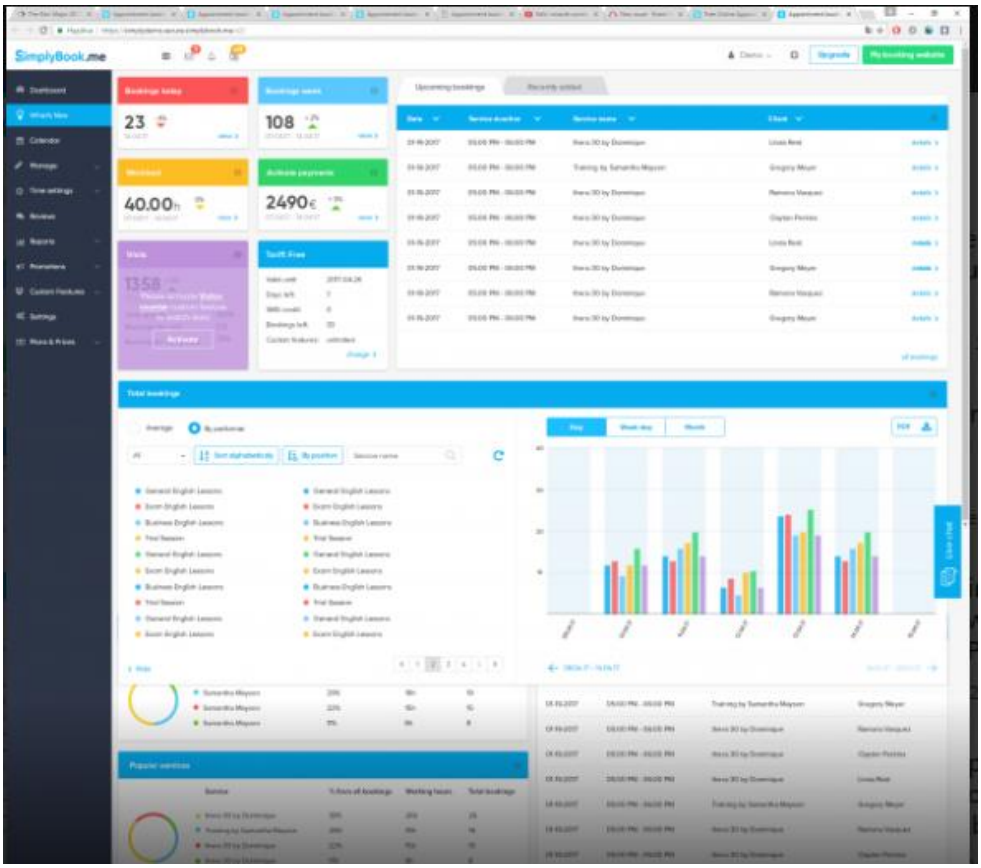


Рис. 1.9. Статистика та аналітика в SimpleBook.me

Співробітники мають можливість синхронізувати свої Outlook та Google календарі з особистим календарем SimplyBook.me. Двостороння синхронізація, що забезпечує уникнення накладок.

Забезпечений системою автоматичний експорт всіх особистих даних в календар. Завдяки регулярному автоматичному експорту бронювань в особистий Google календар.

Google і Outlook календарі надають дані у веб-систему бронювання, і при наявності особистих зустрічей, час буде заблоковано в SimplyBook.me, щоб клієнти користувача не змогли забронювати зайнятий час.

Дослідивши веб-систему SimpleBook.me потрібно звернути увагу на головні особливості:

- Мобільна версія та веб-додаток і для клієнтів і для користувачів
- Інтуїтивно-зрозумілий графічний інтерфейс
- Легке керування подіями
- СМС-нагадування клієнтам
- Аналіз та статистика даних і бронюваннь
- Надійна робота веб-системи
- Безпечний веб-додаток
- Відсутня можливість редагування запису клієнтами, лише користчувачами
- Додаток не є безкоштовним (від 250грн/місяць за використання користувачем)

ВИСНОВКИ ДО РОЗДІЛУ 1

У даному розділі був виконаний аналіз трьох уже розроблених веб-сервісів з подібним функціоналом до системи контролю бронювання певних заходів. Ця робота була виконана з певною метою — дослідити і порівняти програмні рішення заради уникнення помилок у розробці проекту бакалаврської роботи та у формуванні “кращих практик” для написання власного веб-додатку. У наведеній знизу таблиці було порівняно вибрані веб-сервіси за певними критеріями.

Таблиця 1.1 Аналіз аналогів додатку

Наявність функціоналу	Gnom Guru	Запис Онлайн	SimpleBook.me
Створення заходу клієнтом	+	+	+
Автоматичне підтвердження	+	+	-
Веб-версія продукту	-	+	+
СМС-сповіщення / нагадування на електронну пошту	+/+	+/-	+/-
Історія бронювань	+	-	+
Офлайн-версія	+	-	-
Імпорт та експорт подій, користувачів та інших даних	+	+	+
Доступ до подій клієнтами	-	-	-
Створення додаткового календаря при необхідності	+	+	+
Можливість інтегрувати з футбольним полем	+	-	+
Безкоштовний продукт	-	-	-

Роблячи висновок з даної таблиці, слід звернути увагу на те, що найбільшу необхідну функціональність має Gnom Guru. Але попри свої переваги даний мобільний додаток має суттєвий недолік, зокрема дана система не є безкоштовною. ЇЇ ціна розпочинається від 50 грн в місяць збільшуючись в геометричній прогресії за необхідні додаткові сервіси для коректної роботи системи бронювання заходів футбольного поля. Отже, враховуючи даний недолік, було і вирішено створити власний веб-додаток, основаного на принципах роботи сучасних аналогів, і який мав мати всю необхідну функціональність, а також не потребував додаткової плати за її реалізацію.

РОЗДІЛ 2
ОПИС ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ДОДАТКУ

2.1.Підхід до розробки системи

Розвиток людства не стоїть на місці, ринок пропонує велику кількість різноманітних та цікавих технологій для створення корисних систем. Під час аналізу розробки даної системи, було прийнято рішення використовувати всім добре знайому клієнт-серверну архітектуру, даний підхід є дуже популярним в сфері веб-розробки, такий підхід схематично зображений на рис. 2.1.

Рис. 2.1. Схематичне зображення клієнт-серверної архітектури

Дану архітектура логічно поділена на три частини, а саме[7]:

- Клієнтська частина – цю частину ще називають представлення даних, дана частина являє собою сам застосунок, який так звично бачити користувачу, в свою чергу за допомогою використання різних протоколів типу HTTP, клієнт робить запит через мережу до сервера і очікує відповідь на свій запит. Рівень, який розглядається, також використовують для опрацювання найпростішої бізнес-логіки самого додатку, до цього відносяться:
    1. головний інтерфейс для аутентифікації користувача
    2. валідація даних, при веденні в форми
    3. допоміжні легковісні операції

- Серверна частина – або шар обробки, головне завдання даної частини це прийняти від клієнта запит та за допомогою бізнес-логіки, яка написана в серверній частині обробити запит і надати клієнту повноцінну відповідь.
- База даних – це рівень доступу до даних, який несе інформацію згрупованою за певною логікою, найчастіше таке групування формує схему чи таблицю, для покращення сприйняття.

В даному розділі буде проведено дослідження сучасних мов та проаналізовано стек актуальних технології, для розробки веб-додатку. Отже, зробим аналіз сучасних мов та технологій, а саме:

- Для написання серверної частини найчастіше використовують такі технології: Java, PHP, Ruby
- Для проектування клієнтської частина: JS, ReactJS, JSP
- Найбільш вживаними базами даних виступають: PostgreSQL, Cassandra, MongoDB

2.2. Аналіз мов програмування серверної частини

2.2.1. Java

Розглянемо більш детально таку мову програмування як Java. Дана мова є об’єктно-орієнтована і призначена для створення нового програмного забезпечення загального призначення. Java для ефективного створення веб-додатків створила такий фреймворк для Spring це такий open-source фреймворк, який має контейнер з підтримкою IoC[8].

Даний фреймворк не робить прив’язку до якоїсь конкретної моделі програмування, його популярніс серед Java розробників шалено росте, тому що це альтернатива, або її ще можна розглядати як доповнення моделі Enterprise JavaBean. Java Spring Framework надає вже готові вирішення проблем, з якими часто стикаються Java розробники тай в свою чергу, сам конкурентно спроможній бізнес.

Spring надає широку функціональність, та додаткові модулі для ефективної розробки продукту. Всі розробники, які хоч раз створювали додатки з використанням Java Spring Framework можуть відмітити контейнер елементів, який реалізований за допомогою шаблону Inversion of Control. Даний підхід говорить нам про те, що самі об’єкти не будуть брати на себе відповідальність по створенню своїх екземплярів і у випадку якщо об’єкт складний не будуть шукати залежності і впроваджувати їх. Тобто сам термін можна описати як передання залежності по створенню якогось об’єкта зовнішньому компоненту.

В цілому Java Spring Framework це один дуже великий фреймворк, який в свою чергу містить в собі ряд другорядних фреймворків. Кожен з них виконує свою роль і надає можливість не створювати громіздких програмних додатків, а надає можливість розробнику, самому вирішити, які необхідні інструменти для розробки, адже даний фреймворк має модульну структуру, яка схематично зображена на рис. 2.2.

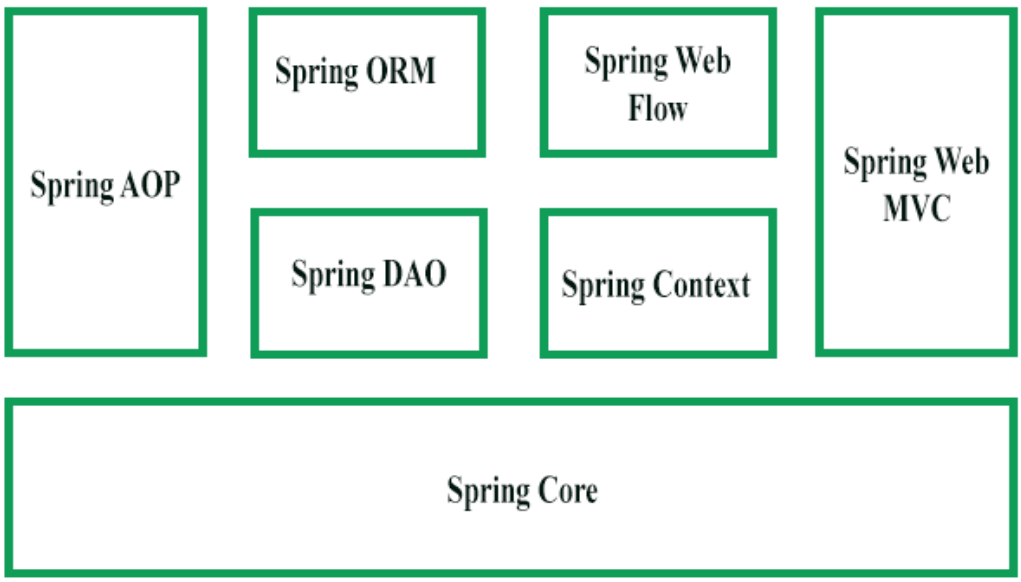


Рис. 2.2. Структура схема фреймворку Java Spring[9]

Головною ідеєю розробки програмного продукту з використанням даного фреймворку є те, що розробнику надається послідовна модель

розробки, ця модель і робить продукт універсальним та конкуренто спроможним серед більшості веб-додатків.

Розберемо детально на які модулі поділений даний фреймворк, виділяють три модулі як: core, data access та web. Java надає вже готові універсальні рішення наприклад Spring MVC реалізує архітектуру патерна Model - View - Controller. Spring MVC працює з використанням DispatcherServlet, він в свою чергу приймає і оброблює HTTP-запити, які приходять від інтерфейсу та формує готову відповідь користувачу, процес роботи зображений на рис. 2.3.

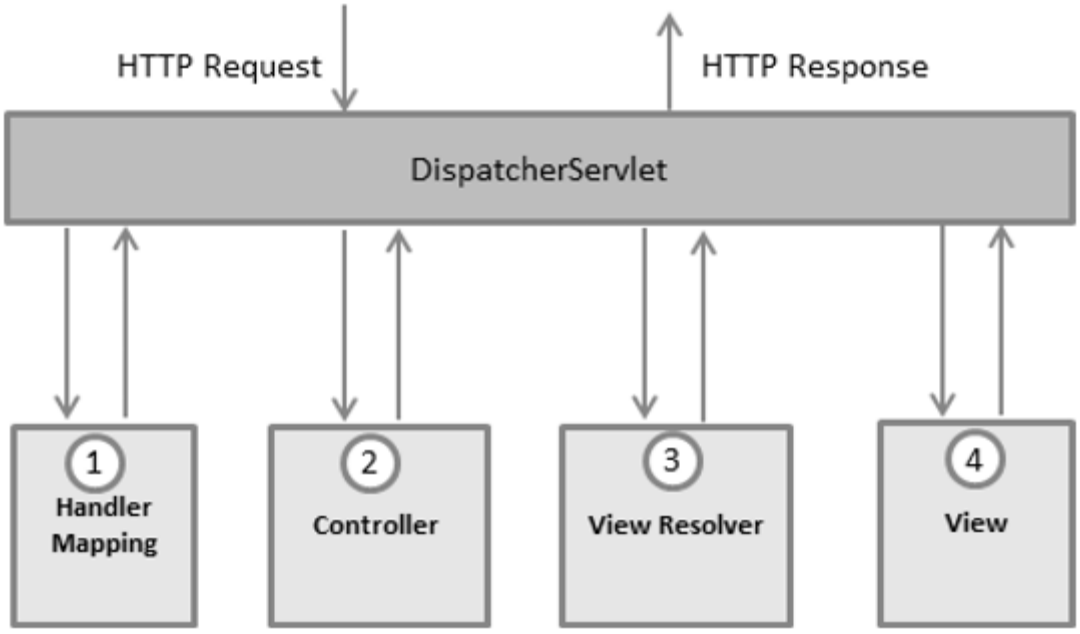


Рис 2.3. Схематичне зображення роботи Spring MVC

Модуль Spring Security надає можливість написання ефективного та гнучкого механізму аутентифікації та авторизації користувачів для кожного додатку.

2.2.2. PHP

Далі мова піде про PHP – дана мова програмування, використовується переважно для написання веб-додаків, найчастіше код написаний на PHP напряму вбудовувався в звичайні HTML сторінки.

Перший реліз даної мови був у 1995, вже через 2 роки PHP набрав такої слави, що його почали використовували близько 1-2% доменів усього світу[10].

Окрім розробки веб-додатків, дана мова знайшла своє призначення для розробки таких речей як:

– Різноманітні десктопні додатки

– Кросплатформні системи управління дронами

– Програмні математичні сервісм

Так як PHP є інтерпретованою мовою, а це означає, що весь код буде обробляєтись інтерпретатором PHP, обробка запиту зображена на рис. 2.4. Він зазвичай реалізований окремим модулем і знаходиться на сервері. Це надає перевагу більш швидко обробляти запити в порівнянні з Perl тощо. Відчутною особливістю даної мови є те , що користувач не має доступу до коду, як це наприклад є в скриптованих мовах програмування, тут все працює навпаки і браузер отримує вже готовий HTML-код.

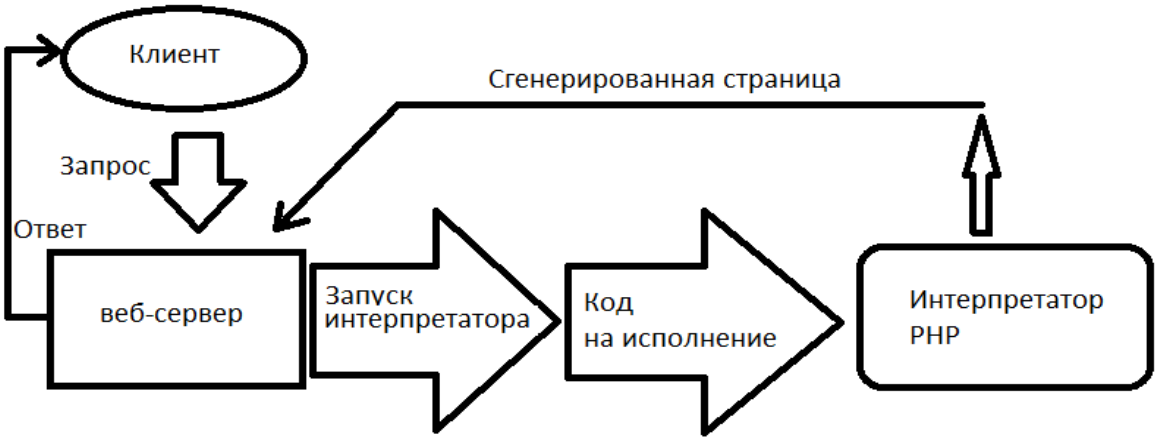


Рис 2.4. Обробка запиту в PHP

Розглянемо основні переваги, які надає PHP:

- Надає продуктивну взаємодію з БД, такими як: MySQL, Microsoft SQL Server тощо.
- Мова є дуже простою для розуміння і вивчення, вона чимось подібна до мови C та з запозиченими конструкціями з Perl.
- Велике аудиторія розробників та відкриті джерела для розробки.

- Ліцензія, яка поширюється на кожний розроблений додаткок забезпечує безкоштовне використання даної мови. Так як сирцевий код є доступний, існує можливість налаштування під свої проблеми.

Дана мова є мультипарадигмальною мовою, до основних особливостей можна віднести підтримку функціонального стилю розробки, процедурного та підтримку ООП . Використання ООП стало доступне лише з п’ятої версії і надало можливість використання: інкапсуляції, наслідування та поліморфізму, інтерфейси тощо. Ядро PHP забезпечує автоматичне керування пам'яттю і тепер розробник не хвилюється про звільнення пам'яті. Відомим PHP-фреймворками в сфері веб-розробки є такі фреймворки як : Laravel, CodeIgniter, Zend.

Перейдемо до недоліків використання PHP:

– Хоч з п’ятої версії мова має підтримку використання ООП, але її не можливо назвати об'єктно-орієнтованою мовою, адже вона є недопрацьованою відсутність індивідуальних змінних та множинного спадкування тощо. Просто присутність трьох принципів ООП, не забезпечує званням об'єктно-орієнтованої мови.

– Великим мінусом є те що основні функцію прямо вбудовані в інтерпретатор і це призвело до того, що глобальний неймспейс містить більш ніж 10000 імен функцій, які не часто й використовуються.

– Відсутність типізації викликає певні незручності при розробці.

2.2.3. Ruby

Мова Ruby є об'єктно-орієнтована з чіткою динамічною типізацією, її синтаксис є доволі простим і дуже схожий на такий що використовується в C++. Ruby надає велику кількість бібліотек, з ефективними рішеннями типових проблем, наприклад це бібліотеки для роботи з мережевими протоколами, математичні функції, засоби для адміністрування тощо[11]. Так як дана мова є інтерпретована вона є безкоштовною і доступна для великої

кількості ОС та платформ. Ruby перейняла дуже багато властивостей від таких мов як: C++, Perl, Lisp, Java.

