ДИПЛОМНА РОБОТА

ДР.ПІс – 06.00.000 ПЗ

Група ПІс-2015

Дзьоба М.А.

2019

ПВНЗ Університет Короля Данила

Кафедра Інформаційних технологій та програмної інженерії

УДК 004

ДИПЛОМНА РОБОТА

Тема ***Розробка веб-сервісу резервування місць в закладах харчування***

Напрям підготовки ***6.050103 — «Програмна інженерія»***
(код і назва спеціальності)

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДР.ПІс – 06.00.000 ПЗ
(позначення)

Студент

Дзьоба М.А.
(підпис) (дата) (розшифрування підпису)

Керівник проекту
д.т.н.,доц. *Мельничук С.І.*
(посада) (підпис) (дата) (розшифрування підпису)

Нормоконтроль
к.т.н. *Мануляк І.З.*
(посада) (підпис) (дата) (розшифрування підпису)

Допускається до захисту
Завідувач кафедри
д.т.н.,доц. *Мельничук С.І.*
(посада) (підпис) (дата) (розшифрування підпису)

2019

ПВНЗ УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет *Інформаційних технологій*
Кафедра *Інформаційних технологій та програмної інженерії*
Напрям підготовки *6.050103 — «Програмна інженерія»*

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІТПІ
_____ *С.І. Мельничук*
“____” _____ 2019 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Студенту *Дзьобі Миколі Анатолійовичу*

1. Тема проекту (роботи) *Розробка веб-сервісу резервування місць у закладах харчування*

Затверджена наказом ректора Університету Короля Данила від *30.10.2018 р. № 20/4*
2. Термін здачі студентом закінченого проекту (роботи) *10.06.2019 р.*
3. Вихідні дані до проекту (роботи) *Java, Spring Boot, VueJs фреймворк, MongoDB база даних.*

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити) *1. Аналіз аналогів веб-сервісу, алгоритми роботи та опис використаних технологій. 2. Розробка алгоритмів та взаємозв’язків між функціональними об’єктами. 3. Опис функціональних можливостей.*

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) *Постановка завдання, вибір технологій, структура програми та бази даних, розробка інтерфейсу, методи реєстрації, настанова користувача.*

6. Консультанти з проекту (роботи), із зазначенням розділів проекту, що стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 30.10.2018

Керівник Мельничук С.І.

Завдання прийняв до виконання Дзьоба М.А.

КАЛЕНДАРНИЙ ПЛАН

Пор №	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1.	Опис наявних сайтів аналогів, їх алгоритми роботи та опис використаних технологій	30.11.2018	
2.	Розробка алгоритмів та дизайну веб-сайту наукової організації	25.12.2018	
3.	Розробка допоміжних функцій та інтерфейсів для веб-сайту наукової організації	21.02.2019	
4.	Оформлення пояснювальної записки	23.05.2019	
5.	Оформлення графічного матеріалу та підготовка до захисту роботи	05.06.2019	

Студент-дипломник *Дзьоба М.А.*
(підпис) (розшифровка підпису)

Керівник проекту *Мельничук С.І.*
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Темою даної дипломної роботи є вихід бізнесу у сфері харчування у інтернет-мережу та економія часу відвідувачів при виборі закладу.

В даній дипломній роботі розробленовеб-сервіс, котрий вирішує проблеми, як власників, які хочуть вийти у інтернет-мережу, так і відвідувачів закладу. Розроблено зручний і швидкий інтерфейс користувача та безпечну і надійну серверну частину.

SUMMARY

The topic of this thesis is the initial business in the field of nutrition on the Internet and economy during the time of the election.

In this thesis, the developed services, which solves the problems of both the owners who want to come to the Internet and visitors of the institution. A useful and fast user interface and a secure and reliable server part are developed.

РЕФЕРАТ

Розрахунково-пояснювальна записка: 66 сторінки, 37 рисунків, 2 таблиці, 2 додатки.

Об’єктом дослідження є заклад харчування.

Метою роботи є дослідження процесу резервування столиків у закладі харчування та створення веб-сервісу для публікування закладу у інтернет-мережу, візуалізація залу для онлайн замовлення відвідувачами.

Відповідно до поставленого завдання в роботі проводиться дослідження процесу формування і обліку замовлень столиків у закладі харчування. Для полегшення даного процесу та зменшення рутинної роботи та затрачуваного часу розроблено веб-додаток для обліку та прийому замовлень онлайн.

Ключові слова: клієнт-сервер, багатошарова архітектура, REST, Spring Framework, Vue.js.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

СУБД – система управління баз даних

SPA – Single Page Application

EE – Enterprise Edition

MVC – Model View Controller

REST – REpresentational State Transfer

JSON – JavaScript Object Notation

ВСТУП

Актуальність теми. Вихід у інтернет-мережу у наш час є невід'ємною частиною успішності бізнесу. Кожного дня конкуренція зростає, а вихід у всесвітню мережу відкриває нові можливості.

Задачі досліджень:

- розглянути процес прийому та обліку заявок на резервування столиків;
- дослідити потреби користувачів веб-сервісу;
- розробити швидкий та зручний інтерфейс та надійну серверну частину.

Об'єкт досліджень – процес прийому та обліку заявок на резервування столиків.

Методи дослідження – методи експерименту та спостереження для визначення ефективності традиційних методів прийому та обліку заявок на резервування, а також метод порівняння для аналізу аналогів.

Результати досліджень. Результатом досліджень став веб-сервіс, який суттєво спрощує прийом та облік заявок на резервування столиків закладу харчування у інтернет-мережі.

1 АНАЛІЗ АНАЛОГІВ ВЕБ-СЕРВІСУ, АЛГОРИТМИ РОБОТИ ТА ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

1.1 Огляд способів забезпечення інформаційних систем резервування

Веб-сервіс розроблявся для вирішення проблеми управління ресурсів закладу харчування, а саме місцями. Тому для використання можливостей програмного продукту було вирішено розробляти його таким чином, щоб заклад переходив на його використанні повністю. Під цим мається на увазі, що заклад прийматиме замовлення не тільки у онлайн режимі, а також відмічатиме ті резервування, які будуть надходити офлайн: дзвінки та резервування у закладі. Таким чином на сайті буде відображатися завжди актуальна інформація про стан замовлень. Також передбачена функція вводу стану стола у тому разі, коли відвідувачі попередньо не робили резервування столу. Всі перелічені функції потрібно обов'язково виконувати персоналу закладу харчування, щоб інформація була завжди актуальною. Беручи до уваги те, що дані операції потрібно виконувати багато разів на день, випливає висновок, що вони повинні бути у швидко доступному місці та виконання їх має бути максимально простим.

Процес замовлення у закладі відбуватиметься у декілька етапів. Після вибору столика потрібно вказати дату, час та якщо потрібно, коментар. У момент надсилання заявки у обліковому записі закладу прийде сповіщення разом із звуковим сигналом, після розгляду замовлення, якщо репутація у клієнта низька, йому зателефонують та підтвердять замовлення. Якщо ж репутація висока, то підтвердження прийде на електронну пошту. Також, якщо замовлення підтверджено, то воно буде виводитися у списку замовлень для всіх користувачів. Аналізуючи процес резервування у закладах харчування можна помітити, що столик, котрий замовили стає недоступний для інших відвідувачів в середньому 1 годину до замовлення. Але у кожному закладі різний середній час відвідування користувачів, тому для більшої ефективності сервісу час

блокування столику перед резервуванням заклад може вказувати свій (за замовчуванням 1 година). Тобто, якщо час введений користувачем занадто близький із часом вже наявного замовлення, то замовлення зробити буде неможливо через блокування столика. Замовлення можна скасувати, але це негативно відобразиться на рівні репутації. Слід зазначити, що рівень репутації є загальним для усіх закладів, а не для кожного окремо. Скасувати замовлення може також і адміністратор закладу, у такому разі на заклад буде начислено штрафні бали. Штрафні бали напряму впливають на рейтинг закладу у веб-сервісі. Процес скасування супроводжується сповіщенням обох сторін є, як сповіщенням на сайті, так і листом на електронну пошту адресу. При скасуванні замовлення закладом вони зобов'язані зателефонувати клієнту та повідомити причину скасування.

1.2 Аналіз наявних аналогів

Перш ніж створити будь-який продукт потрібно чітко визначити специфікації та ретельно спланувати всі кроки реалізації. Не винятком є розробка програмного продукту. Від цього залежить якість кінцевого результату, а якість завжди визначає успіх.

Проаналізувавши наявні аналоги, можна зрозуміти, що вони мають хоча б один із перелічених недоліків: немає візуалізації залу, дуже вузький вибір закладів, низька швидкість обробки замовлень, довга процедура додавання свого закладу, недостатня кількість інформації про заклад, робота тільки із одним закладом. Звідси можна зробити висновок, що проекти є неуспішними і найближчим часом закриються. Для детального аналізу пропоную взяти сервіси під назвою Letsbar, Mistercat та Reston.

Перейшовши на веб-сайт сервісу Letsbar, можна помітити досить простий, але в той же час зручний і зрозумілий інтерфейс. Одразу ж на головній сторінці нас зустрічає красиво оформлений список закладів і всією необхідною

інформацією (рис. 1.1). Для онлайн резервування є доступним тільки один заклад.

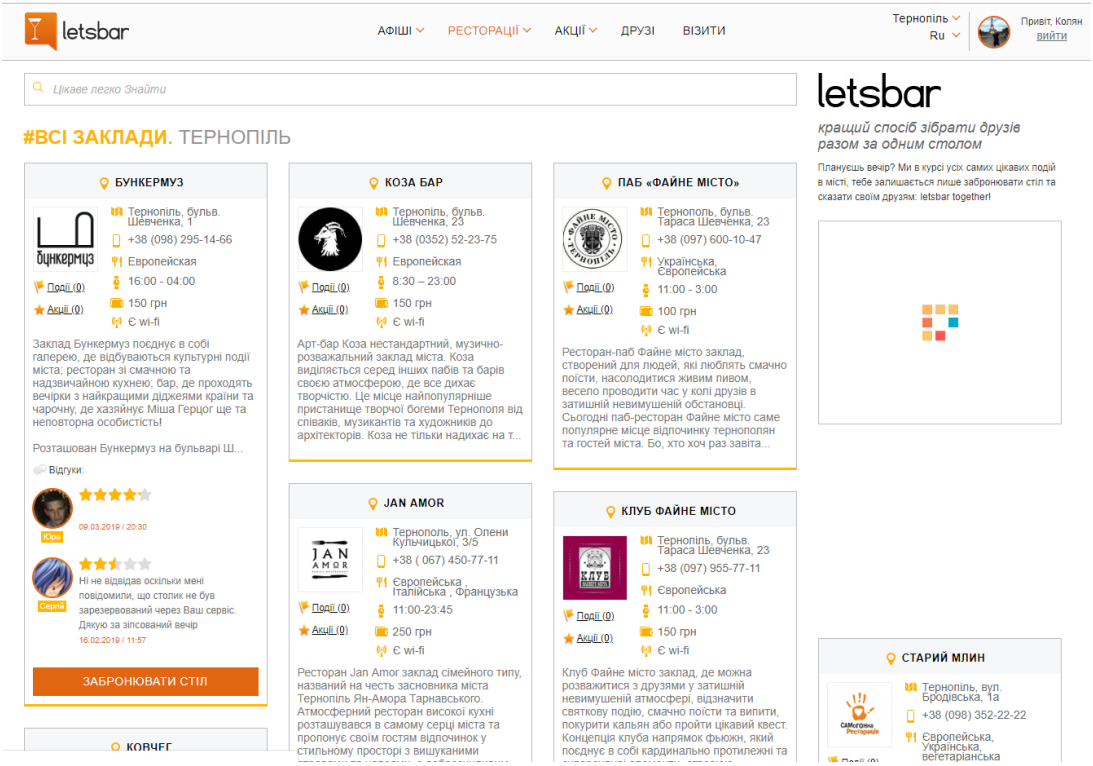

Рисунок 1.1 – Головна сторінка сервісу Letsbar

Переваги:

- візуалізація залу;
- відгуки і оцінки користувачів;
- багато інформації про заклад.