Використання даної мови не обмежується написанням одних веб-додатків, також дану мову використовують для:

– створення різноманітних сервісів

– автоматизація роботи сервісів та проведення налаштувань;

– написання програмного забезпечення для ведення обліку та адміністрування.

Розробники, які використовують Ruby, підпорядковуються таким підходам для ефективного створення програмного забезпечення, а саме:

- Convention over configuration даний принцип використовує численні угоди по конфігурації, тобто стандарний набір конфігурацій для більшості додатків. А отже такий набір конфігурацій в рази ефективніший та спрощує створення додатків. Адже так звана явна специфікація налаштувань потрібна для нестандартних рішень.
- Do not repeat yourself – даний принцип надає фреймворку перевикористання програмного коду.
- Автоматизоване тестування – використання засобів автоматичного тестування, а саме: модульного, інтеграційного і функціонального.

Для написання програмного продукту за архітектурою Rest потрібно виконувати певний набір правил, який називається Restful. До цих правил відносяться:

- Client-Server – вся система має бути розбита на клієнтів та на сервера.
- Stateless – сервер не має права зберігати жодної інформації про клієнта.
- Cashe - кожен responce повинен бути помічений чи він кешований чи ні, адже клієнт може отримувати застарівші данні.
- Uniform Interface - єдиний набір правил, який виступає в формі інтерфейсу між клієнтом і сервером.

- Layered System – всю розроблену систему можна поділити на рівні так щоб компонент вищого рівня знав про компонент нищого рівня, але не навпаки.
- Code on Demand – забезпечує роботу кода на стороні клієнту.

Ruby дуже швидко адаптується до нових ринкових тенденцій в сфері розробки веб-додатків, відомим фактом є те, що фреймворк Ruby on Rails був з перших, хто реалізував Rest архітектуру для створення програмного забезпечення, принцип роботи даної архітектури зображений на рис. 2.5. До веб-фреймворків, окрім Rails відносяться: Nitro, Webby, Sinatra.

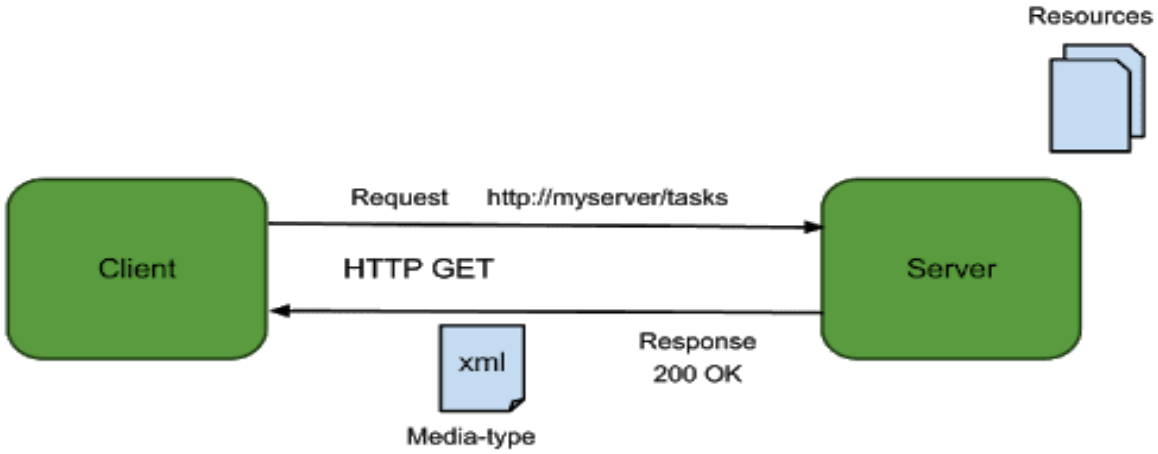


Рис 2.4. Принцип роботи Rest архітектури

Основні переваги використання Ruby on Rails:

- При необхідності підтримка AJAX
- Простота розробки додатків електронної комерції
- Підтримка JavaScript для покращення анімації
- Використання ODC, наявність унікальних конфігурацій змісту, що прискорює швидкість розробки продукту

2.3. Аналіз СУБД

2.3.1. PostgreSQL

PostgreSQL є найбільш популярною системою управління базами даних, які використовують сьогодні на ринку, вона є відкритою системою, представлена в форматі реляційної бази даних[12]. Це забезпечує

маштабованість та представлення технічних стандартів для відповідних систем.

При розробці доволі простих сайтів PostgreSQL використовується не так вже й часто, в порівнянні з MySQL чи MariaDB. Якщо розглядувати тенденцію використання при проектуванні складних систем PostgreSQL випереджає MySQL та MariaDB. Найчастіше з такими технологіями як Spring Framework, Ruby on Rails, Yii, Symfony, Django тощо використовують PostgreSQL для розробки.

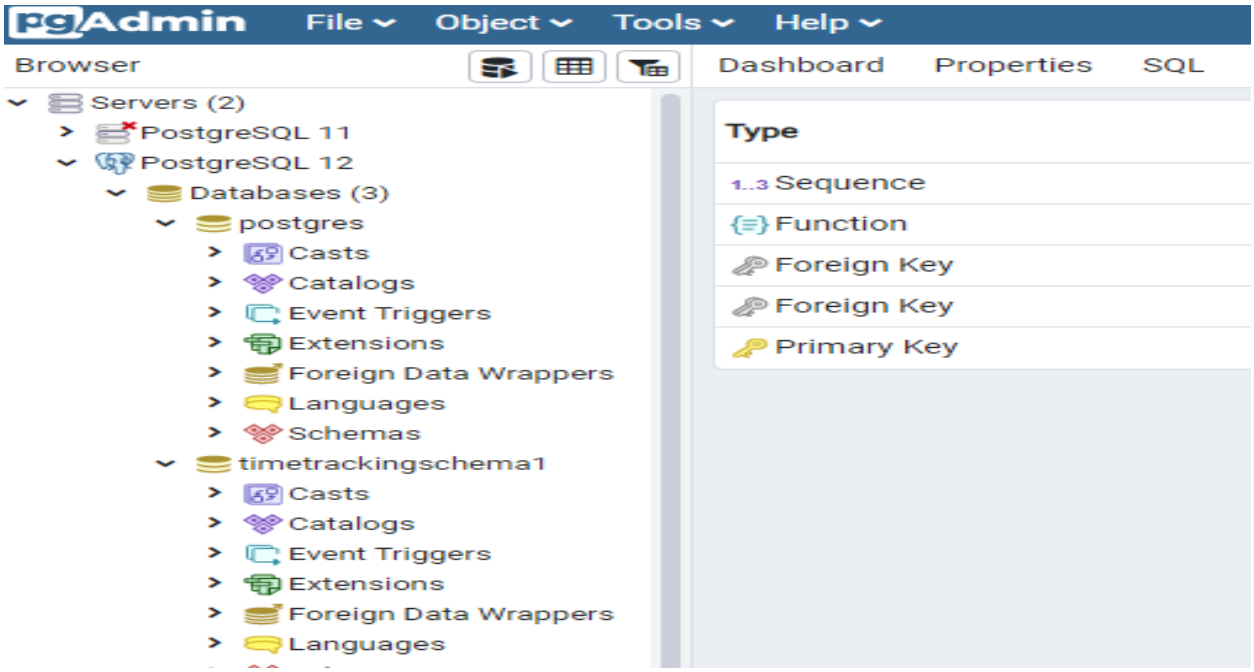


Рис 2.5. Головний екран PostgreSQL

Розгянемо особливості використання, PostgreSQL надає широкий функціонал таких інструментів як:

- PostgreSQL надає можливість використання функції – це блоки коду, які виконуються на сервері, а не на клієнті. Використання функцій можна забезпечити за допомогою використання різних мов програмування. Функції може використовувати як і розробник так і поточний користувач.
- Існує певний набір правил, що забезпечує механізм створення користувацьких обробників наприклад звичайних DML-операцій, операції вибірки тощо. Відмінність від тригерів полягає в тому, що ці

правила спрацьовують до самого процесу виконання, тобто на етапі розбору запиту.

- Тригери схожі на функції, але існує різниця. Наведемо приклад при використанні операція INSERT створюєм тригер, перевіряючий додану запис на певні умови. Самі ж тригери асоціюються з таблицями.
- Можливо одночасно модифікувати БД за допомогою використання механізму Multiversion Concurrency Control цей механізм підтримує роботу ACID і забезпечує неблокуюче читання.
- Є також можливість використовувати індекси таких типів як: R-дерево, B-дерево, хеш, GiST, GIN тощо. В нестадартних випадках є можлиість створення власних ідексів для вирішення сцецифічних проблем.
- Присутні багато різноманітних розширень, наприклад власні перетворення типів, типи даних, функції, індекси, оператори тощо.
- Присутній механізм спадкування. Він реалізований на рівні таблиць. Тобто таблиці успадковують характеристики і набори полів від батьківських таблиць. Дані, додані в таблицю-нащадок, автоматично будуть в запитах до батьківських таблиць.
- Дані відповідають ACID принципам.
- Стабільність релізів та оновлень при виявленні різноманітних багів.

Перейдемо до розгляду недоліків використання PostgreSQL:

- Перш за все слід знати, що PostgreSQL повільніший ніж MySQL навіть при використанні операцій читання чи запису.
- Складний для освоєння ніж аналоги.
- Відсутність підтримки стандартів ANSI SQL 92 та ANSI SQL 99.

2.3.2. Apache Cassandra

Cassandra – це високопродуктивна та найчастіше широко використовувана база даних NoSQL, головний екран даної бази даних зображений на рис. 2.6. Дана БД створена для обробки великих об’ємів інформації, вона відноситься до типу баз як ключ-значення[13]. Це означає,

що дані зберігаються як сукупність пар, де ключ виступає унікальним ідентифікатором, аналогом є принцип взаємодії звичайна HashMap. NoSQL повна назва Not Only SQL – це така база даних, яка відрізняється від реляційної тим, що механізм взаємодії з даними зовсім інший. Різниця в структурі даних дозволяє NoSQL працювати набагато скоріше.

Як вже згадувалось Cassandra працює з простором ключів, що в свою чергу відповідає термінології схемам баз даних в реляційної моделі. В просторі ключів можуть використовуватись кілька колонкових сімейств.

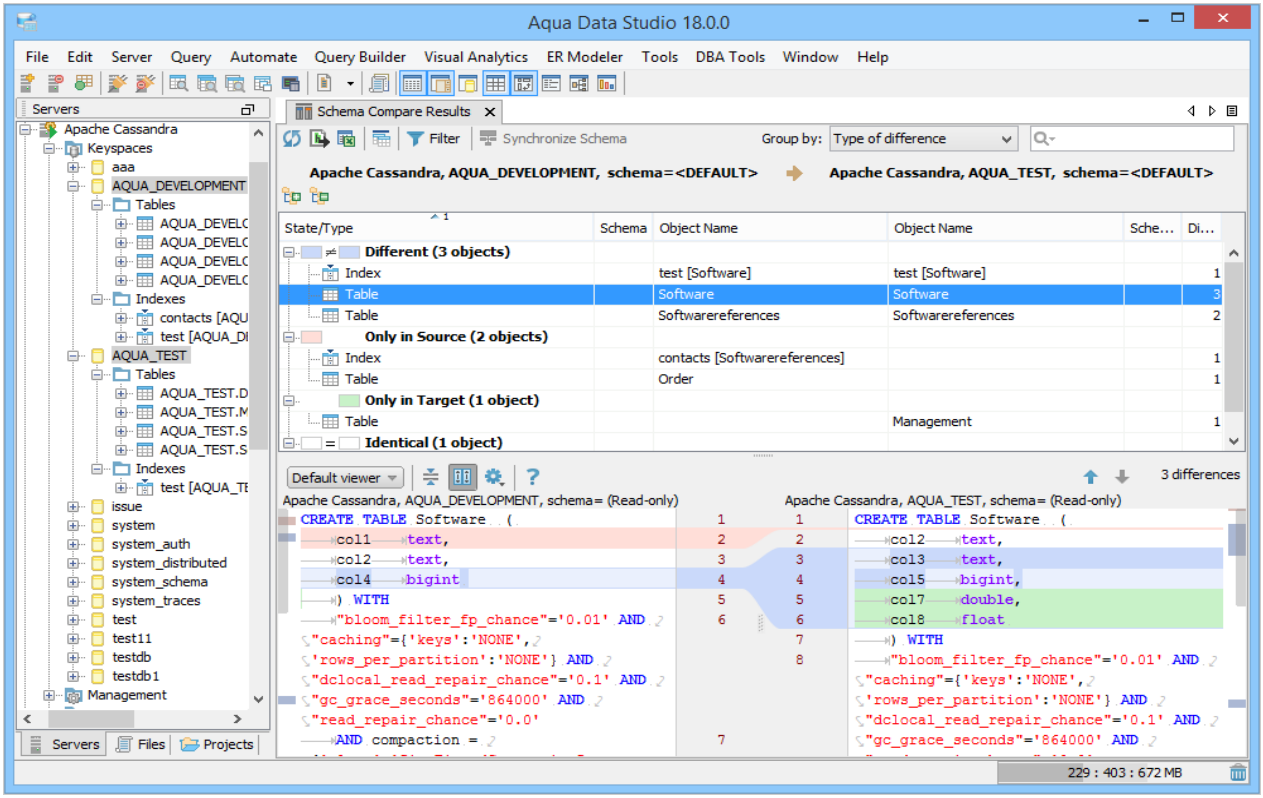


Рис. 2.6. Головний екран Apache Cassandra

Дана база даних побудована як децентралізована розподілена система. В свою чергу вона складається з декількох вузлів, за кожним з цих вузлів закріплені певні данні, які постійно змінюються, дана архітектура зображена на рис. 2.7.

Касандра не підтримує концепцію master/slave, як це реалізовано в СУБД, які використовуються в BigData, таких як Apache Hadoop, HBase, HDFS. Концепція master/slave полягає в тому, що один з серверів використовують для керування іншим компонентом кластера. Касандра в

свою чергу при розподілі елементів по вузлах використовує послідовне хешування. Відбувається застосування хеш-алгоритма для кожного елемента при обчисленні хеш-значень ключів чи то ім'я стовпця, ідентифікатору рядка тощо. Всі можливі хеш-значення формують діапазон і цей сформований простір ключів, розподіляється між вузлами самого кластера. Це означає, що кожному елементу даних призначений свій особистий вузол і цей вузол керує життєвим циклом елементів.

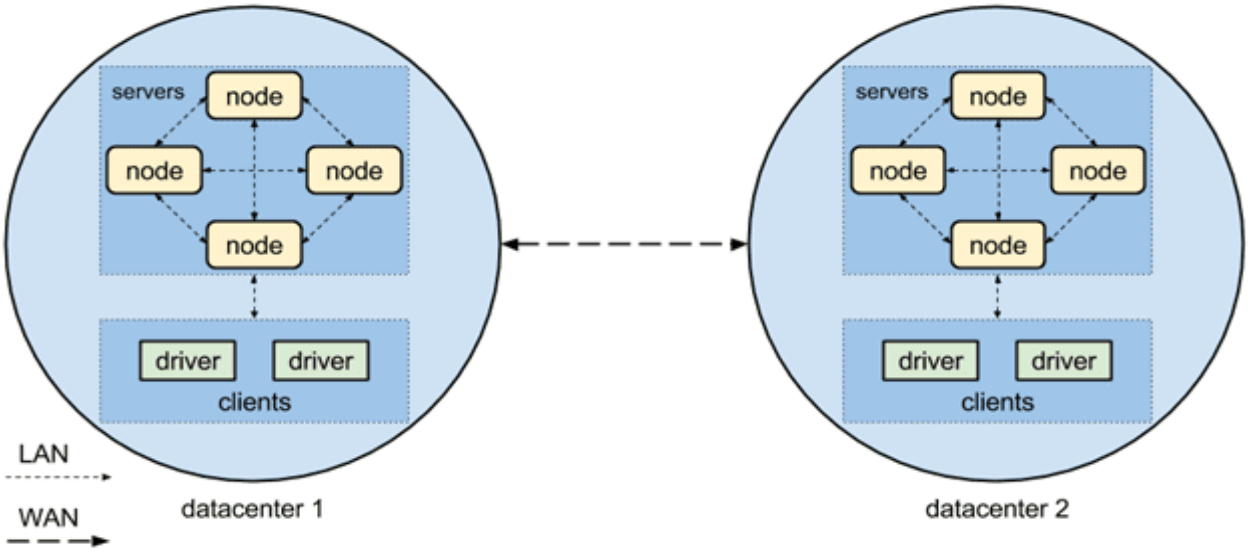


Рис. 2.7. Кластерна архітектура Apache Cassandra

Основні переваги, які виділяють розробники це:

- Надійність в плані безпеки
- Підтримка виконання транзакцій, які чітко слідують принципам ACID
- Дані розподіляються між вузлами самого кластера прозоро для користувача
- Гнучка масштабованість
- Висока швидкість роботи з даних та різноманіття формату даних

2.3.3. MongoDB

Також розглянемо інший тип баз даних і в даному випадку це буде документоорієнтована БД. MongoDB – це документоорієнтована система управління базами даних, яка є дуже популярною і широко використовуваною сьогодні на ринку. ЇЇ класифікують як NoSQL базу даних.

Це означає, що вона не потребує опису схеми таблиць. Взамін вона використовує JSON-подібні документи[14].

MongoDB дає можливість використовувати «ad-hoc»-запити. Дані запити дозволяють повертати конкретні поля документів, які необхідно та користувацькі функції, які написані на JavaScript. Дуже корисною функцією являється робота з репліками баз даних, тобто дані копії даних розміщені на різних вузлах. Дана функція забезпечує захист даних і захищає від втрати даних при виході одного з вузлів. Кожна репліка може бути в ролі основної, так і допоміжною. В цілому всі операції ведуться тільки з основною реплікою, в свою чергу допоміжні просто підтримують копії роблять запити на основну, щоб підтримати дані в актуальному стані, приклад такої взаємодії зображено на рис. 2.8. У випадку проблем чи збоїв автоматично обирається натупна репліка основною.

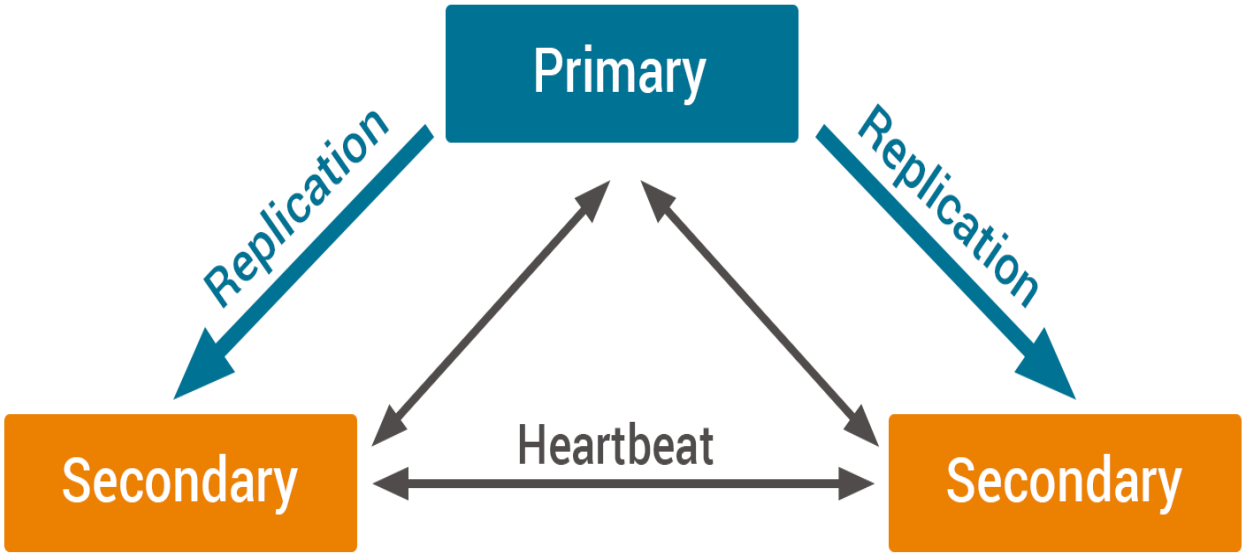


Рис. 2.8. Схема взаємодії реплік в MongoDB

Дана БД є горизонтально масштабованою системою. Вона побудована на техніці сегментації об’єктів бази даних. Все це означає, що відбувається розподілення її частин по різним вузлам кластеру, таке маштабування схематично зображено на рис. 2.9

Перше, що потрібно пам’ятати, що відразу коли створюється система, відразу обирається ключ сегментування. Даний ключ і визначає всі критерії,

за яким будуть розподілятись дані по вузлам. Використовуючи даний підхід, кожен вузол кластеру може приймати запити. Це й принцип і гарантує збалансоване навантаження на систему.

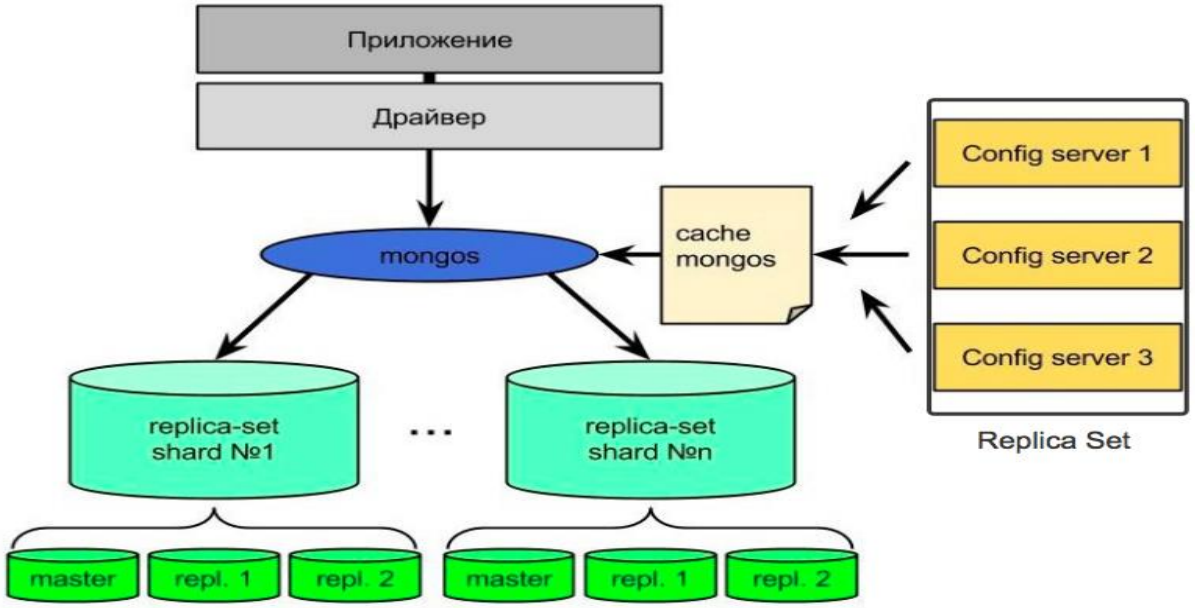


Рис.2.9 Схема горизонтального масштабування в MongoDB

MongoDB надає зовсім новий підхід до побудови баз даних. В цьому підході немає таблиць, схем, запитів SQL, зовнішніх ключів і багатьох інших речей, які ми так звикли бачити в об'єктно-реляційних БД. На відміну від реляційних баз даних MongoDB пропонує документо-орієнтовану модель даних. Документо-орієнтована модель працює набагато скоріше та має кращу масштабованість, а отже її й буде простіше використовувати. Навіть враховуючи всі недоліки традиційних баз даних і сильні сторони MongoDB, потрібно розуміти, що завдання бувають різні. І тому для вирішення кожної з проблем потрібно використовувати кращий підхід. Тобто в якійсь ситуації краще використати MongoDB, яке й справді збільшить продуктивність в ситуаціях в ситуації, коли ми зберігаєм складні за структурою дані. В інших ситуація краще використовувати об'єктно-реляційних БД.

2.4. Аналіз технологій для клієнтської частини

2.4.1. JavaScript

JavaScript – це високорівнева, багатопарадигмова мова, яка має здатність компілюватися під час роботи програми, найчастіше виконується у браузері. Блоки поділяються на класові, функціональні, умовні та інші обмежуються один від одного фігурними дужками. Мова широко розвивається відповідно до стандарту ECMAScript.

Разом з HTML (мова розмітки) та CSS (інструментом стилізації) мова програмування задає базу сучасного онлайн-простору, відповідно усі найсучасніші веб-браузери мають ядро для роботи JavaScript скриптів[15].

Дана мова - прототипно-орієнтованою, це означає, що вона не містила підтримку класів до 2015 року. Змінила ситуацію пізня версія стандарту ECMAScript , що надає підтримку для об'єктно-орієнтованого програмування, використовуючи функцію-клас.

JavaScript - має динамічну типізація мови. Дані програми можуть редагувати формат, а також використовуватися для різних цілей. Наявні найпопулярніші приведення типів: явна та неявна. Вони необхідні під час виконання операцій порівняння булевих значень, конкатенації строк та коректної інтерполяції рядків.

JavaScript надає зручний програмний інтерфейс, щоб мати доступ до DOM дерева веб-сторінки й BOM інтерфейсу веб-вікна і вкладок веб-браузера.

Використовуючи стандартні вбудовані можливості наявні методи для обробки наступних типів даних:

- Текст/рядки
- Числа та всі константи, які пов'язані з програмуваннями та математикою, наприклад, максимальне ціле позитивне число та число ПІ
- Звичайні масиви, хешмапи і хешнабори
- Прототипи об'єктів

- Функції. У даній мові функції є змінними, тому з функціями проводяться всі операції, що і проводяться з усіма типами
- Регулярні вирази. Вбудовано велику кількість функцій для їх використання: пошук, порівняння, тестування і так далі.

Для JavaScript створений NPM - глобальний менеджер пакетів, який надає змогу розробникам програмного забезпечення розроблювати та поширювати спільноті свої модулі, а іншим розробникам скачувати їх та добавляти у свої продукти. Всі сучасних додатки, написані JavaScript використовують NPM. При створенні пустого проекту додається документ package.json, який має всю детальну інформацію про додані в систему пакети. Це надає змогу ігнорувати модулі інших розробників до робочої системи контролю версій, наприклад Github, а тільки потрібний файл, пришвидшуючи роботу програміста. Всім стороннім розробникам потрібно використовуючи спеціальну команду встановити необхідні модулі з package.json на локальну систему.

Бандлери пришвидшують завантаження створених систем написаних на мові програмування JavaScript. Бандлери – це програми, які об’єднують розроблені модулі програмістів в єдиний скрипт, що пізніше підключається до веб-сторінки клієнта. Найвідомішим з наявних – це Webpack.

Створено велику кількість фреймворків та бібліотек JavaScript, які мають свої особливості і мету:

- Для front-розробки: Vue.js, React.js, Angular
- Для десктоп-додатків: Electron.js
- Для мобільних додатків: React Native та Ionic
- Для доступ до програм мобільних пристроїв: Cordova
- Для створення back -розробки: Express.js

Модель циклу подій, яка зображена на рис 2.10, допомагає динамічно обробляти тригери інтерфейсу, доступатися до системи файлів пристрою, відстежувати дії клієнта, отримувати відповіді з сторони back-розробки і так

далі. Вона використовує у своїй роботі чергу подій. Для роботи JavaScript зазвичай використовує 4 потоки (при необхідності можна змінити), дана мова програмування делегує потокам високозатратні задачі, такі як, робота з системою файлів, дерева DOM, БД, запитів до back-серверу і тому подібні операції, а скрипт JS в цей час продовжує виконання програми.

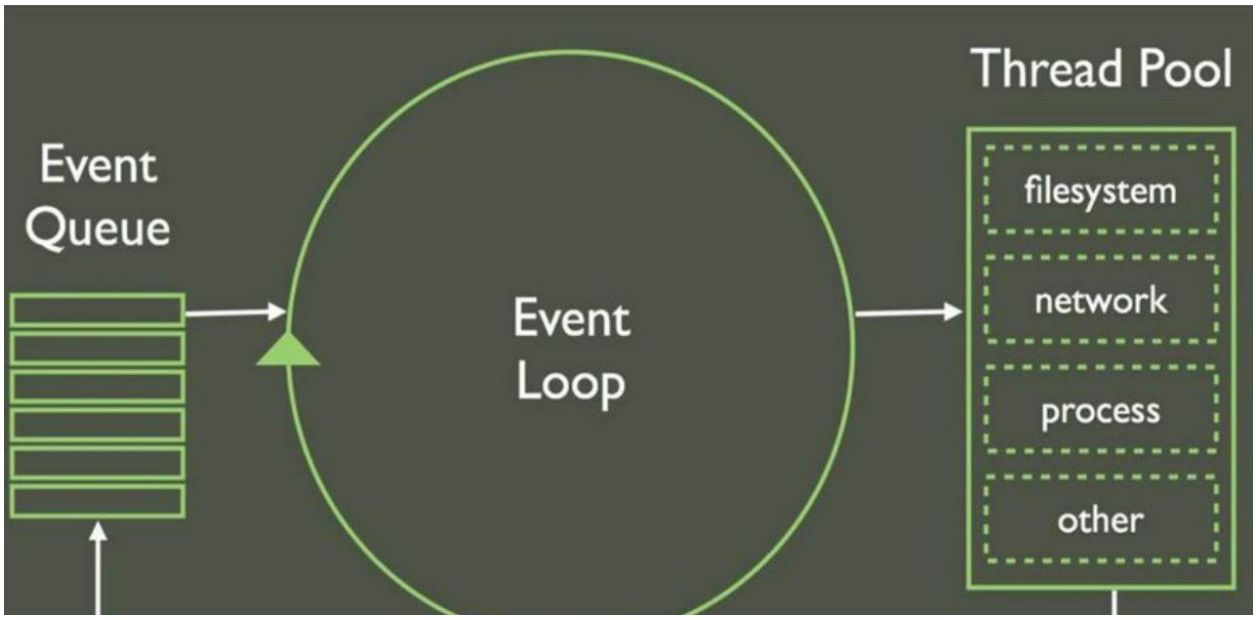


Рис. 2.10. Схема роботи подій JS

2.4.2. React.js

React.js – це бібліотека, яка створена компанією Facebook разом з групою програмістів та організацій, які надає можливості для веб-розробки збоку клієнтської частини. Багато програмістів у зв'язку з великою функціональністю бібліотеки вважають її фреймворком. Найчастіше використовується для створення багатосторінкових веб-сайтів, а також мобільних додатків.

Характерною рисою React.js є робота з тіньовим деревом DOM[16], використовуючи хешовану копію структури браузера. React необхідний в ньому для висвітлення змін між теперішнім та наступним станом графічного інтерфейсу. Використовуючи даний підхід, веб-браузер оновлює тільки необхідну частину дерева DOM, попри те, що програмно зображено цілком нову веб-сторінку, що забезпечує швидкодію системи.

Схема зв'язування даних зображена на рис. 2.11.

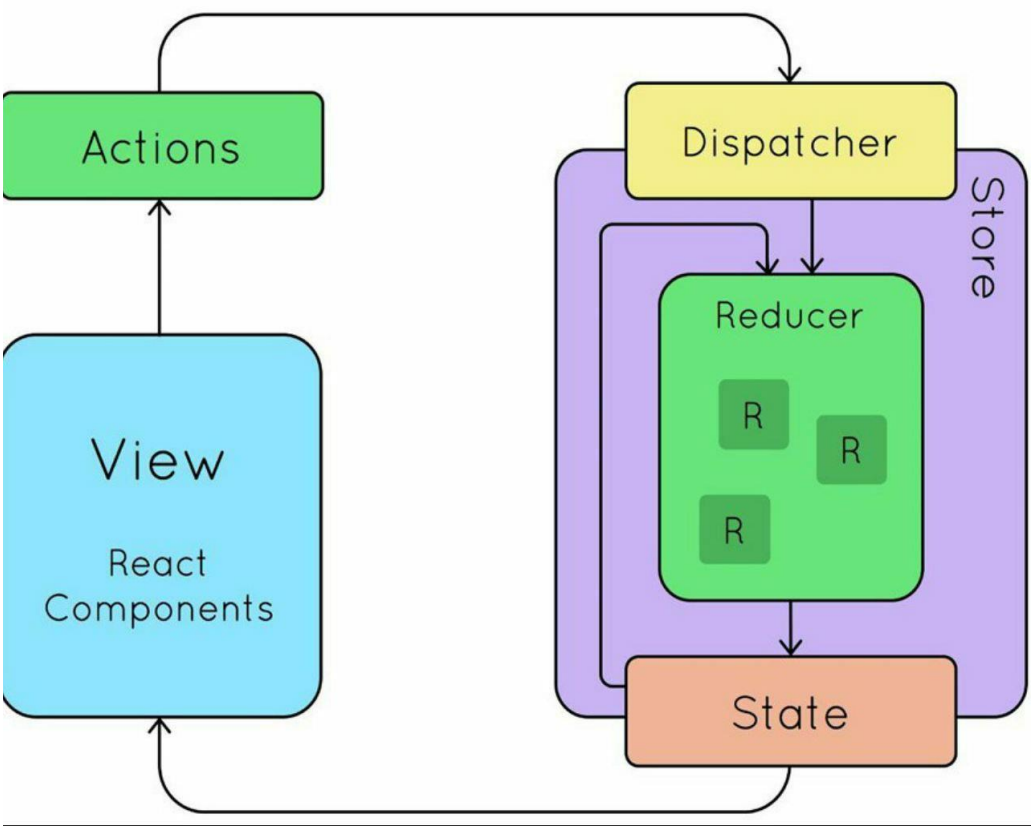


Рис. 2.11. Схема зв’язування даних у React

Бібліотека надає розробнику прикладний інтерфейс контексту, щоб керувати всіма даними юзера в клієнтській частині продукту, але для масштабних систем необхідні зовнішні модулі. Найчастіше для цього використовується фреймворк Redux із обробниками даних проміжкового типу.

Для створення компонентів React працює з засобом JavaScript XML – мова, яка працює з синтаксисом подібним до мови розмітки HTML, щоб створити користувацький інтерфейс.

Компоненти бібліотеки використовують локальні та глобальні дані. Локальні створюються для даної частини компоненту. Глобальні передаються із компонента вищого порядку. React має життєвий цикл, який складається з таких елементів:

- Render. Це відображення частини користувацького інтерфейсу, який викликається при зміні даних, що використовують даний елемент.

- ComponentDidMount. Це метод, що викликається, коли елемент поміщається у дерево DOM веб-сторінки браузера, зазвичай в ньому виконуються запити користувача до back-сервера для завантаження компонентів для представлення
- ComponentWillUnmount. Це метод, що викликається перед тим, як елементу буде видалений з дерева DOM
- ShouldComponentUpdate. Це метод, який викликається перед рендерингом компоненту. Даний метод використовується також для відміни відображення елементу інтерфейсу, який повторюється, повернувши false.

Життєві цикли React мають значне більше методів, від переліку, який вказано, але інші методи є специфічними і тому мають менший спектр застосування. Деякі з них: підписка компонентами на оновлення інших частин додатку, надання значень певним змінним компонентів інтерфейсу.

2.4.3. JSP

JSP створена організацією Sun Microsystems і стала доповненням для Java Servlets. JSP збільшує швидкодію та простоту створення веб-додатків, використовуючи шаблони програмування.

Щоб розуміти переваги та архітектуру JSP потрібно розуміти Java Servlets, тому що вони пов’язані між собою. JSP - це шаблони веб-сторінок мови розмітки HTML. Відмінністю від схожих технологій в тому, що програмний код всередині тегів не інтерпретується при запиті під до веб-сторінки, а компілюється в Servlet[17].

Програмний код компілюється схоже на те, наче код написаний всередині Servlet. JSP компілює сторінки у Servlet є дорогою операцією, проте створюється одноразово або при першому запиті до веб-сторінки, або при створенні контейнера сервлетів.

Архітектура використання JSP зображена на рис 2.12

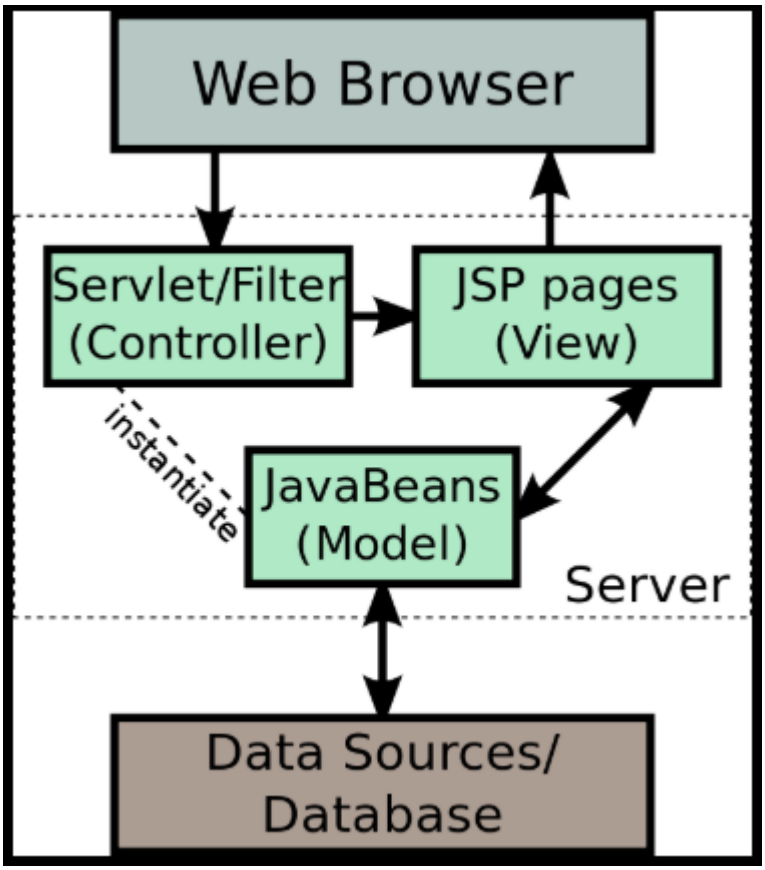


Рис.2.12. Архітектура JSP моделі

JSP поєднує використання шаблонів проектування для створення веб-сайтів, і також всі переваги мови Java, тому дана технологія поширилася в усіх сферах: і при розробці комерційних, так і для безкоштовних веб-проектів.