Недоліки:

- 3D візуалізація залу;
- незручний інтерфейс;
- немає списку наявних замовлень;
- малий вибір закладів;
- важко розмістити заклад.

Сервіс Mistercat надає послуги резервування тільки для однієї мережі ресторанів. Також в переліку крім назви та фото немає іншої інформації, що є дуже незручним для користувача, який не визначився із закладом. Перейшовши

на сторінку закладу спостерігаємо таку ж картину. Немає інформації про адресу, графік роботи, середній чек, меню, додаткових фото і тому подібне. Основним недоліком цього веб-додатку є відсутність візуалізації залу (рис 1.2). Процес замовлення столику є максимально незручним для користувача, а відсутність будь-якої іншої інформації про заклад є показником низької якості продукту.

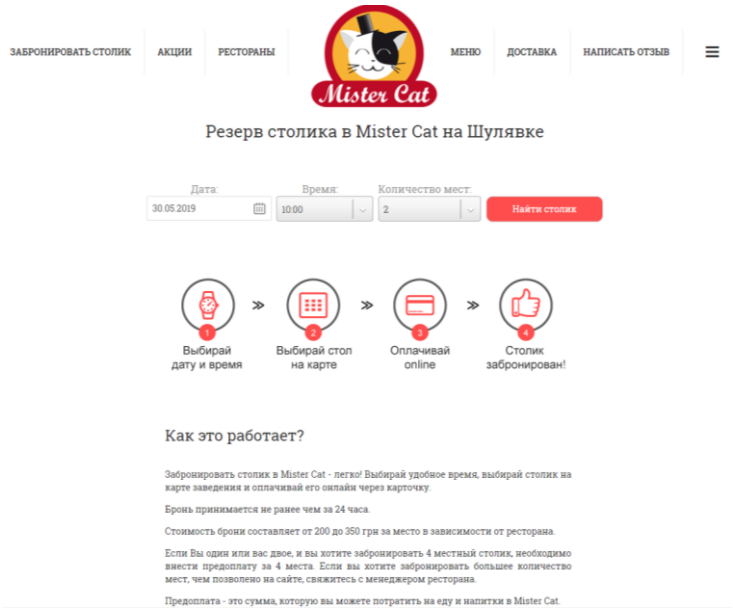

Рисунок 1.2 – Сторінка замовлення закладу сервісом Mistercat

Переваги:

- зручний інтерфейс.

Недоліки:

- відсутність візуалізації залу;
- недостатньо інформації про заклад;
- немає списку замовлень;
- заклади тільки однієї франшизи.

Сервіс Reston відрізняється своїм привабливим на вигляд користувацьким інтерфейсом, функцією пошуку із широкими можливостями. На головній сторінці можна побачити функції «Нові заклади», «Зі знижкою 20-50%», а також «Найближчий заклад» (рис. 1.3). На сайті присутні також рубрики «Спеціальні пропозиції», «Топ закладів тижня», «Популярні заклади» та «Reston

рекомендує». Великим плюсом для користувача є також огляди закладів та блок новин.

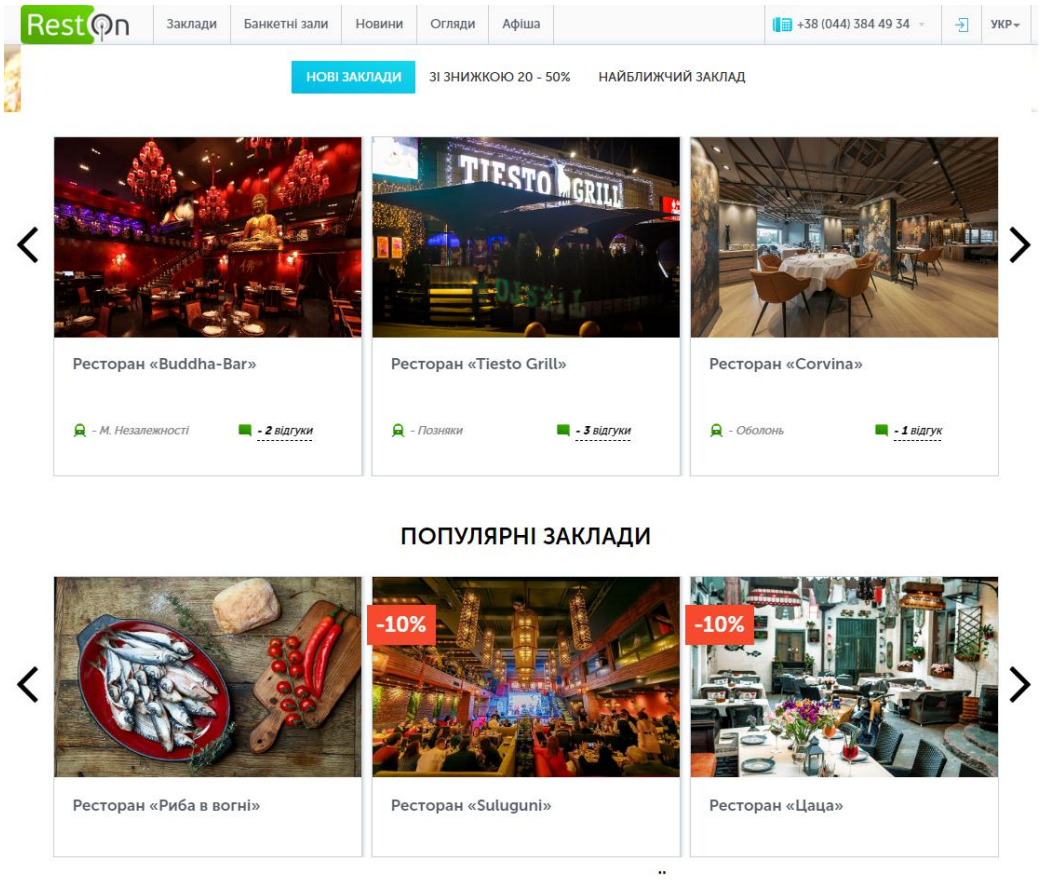

Рисунок 1.3 – Головна сторінка сервісу Reston

Основними недоліками сервісу Reston є низька швидкість роботи сайту та відсутність візуалізації залу.

Переваги:

- зручний інтерфейс;
- широкий вибір закладів;
- великий набір функціоналу.

Недоліки:

- відсутність візуалізації залу;
- повільна робота сайту;
- недостатньо інформації про заклад.

З огляду на усі перелічені вище огляди аналогів стає очевидним, що дипломний проект стане універсальним вирішенням проблем, метою якого буде зберігання часу для користувачів, які часто відвідують заклади громадського харчування та власників закладів, які вирішили вийти у мережу інтернет.

1.3 Функціональні вимоги веб-сервісу

Сфера харчування розвивається так само стрімко, як і усі сфери у наші дні. Нинішній світ кардинально відрізняється від того, який був декілька десятків років тому. Зараз ми не уявляємо свого життя без гаджетів, а тим більше без інтернету. В нинішньому світі наявність інтернет сервісів кардинально впливає на успішність бізнесу, а величезна кількість людей не уявляє своїх покупок без інтернету. І на це є декілька причин – економія часу, великий список пропозицій у одному місці, можливість переглядати відгуки.

Задля зберігання і розширення своєї справи нинішній стан речей змушує ставати все дедалі медійнішими та приваблювати клієнтів пропозиціями кращими ніж у конкурентів у мережі інтернет.

За допомогою різноманітних технологій та впровадження у використання нових програм та додатків можна зробити значно ефективнішим використання ресурсів закладу (для бізнесу) та зекономити час на вибір закладу та резервування місця у ньому (для клієнта).

Коли бізнес вирішує виходити на ринок у інтернеті, невід’ємною частину цього процесу є вибір компанії, яка б реалізувала веб-сайт. Через це витрачається велика кількість часу на визначення вимог, перевірка виконаної роботи, вказування правок і тому подібне. Не завжди кінцевий результат може задовільнити замовника, а це за собою тягне ряд інших проблем таких як втрата часу та грошей.

Виходячи з цього, потреба у створенні веб-додатку є беззаперечною. Такий додаток матиме великі перевагу над конкурентами.

Першою, і однією з найважливіших переваг, є економія часу і мінімальна кількість зусиль і ресурсів, необхідних для отримання бажаного результату: розміщення всієї інформації про заклад у мережі інтернет, отримання панелі керування для ефективного керування місцями у закладі. Ніякої однозначно застарілої, не ефективної та рутинної роботи виконувати більше не потрібно. Вся статистика закладу, списки замовлень та панель керування сторінкою буде на екрані.

Друга перевага в тому, що доступ до додатку можна отримати з будь-якого місця в будь-який час. Додаток є повністю адаптований під всі типи користувацьких пристроїв, тому ним можна легко користуватися за допомогою комп’ютера, планшета чи телефона. Єдиною умовою є наявність підключення до інтернету. Доступ до інтернету зараз доступний практично всім, тому така умова не викликатиме жодних проблем чи незручностей для потенційних користувачів.

Добре спроектований максимально простий і зрозумілий інтерфейс програми дасть змогу швидко освоїти сервіс і приступити до його експлуатації з першого ж знайомства з додатком.

Також важливою перевагою є добре спроектована і розроблена структура проекту, яка надає можливість додавати та інтегрувати нові функції, для покращення додатку, легко і швидко.

Додаток має бути зручним у користуванні (простий, зрозумілий інтерфейс), швидким (оперативно обробляти вхідні і вихідні запити), і повинен працювати коректно (видавати чіткі правильні результати).

Веб-додаток має клієнт-серверну структуру, клієнтська частина якого виконується в браузері, і яка за допомогою запитів спілкується з сервером. Таким чином, при написанні веб-додатку необхідно реалізовувати як клієнтську частину (браузерну, яка подається у вигляді HTML розмітки, CSS стилів, JavaScript коду, відповідального для з'єднання з сервером) так і серверну частину, яка повинна обробляти запити клієнтів.

Для написання якісного веб-додатку необхідно мати великий набір навичок. Для клієнтської частини це:

- навички дизайну і мови розмітки сторінок HTML або її препроцесорів;
- основи скриптового програмування з використанням JavaScript для взаємодії з сервером і підвищення інтерактивності сторінок.

Головними вимогами до розробки даного продукту є:

- зручний інтерфейс, який надасть можливість почати використання програми вже після першого знайомства;
- максимальна швидкість обробки даних, для забезпечення оперативної обробки вхідних і вихідних запитів;
- коректна робота і видавати чітких релевантних результатів;
- стабільність роботи, забезпечення умов для безперебійного функціонування, надійні сервери можуть забезпечити стабільність роботи веб-додатку;
- безпека даних, полягає в тому, що всі персональні дані (паролі, адреси електронних скриньок і тд.) будуть надійно зашифровані.