Бібліотека тегів інтегрують звичайні, власні або сторонні у веб-сторінки. Оскільки бібліотека тегів забезпечує легке використання та створення компоненти надала популярності.

JSP не підв’язана до програмної платформи, отже технологія є гарним засобом для створення додатків у гетерогенних середовищах.

Продуктивність JSP сторінок є обмеженою даними особливостями архітектури:

- веб-сторінки компілюються в Servlet, що є потребує багато часу
- Servlet виконується лише в програмному середовищі Java, тому тільки в режимі інтерпретації.

Проте дані обмеження перекриваються додатковими особливостями. Сучасні веб-контейнери мають кластеризацію веб-серверів, тому дане навантаження виконується на стороні апаратного забезпечення, що є виправданим рішенням, як і економічно, так і простотою. Оскільки, компіляція в Servlet виконується лише раз, то це не відчувається для продуктивності веб-системи за довгий період.

Основними перевагами JSP є:

- Простота створення програмного коду
- Легке застосування великої кількості сторонніх бібліотек
- Різноманітні програмні середовища розробки коду

Звертаючи увагу на дані фактори, JSP технологія найперспективніша базова технологія для розробки клієнтської частини веб-додатку.

Проте при розробці складних веб-систем обмеження, які існують при використані JSP є значною проблемою для розвитку даної технології.

ВИСНОВКИ ДО РОЗДІЛУ 2

У розділі 2 було розглянуто основні сучасні інструменти та технології для розробки веб-додатків. Досліджені вихідні дані систематизовані й детально проаналізовані з ціллю їх використання у подальшій бакалаврській роботі визначеного набору технологій та інструментів, що будуть присутніми для створення онлайн системи бронювання заходів спортивного комплексу.

Розглянуті і дослідженні веб-технології у даній бакалаврській роботі:

- Для розробки серверної частини: Java, PHP, Ruby
- Для створення клієнтської частини: JS, ReactJS, JSP
- Системи управління базами даних: PostgreSQL, Cassandra, MongoDB

На основі досліджень даного розділу обираємо JavaScript для розробки клієнтської частини, оскільки має дані переваги: стабільна технологія розробки, легкість створення та підтримки програмного коду для немасштабних проектів, широкий спектр можливостей з використання зовнішніх модулів, багатофункціональність, якісна реалізація багатопоточності, інтеграція з HTML/CSS.

В якості СУБД було обрано PostgreSQL, оскільки – дана СУБД працює в реальному часі, має доступ до даних сервісу з будь-якого пристрою, славиться своєю стабільністю та надійністю роботи, працює з усіма необхідними типами даних та легко інтегрується з вибраними в даній бакалаврській роботі технологіями

Мову програмування Java, та зокрема фреймворк Spring було обрано для реалізації серверної частини, оскільки дані технології мають ряд переваг: ідеально пристосовані для створення веб-додатків, маючи для цього необхідні модулі для розробки (Spring Security, Spring Data, Spring Boot і тд), кросплатформеність, паралельність, швидкість розробки та доступна підтримка програмного забезпечення.

РОЗДІЛ 3
СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Вимоги до розроблюваного веб-додатку

Проаналізувавши наявні аналоги веб-застосунків систем з функціональністю бронювання заходів у першому розділі були сформовані вимоги до усіх можливих користувачів.

Вимоги до гостьового користувача:

- Можливість пройти аутентифікацію
- Після аутентифікації проходження процесу авторизації

Вимоги до звичайного користувача:

- Календар бронювань з закритими даними користувачів.
- Можливість додання, відміни та редагування власного замовлення.
- Отримання e-mail-листа на електрону пошту при відміні та зміні замовлення адміністратором.
- Отримання e-mail-листа на електрону пошту при додавані, відміні та зміні замовлення особисто.
- Перегляд усіх власних замовлень.
- Зміна даних власного облікового запису.

Вимоги до адміністратора:

- Календар з всією інформацією резервування.
- Можливість відміни та редагування замовлення інших користувачів.
- Надсилання e-mail користувачу при відміні та зміні замовлення адміністратором.
- Можливість додання нового користувача.
- Можливість додання нового стадіону.
- Можливість додання нового поля.

- Можливість додання нового університету.
- Можливість редагування даних існуючого користувача.
- Можливість редагування даних існуючого стадіону.
- Можливість редагування даних існуючого поля.
- Можливість редагування даних існуючого університету.
- Перегляд архіву всіх замовлень користувачів.

3.2.Архітектура MVC веб-застосунку

MVC (Model, View, Controller) – це паттерн проектування архітектури, що пов’язаний з глобальним проектуванням веб-додатку, що беруть за базі створенні програмного забезпечення. Мета даної архітектури - це створення системи розділивши її на три основні компоненти[18], які пов’язані один з одним, що зображено на рис 3.1:

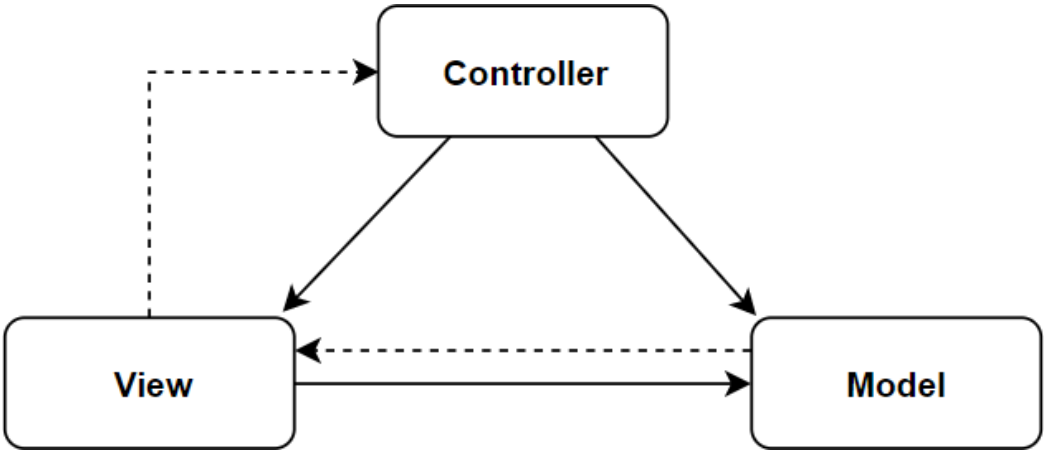


Рис.3.1. Зв'язок компонентів архітектури MVC

- Це Model – це модель, яка відповідає за дані програми
- View – представлення, зазвичай графічний інтерфейс для взаємодії з користувачем, для отримання від них запитів
- Controller – це компонент, який відповідає за правильну роботи процесу програми, а саме приймає запит від View та повертає відповідь, перед цим передавши це Model.

Найчастіше даний архітектурний паттерн створює веб-застосунки з користувацьким графічним інтерфейсом. Java, C#, Ruby, Python та інші мови програмування використовують архітектуру Model-View-Controller для продуктивного створення програмних додатків.

Суть архітектури – розробка масштабованого розширюваного веб-застосунку, та легкій функціонала та клієнтської частини. Додатковою особливістю є повторне використання коду, що забезпечує продуктивність та зменшує кількість помилок, як програмних, так і програміста. Одне з найголовніших, це відокремлення серверної частини від користувацької що забезпечує легкість, простоту та швидку розробку. Величезні системи, використовуючи архітектуру MVC, стають більш зрозумілі та краще підтримуються в подальшому.

Model – це компонент архітектури MVC, що має в собі бізнес-логіку веб-системи. Модель не залежить від інших компонентів шаблону проектування. Дані у Model редагуються, проте представлення моделей залишається незмінним. Основні характеристики моделі:

- Інкапсуляція всіх даних програми.
- Виконання бізнес-логіки веб-застосунку.
- Виконується робота з базою даних

View – це компонент архітектури MVC, що відображає дані користувачеві веб-застосунку, а також від них отримує запити та надсилає їх на Controller. View ніяким чином не впливає на Model, а лише отримує дані та створює їх відображення. Найпопулярніше представлення – це веб-сторінка написана мовою розмітки HTML.

Controller – це компонент архітектури MVC, що керує програмним застосунком, а саме отримує запит від View, перетворює у команди та передає Service Model, де виконується бізнес-логіка і повертається результат до Controller, який повертає ModelAndView. Controller відстежує дії клієнта на веб-сайті – переміщення курсора, ввід користувацьких даних, натискання

кнопок та інші. Далі отримавши дію, він надає запит до Model або до View для виконання операції клієнта.

Крім того наявні і похідні архітектури :

- MVP (Model, View, Presenter) – найчастіше використовується, при неможливості логічного поєднання даних. Model та View ті ж самі, а Presenter являє собою інтерфейс контролера, що забезпечує легку його підміну. Це забезпечує кросплатформеність, розширення та полегшує тестування. Явним прикладом є Windows Form.
- MVVM (Model, View, View, Model) – використовується у програмних рішеннях, основане на тому, інтерфейси представлення дуже тісно повязані з моделю.

У даній програмі використовується шаблон проектування MVC, що зображено на рис 3.2.

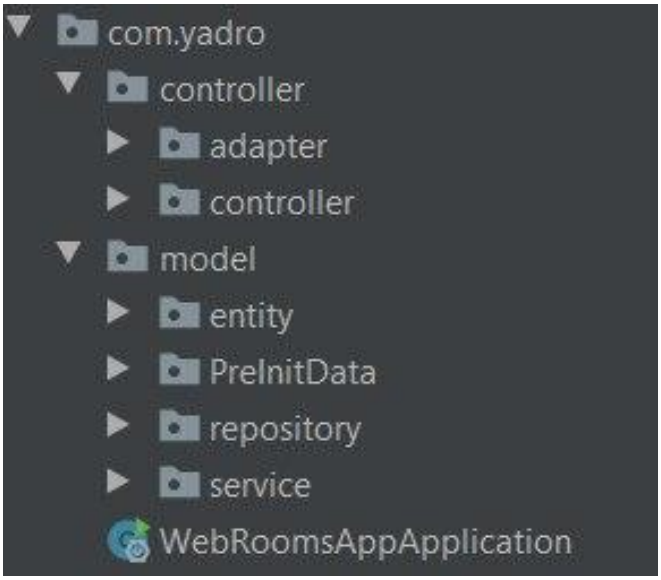


Рис. 3.2. Структурна схема програмного забезпечення

3.3.Список User Stories

Перед тим, як перейти до User Stories, необхідно виділити три ролі: неавторизований користувач, авторизований користувач та адміністратор, які зображені на рис 3.3. А також виділити три основні права: View – право

перегляду, Access – мати доступ до даних, Create, Edit, Delete, Update – мати доступ до створення, редагування, видалення певних даних

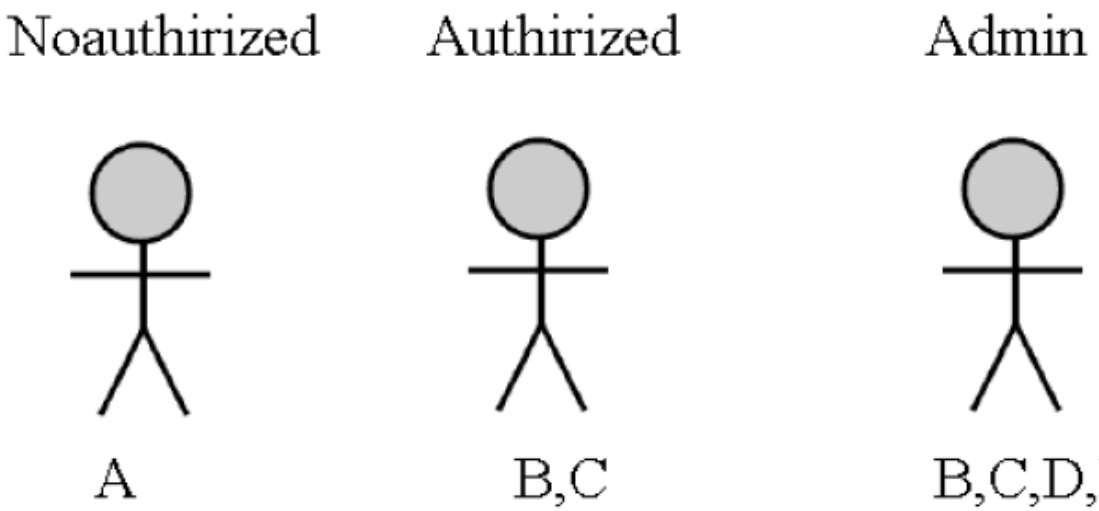


Рис. 3.3. Ролі користувачів у веб-застосунку

Неавторизований користувач (далі як НК):

1.View…

1.1 НК має право переглянути стартову сторінку

1.2. НК має право переглянути сторінку для аутентифікації

2.Access…

2.1 НК має доступ для введення своїх особистих даних (логіну або поштової скриньки та паролю) для входження в акаунт

2.2 НК має право зберегти пароль та логін для наступного входу

2.3 НК має право зберегти акаунт для наступної сесії

3. Create, Edit, Delete, Update…

3.1 НК має можливість на приховання особистого паролю

3.2 НК має можливість перегляду незашифрованого особистого паролю

Авторизований користувач (далі як АК):

1.View…

1.1 АК має право переглянути привітальну сторінку для авторизованих користувачів

1.2 АК має право переглянути свої особисті дані

1.3 АК має право переглянути можливі стадіони для бронювання та їх характеристики

1.4 АК має право переглянути можливі поля для бронювання та їх характеристики

1.5 АК має право переглянути можливі поля для бронювання та їх характеристики

1.6 АК має право переглянути календар бронювання, їхню назву події, час, але крім особистих даних користувачів

1.7 АК має право переглянути історію свого бронювання

1.8 АК має право дізнатися точний час в даний момент

2.Access…

2.1 АК має доступ до привітальної сторінки для авторизованих користувачів

2.2 АК має доступ до своїх особистих даних

2.3 АК має доступ до календаря бронювання поля тільки для своїх замовлень

2.4 АК має доступ до історії свого бронювання

3.Create, Edit, Delete, Update…

3.1 АК має можливість забронювати поле на зручний йому час

3.2 АК має можливість змінити час свого бронювання

3.3 АК має можливість видалити своє бронювання

3.4 АК має можливість редагувати назву свого бронювання

3.5 АК має можливість редагувати свої дані:

3.5.1 Email

3.5.2 Address

3.5.3 Group

3.5.4 Name

3.5.5 Surname

3.5.6 Password

3.6 АК отримує електронний лист на свою пошту при:

3.6.1 При бронюванні поля

3.6.2 При редагуванні даних бронювання поля

3.6.3 При видаленні бронювання поля

3.6.4 Дії 3.6.2-3.6.3 збоку адміна над бронюванням АК

3.7 АК має можливість вийти з свого акаунта

Адмін:

1.View…

1.1 Адмін має аналогічні до прав користувача у View 1.1-1.8

1.2 Адмін має право переглянути список всіх користувачів

1.3 Адмін має право переглянути дані кожного з користувачів

1.4 Адмін має право переглянути список всіх стадіонів

1.5 Адмін має право переглянути дані кожного з стадіонів

1.6 Адмін має право переглянути список всіх полів

1.7 Адмін має право переглянути дані кожного з полів

1.8 Адмін має право переглянути всі бронювання усіх користувачів

2.Access…

2.1 Адмін має доступ до привітальної сторінки для авторизованих користувачів

2.2 Адмін має доступ до своїх особистих даних

2.3 Адмін має доступ до календаря бронювання поля, до всіх бронювань, як своїх, так і користувацьких

2.4 Адмін має доступ до історії свого та всіх бронювань

2.5 Адмін має доступ до списку та редагування:

2.5.1 Користувачів

2.5.2 Стадіонів

2.5.3 Полів

3.Create, Edit, Delete, Update…

3.1 Адмін має право виконати аналогічні пункти 3.1 - 3.7 користувача Create, Edit, Delete, Update…

3.2 Адмін має можливість створити, видалити, редагувати та зберегти списки таких об’єктів, як:

3.2.1 Поле

3.2.2 Стадіон

3.2.3 Університет

3.4.Beans створеного веб-застосунку

Bean - це POJO-класи Java[19] (це класи, які не містять логіки програми, а відповідають лише за створення та обробку об’єкта), які можна використовувати багато раз у своєму коді, що прискорює процес створення веб-додатку. Beans призвели до створення сучасного типу програмування - збірки додатків з бінів/компонентів, при цьому програмісту необхідно знати лише інтерфейси, тому що реалізація повинна бути легко розширювана.

JavaBean[20] - це POJO-об'єкти, які інкапсулюють в об'єкті програмний код, об’єкти та інші дані. JavaBean має методи відкриті для доступу віддаленого.

У веб-застосунку існує п’ять bean, які виступають моделями і забезпечують логіку програми:

1. User – це користувач веб-застосунку, який надсилає запити для дій з бронювань заходів спортивного комплексу
2. University – це університет, в якому присутній стадіон та в якому навчаються студенти, які бронюють поле
3. Stadium – це стадіон, який міститься поля для бронювання
4. Pitch – це поле (половина стадіону), яке може бронювати
5. Event – це бронювання заходу спортивного комплексу

3.5. Controller

Реалізація контролера в коді виступає клас, який призначений для обробки запитів від клієнта і повернення відповідних результатів. В свою чергу сам контролер найчастіше робить виклики сервісних методів в потрібному порядку і повертає потрібні результати клієнту. Не потрібно плутатити з бізнес-логікою, логікою обробки даних. Для дотримання бізнес-логіки створюються додаткові класи, які обробляють дані.

При тестуванні методів клієнта найчастіше використовують POstMan, за допомогою якого здійснюють запити на контролер

Присутні дані контролери в програмному забезпеченні: UserController, EventController, PitchController, StadiumController, UniversityController, які відповідають за обробку даних описаних раніше моделей.

На рис 3.4 зображено структуру схеми програми у веб-застосунку і можна побачити, яку роль відіграє Controller, Service, Repository та інші компоненти.

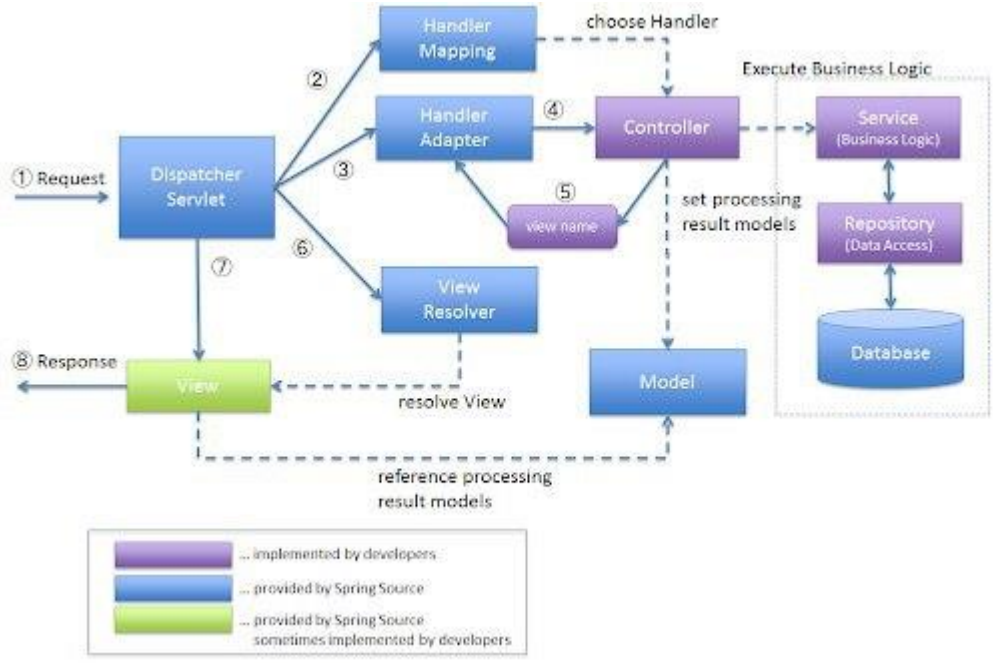


Рис. 3.4. Структура схеми програми у веб-застосунку

3.6.Service та Repository програмного забезпечення

Сервіс виконує чітко описану логіку: валідацію даних, обробке різноманітних помилок, отримання моделей, для цього він звертається до бази даних за допомогою використання репозиторіїв.

Реалізацією сервісу виступає клас, який помічений анотацією @Service. Даний клас містить в собі майже всю бізнес логіку проекта.

Шар, який відповідає за взаємодію з даними , які зберігаються в базі даних називається репозиторій. В коді він реалізований як клас , який позначений анотацією @Repository . Дана анотація означає, що клас функціонує як репозиторій.

Присутні такі сервіси, репозиторії в даному програмному забезпеченні:

- UserService – сервіс, який оброблює бізнес-логіку клієнта
- EventService – сервіс, який оброблює бізнес-логіку бронювання заходів спортивного комплексу
- PitchService – сервіс, який оброблює бізнес-логіку полів
- MailService – сервіс, який оброблює бізнес-логіку надсилання електроних листів при дій з бронюваннями
- StadiumService – сервіс, який оброблює бізнес-логіку стадіонів
- UniversityService – сервіс, оброблює бізнес-логіку університетів
- UserRepository – це репозиторій, що звертається до бази даних для роботи з моделю користувача
- EventRepository – це репозиторій, що звертається до бази даних для роботи з моделю бронювання
- UniversityRepository – це репозиторій, що звертається до бази даних для роботи з моделю університета
- PitchRepository – це репозиторій, що звертається до бази даних для роботи з моделю поля
- StadiumRepository – це репозиторій, що звертається до бази даних для роботи з моделю стадіона

ВИСНОВКИ ДО РОЗДІЛУ 3

Проаналізувавши у другому розділі наявні аналоги веб-додатків з подібною функціональністю бронювання заходів, у третьому розділі було створено вимоги для розробки програмного забезпечення. Складені критерії для кожного типу користувача — невідомого (неавторизованого) користувача, авторизованого користувача та користувача з правами адміністратора. Дані вимоги були реалізовано у веб-застосунку та успішно протестовані.

При написанні даної бакалаврської роботи було обрано реалізацію клієнт-серверної архітектури, а саме Model-View-Controller, адже вона має таку низку переваг як: швидка та паралельна обробка запитів сервером від користувачів, розподілення системи на клієнтську та серверну частини та можливість добавити додатковий рівень безпеки.

Проаналізовано компоненти серверної частини клієнт-серверної архітектури даного програмного забезпечення. Було описано усі об’єкти, які застосовуються у веб-додатку, а саме: контролери, сервіси, репозиторії та сутності.

Для реалізації серверної частини програмного забезпечення даної бакалаврської роботи використовував фреймворк Spring мови програмування Java. JavaScript, CSS, HTML було обрано для створення графічного інтерфейсу. Для зберігання даних веб-додатку була задіяна PostgreSQL.

РОЗДІЛ 4
АНАЛІЗ ТА ПЕРЕВІРКА ЕКСПЛАУТАЦІЇ ВЕБ-ЗАСТОСУНКУ

4.1.Інструкція користувача та демонстрація проекта

У даній бакалаврській роботі результатом є веб-система бронювання заходів з зручним та зрозумілим графічним інтерфейсом, тому для наглядності зображення з демонстрацією даного застосунка буде додано до відповідних пунктів інструкції користувача.

Перед запуском програмного додатку з боку адміністратора потрібно переконатися, що усі вимоги до апаратного та програмного забезпечення були виконані.

Наразі веб-додаток підтримує три ролі користувачів з різними правами, доступом та функціональністю: невідомий (неавторизований) користувач, авторизований користувач та адміністратор. Для кожної з ролей буде описано інструкцію та для кращого візуального сприйняття будуть прикріплені скріншоти системи.

4.1.1. Інструкція для невідомого користувача:

1.Як невідомий користувач, я можу ввести логін, щоб увійти в систему

2.ЯК невідомий користувач, я можу приховати свій пароль, щоб його не побачили зловмисники.

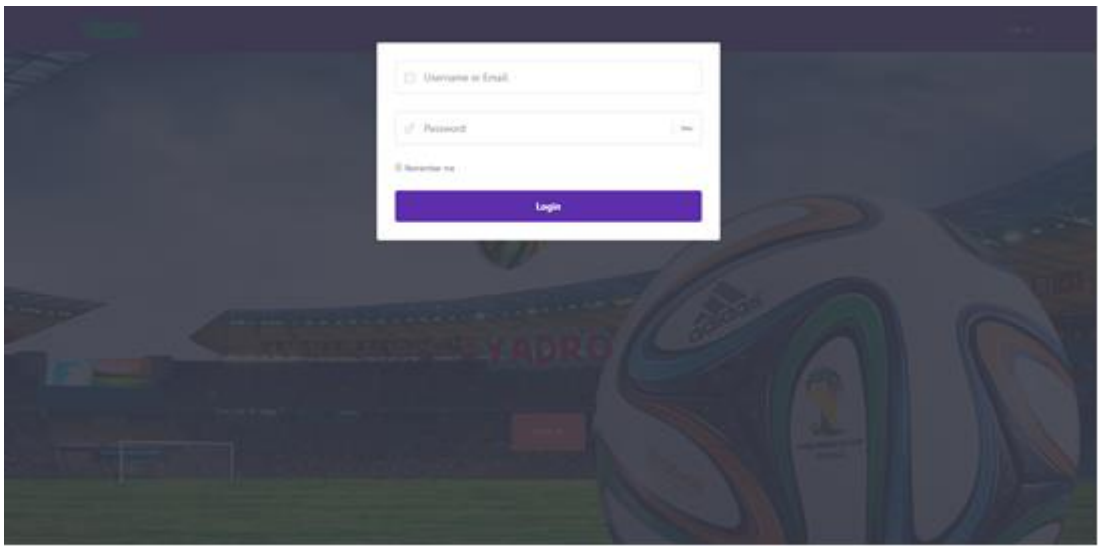


Рис. 4.1. Форма авторизації користувача

4.1.2. Інструкція для авторизованого користувача:

1. Як авторизований користувач я можу перегляну інформацію, яка прикріплена на головній сторінці веб-застосунка.

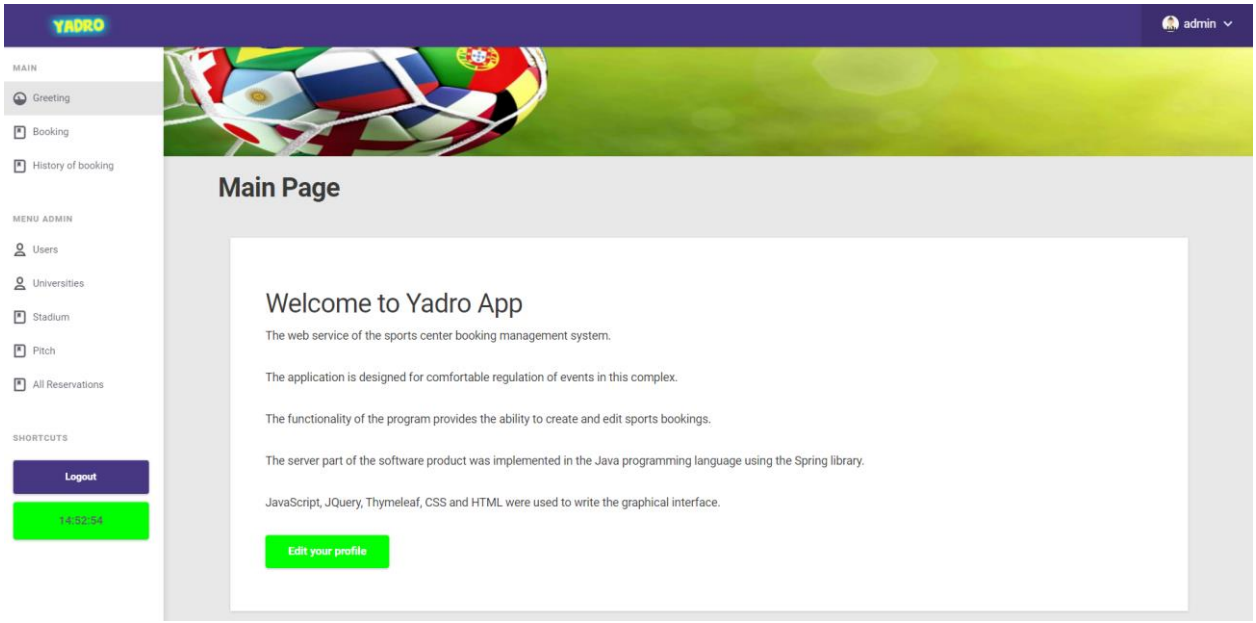


Рис. 4.2. Головна сторінка сайту

2. Як авторизований користувач я можу перейти в свій особистий кабінет, щоб змінити свої дані.

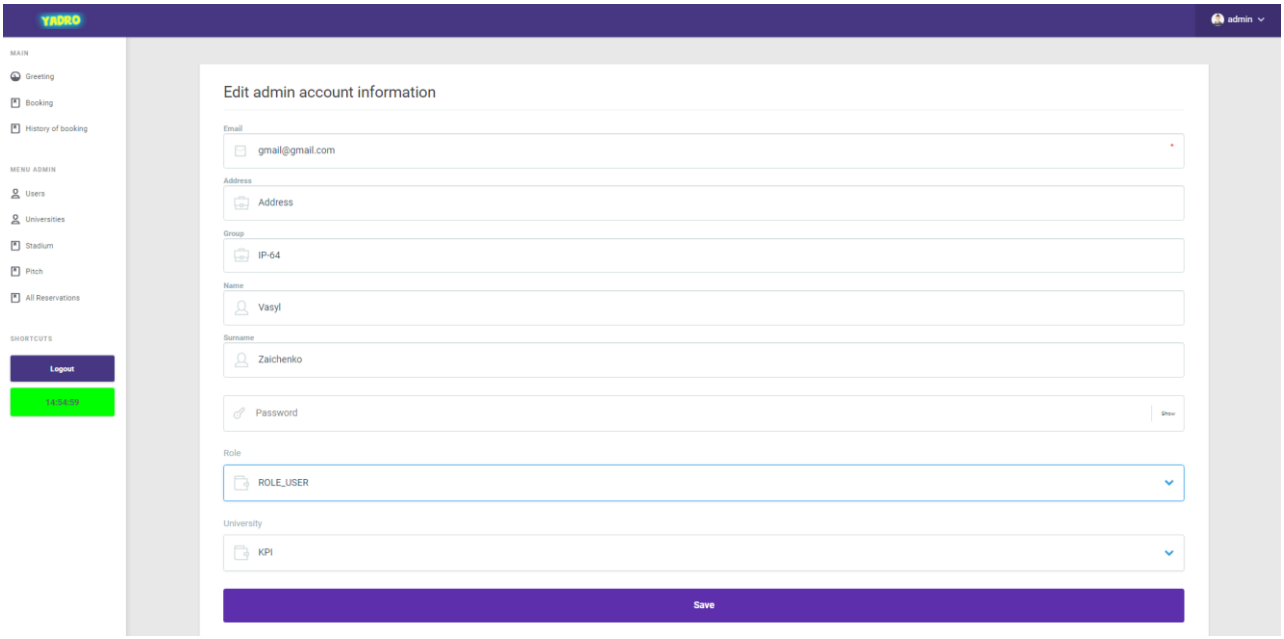


Рис. 4.3. Особистий кабінет користувача

3. Як авторизований користувач я можу перейти на сторінку університету, щоб побачити можливі стадіони.

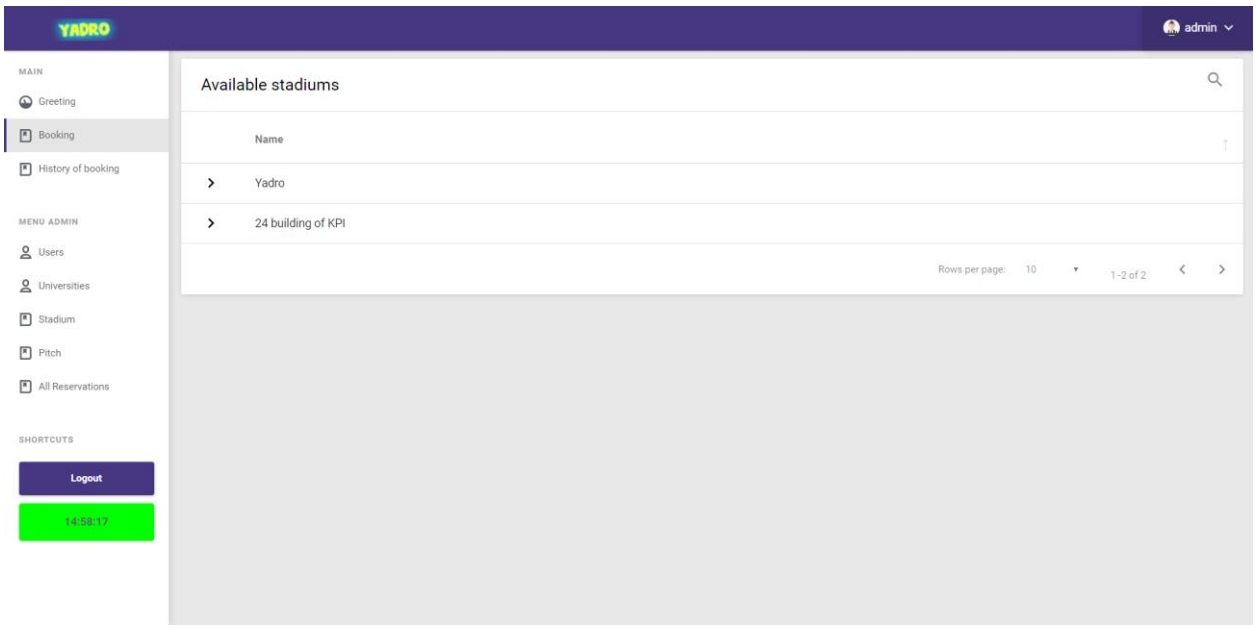


Рис. 4.4. Сторінка університету з наявними стадіонами

4. Як авторизований користувач я можу перейти на сторінку обраного стадіону, щоб побачити можливі футбольні поля та їх характеристики.

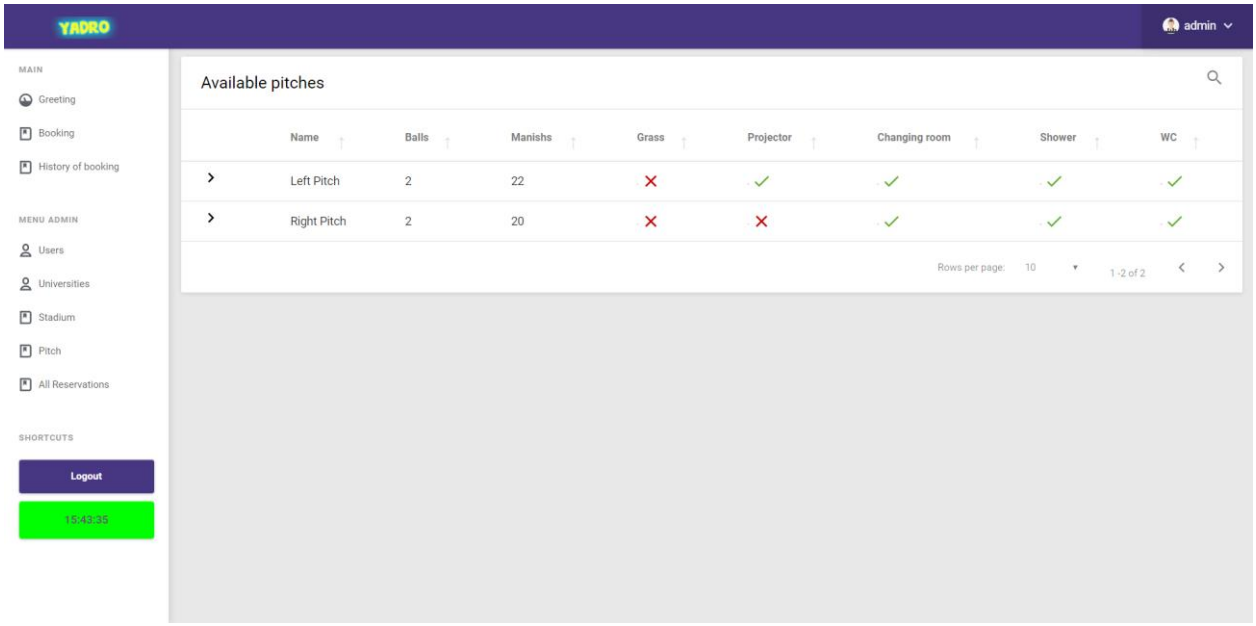


Рис. 4.4. Сторінка стадіону з його характеристиками

5. Як авторизований користувач я можу обрати футбольне поле, щоб перейти до календарю бронювання