Основні функціональні можливості:

- візуалізація залу;
- зручний інструмент для створення плану залу;
- зв’язок користувачів із сервером в режимі реального часу для оповіщення їх про зміни у списку замовлень;
- можливість переглядання наявних замовлень для кожного столика;
- можливість переглядання списку замовлень на конкретну дату;
- система репутації клієнта.

Опис кожної функції і етапу розробки буде детально розглянено в наступних розділах.

1.4 Вибір засобів розробки та постановка задачі

Останніми роками бізнес громадського харчування почав стрімко зростати. Споживачі почали вимагати все більш якіснішого сервісу, зручності

обслуговування та індивідуальності закладів. В той же час стрімке збільшення пропозиції на цьому ринку стимулює розширюватися та розвиватися. При цьому потрібно враховувати економічну складову у країні та слідкувати за доступністю. У цьому різноманітті пропозицій користувачу важко слідкувати за цікавими та вигідними пропозиціями. Тому даний веб-сервіс повинен задовільняти потребу споживачів у відслідковуванні закладів.

Сервіс пропонує користувачам зручний спосіб ознайомлення і закладом, а також швидко та зручно замовити місце. У свою чергу сервіс надає велику кількість функцій для бізнесу. Серед них: вихід у інтернет-мережу, маніторинг ресурсів закладу, а також зручний спосіб прийому заявок на резерв столиків.

Головною метою веб-сервісу є зручний та швидкий спосіб резервування для користувачів та оптимізація процесу обліку заявок на резервування для закладів.

Створення якісного сайту вимагає ретельного планування, чітких специфікацій і якісно виконаної роботи. Перш за все сайт має бути зручним у користуванні (простий, зрозумілий інтерфейс), швидким (оперативно обробляти вхідні і вихідні запити), і повинен працювати коректно (видавати чіткі правильні результати).

Сайт має клієнт-серверну структуру, клієнтська частина якого виконується в браузері, і яка за допомогою запитів спілкується з сервером. Таким чином, при написанні веб-додатку необхідно реалізовувати серверну частину, в якій будуть виводитися запити від користувачів, а також створюватися нові інформаційні блоки для сайту.

Серверна частина повинна коректно і надійно опрацьовувати запити клієнта, зберігати сесії клієнтів, і всю інформацію, що вводиться, бути стійкою до можливих атак з боку зловмисників, а також адміністративна панель повинна бути зручною у користуванні і мати приємний інтерфейс.

На даний час найбільш надійнішою технологією для реалізації серверної частини є Java EE та фреймворк Spring.

Для розробки даного програмного забезпечення було обрано наступний стек технологій:

Основні технології:

- RESTful API – набір URI, HTTP викликів до цих URI і деяку кількість подань ресурсів в форматі JSON і / або XML, багато з яких будуть містити перехресні посилання [1];
- Spring Framework (або коротко Spring) – універсальний фреймворк з відкритим вихідним кодом для Java-платформи [2];
- MongoDB – документоорієнтована система управління базами даних з відкритим вихідним кодом, яка не потребує опису схеми таблиць [3];
- VueJS (версія 2.6.8), це прогресивний фреймворк JS для створення користувацьких інтерфейсів [4];
- Vuex (версія 3.1.0), патерн управління станом і бібліотека для додатків на Vue.js;
- Vue Router (версія 3.0.2), офіційна бібліотека маршрутизації для Vue.js;
- Vuetify – UI фреймворк для Vue.

Бібліотеки:

- Moment (версія 4.0.0), для роботи з часом і датами;
- Vue-cookie – бібліотека для роботи із cookie-файлами;
- Vuelidate – бібліотека для перевірки правильності вводу форми;
- Sockjs-client – бібліотека для реалізації технології спілкування клієнта із сервером у реальному часі [5];

Репозиторій:

- Github

Планувальник:

- Webpack

Spring Framework (або коротко Spring) – універсальний фреймворк з відкритим вихідним кодом для Java-платформи. Незважаючи на те, що Spring Framework не забезпечує будь-яку конкретну модель програмування, він став широко поширеним в Java-співтоваристві головним чином як альтернатива і

заміна моделі Enterprise JavaBeans. Spring Framework надає велику свободу Java-розробникам в проектуванні; крім того, він надає добре документовані і легкі у використанні засоби вирішення проблем, що виникають при створенні додатків корпоративного масштабу. Зазвичай Spring описують як полегшену платформу для побудови Java-додатків, але з цим твердженням пов'язані два цікавих моменти [6].

VueJS – це прогресивний веб фреймворк JavaScript для створення користувацьких інтерфейсів. В відмінності від феймворків-монолітів, Vue створено придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня представлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторонніх додатків (SPA, Single-Page Applications), якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками [7].

Наступним не менш важливим етапом є вибір бази даних, для даного проекту було обрано MongoDB. MongoDB - документоорієнтованих система управління базами даних з відкритим вихідним кодом, яка не потребує опису схеми таблиць. Класифікована як NoSQL, використовує JSON-подібні документи і схему бази даних. Написана на мові C ++. Вся система MongoDB може представляти не тільки одну базу даних, що знаходиться на одному фізичному сервері. Функціональність MongoDB дозволяє розташувати кілька баз даних на декількох фізичних серверах, і ці бази даних зможуть легко обмінюватися даними і зберігати цілісність.

Одним з найважливіших аспектів побудови ефективного і швидкісного сайту є оптимізація і автоматизація. Для цього існує безліч рішень і одним з них є Webpack.

Webpack - це інструмент збірки веб-додатку, що дозволяє автоматизувати повторювані завдання, такі як складання і мініфікація CSS- і JS-файлів, запуск тестів, перезавантаження браузера і т.д. Тим самим Gulp прискорює і оптимізує процес веб-розробки. Він використовує Stream і є дуже швидким.

Для прикладу проект, який має близько тисячі файлів, GruntJS потрібно приблизно 2.5 секунди на збірку і 2 секунди на обробку autoprefixer. Gulp може виконати цю роботу за 0.5 секунди вигравочи у GruntJS мінімум в 4 рази.

Установка Webpack:

```
npm install -g webpack
```

Ініціалізація Webpack:

```
export default function(webpack, plugins, args, config,
taskTarget, browserSync) {
 let dirs = config.directories;
```

Для роботи з Webpack необхідно створити завдання, які відповідатимуть за кожну функцію. Необхідні завдання призначені для розробки і звичайно зміни файлів відстежуються на сервері за допомогою задачі ‘watch’:

```
webpack.task('watch', () => {
  if (!args.production) {
   webpack.watch([
    path.join(dirs.source, dirs.styles, '**/*.{scss,sass}'),
    path.join(dirs.source, dirs.modules, '**/*.{scss,sass}')
   ], ['sass']);  } });}
```

2 РОЗРОБКА АЛГОРИТМІВ РЕЗЕРВУВАННЯ ТА ВЗАЄМОЗВ'ЯЗКІВ МІЖ ФУНКЦІОНАЛЬНИМИ ОБ'ЄКТАМИ

2.1 Розробка структури веб-сервісу та бази даних

Багато просунутих користувачів інтернет-мережі до сих пір не можуть визначити, чим відрізняється додаток від сайту. Напевно, це один з найпоширеніших запитів щодо веб-розробки, які задають користувачі. І правда, різниця розмита настільки, що навіть сам творець перед проектуванням не завжди знає, що він буде розробляти. Найпоширеніша відповідь: це майже одне і теж. Тільки сайт не завжди додаток, а додаток завжди сайт. Якщо ви знайомі з колами Ейлера, то такий малюнок зможе вам трохи прояснити ситуацію (рис.2.1). Деякі веб-сайти є веб-додатками, в той час як всі додатки – сайти. Їх відмінність можна визначити тим, що сайт – це інформаційна сторінка. На ній розміщені дані для читання і обмежений функціонал, який забезпечує доставку цієї інформації користувачеві. При створенні структури сайту-недодатку увага приділяється лише тому, щоб користувач знайшов сторінку в мережі і при використанні відчував себе комфортно [8].

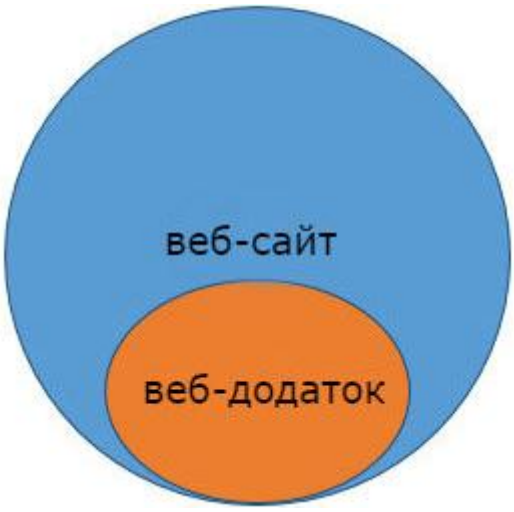

Рисунок 2.1– Порівняння веб-сайту та веб-додатку

Проектуючи додаток, не можна забувати, що воно повинно виконувати і певні бізнес-функції. Наприклад, купівля квітів, замовлення послуг та інше. Для додатку характерна динаміка – сторінка в браузері змінюється відповідно з особистістю користувача і його поведінковими реакціями. Для цього потрібно правильно облаштувати архітектуру системи «питання-відповідь», яка і буде частиною web-структури. Але динамічний веб-сайт – норма для нашого часу. «Статика» використовується лише на сайтах-візитках, але навіть їх намагаються прикрасити динамічними елементами. Звичайно, якщо є динаміка, побудована на HTML-файлах, які посилаються один на одного, це можна називати «чистим» сайтом. Але в цілому, якщо говорити про сучасну web-структуру, справжньої різниці між сайтом і додатком просто не існує.

Програму з хорошою архітектурою легше розширювати і змінювати, а також тестувати, налагоджувати і розуміти. Тобто, насправді можна сформулювати список цілком розумних і універсальних критеріїв:

- ефективність системи. В першу чергу програма, звичайно ж, повинна вирішувати поставлені завдання і добре виконувати свої функції, причому в різних умовах. Сюди можна віднести такі характеристики, як надійність, безпеку, продуктивність, здатність справлятися зі збільшенням навантаження (масштабованість) і т.п;
- гнучкість системи. Будь-який додаток доводиться міняти з часом – змінюються вимоги, додаються нові. Чим швидше і зручніше можна внести зміни в існуючий функціонал, чим менше проблем і помилок це викличе - тим гнучкіша і конкурентоздатна система;
- можливість розширення системи. Можливість додавати в систему нові сутності та функції, не порушуючи її основної структури. На початковому етапі в систему має сенс закладати лише основний і самий необхідний функціонал (принцип YAGNI - you is not gonna need it, «Вам це не знадобиться») Але при цьому архітектура повинна дозволяти легко нарощувати додатковий функціонал в міру необхідності. Причому так, щоб внесення найбільш ймовірних змін вимагало найменших зусиль;

- масштабованість процесу розробки. Можливість скоротити термін розробки за рахунок додавання до проекту нових людей. Архітектура повинна дозволяти розпаралелити процес розробки, так щоб безліч людей могли працювати над програмою одночасно;
- тестованість. Код, який легше тестувати, буде містити менше помилок і надійніше працювати. Але тести не тільки покращують якість коду. Багато розробників приходять до висновку, що вимога «хорошою тестованості» є також спрямовуючою силою, автоматично веде до хорошого дизайну;
- можливість повторного використання. Систему бажано проектувати так, щоб її фрагменти можна було повторно використовувати в інших системах [9].