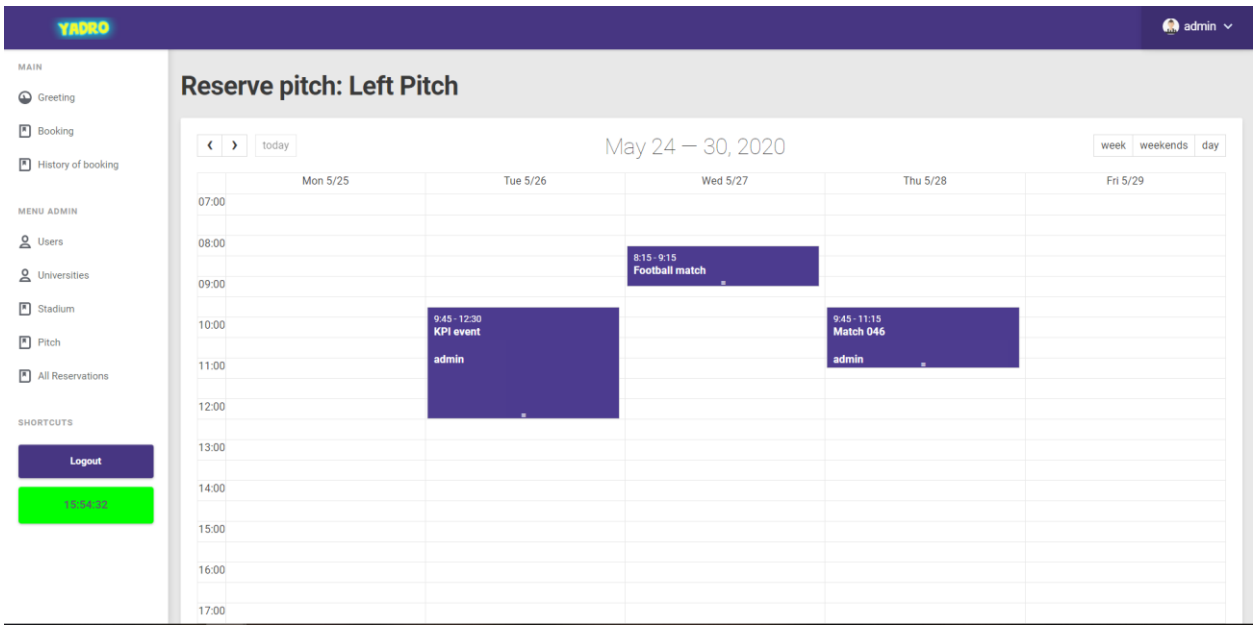


Рис. 4.5. Сторінка з календарем бронювань

6. Як авторизований користувач я можу перейти до календаря бронювань, щоб:

6.1 Додати собі бронювання поля на обраний час

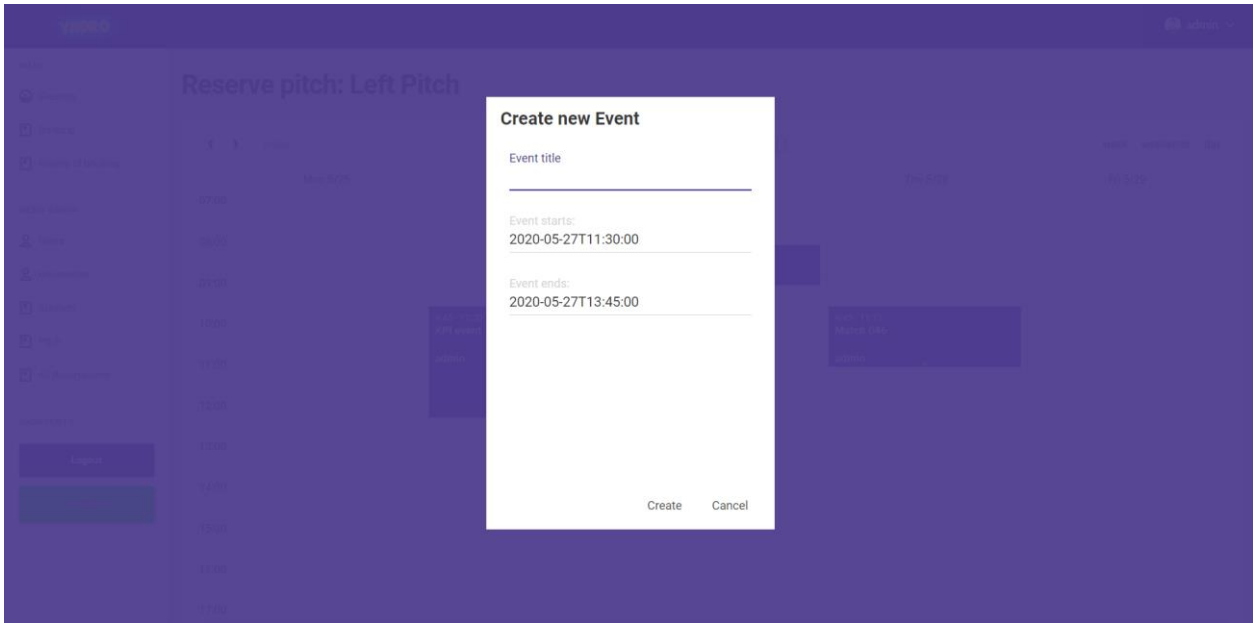


Рис. 4.6. Форма для заповнення заходу

6.1.1 Отримати електронний лист на поштову скриньку з даними про добавлення бронювання

6.2 Видалити своє бронювання

6.2.1 Отримати електронний лист на поштову скриньку з даними про видалення бронювання

6.3 Змінити час свого бронювання

6.3.1 Отримати електронний лист на поштову скриньку з даними про зміну бронювання

7. Як авторизований користувач я додаю бронювання, щоб вказати назву події, час початку та її завершення

8. Як авторизований користувач я захожу на історію бронювань, щоб:

8.1 Побачити всі мої замовлення

8.2 Видалити замовлення

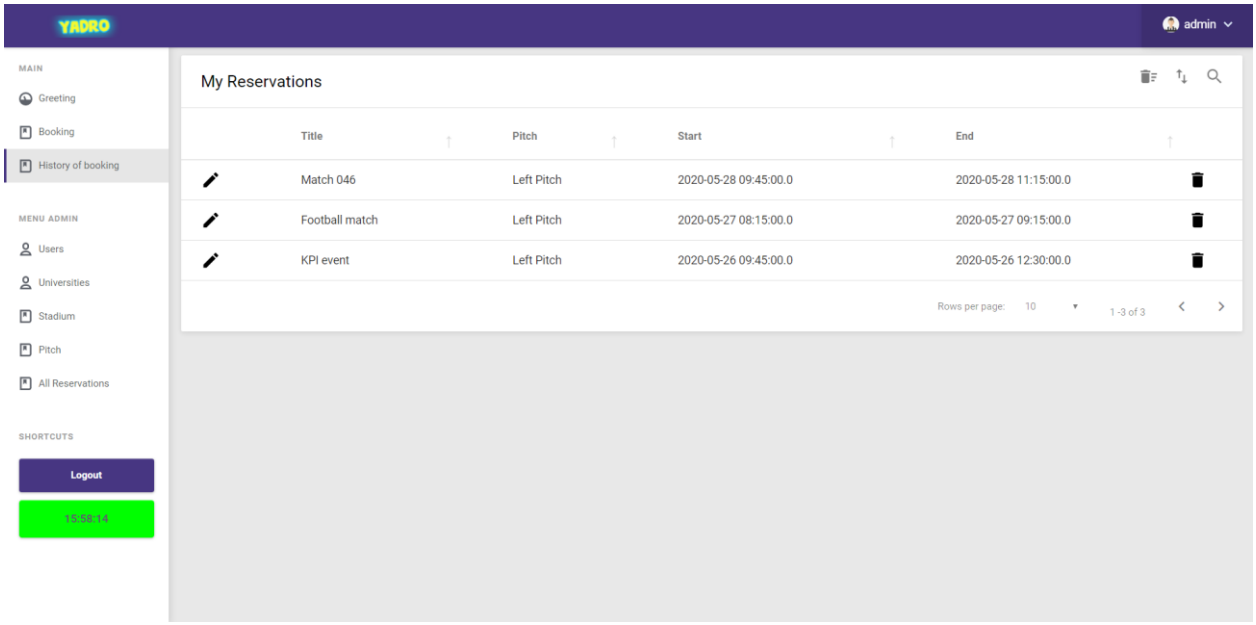


Рис. 4.7. Історія бронювань заходів

9. Як авторизований користувач я натискаю на кнопку особистого кабінету (справа зверху), щоб:

9.1 Перейти до сторінки зміни даних користувача

9.2 Вийти з акаунта

10. Як авторизований користувач я натискаю на кнопку Logout, щоб вийти з акаунту.

4.1.3. Інструкція для користувача з правами адміністратора:

1.Як адміністратор я можу перейти в свій особистий кабінет, щоб змінити свої дані.

2. Як адміністратор я можу перейти на сторінку університету, щоб побачити можливі стадіони.

3. Як адміністратор я можу перейти на сторінку обраного стадіону, щоб побачити можливі футбольні поля та їх характеристики.

4. Як адміністратор я можу обрати футбольне поле, щоб перейти до календарю бронювання

5. Як адміністратор я можу перейти до календаря бронювань, щоб :

5.1 Додати собі бронювання поля на обраний час

5.1.1 Отримати електронний лист на поштову скриньку з даними про добавляння бронювання

5.2 Видалити своє бронювання

5.2.1 Отримати електронний лист на поштову скриньку з даними про видалення бронювання

5.3 Змінити час свого бронювання

5.3.1 Отримати електронний лист на поштову скриньку з даними про зміну бронювання

6. Як адміністратор я додаю бронювання, щоб вказати назву події, час початку та її завершення

7. Як адміністратор я захожу на мою історію бронювань, щоб:

7.1 Побачити всі мої замовлення

7.2 Видалити замовлення

8. Як адміністратор я натискаю на кнопку особистого кабінету (справа зверху), щоб:

8.1 Перейти до сторінки зміни даних користувача

8.2 Вийти з акаунта

9. Як адміністратор я натискаю на кнопку Logout, щоб вийти з акаунту

10. Як адміністратор я натискаю кнопку Add User, щоб:

10.1 Добавити новий акаунт

10.2 Надати йому роль

11. Як адміністратор, я перехожу на список користувачів, щоб

11.1 Побачити всіх зареєстрованих користувачів

11.2 Перейти на сторінку редагування користувача

11.3 Видалити користувача

11.4 Завантажити файл всіх користувачів

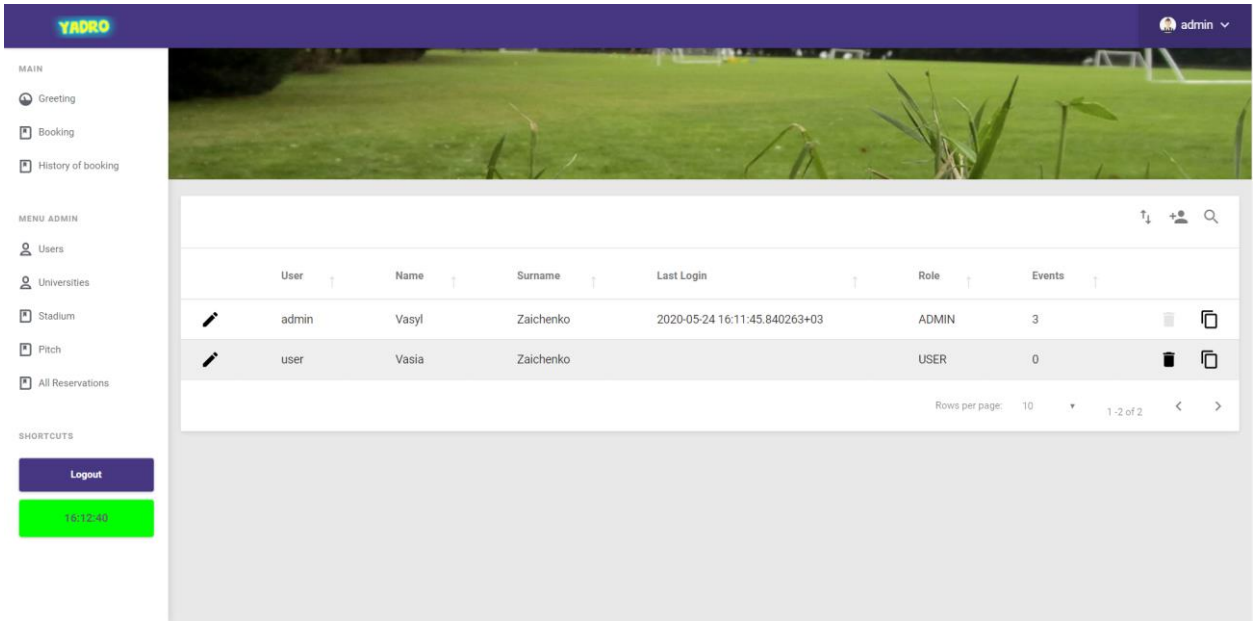


Рис. 4.8. Список усіх зареєстрованих користувачів

12. Я переходжу на сторінку редагування користувача, щоб

12.1 Змінити його дані

12.2 Надати йому роль

13. Як адміністратор я натискаю кнопку Add University, щоб добавити новий університет

14. Як адміністратор, я переходжу на список університетів, щоб

14.1 Побачити всі університети

14.2 Перейти на сторінку редагування університету

14.3 Видалити університет

14.4 Завантажити файл всіх університетів

15. Я переходжу на сторінку редагування університету щоб змінити його дані

16. Як адміністратор я натискаю кнопку Add Pitch, щоб добавити нове поле

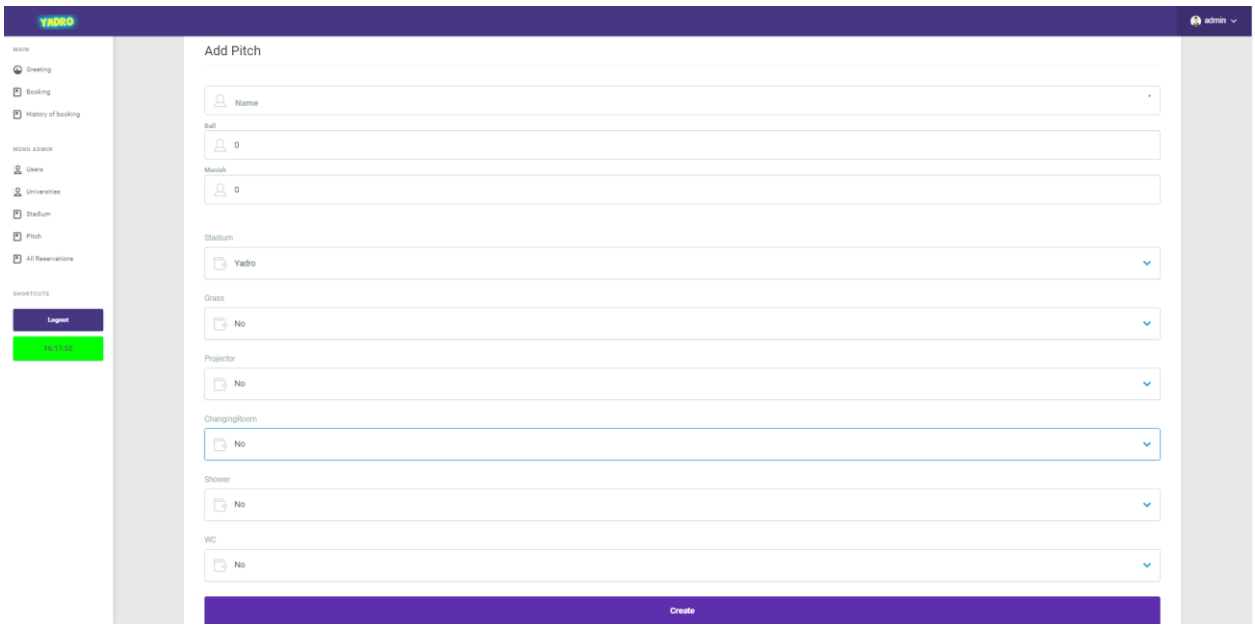


Рис. 4.9. Додавання нового поля у стадіону

17. Як адміністратор, я переходжу на список полів, щоб

    17.1 Побачити всі поля

    17.2 Перейти на сторінку редагування поля

    17.3 Видалити поле

    17.4 Завантажити файл всіх полів

18. Я переходжу на сторінку редагування поля, щоб змінити його дані

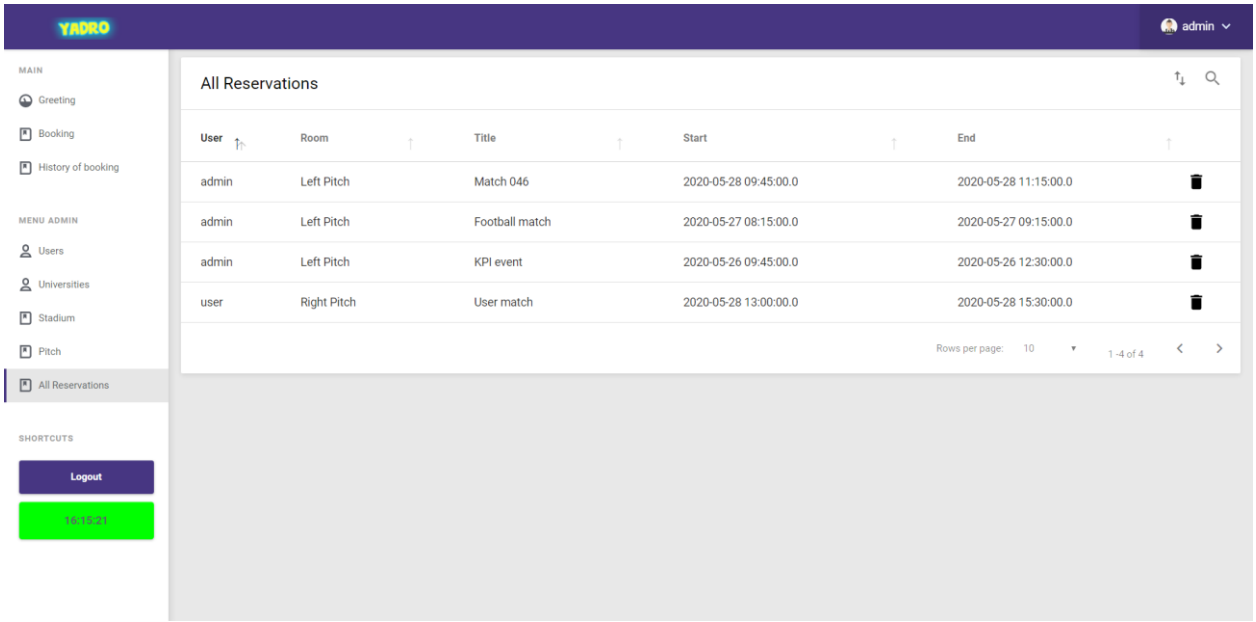


Рис. 4.10. Список усіх наявних резервувань

19. Я переходжу на всі бронювання, щоб:

19.1 Побачити всі бронювання усіх користувачів

19.2 Завантажити файл всіх бронювань

19.3 Видалити бронювання

19.3.1 Відсилання електроного листа власнику бронювання на поштову скриньку з текст про видалення бронювання

4.2.Тестування, як прискорення пошуку помилок систем

З часом кожен додаток досягає того розвитку, що буде майже не можливо знайти помилки в роботі програми особисто, тобто вручну. Для ведення контролю якості покривають програмний код тестами, адже написання тестів - це невід'ємна частина створення програмного забезпечення. В цілому тестування спрямоване на раннє виявлення помилок.

При написанні даного веб-застосунку покриття тестами та перевірка на правильність проводилось на кожній ітерації стоврення певного функціоналу системи. Для забезпечення якості продукту було використані наступні методи:

- Написання юніт-тестів для покриття основного функціоналу на правильність роботи.
- Ручне тестування була також присутнім для перевірки роботоспроможності полів для вводу та кнопок взаємодії.
- Було використано PostMan[21] для перевірки коректної роботи контроллерів серверної частини.