База даних слугує основою для створення і розвитку всіх необхідних функції веб-додатку. Також важливо розробити базу даних, яка буде захищена від можливих атак, збережені дані повинні бути добре зашифровані.

Основні функціональні можливості:

- візуалізація залу;
- зручний інструмент для створення плану залу;
- зв’язок користувачів із сервером в режимі реального часу для оповіщення їх про зміни у списку замовлень;
- можливість переглядання наявних замовлень для кожного столика;
- можливість переглядання списку замовлень на конкретну дату;
- система репутації клієнта;

Так як була вибрана нереляційна СУБД – MongoDB, вона має ряд відмінностей перед звичайними реляційними БД (табл. 2.1). Надалі використовуватимуться саме ці визначення.

Таблиця 2.1 – Відмінності визначень MongoDB від SQL

SQL	MongoDB
Таблиця	Колекція
Ряд	Документ
Стовпець	Поле
Первинний ключ	ObjectId
Вкладена таблиця	Вбудований документ

Вся суть NoSQL баз даних полягає у тому, що немає жорсткого планування структури і зв'язку між таблицями, як в SQL. В цьому і полягає їхня основна перевага. Можна з легкістю додати в колекцію новий документ, у якого є менше чи більше полів, у такому випадку не виникне ніякої помилки, якщо це не забороняє валідація на стороні серверу. Також не потрібно створювати вручну саму Базу Даних чи колекції. При першому запиті, якщо колекції чи БД не існуватиме – вона створиться автоматично.

Для того, щоб зрозуміти структуру будуть використані скріншоти колекцій то документів із графічного користувацького інтерфейсу для СУБД MongoDB - MongoDB Compass Community (рис. 2.2)

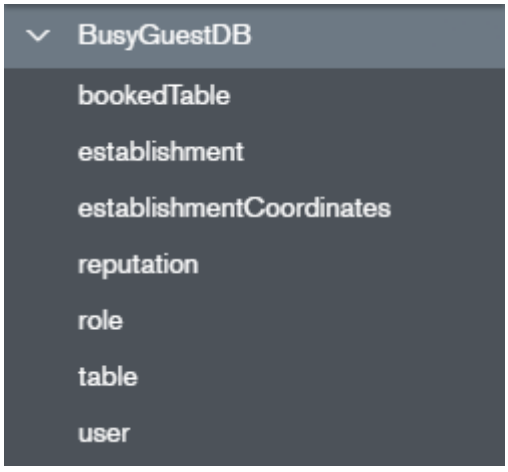

Рисунок 2.2 – Структура бази даних

MongoDB реалізує новий підхід до побудови баз даних, де немає таблиць, схем, запитів SQL, зовнішніх ключів і багатьох інших речей, які притаманні

об'єктно-реляційних баз даних. З часів динозаврів було звичайною справою зберігати всі дані в реляційних базах даних (MS SQL, MySQL, Oracle, PostgresSQL). При цьому було не так важливо, а чи підходять реляційні бази даних для зберігання даного типу даних чи ні. На відміну від реляційних баз даних MongoDB пропонує документо-орієнтовану модель даних, завдяки чому MongoDB працює швидше, має кращу масштабованість, її легше використовувати. Але, навіть враховуючи всі недоліки традиційних баз даних і гідності MongoDB, важливо розуміти, що завдання бувають різні і методи їх вирішення бувають різні. В якійсь ситуації MongoDB дійсно поліпшить продуктивність вашої програми, наприклад, якщо треба зберігати складні за структурою дані. В іншій же ситуації краще буде використовувати традиційні реляційні бази даних. Крім того, можна використовувати змішані підхід: зберігати один тип даних в MongoDB, а інший тип даних – в традиційних БД [10].

База даних складається зі 7 документів:

- замовлений столик;
- заклад;
- координати столиків;
- репутація клієнта;
- ролі;
- столик;
- користувач.

Кожна таблиця містить поле id, яке слугує ідентифікатором для кожного документу. Значенням поля id завжди є автоматично згенерований унікальний ключ за спеціальним алгоритмом. Значення поля _id ніколи не повторюється. Перша колекція User буде слугувати для збереження даних про зареєстрованих користувачів. Схематичний вигляд рядків в першій колекції (рис. 2.3).

Кожен документ таблиці з даними про користувачів буде складатись з таких атрибутів:

- _id – унікальний ідентифікатор документу, згенерований випадково;

- name – повне ім'я користувача;
- userpic – посилання на фото;
- email – електронна пошта;
- reputation – репутація користувача (якщо це клієнт);
- phone – номер телефону користувача;
- role – роль користувача.

Рисунок 2.3 – Колекція User в базі даних

Другою є колекція під назвою Establishment призначена для збереження інформації про заклади. Схематичний вигляд документу в другій колекції (рис. 2.4).

Кожен документ з даними про зняття буде складатись з таких атрибутів:

- _id – унікальний ідентифікатор документу, згенерований випадково;
- email – емейл адреса закладу, яка є також унікальним полем і зв'язує аккаунт із закладом;
- address – адреса закладу;
- phone – список номерів телефону закладу;
- workSchedule – графік роботи закладу;
- descriprion – опис закладу;
- show – прапорець, який дозволяє показувати/сховувати заклад;
- accepted – прапорець, який показує чи підтверджений заклад адміністрацією.

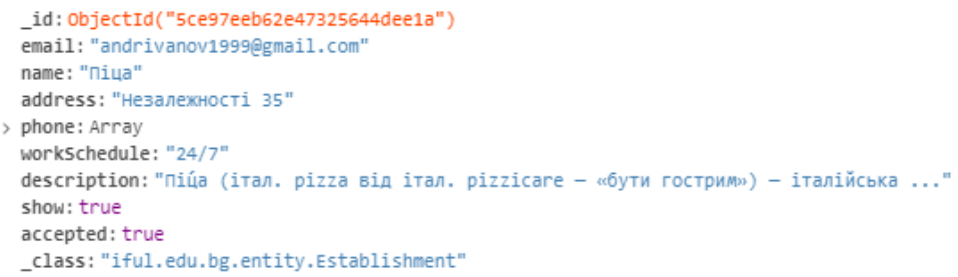

Рисунок 2.4 – Колекція Establishment в базі даних

Щоб вказувати столик, який клієнт хоче зарезервувати, також потрібно колекція (рис. 2.5).

Колекція зі списком всіх столиків матиме назву Table та буде складатись з таких атрибутів:

- _id – унікальний ідентифікатор документу, згенерований випадково;
- estb – вкладений документ закладу у якому столик замовлений;
- tableNum – номер столика;
- seats – кількість місць за столиком;
- status – стан столика на даний момент.

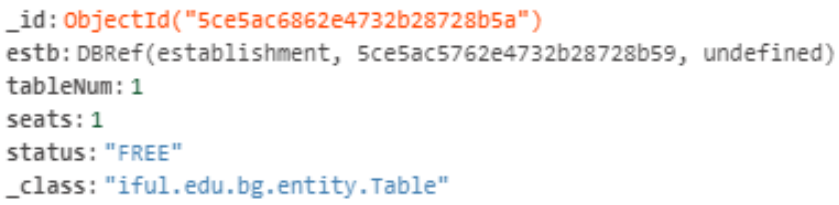

Рисунок 2.5 – Колекція Tablse в базі даних

Колекція BookedTable є однією із ключових. Саме вона зберігає у собі всі дані про замовлення столика. Користувачу потрібно вибрати заклад, столик та бажану дату, а адміністратору закладу потрібно буде опрацювати замовлення і підтвердити його. Схематичний вигляд колекції (рис. 2.6).

Колекція складатиметься із семи атрибутів:

- _id – унікальний ідентифікатор документу, згенерований випадково;
- table – вкладений документ столика, який замовлено;

- estb – вкладений документ закладу у якому замовлено столик;
- user – вкладений документ користувача, який зробив замовлення;
- bookedOn – дата та час, на яку замовлено столик;
- comment – особливі побажання клієнта;
- accepted – прапорець, який вказує чи замовлення прийняте;
- done – прапорець, який вказує чи замовлення виконане.

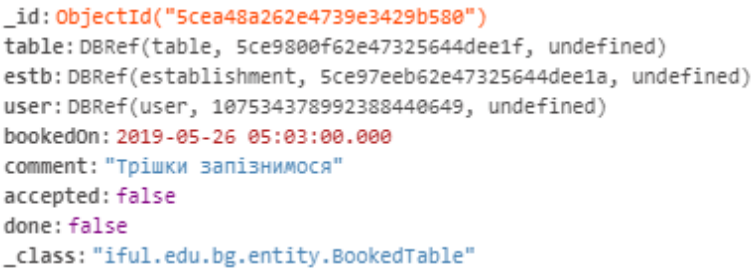

Рисунок 2.6 – Колекція BookedTable в базі даних

Наступна колекція необхідна для збереження інформації про наявні ролі у систему. Таблиця матиме назву Role. Схематичний вигляд документу в п’ятій колекції із наявними ролями в базі даних (рис. 2.7).

Колекція складатиметься з двох атрибутів:

- _id – унікальний ідентифікатор документу, згенерований випадково;
- name – назва ролі.

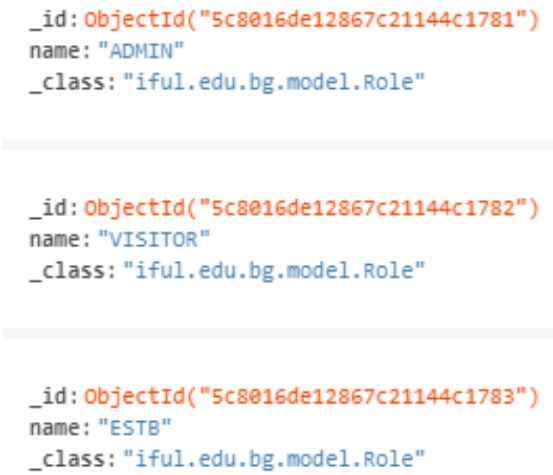

Рисунок 2.7 – Колекція Role в базі даних

Наступна колекція слугуватиме для збереження інформації про репутацію клієнта. При реєстрації для кожного нового клієнта буде створюватися запис у колекції Reputation для подальшого ведення репутації клієнта і визначення ступеня його надійності. Схематичний вигляд атрибут в шостій колекції (рис. 2.8).

Колекція складатиметься з 4 атрибутів:

- _id – унікальний ідентифікатор документу, згенерований випадково;
- name – поле, яке вказує чи надійний клієнт;
- orders – кількість замовлень;
- successfulOrders – кількість виконаних замовлень;

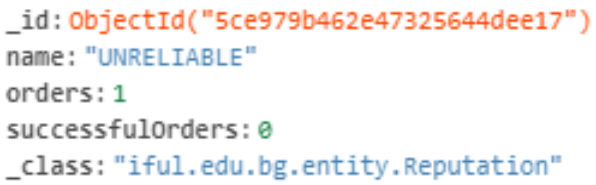

Рисунок 2.8 – Колекція Reputation в базі даних

Остання колекція має назву EstablishmentCoordinates і зберігає в собі координати і параметри столиків, які потрібно зберігати для коректного відображення їх у візуалізаторі залу закладу.

Схематичний вигляд атрибутів в сьомій колекції (рис. 2.9).

Колекція складатиметься із п’яти атрибутів:

- id – унікальний ідентифікатор документу, згенерований випадково;
- width – ширина залу;
- height – довжина залу;
- tables – параметри столика, збережені у форматі «Ключ-Значення»;
- signs – параметри допоміжних знаків, такі як «WC» та «Exit».

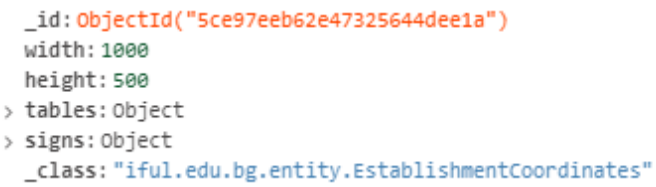

Рисунок 2.9 – Колекція EstablishmentCoordinates в базі даних

У підсумку ми отримали 7 колекцій у базі даних, які надійно зберігатимуть дані та швидко реагуватимуть на маніпуляції користувача завдяки потужності СУБД MongoDB.

2.2 Розробка структури інтерфейсу користувача

Реалізовуючи інтерфейс користувача веб-сервісу слід чітко розуміти вимоги, які потрібно виконувати. Для полегшення процесу розробки прийнято створювати Use Cases. Юзкейс – це перелік дій, сценарій по якому користувач взаємодіє з додатком, програмою для виконання якої-небудь дії для досягнення конкретної мети. Тестування по юзкейсам проводиться для того щоб виявити додаткові логічні діри і баги в додатку, які складно знайти в тестуванні індивідуальних модулів, частин додатка окремо один від одного [11].

На рисунку 2.10 зображено юзкейс для можливостей всіх типів користувачів у сервісі.

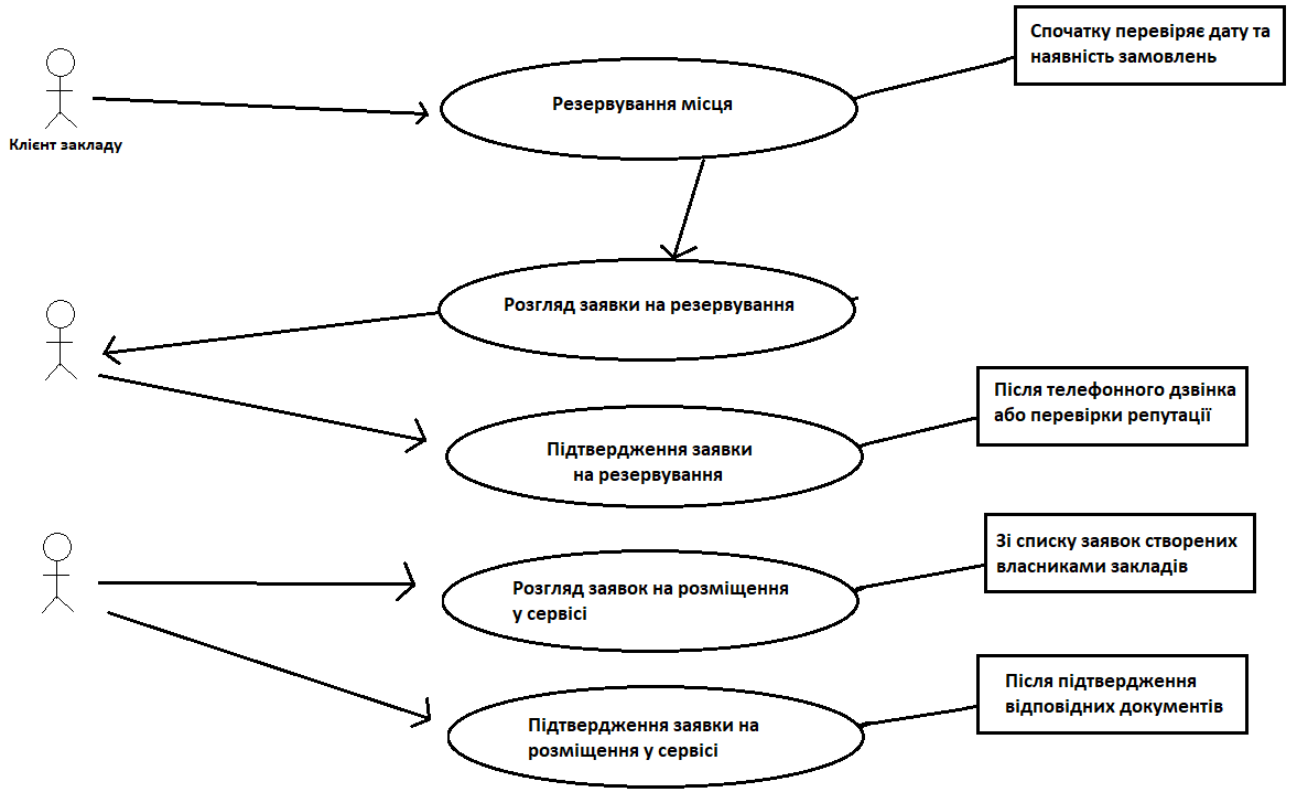

Рисунок 2.10 – Юзкейс для можливостей типів користувачів сервісу

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них [12].

Клієнт-серверну архітектуру можна означити, як концепцію інформаційної мережі в якій основна частина її ресурсів зосереджена в серверах, обслуговуючих своїх клієнтів (рис. 2.11). Така архітектура визначає такі типи компонентів:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами [13].

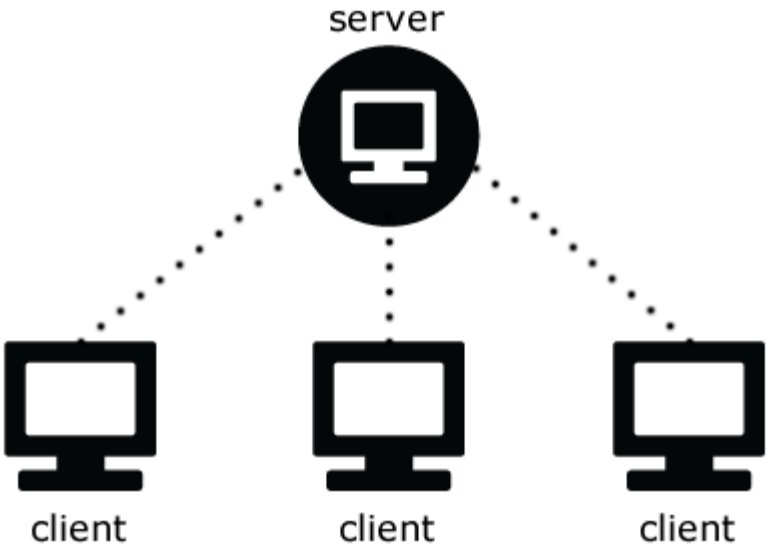

Рисунок 2.11 – Клієнт-серверна архітектура

Для з'єднання клієнтського і серверного рівня між собою, а вони знаходяться на різних хостингах, додамо в файл index.html, який повертається контролером, підключення до скрипту, який знаходиться на хостингу разом із фронтендом:

```
<script th:src="${isDevMode} ? 'http://localhost:3000/main.js' :
'/js/main.js'"></script>
```

У самому підключенні скрипту є перевірка на те, чи у продакшині проект чи ні, якщо так, то файл береться із хостингу. У файлі main.js відбувається підключення усіх бібліотек у проект:

```
import Vue from 'vue'
import 'api/resource'
import App from 'pages/App.vue'
import '@babel/polyfill'
import router from 'router/router.js'
import store from 'store/store.js'
import { connect } from './util/ws'
```

```
import Vuetify from 'vuetify'
import 'vuetify/dist/vuetify.min.css'
var VueCookie = require('vue-cookie');
import Vuelidate from 'vuelidate'
import VueDraggableResizable from 'vue-draggable-resizable'
import 'vue-draggable-resizable/dist/VueDraggableResizable.css'

Vue.component('vue-draggable-resizable', VueDraggableResizable)
Vue.use(Vuelidate)
Vue.use(VueCookie);
Vue.use(Vuetify)
connect()
Vue.use(require('vue-moment'));
new Vue({
  el: '#app',
  router,
  store,
  render: a => a(App),
})
```

Після підключення можна користуватися усіма підключеними бібліотеками без обмежень.

При запуску сайту, користувач потрапляє у файл App.vue, де й відбуваються підключення всіх інших файлів, перенаправлення і тд.

```
  <v-app>
       <v-toolbar app>
          <v-toolbar-title>BusyGuest</v-toolbar-title>
          <v-toolbar-items class="hidden-sm-and-down">
             <v-btn flat>
               <router-link to="/">Home</router-link>
             </v-btn>
           </v-toolbar-items>

          <v-spacer></v-spacer>
         <div v-if="profile">
          <span>{{profile.name}}</span>
           <a href="/logout">Вийти</a>
         </div>
       </v-toolbar>

       <v-content>
          <div v-if="!profile">
              <v-btn color="primary" href="/login">Вхід</v-btn>
              <v-btn color="primary" @click="createEstb">Вхід для
підприємців</v-btn>
         </div>
         <div v-if="role=='VISITOR'">
          <books></books>
```

```
        </div>
        <v-container fluid>
         <sign-up-stepper v-if="profile && profile.phone=='' ||
showStepper"></sign-up-stepper>
             <div v-else>
               <div>
                 <router-view></router-view>
               </div>
             </div>
              </v-container>
      </v-content>
  </v-app>
```

На початку файлу йде підключення хедеру за допомогою тегу <v-toolbar> </v-toolbar>, далі у тегу <v-content></v-content> йде підключення усього, що стосується контенту: списку активних замовлень клієнта (тег компоненту <books></books>), тегу <router-view></router-view>, який/ підвантажує і відображає потрібний компонент.

```
const routes = [
  {path: '/establishment/:id', component: Establishment, props:
true},
  {path: '/establishment', component: CreateEstablishment, props:
true},
  {path: '/', component: EstablishmentsList, props: true},
  {path: '/bookedTable/:id', component: BookedTable, props: true},
  {path: '/admin', component: AdminPanel, props: true},

  { path: '*', component: EstablishmentsList}
 ]
```

Як видно із коду вище, для кожного шляху router-view буде підвантажувати інший компонент. Таким чином на головній сторінці буде показуватися компонент EstablishmentsList.vue, який відповідає за показ списку усіх закладів.