Використання методів, які описані вище, забезпечили виявлення критичних помилок серверної частини додатку та спростили пошук помилок на клієнтській частині

4.3.Порівняння розробки з існуючими аналогами

Розробка додатку почалась з дослідження аналогів, які було розглянуті в розділі 1. Дана система реалізована з дотриманням усіх вимог, які були зазначені в технічному завданні, а саме:

- Можливість пройти аутентифікацію та авторизацію для того, щоб потрапити в особистий кабінет.
- Можливість заповнення календаря бронювань
- При додавані, відміні чи зміні бронювання, приходить відповідний електронний лист на пошту
- Редагування даних в особистому кабінеті
- Додавання нового поля, стадіону, користувача, університету
- Можливість редагувати данні поля, стадіону, користувача та університету
- Перегляд усіх зроблених замовлень користувачами

Реалізована система має наступні переваги перед аналогами, а саме:

- Інтуїтивно зрозумілий користувацький інтерфейс
- Розсилка подій на пошту
- Спроектований під студмістечко
- Чітко визначені повноваги адміна для керування системою
- Відсутність реклами

Створення даної системи є дуже цікавим рішенням для ефективного керування бронювання заходів спортивного комплексу, веб-застосунок може повністю вирішити всі проблеми і надати можливість швидкого та зручного бронювання поля без потреби зустрічі з тренерським штабом.

4.4.Ідеї для вдосконалення й розширення системи

Так як ринок не стоїть на місці і все постійно динамічно розвивається, вже створені сервіси замінюються на аналогічні з більш приємним інтерфейсом, збільшеним функціоналом, ось чому потрібно розвивати та

підтримувати систему. Для подальшого розвитку даної системи хочеться відзначити такі кроки:

1. Додати автотестування сайту, адже перевірка помилок на стадії розробки обходиться значно дешевше, ніж коли вона в релізі.
2. Цікавою ідеєю для розвитку є добавлення полів київських університетів та полів, які належать приватним та державним організаціям.
3. Викладення веб-додатку на домен kpi.
4. Додати можливість оплатити онлайн платні поля.
5. Використати більш сучасний фреймвор для клієнтської частини.
6. Створити телеграм бота, який буде надавати актуальну інформацію про завантаженість поля.
7. Створення можливості переглядати статистику замовлень.

ВИСНОВКИ ДО РОЗДІЛУ 4

У четвертому розділі було продемонстровано реалізований веб-застосунок та розглянуто функціональність кожного типу користувача. Для кожної з ролей було описано інструкцію та для кращого візуального її сприйняття були прикріплені зображення з відповідними можливостями системи.

Проведено тестування програмного забезпечення продукту. Були виконані модульні, інтеграційні, функціональні, нефункціональні та інші типи тестів. Завдяки перевірки веб-додатку за допомоги вищевказаних тестів успішно виявлено та виправлено помилки серверної та клієнтської частини продукту.

Провівши порівняння переваг та недоліків даного веб-застосунку з наявними аналогами із подібною функціональністю, можна сказати, що розроблене програмне забезпечення є чудовим рішенням, який вирішує усі поставленні вимоги для забезпечення комфортного та легкого бронювання заходів спортивного комплексу.

Було проведено аналіз недоліків даного веб-додатку з метою покращення функціональності не лише серверної частини, а й графічного інтерфейсу.

ВИСНОВКИ

Результатом даної бакалаврської роботи є реалізація зручної системи з функціональністю швидкого та комфортного бронювання заходів спортивного комплексу.

В ході даної бакалаврської роботи, було зроблено наступні висновки:

1. Проаналізовано наявні аналоги з подібною функціональністю та було виділено основні вимоги до даного програмного забезпечення спираючись на порівняні існуючих систем.
2. Зробивши аналіз поширених засобів для розробки веб-застосунків, було вирішено: для створення серверної частини даної бакалаврської роботи використовувати фреймворк Spring мови програмування Java; HTML, CSS, JavaScript обрати для реалізації графічного інтерфейсу; задіяти PostgreSQL для зберігання даних веб-додатку.
3. При реалізації даного програмного забезпечення було дотримано усі вимоги та найкращі практики, які були сформовані при порівняння наявних аналогів. Було реалізовано поставлений функціонал для комфортного бронювання заходів. Створено адміністративну частину веб-додатку для зручного регулювання резервації подій спортивного комплексу. Забезпечено надійність програмного забезпечення та додано авторизацію та аутентифікацію користувача.
4. Було проведено тестування даної системи, виконані тести різних типів для глибокої перевірки функціоналу. За допомоги вищевказаних тестів було успішно виявлено та виправлено помилки серверної та клієнтської частини веб-додатку. Додано набір рекомендацій для подальшого розширення та вдосконалення програмного забезпечення.

При виконанні даної бакалаврської роботи було покращено теоретичні знання для створення клієнт-серверних веб-додатків, а також закріплено на практиці цих навичків, реалізувавши систему контролювання бронювання заходів спортивного комплексу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Запис клієнтів | онлайн | Мобільна CRM [Електронний ресурс]. Режим доступу: https://gnom.guru/ru (дата звернення 15.04.2019)

2. ГномГуру CRM: Запис клієнтів [Електронний ресурс]. Режим доступу: https://play.google.com/store/apps/details?id=guru.gnom_dev&hl=ru (дата звернення 15.04.2019)

3. ЗапісьОнлайн - сервіс онлайн записи - сервис онлайн записи [Електронний ресурс]. Режим доступу: https://xn--80aatfgkndeix6k.xn--p1ai/ (дата звернення 15.04.2019)

4. Сервіс онлайн запису [Електронний ресурс]. Режим доступу: https://xn--80aatfgkndeix6k.xn--p1ai/price (дата звернення 15.04.2019)

5. SimplyBook.me - Free Appointment Scheduling Software [Електронний ресурс]. Режим доступу: https://simplybook.me/ (дата звернення 15.04.2019)

6. SimplyBook.me | DOU [Електронний ресурс]. Режим доступу: https://jobs.dou.ua/companies/notando/ (дата звернення 15.04.2019)

7. Клієнт-серверна архітектура та ролі серверів [Електронний ресурс]. Режим доступу: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893в8048229 (дата звернення 15.04.2019)

8. Java - об'єктно-орієнтована мова [Електронний ресурс]. Режим доступу: https://habr.com/ru/hub/java/ (дата звернення 15.04.2019)

9. Spring | Home [Електронний ресурс]. Режим доступу: https://spring.io/ (дата звернення 15.04.2019)

10. PHP – найбільш популярна мова для веб програмування [Електронний ресурс]. Режим доступу: https://chili-web.com.ua/php-5/ (дата звернення 15.04.2019)

11. Переваги використання технологій Ruby та Ruby on Rails [Електронний ресурс]. Режим доступу: http://eprints.zu.edu.ua/21108/ (дата звернення 15.04.2019)

12. Google Keep [Електронний ресурс]. Режим доступу: https://keep.google.com (дата звернення 15.04.2019)

13. NoSQL бази даних — переваги та недоліки [Електронний ресурс]. Режим доступу: https://www.quality-assurance-group.com/nosql-perevagy-ta-nedoliky-nerelyatsijnyh-baz-danyh/ (дата звернення 15.04.2019)

14. 5 причин використовувати Монго (MongoDB) [Електронний ресурс]. Режим доступу: https://echo.lviv.ua/dev/9693 (дата звернення 15.04.2019)

15. Мова JavaScript та її можливості [Електронний ресурс]. Режим доступу: /https://sites.google.com/site/webtehnologiietawebdizajn/mova-javascript-ta-ieie-mozlivosti (дата звернення 15.04.2019)

16. Початок роботи – React [Електронний ресурс]. Режим доступу: https://uk.reactjs.org/docs/getting-started.html (дата звернення 15.04.2019)

17. JSP - JavaStudy [Електронний ресурс]. Режим доступу: http://javastudy.ru/interview/jee-jsp-questions-answers/ (дата звернення 13.05.2019)

18. Що таке MVC і які переваги він надає? [Електронний ресурс]. Режим доступу: http://web-programming.in.ua/view/post/scho-take-mvc-i-yaki-perevagu-vin-nadae (дата звернення 13.05.2019)

19. Java - Відмінність POJO від звичайного класу, Java [Електронний ресурс]. Режим доступу: https://proselyte.net/tutorials/spring-tutorial-full-version/introduction-into-beans/ (дата звернення 13.05.2019)

20. Керівництво по Spring. Введення в Bean-и. [Електронний ресурс]. Режим доступу: https://habr.com/ru/company/ruvds/blog/422893/ (дата звернення 13.05.2019)

21. Postman - ваш помічник у тестуванні API [Електронний ресурс]. Режим доступу: https://software-testing.ru/library/testing/testing-tools/2638-postman (дата звернення 13.05.2019)

ДОДАТОК 1

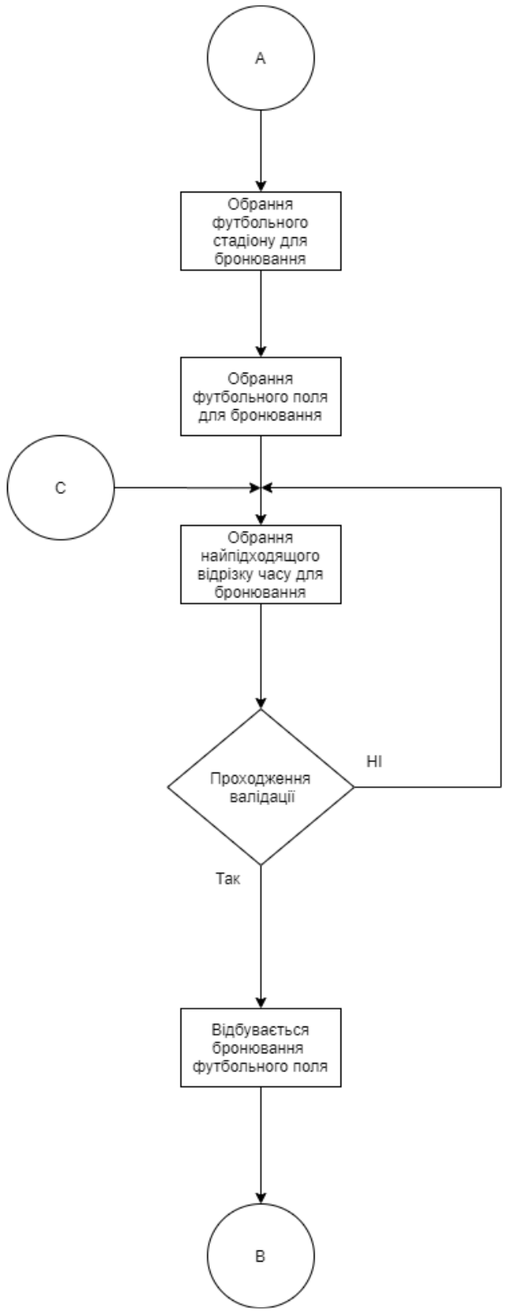
Система контролю бронювання заходів спортивного комплексу

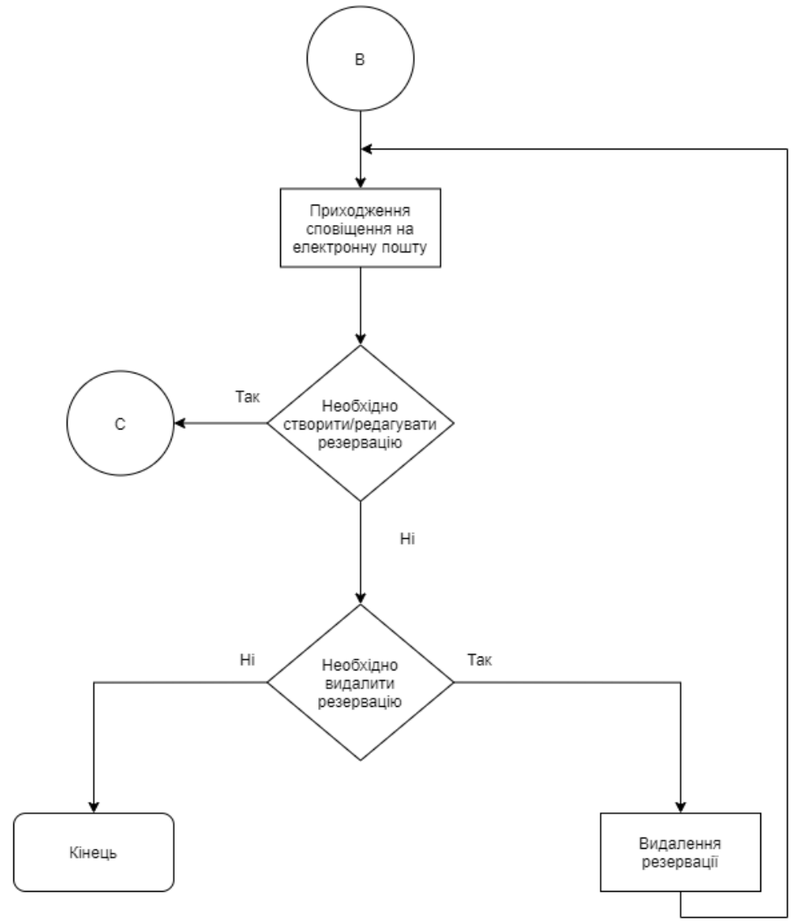
Алгоритм дій програмного забезпечення

ІАЛЦ.467100.004 Д1

Аркушів 1

Київ 2020 р





					ІАЛЦ.467100.004 Д1		
Зм.	Арк.	№ докум.	Підпис	Дата	*Система контролю бронювання заходів спортивного комплексу Алгоритм дій програмного забезпечення*	Літ.	Маса / Масштаб
Розроб.		Зайченко В. В.					
Перевір.		Аленін О. І.				Арк.	Аркушів
Н. контр.		Сімоненко В.П.			*Дипломна робота*	*КПІ ФІОТ кафедра ОТ гр. ІП-64*	
Затверд.							

ДОДАТОК 2

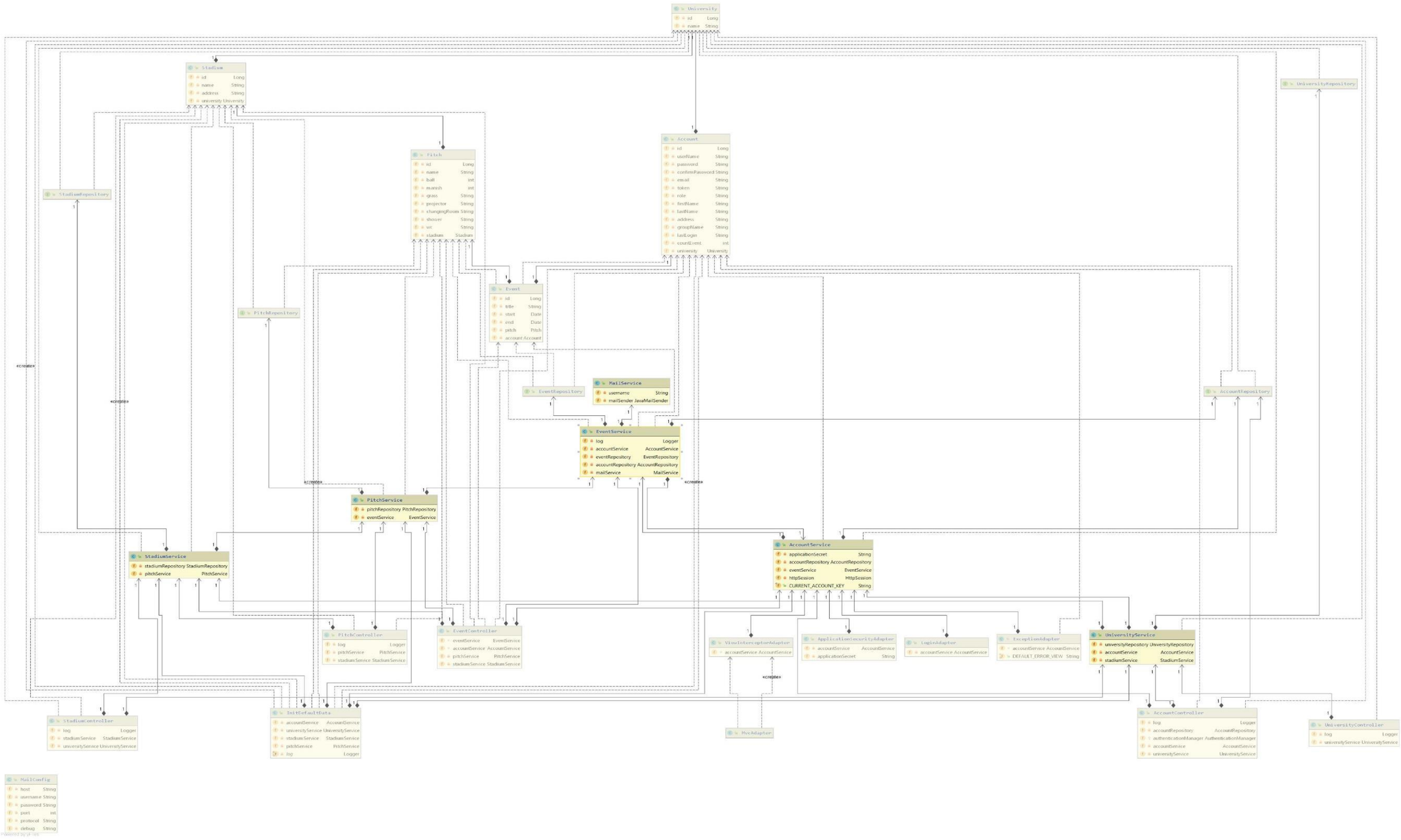
Система контролю бронювання заходів спортивного комплексу

Функціональна схема (діаграма класів)

ІАЛЦ.467100.005 Д2

Аркушів 1

Київ 2020 р



					ІАЛЦ.467100.005 Д2		
Зм.	Арк.	№ докум.	Підпис	Дата	*Система контролю бронювання заходів спортивного комплексу* *Функціональна схема (діаграма класів)*	Літ.	Маса / Масштаб
Розроб.		Зайченко В. В.					
Перевір.		Аленін О. І.				Арк.	Аркушів
Н. контр.		Сімоненко В.П.			*Дипломна робота*	*КПІ ФІОТ кафедра ОТ гр. ІП-64*	
Затверд.							