Код EstablishmentsList.vue:

```
<template>
     <v-flex xs12 sm6 offset-sm3>
          <h2>Список закладів</h2>
          <v-list two-line>
               <template v-for="establishment in establishments" >
```

```
                <establishment-row :establishment="establishment"
:key="establishment._id"/>
            </template>

          </v-list>
     </v-flex>
</template>
```

У середині файлу EstablishmentsList.vue, функція циклу v-for створює файл EstablishmentRow.vue для кожного закладу окремо, де вже у середині заповнюється інформація індивідуально до кожного файлу. Такий підхід проектування допомогає розбивати код на більшу кількість файлів, що робить код більш зрозумілим і логічно розділеним. Файл EstablishmentRow.vue:

```
<template>
   <div v-show="establishment.show">
     <div v-if="role=='ESTB'">
          <div v-if="profile.email===establishment.email">
          {{this.$router.push(`/establishment/`+establishment._id)}}
      </div>
      </div>
      <div v-else>
       <v-list-tile :key="establishment.name" avatar
@click="pushToEstb">
        <v-list-tile-avatar>
         <img :src="profile.userpic">
        </v-list-tile-avatar>
        <v-list-tile-content>
         <v-list-tile-title v-html="establishment.name"></v-list-
tile-title>
         <v-tooltip bottom>
                    <template v-slot:activator="{ on }">
               <v-list-tile-sub-title v-on="on" v-
html="establishment.description"></v-list-tile-sub-title>
           </template>
                  <span>{{establishment.description}}</span>
                </v-tooltip>
       </v-list-tile-content>
      </v-list-tile>
      </div>
    </div>
</template>
```

Для переходу після кліку на заклад зі списку використовується функція goToEstb(), яка виконує доступ до об'єкту $router та за допомогою методу push()

вводить у об'єкт router шлях до закладу, а компонент router-view переправляє на файл Establishment.vue. Код методу goToEstb():

```
goToEstb(){
   this.$router.push(`/establishment/`+this.establishment._id)
}
```

Потрапивши на сторінку закладу код програми визначає ваші права та показує відповідний функціонал. Так, якщо ви клієнт, то вам недоступні функції редагування даних закладу, додавання чи маніпулювання столами. Також, якщо користувач наділений правами закладу, то йому не доступні функції переглядання інших закладів, окрім свого. Для цього також передбачені перевірки. Заклад може приховувати свою сторінку, у такому разі його не зможе ніхто побачити, окрім власника і адміністрації. Код файлу Establishment.vue:

```
<template>
  <v-app>
     <div v-if="role=='ADMIN'">
          <v-btn
@click="acceptEstablishment">{{establishment.accepted ? 'Заховати'
: 'Підтвердити'}}</v-btn>
     </div>
     <div v-if="establishment.show || role!='VISITOR'">
     <div v-if="role==='ESTB' &&
profile.email!=establishment.email">
          Не ваш заклад!
            </div>
             <div v-else>
               <div v-if="role==='ESTB' && establishment.accepted">
                    <div v-if="!establishment.show">
                         <h3>Заклад не оприлюднено! Після заповнення
всієї інформації, не забудьте опублікувати!</h3>
                         <input type="button"
@click="changeShowStatus()" value="Опублікувати">
                    </div>
                    <div v-else>
                         <h3>Заклад оприлюднено ^ ^</h3>
                         <input type="button"
@click="changeShowStatus()" value="Заховати">
                    </div>
               </div>

               <div v-if="role==='ESTB' && !establishment.accepted">
                    <h3>Заклад знаходиться на розгляді в
адміністрації!</h3>
               </div>
```

```
            <div>
                  <div style="text-align: center;">
                       <h1 style="display: inline;" v-
show="editElementId != 'name'" id="name">
                            {{establishment.name}}
                       </h1>
                       <span v-show="editElementId === 'name'">
                            <input style="border: 2px solid
#87CEFA; font-size: 30px;" type="text" v-
model="establishment.name">
                       </span>
                       <button v-if="role==`ESTB`" v-
on:click="edit('name')"><v-icon color="green">edit</v-
icon></button>
                  </div>

                            <v-container fluid grid-list-md >
                              <v-layout align-start justify-space-
around fill-height wrap>
                               <v-flex xs6 order-lg2>
                                <v-carousel>
                                        <v-carousel-item
                                         v-for="(item,i) in items"
                                         :key="i"
                                         :src="item.src"
                                        ></v-carousel-item>
                                       </v-carousel>
                               </v-flex>
                               <v-flex xs6 order-lg2>
                                <establishment-info
:establishment="establishment"></establishment-info>
                               </v-flex>
                              </v-layout >
                             </v-container>

             </div>
                  <h3>Замовити столик: </h3>

            <tables-list :establishment="establishment"></tables-
list>
          </div>
      </div>

      <div v-else>
         Заклад не оприлюднено!
      </div>
  </v-app>
    </template>
```

Як можна побачити із прикладу коду, внизу вказано тег <tables-list>, який посилається на файл TablesList.vue, котрий генерує файли TablesRow.vue, які у

свою чергу заповнюють даними конкретного стола. Загалом принцип такий же, як і з файлом EstablishmentsList.vue. Генерація списку замовлень для кожного столу відбувається так само.

Окремо хотілося б відзначити процес реєстрації користувачів. Для найбільшої зручності було прийнято рішення використовувати реєстрацію за допомогою OAuth технології сервісу Google. Це допомагає обходити заповнення майже усіх потрібних даних користувачем вручну. Для збереження актуальності номеру, що є дуже важливим, так як при низькій репутації, оператор телефонуватиме для підтвердження замовлення, було прийнято рішення використовувати stepper. Код файлу SignUpStepper.vue нижче:

```
<template>
     <v-stepper v-model="e6" vertical>
       <v-stepper-step :complete="e6 > 1" step="1">
        Залогіньтесь через зручну соцмережу:
       </v-stepper-step>

       <v-stepper-content step="1">
        <v-card color="grey lighten-1" class="mb-5" height="50px"
width="300px">
          <a v-if="!profile" href="/login"
@click="saveStep(1)">Google</a>
          <span v-else>{{profile.name}}, ви успішно залогінилися!
Переходься то наступного кроку.</span>
        </v-card>
        <v-btn v-show="profile" color="primary" @click="e6 =
2">Continue</v-btn>
        <v-btn flat @click="closeStepper()">Cancel</v-btn>
       </v-stepper-content>

       <v-stepper-step :complete="e6 > 2" step="2">Вкажіть номер
телефону</v-stepper-step>

       <v-stepper-content step="2">
        <v-card color="grey lighten-1" class="mb-5" height="200px">
             <span v-if="incorrectPhone">{{incorrectPhone}}</span>
             <v-flex xs12 sm6 md3>
               <v-text-field
                label="Solo"
                placeholder="Номер телефону..."
                solo
                v-model="phone" type="tel" mask="(+380) #########"
               ></v-text-field>
              </v-flex>
```

```
            <input type="tel" id="phone" name="phone" pattern="[0-
9]{10,13}" required>
        </v-card>
        <v-btn color="primary" @click="inputPhone()">Ввести</v-btn>
        <v-btn flat @click="closeStepper()">Cancel</v-btn>
       </v-stepper-content>
      </v-stepper>
     </template>
```

Після логіну через сервіс Google, перевіряється наявність номеру телефону, якщо його немає, то пропонується ввести. Після введення номеру телефону компонент більше ніколи не буде показуватися користувачу. Якщо мова йде про підприємця, який бажає додати свій заклад до системи, то йому потрібно входити у систему через призначену для цього кнопку. Система зареєструє його як заклад, також запропонує вказати свій актуальний номер телефону. Після вводу телефону, користувача перенаправить на сторінку заповнення даних про свій заклад. Одразу ж після натискання кнопки «Відправити», заявка відішлеться на розгляд адміністрації. До її підтвердження клієнти не можуть побачити заклад у списку. Лістинг файлу CreateEstablishment.vue:

```
<template>
     <v-stepper v-model="e6" vertical>
       <v-stepper-step :complete="e6 > 1" step="1">
        Залогіньтесь через зручну соцмережу:
       </v-stepper-step>

       <v-stepper-content step="1">
        <v-card color="grey lighten-1" class="mb-5" height="50px"
width="300px">
         <a v-if="!profile" href="/login"
@click="saveStep(1)">Google</a>
         <span v-else>{{profile.name}}, ви успішно залогінилися!
Переходься то наступного кроку.</span>
        </v-card>
        <v-btn v-show="profile" color="primary" @click="e6 =
2">Continue</v-btn>
        <v-btn flat @click="closeStepper()">Cancel</v-btn>
       </v-stepper-content>

       <v-stepper-step :complete="e6 > 2" step="2">Вкажіть номер
телефону</v-stepper-step>
```

```
      <v-stepper-content step="2">
       <v-card color="grey lighten-1" class="mb-5" height="200px">
            <span v-if="incorrectPhone">{{incorrectPhone}}</span>
            <v-flex xs12 sm6 md3>
              <v-text-field
               label="Solo"
               placeholder="Номер телефону..."
               solo
               v-model="phone" type="tel" mask="(+380) #########"
              ></v-text-field>
             </v-flex>
            <input type="tel" id="phone" name="phone" pattern="[0-
9]{10,13}" required>
        </v-card>
        <v-btn color="primary" @click="inputPhone()">Ввести</v-btn>
        <v-btn flat @click="closeStepper()">Cancel</v-btn>
       </v-stepper-content>
      </v-stepper>
     </template>
```

Як можна зрозуміти у висновку фреймворк Vue.js дуже спрощує розробку клієнтської частини, якщо використовувати всі його можливості на повну.

2.3 Розробка функціональної частини на стороні серверу

Індустрії програмного забезпечення притаманна властивість, що будь-який термін може мати багато суперечливих трактувань. Ймовірно, найбільш абстрактним виявилось поняття архітектура. Зрештою, можна назвати два загальних варіанта. Перший пов’язаний з розбиттям системи на найбільш значимі складові частини; в другому випадку маються на увазі деякі конструктивні рішення, котрі після їх прийняття важко піддаються внесенню змін. Також, росте розуміння того, що існує більше одного способу описання архітектури і ступінь важливості кожного з них змінюється з плином життєвого циклу системи.

В даному розділі представлено структуру системи після декомпозиції у вигляді архітектурних шарів (layers). В більшості корпоративних додатків відслідковується та чи інша форма архітектурного «розшарування», але в деяких ситуаціях більшого значення можуть набувати інші підходи, пов’язані,

наприклад з організацією каналів (pipes) або фільтрів (filters). Однак увага у даному розділі концентрується на архітектурі шарів як на найбільш плідній структурній моделі.

Model-View-Controller – схема поділу даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, уявлення і контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно (рис. 2.12).

- модель (Model) надає дані і реагує на команди контролера, змінюючи свій стан;
- подання (View) відповідає за відображення даних моделі користувачеві, реагуючи на зміни моделі;
- контролер (Controller) інтерпретує дії користувача, сповіщаючи модель про необхідність змін [14].

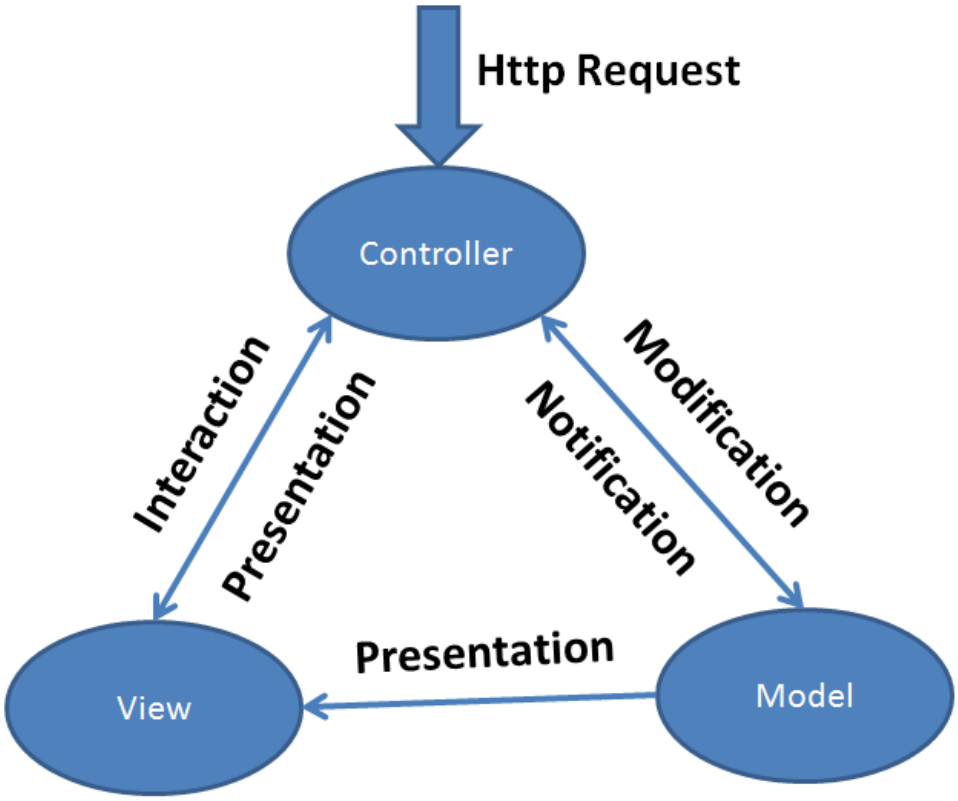

Рисунок 2.12 – Схема поділу даних MVC

REST – це архітектура для розробки веб-сервісів, що намагається імітувати архітектури, котрі використовують HTTP чи схожі протоколи, шляхом обмеження інтерфейсів набором стандартних і добре знайомих розробникам операцій (наприклад, GET, PUT/PATCH, POST та DELETE для HTTP).

Акцент зроблений на взаємодії з ресурсами, що мають збережений стан, а не з повідомленнями та операціями. Можна сказати, що ця архітектура розроблена для того, щоб показати, що існуючого HTTP достатньо для побудови веб-сервісу, та демонстрації масштабування.

Головною ідеєю REST було скоріш використання добре розвиненого HTTP для передачі даних між пристроями, ніж використання протоколу, побудованого поверх рівня HTTP, для передачі повідомлень. Додаток, розроблений слідуючи принципам даного архітектурного підходу, буде використовувати HTTP для виконання викликів між пристроями не спираючись на складні об’єктні механізми на кшталт CORBA (Common Object Request Broker Architecture), RPC (Remote Procedure Call), чи SOAP. Відповідно, REST-додатки використовують функції HTTP запиту для надсилання, зчитування та видалення даних, використовуючи повну функціональність HTTP CRUD (Create, Read, Update і Delete) операцій. До того ж REST може використовувати HTTPS, надаючи безпечне передавання даних [15].

CRUD операції разом з функціями HTTP REST та відповідними SQL операціями показані у таблиці 2.2. На відміну від SOAP, REST принципи легкі для розуміння та застосування розробнику, котрий знайомий з використанням HTTP.

Таблиця 2.2 – Відповідність HTTP-функцій SQL-операціям

CRUD-операції	Ключові слова REST (HTTP)	Оператори SQL (база даних)
CREATE – створити чи додати нові сутності	POST	INSERT
UPDATE – оновити чи редагувати існуючі дані	PUT	UPDATE
READ – зчитати, отримати дані	GET	SELECT
DELETE – видалити інсуючі дані	DELETE	DELETE/DROP

Для спрощення розробки алгоритмів на стороні серверу створюють блок-схеми. Блок-схема – це графічне зображення алгоритму у вигляді пов'язаних між собою за допомогою стрілок (ліній переходу) і блоків - графічних символів, кожен з яких відповідає одному кроку алгоритму [16]. Усередині блоку дається опис відповідного дії. На рисунку 2.13 зображена блок-схема входу на сайт. Блок-схема резервування столу – рисунок 2.14.

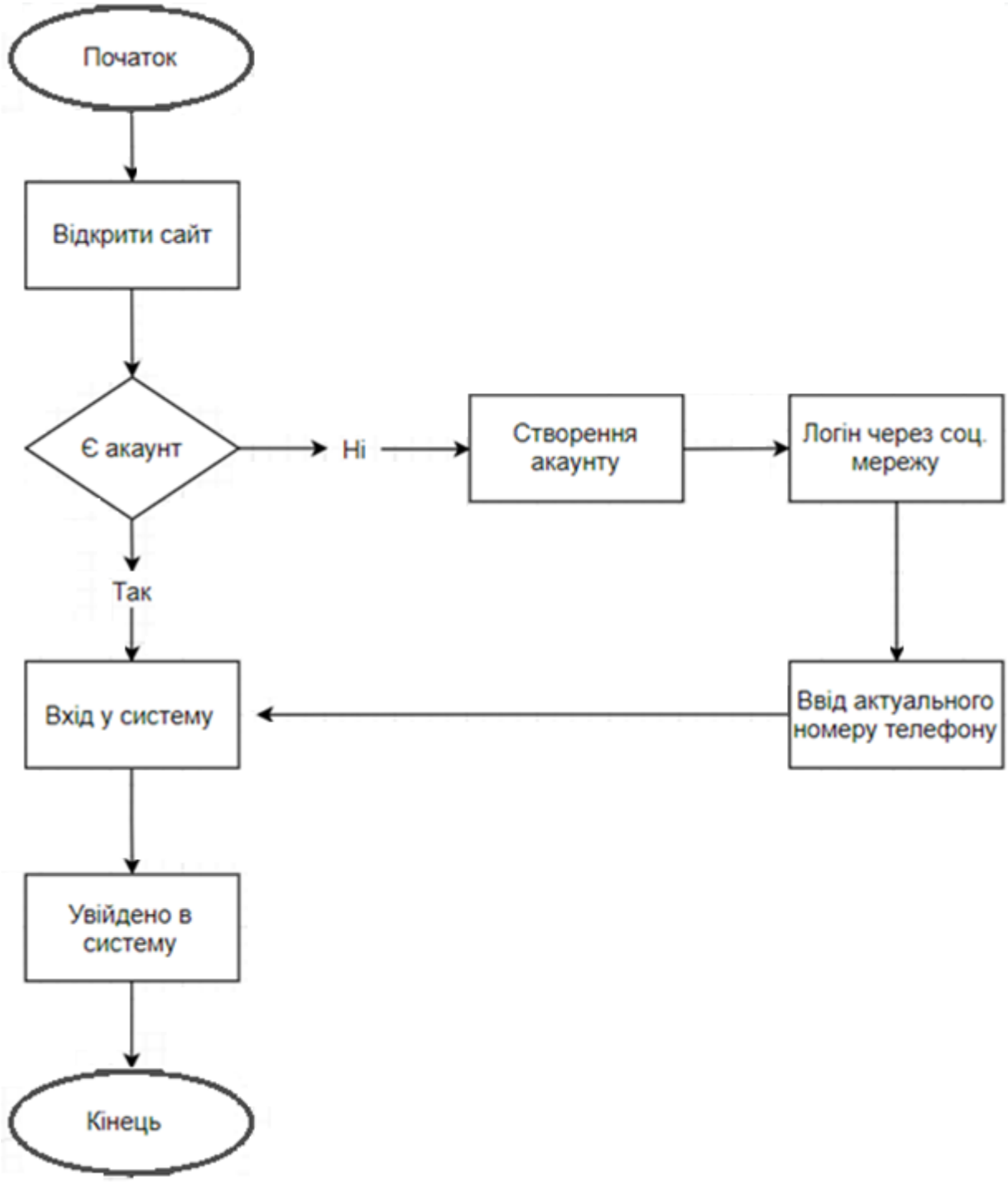

Рисунок 2.13 – Блок-схема входу на сайт

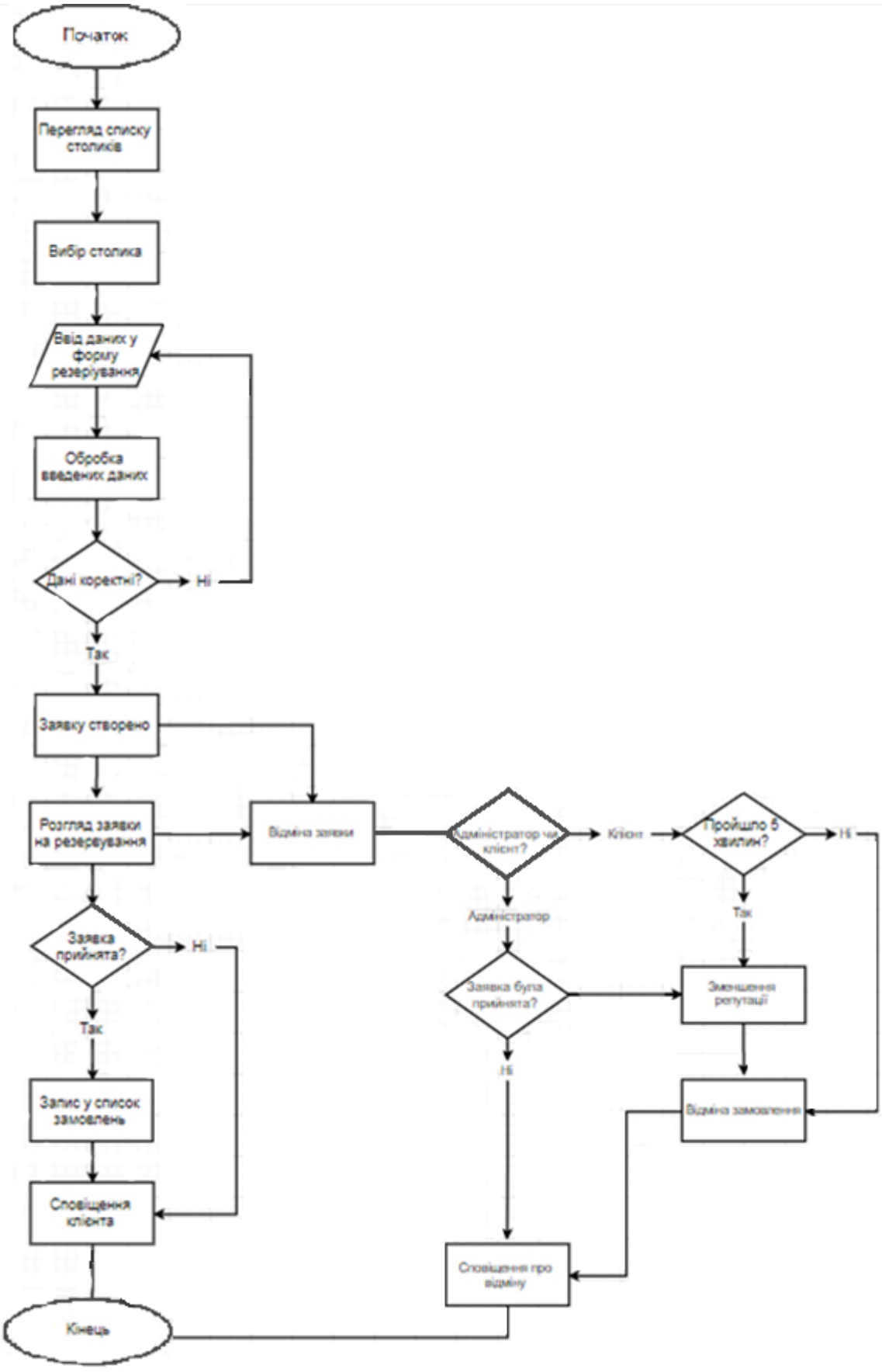

Рисунок 2.14 – Блок-схема алгоритму резервування столу

Для того, щоб серверна частина запустилася і розпізналася, як Spring Boot проект, нам потрібно включити анотацію @SpringBootApplication у клас, який запускає весь проект. Дана анотація заміняє собою анотації @Configuration, @EnableAutoConfiguration, та @ComponentScan. Приклад коду:

```
@SpringBootApplication(scanBasePackages = "iful.edu.bg")
public class BusyGuestApplication {
     public static void main(String[] args) {
          SpringApplication.run(BusyGuestApplication.class, args);
     }
}
```