ДОДАТОК З

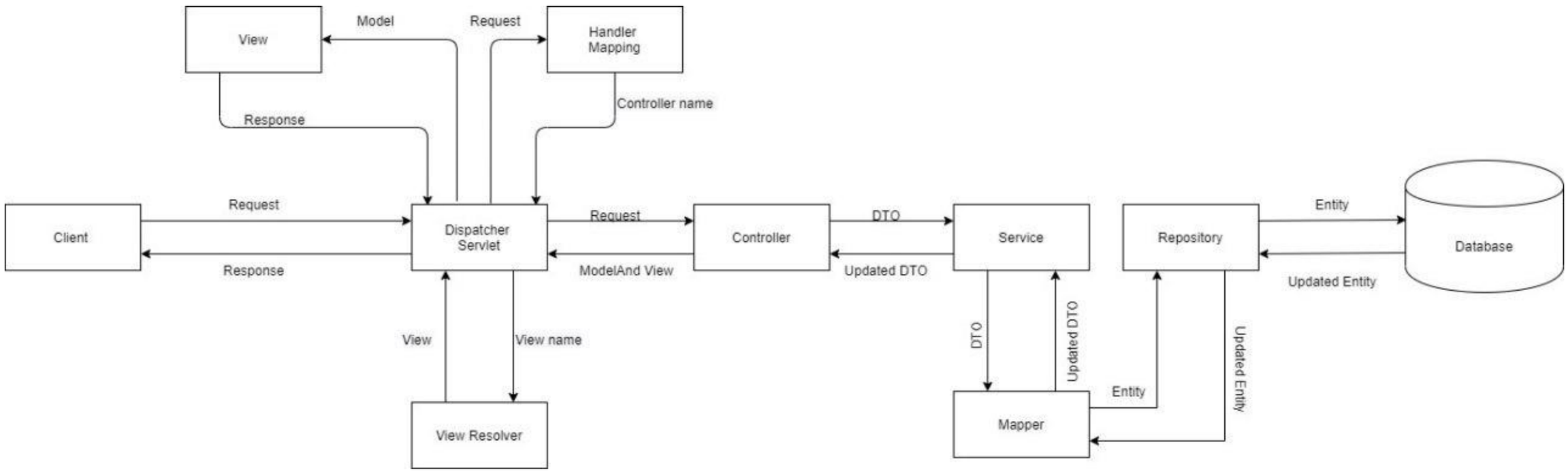
Система контролю бронювання заходів спортивного комплексу

Структурна схема веб-застосунку

ІАЛЦ.467100.006 ДЗ

Аркушів 1

Київ 2020 р



					ІАЛЦ.467100.006 ДЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	*Система контролю бронювання заходів спортивного комплексу Структурна схема веб-застосунку*	Літ.	Маса	Масштаб
Розроб.		Зайченко В. В.						
Перевір.		Аленін О. І.				Арк.	Аркушів	
Н. контр.		Сімоненко В.П.			*Дипломна робота*	*КПІ ФІОТ кафедра ОТ гр. ІП-64*		
Затверд.								

ДОДАТОК 4

Система контролю бронювання заходів спортивного комплексу

Текст програми
ІАЛЦ.467100.007.Д4

Аркушів 10

Київ – 2020 року

```
@Service
public class EventService {

    private Logger log = LoggerFactory.getLogger(EventService.class);

    @Autowired
    private AccountService accountService;

    @Autowired
    private EventRepository eventRepository;

    @Autowired
    private AccountRepository accountRepository;

    @Autowired
    private MailService mailService;

    public Event findById(long id) {
        return eventRepository.findById(id);
    }

    public List<Event> findByAccount(Account account) {
        return eventRepository.findByAccount(account);
    }

    public List<Event> list() {
        return eventRepository.findAll();
    }

    public List<Event> findByDatesBetween(String start, String end, String pitch) {
        Date startDate = null;
        Date endDate = null;
        SimpleDateFormat inputDateFormat = new SimpleDateFormat("yyyy-MM-dd");

        try {
            startDate = inputDateFormat.parse(start);
            endDate = inputDateFormat.parse(end);
        } catch (ParseException e) {
            e.printStackTrace();
        }

        long r = Long.parseLong(pitch);
        return eventRepository.findByDatesBetween(startDate, endDate, r);
    }

    public Event add(Event event) {
        Date now = new Date();

        List<Event> events = eventRepository.findByDatesBetweenOr(event.getStart(),
event.getEnd(), event.getpitch().getId());
        if (events.size() == 0
                && (now.getTime() - event.getStart().getTime()) < 0) {
            Event created = eventRepository.save(event);
            String message = String.format(
                    "Hello %s! \n" +
                            "%s since %s to %s was reserved by you. \n" +
                            "Have a nice day! \n\n\n" +
                            "Best wishes,\n" +
                            "Team 'stadium Activity'",
                    accountService.getLoggedInAccount().getFirstName(),
```

```
                    created.getpitch().getName(),

                    created.getEnd()
            );
            mailService.send(created.getAccount().getEmail(), "Reserved pitch in
stadium", message);

accountRepository.updateCountEvent(accountService.getLoggedInAccount().getUserName(),
countInAccount(accountService.getLoggedInAccount()));
            return created;
        } else {
            return null;
        }
    }

    public Event update(Event event) {
        List<Event> events =
eventRepository.findByDatesBetweenOrAnd(event.getStart(), event.getEnd(),
event.getpitch().getId(), event.getId());
        if (events.size() == 0) {
            String message = String.format(
                    "Hello %s! \n" +
                            "%s since %s to %s was updated. \n" +
                            "Have a nice day! \n\n\n" +
                            "Best wishes,\n" +
                            "Team 'stadium Activity'",
                    accountService.getLoggedInAccount().getFirstName(),
                    event.getpitch().getName(),
                    event.getStart(),
                    event.getEnd()
            );
            mailService.send(event.getAccount().getEmail(), "Update reserve pitch in
stadium", message);
            return eventRepository.save(event);
        } else {
            return null;
        }
    }

    public Boolean delete(Long id) {
        Event event = eventRepository.findById(id);
        String message = String.format(
                "Hello %s! \n" +
                        "%s since %s to %s was deleted. \n" +
                        "Have a nice day! \n\n\n" +
                        "Best wishes,\n" +
                        "Team 'stadium Activity'",
                accountService.getLoggedInAccount().getFirstName(),
                event.getpitch().getName(),
                event.getStart(),
                event.getEnd()
        );
        mailService.send(event.getAccount().getEmail(), "Delete reserve pitch in
stadium", message);
        this.eventRepository.delete(id);
        return true;
    }

    public int countInpitch(pitch pitch) {
        return this.eventRepository.findBypitch(pitch).size();
    }
```

```
    public List<Event> pitchEventList(pitch pitch) {

    }

    public int countInAccount(Account account) {
        return this.eventRepository.findByAccount(account).size();
    }

    public List<Event> accountEventList(Account account) {
        return this.eventRepository.findByAccount(account);
    }

    public List<List<String>> listTable() {
        List<Event> events = this.eventRepository.findAll();
        List<List<String>> eventsList = new LinkedList<List<String>>();

        Comparator<Event> comparator = new Comparator<Event>() {
            @Override
            public int compare(Event left, Event right) {
                return (int) (left.getId() - right.getId());
            }
        };
        Collections.sort(events, comparator);

        for (Event event : events) {
            List<String> eventData = new ArrayList<String>();
            //eventData.add("<a style=\"color: #f9b012\" href=\"/event/edit/" +
event.getId() + "\"><svg xmlns=\"http://www.w3.org/2000/svg\" viewBox=\"0 0 24 24\"
width=\"24\" height=\"24\"><path d=\"M3 17.25V21h3.75L17.81 9.94l-3.75-3.75L3
17.25z\"></path><path d=\"M20.71 7.04c.39-.39.39-1.02 0-1.41l-2.34-2.34c-.39-.39-
1.02-.39-1.41 0l-1.83 1.83 3.75 3.75 1.83-1.83z\"></path></svg></a>");
            eventData.add(event.getAccount().getUserName());
            eventData.add(event.getpitch().getName());
            eventData.add(event.getpitch().getstadium().getName());
            eventData.add(event.getTitle());
            eventData.add(String.valueOf(event.getStart()));
            eventData.add(String.valueOf(event.getEnd()));
            eventsList.add(eventData);
        }
        return eventsList;
    }

    public List<List<String>> listTable(Account account) {
        List<Event> events = this.eventRepository.findByAccount(account);
        List<List<String>> eventsList = new LinkedList<List<String>>();

        Comparator<Event> comparator = new Comparator<Event>() {
            @Override
            public int compare(Event left, Event right) {
                return (int) (left.getId() - right.getId());
            }
        };
        Collections.sort(events, comparator);

        for (Event event : events) {
            List<String> eventData = new ArrayList<String>();
             eventData.add(event.getTitle());
             eventData.add(event.getpitch().getName());
             eventData.add(event.getpitch().getstadium().getName());
             eventData.add(String.valueOf(event.getStart()));
             eventData.add(String.valueOf(event.getEnd()));
             ventsList.add(eventData);
```

```
        }

    }

    public Map<String, Object> listExport(Account a) {
        List<Event> events = new ArrayList<>();
        if (a == null) {
            events = eventRepository.findAll();
        } else {
            events = eventRepository.findByAccount(a);
        }

        Comparator<Event> comparator = new Comparator<Event>() {
            @Override
            public int compare(Event left, Event right) {
                return (int) (left.getId() - right.getId());
            }
        };
        Collections.sort(events, comparator);

        Map<String, Object> linkedHashMap = new LinkedHashMap<String, Object>();
        for (Event event : events) {
            Map<String, Object> linkedHashMapMap = new LinkedHashMap<String,
Object>();
            linkedHashMapMap.put("id", event.getId());
            linkedHashMapMap.put("title", event.getTitle());
            linkedHashMapMap.put("pitch", event.getpitch());
            linkedHashMapMap.put("start", event.getStart());
            linkedHashMapMap.put("end", event.getEnd());
            linkedHashMapMap.put("account", event.getAccount().getUserName());
            linkedHashMap.put("event " + String.valueOf(event.getId()),
linkedHashMapMap);
        }
        return linkedHashMap;
    }
}



<script th:inline="javascript">
    /*<![CDATA[*/
    $(function () {
        $('form.material').materialForm();
    });

    $(function () {

        var dialog,
            form,
            tips = $(".validateTips"),
            moved = false,
            eventToDeleteId,
            eventInEdit,
            account = /*[[${g_account}]]*/ null,
            rooom = /*[[${room}]]*/ null,
            rooomId = /*[[${room.id}]]*/ null;

        function validateDateRange(start, end) {

            if (!start.isValid()) {
                alert("Start date is invalid");
                return false;
```

```
            }

            if (!end.isValid()) {
                alert("End date is invalid");
                return false;
            }

            if (start.isAfter(end)) {
                alert("End date must be after start date");
                return false;
            }

            if (start.isSame(end)) {
                alert("End date must be after start date");
                return false;
            }
            return true;
        }

        function isOverlapping(start, end, id) {
            var array = $('#calendar').fullCalendar('clientEvents');
            for (i in array) {
                if (end > array[i].start && start < array[i].end && id !=
array[i].id) {
                    alert("Can't be created, change starting and ending time.");
                    return true;
                }
            }
            return false;
        }

        function removeEvent() {

            var eventStart = moment(modstartdateandtime.value);
            var eventEnd = moment(modenddateandtime.value);

            var eventData;
            if (moduid.value) {
                eventData = {
                    id: moduid.value,
                    title: modtitle.value,
                    start: eventStart,
                    end: eventEnd,
                    account: account,
                    room: rooom
                };

                editDialog.dialog("close");

                $.ajax({
                    type: "DELETE",
                    url: "/calendar/removeevent",
                    data: JSON.stringify(eventData),
                    contentType: "application/json; charset=utf-8",
                    dataType: "json",
                    success: function (data) {
                        $('#calendar').fullCalendar('removeEvents', eventToDeleteId);
                    },
                    failure: function (errMsg) {
                        alert(errMsg);
                    }
                });
```

```
            }

        }

        function editEvent(event, elements) {
            var eventStart = moment(event.start).format("YYYY-MM-DD  HH:mm:ss");
//moment(event.start);
            var eventEnd = moment(event.end).format("YYYY-MM-DD  HH:mm:ss");

            //alert (eventStart + "   " + eventEnd + "   " + event.end);
            modtitle.value = event.title;
            modstartdateandtime.value = eventStart;
            modenddateandtime.value = eventEnd;
            moduid.value = event.id;
            eventToDeleteId = event.id;
            editDialog.dialog("open");
        }

        function saveEvent() {
            var valid = true;

            var eventStart = moment(modstartdateandtime.value);
            var eventEnd = moment(modenddateandtime.value);

            valid = valid && startdateandtime.value;
            if (modtitle.value) {
                valid = true;
            }
            if (moved) {
                valid = true;
            } else {
                valid = !isOverlapping(eventStart, eventEnd, moduid.value);
            }
            if (!validateDateRange(eventStart, eventEnd)) {
                valid = false;
            }


            if (valid) {
                var eventData;
                if (modtitle.value) {
                    eventData = {
                        id: moduid.value,
                        title: modtitle.value,
                        start: eventStart,
                        end: eventEnd,
                        account: account,
                        room: rooom
                    };

                }
                $('#calendar').fullCalendar('unselect');

                editDialog.dialog("close");

                $.ajax({
                    type: "PATCH",
                    url: "/calendar/updateevent",
                    data: JSON.stringify(eventData),
                    contentType: "application/json; charset=utf-8",
                    dataType: "json",
```

```
                        if (moved) {

                        }
                        if (modtitle.value) {
                            var item = $("#calendar").fullCalendar('clientEvents',
eventInEdit.id);
                            item[0].title = modtitle.value;
                            item[0].start = modstartdateandtime.value;
                            item[0].end = modenddateandtime.value;
                            $('#calendar').fullCalendar('updateEvent', item[0]);
                        }
                    },
                    failure: function (errMsg) {
                        alert(errMsg);
                    }
                });
            }
            moved = false
            return valid;
        }
    function addEvent(start, end) {
            var valid = true;

            var eventStart = moment(startdateandtime.value);
            var eventEnd = moment(enddateandtime.value);


            valid = valid && newtitle.value;
            valid = valid && startdateandtime.value;

            if (validateDateRange(eventStart, eventEnd)) {
                valid = !isOverlapping(eventStart, eventEnd, uid.value);
            }

            if (valid) {
                var eventData;
                if (newtitle.value) {
                    eventData = {
                        title: newtitle.value,
                        start: eventStart,
                        end: eventEnd,
                        account: account,
                        room: rooom
                    };
                    //$('#calendar').fullCalendar('renderEvent', eventData, true); //
stick? = true
                }

                $('#calendar').fullCalendar('unselect');
                dialog.dialog("close");

                $.ajax({
                    type: "POST",
                    url: "/calendar/addevent",
                    data: JSON.stringify(eventData),
                    contentType: "application/json; charset=utf-8",
                    dataType: "json",
                    success: function (data) {
                        $('#calendar').fullCalendar('renderEvent', data, true); //
stick? = true
                    },
                    failure: function (errMsg) {
```

```
                    alert(errMsg);
                }
            });


        }
        return valid;
    }

    dialog = $("#dialog-form").dialog({
        autoOpen: false,
        height: 900,
        width: 1900,
        modal: true,
        buttons: {
            "Create": addEvent,
            Cancel: function () {
                dialog.dialog("close");
            }
        },
        close: function () {
            form[0].reset();
        }
    });

    editDialog = $("#edit-dialog-form").dialog({
        autoOpen: false,
        height: 900,
        width: 1900,
        modal: true,
        buttons: {
            Save: function () {
                saveEvent();
            },
            Delete: function () {
                removeEvent();
                $('#calendar').fullCalendar('removeEvents', eventToDeleteId);
            },
            Cancel: function () {
                editDialog.dialog("close");
            }
        },
        close: function () {
            form[0].reset();
        }
    });

    form = dialog.find("form").on("submit", function (event) {
        event.preventDefault();
        addEvent();
    });

    $("#create-event").button().on("click", function () {
        dialog.dialog("open");
    });

    $(document).ready(function () {

        $('#calendar').fullCalendar({
            firstDay: 1,
            header: {
                left: 'prev,next today',
```

```
                    center: 'title',
                    right: 'weekDays, agendaDay'
                },
                monthNamesShort: ['January', 'February', 'March', 'April', 'May',
'June', 'July', 'August', 'September', 'October', 'November', 'December'],
                dayNamesShort: ["Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"],
                buttonText: {
                    prev: "PREV",
                    next: "NEXT",
                    today: "now",
                    week: "7 DAY",
                    day: "1 DAY"
                },
                views: {
                    // agendaFourDay: {
                    //     type: 'agenda',
                    //     duration: { days: 5 },
                    //     buttonText: '4 day'
                    // },
                    weekDays: {
                        buttonText: '7 days',
                        type: 'agendaWeek'
                        // hiddenDays: [0, 14]
                        // },
                        // weekEnds: {
                        //     buttonText: 'weekend  !important',
                        //     type: 'agendaWeek',
                        //     hiddenDays: [1, 2, 3, 4, 5]
                    }
                },
                defaultDate: moment().format("YYYY-MM-DD"),
                editable: false,
                eventRender: function (event, element) {
                    if (event.account === account) {
                        event.editable = true;
                        event.durationEditable = true;
                    }
                },
                eventDurationEditable: true,
                eventAfterRender: function (event, element, view) {
                    if (event.account.userName === account.userName) {
                        event.editable = true;
                        event.durationEditable = true;
                        element.css('background-color', '#006600');
                    } else {
                        event.editable = false;
                        event.durationEditable = false;
                        element.css('background-color', '#ff0000');
                        element.css('color', '#000000');
                        // element.css('opacity', '.80')
                    }
                    element.find('.fc-title').append("<br \/>" + "Reserved by" + "\n"
+ event.account.userName);
                },
                eventLimit: true, // allow "more" link when too many events
                events: {
                    url: '/calendar/events',
                    data: function () { // a function that returns an object
                        return {
                            room: rooomId
                        };
```

```
                    },

                    error: function () {
                        alert('there was an error while fetching events!');
                    }
                },
                selectable: true,
                selectHelper: true,
                select: function (start, end) {
                    startdateandtime.value = moment(start).format("YYYY-MM-
DDTHH:mm:ss");
                    enddateandtime.value = moment(end).format("YYYY-MM-DDTHH:mm:ss");
                    dialog.dialog("open");
                },
                eventClick: function (event, element) {
                    if (event.account.userName === account.userName) {
                        eventInEdit = event;
                        editEvent(event, element);
                    }
                },
                eventMouseover: function (event, jsEvent, view) {

                },
                eventMouseout: function (event, jsEvent, view) {
                },
                loading: function (bool) {
                    $('#loading').toggle(bool);
                },
                eventDrop: function (event) {
                    eventInEdit = event;
                    var eventStart = moment(event.start).format("YYYY-MM-
DDTHH:mm:ss"); //moment(event.start);
                    var eventEnd = moment(event.end).format("YYYY-MM-DDTHH:mm:ss");

                    modtitle.value = event.title;
                    modstartdateandtime.value = eventStart;
                    modenddateandtime.value = eventEnd;
                    moduid.value = event.id;

                    moved = true;
                    saveEvent();
                },
                eventResize: function (event) {
                    eventInEdit = event;
                    var eventStart = moment(event.start).format("YYYY-MM-
DDTHH:mm:ss"); //moment(event.start);
                    var eventEnd = moment(event.end).format("YYYY-MM-DDTHH:mm:ss");

                    modtitle.value = event.title;
                    modstartdateandtime.value = eventStart;
                    modenddateandtime.value = eventEnd;
                    moduid.value = event.id;

                    moved = true;
                    saveEvent();

                });
            });
        });
    /*]]>*/
</script>
```