При переході на домашню сторінку, а саме за шляхом «/» серверна частина обробляє запит, записує деякі дані в об’єкт Model та повертає користувачу файл index.html, з цього моменту користувацька частина працюватиме із Vue.js та серверна частина не займатиметься обробкою html-сторінок. В об’єкт Model записуються такі дані, як дані про користувача «profile» та роль користувача «role». Цей запит обробляється контролером. Для позначення класу, як контролера, потрібно використати анотацію @Controller. Анотація @Controller вказує, що даний клас, як багато здогадуються, грає роль контролера. Тому немає необхідності успадкування будь-якого базового класу контролера або використання на Servlet API. Хоча, звичайно, можливості класу Servlet також підтримується. Основна мета анотації @Controller – призначати шаблон даного класу, показуючи його роль. Це означає, що диспетчер буде сканувати Controller-класи на предмет реалізованих методів, перевіряючи @RequestMapping анотації [17]. Приклад коду:

```
     @Controller
     @RequestMapping("/")
     public class MainController {
          @Autowired
```

```
        private EstablishmentServiceImpl establishmentServiceImpl;

        @Autowired

        private UserRepository userRepository;

        @Autowired

        private TableRepository tableRepository;

        @GetMapping

        public String main(Model model, @AuthenticationPrincipal
User user) throws Exception {

            HashMap<Object, Object> data = new HashMap<>();

            if (user != null) {

                Optional<User> usse =
userRepository.findById(user.getId());

                user.setPhone(usse.get().getPhone());

                data.put("profile", user);

                data.put("role", user.getRole().getName());

            }

            model.addAttribute("frontendData", data);

            return "index";

        }

    }
```

Обробка усіх наступних запитів на сервер буде оброблятися REST контролером. Кожен контролер обов'язково має анотацію @Controller або @RestController, остання розширює @Controller і додає до нього @ResponseBody, тим самим кажучи, що всі методи класу повертають саме ті дані, які треба переслати клієнту, ігноруючи MVC частина (яку я теж розгляну окремо) . Тобто, ця анотація вказує фреймворку, що контролер буде повертати не готовий html-файл, а «сирі» дані у форматі json, які будуть оброблятися і відображатися вже на клієнтській частині за допомогою Vue.js. Для того, щоб контролер умів сереалізовувати та десереалізовувати дані у потрібному форматі,

у нашому випадку це json, потрібно використовувати відповідну бібліотеку. У випадку із фреймворком Spring Boot у ньому вже включена бібліотека Jackson, яка призначена саме для цього. Приклад REST контролера:

```
@RestController

@RequestMapping("/visitor")

public class VisitorController {

    private final BookedTableServiceImpl
bookedTableServiceImpl;

    public VisitorController(BookedTableServiceImpl
bookedTableServiceImpl) {

        super();

        this.bookedTableServiceImpl = bookedTableServiceImpl;

    }

    @GetMapping("bookedTable/{id}")

    public BookedTable getBookedTable(@PathVariable("id")
String id) throws Exception {

        return bookedTableServiceImpl.getBookedTableById(id);

    }

    @GetMapping("{tableId}/bookedTable")

    public List<BookedTable>
getEstablishmentBookedTable(@PathVariable("tableId") String id)
throws Exception {

        return
bookedTableServiceImpl.getActualBookedTableByTableId(id);

    }

    @GetMapping("{userId}/userBookedTable")

    public List<BookedTable>
getUserBookedTables(@PathVariable("userId") String id) throws
Exception {

        return
bookedTableServiceImpl.getBookedTableByUserId(id);

    }
```

```
}
```

Кожен метод у класі, котрий позначений анотацією @RequestMapping, або її скороченнями @PostMapping, @GetMapping, @PutMapping, @DeleteMapping, обробляє запит за певним шляхом, вказаному у параметрі value цієї анотації. Метод може приймати аргументи, повертати значення чи не повертати значення, викидувати виключення і т.д. Загалом усі характеристики звичайного методу у мові java збережені. Приклад коду методу у контролері:

```
@GetMapping("bookedTable/{id}")
public BookedTable getBookedTable(@PathVariable("id") String id)
throws Exception {
    return bookedTableServiceImpl.getBookedTableById(id);
}
```

Для реалізації спілкування клієнта із сервером у режимі реального часу, на серверній частині було реалізовано WebSocket. Тобто за допомогою цього протоколу можна передавати і приймати повідомлення одночасно. Він дозволяє в режимі реального часу обмінюватися повідомленнями між сервером і клієнтом (браузером). WebSocket встановлює одне єдине постійне з'єднання клієнта з сервером, за яким відбувається двосторонній обмін інформацією. Так як з'єднання з клієнтом і сервером не закривається (він тримається відкритим постійно), це дозволяє уникнути передачі зайвих даних (HTTP-заголовки) [18]. Коли користувач змінює, видаляє чи додає дані, всі його запити потрапляють у контролер, саме тут повідомляються всі користувачі веб-сервісу про зміни. Для початку створюється об’єкт класу (приклад коду у додатках) WsSender , котрий і являє собою веб-сокет. Після цього, при будь-яких змінах на сервері у цей об’єкт записується ці дані, після чого цей об’єкт повідомляє про зміни усіх користувачів. Ось як це виглядає у коді при зміні даних замовлення:

```
    @PutMapping("/bookedTable/{id}")
```

```
        public BookedTable updateBookedTable(@PathVariable("id")
BookedTable bookedTableFromDB, @RequestBody BookedTable
bookedTable) throws Exception {

            BeanUtils.copyProperties(bookedTable,
bookedTableFromDB, "id");

            BookedTable updatedBookedTable =
bookedTableServiceImpl.createBookedTable(bookedTableFromDB);

        wsSender.accept(EventType.UPDATE, updatedBookedTable);

        return updatedBookedTable;

         }
```

Доступ до бази даних здійснюється за допомогою JpaRepository. JpaRepository – це інтерфейс фреймворка Spring Data надає набір стандартних методів JPA для роботи з БД [19]. Приклад коду репозиторію:

```
    public interface BookedTableRepository extends
MongoRepository<BookedTable, String> {

        @Query("{ 'user.id' : ?0, 'estb._id' : ?1, 'bookedOn':
{$gt:?2}}")

        Optional<BookedTable>
findActualBookedTablesByEstablishmentId(String userId, String
estbId, Date date);

        Optional<BookedTable> findByEstablishmentId(String estbId,
Date date);

    }
```

Для реалізації бізнес логіки використовуються сервіси. Service – це Java клас, який надає з себе основну (Бізнес-Логіку), позначається анотацією @Service [20]. В основному сервіс використовує готові DAO / Repositories або ж інші сервіси, для того щоб надати кінцеві дані для призначеного для користувача інтерфейсу. При створенні сервісів спочатку створюють інтерфейс, для реалізації поліморфізму, - одного із основних принципів ООП, у якому описані методи, котрі повинні бути у нашій програмі. Після чого цей інтерфейс реалізовується

класом, котрий реалізовуватиме бізнес логіку. Приклад сервісу для бронювання столиків:

```
@Service

public class BookedTableServiceImpl implements
BookedTableService {

        @Autowired

        private BookedTableRepository bookedTableRepository;

        @Override

        public BookedTable createBookedTable(BookedTable
bookedTable) {

             return bookedTableRepository.save(bookedTable);

        }

        @Override

        public List<BookedTable>
getActualBookedTableByTableId(String id) throws Exception {

             Table table = tableServiceImpl.getEstbTableById(id);

             return
bookedTableRepository.findActualBookedTables(table.get_id(), new
Date(System.currentTimeMillis()));

        }

        @Override

        public BookedTable getBookedTableById(String id) throws
Exception {

             Optional<BookedTable> bookedTable =
bookedTableRepository.findById(id);

             if(bookedTable.isPresent())

                  return bookedTable.get();

             else throw new Exception("Not found");}

}
```

Завдяки фреймворку Spring ми отримали ефективний та зручний для читання код.

3 ОПИС ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ

3.1 Розробка методу реєстрації користувача

Однією із важливих особливостей проекту є легкість розміщення свого закладу. Задля цього було вирішено розробити реєстрацію за допомогою логіну через соціальні мережі протоколом OAuth.

OAuth – це відкритий протокол авторизації, що дозволяє надавати третій стороні обмежений доступ до захищених ресурсів без необхідності передавати логін і пароль. Наприклад, користувач, який хоче надати сервісу соцмережі доступ до контактів своєї пошти, не повинен повідомляти соціальної мережі свою поштову пароль. Замість цього, користувач проходить авторизацію в поштовому сервісі, який надає сервісу соцмережі доступ до адресної книги. OAuth заснований на використанні базових веб-технологій: HTTP-запити, редирект і т. П. Тому використання OAuth можливо на будь-якій платформі з доступом до інтернету і браузеру: на сайтах, в мобільних і desktop-додатках, плагинах для браузерів [21].

Загальна схема роботи програми, що використовує OAuth, така:

1. отримання авторизації;
2. звернення до захищених ресурсів.

Результатом авторизації є access token – якийсь ключ (зазвичай просто набір символів), пред'явлення якого є пропуском до захищених ресурсів. Звернення до них в найпростішому випадку відбувається по HTTPS із зазначенням в заголовках або в якості одного з параметрів отриманого access token'a. У протоколі описано кілька варіантів авторизації, що підходять для різних ситуацій:

- авторизація для додатків, що мають серверну частину (найчастіше, це сайти і веб-додатки);

- авторизація для повністю клієнтських додатків (мобільні і desktop-додатки);
- авторизація за логіном і паролем;
- відновлення попередньої авторизації [22].

Для розміщення закладу при вході потрібно використати кнопку «Вхід для підприємців», після чого користувач почне реєструватися, як власник закладу (рис. 3.1).

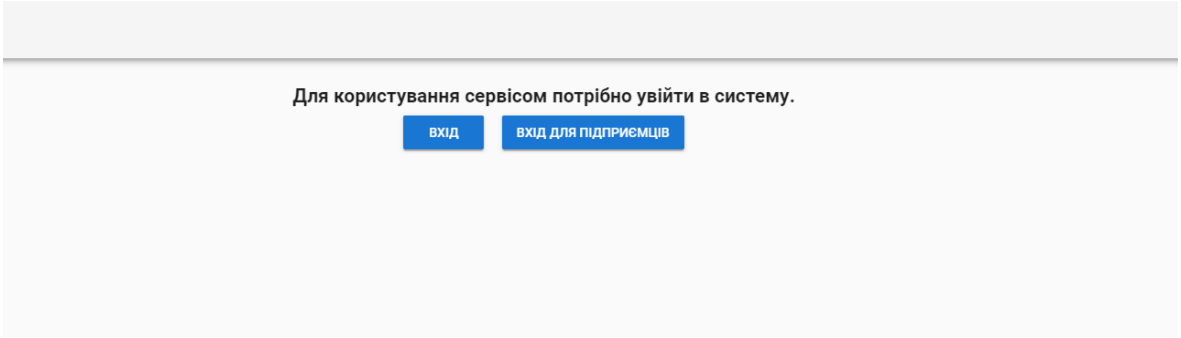

Рисунок 3.1 – Панель входу користувача

Для реєстрації запропонується залогінитися через одну із запропонованих соціальних мереж (рис. 3.2). Це зроблено для того, щоб максимально спростити процес реєстрації у веб-сервісі.

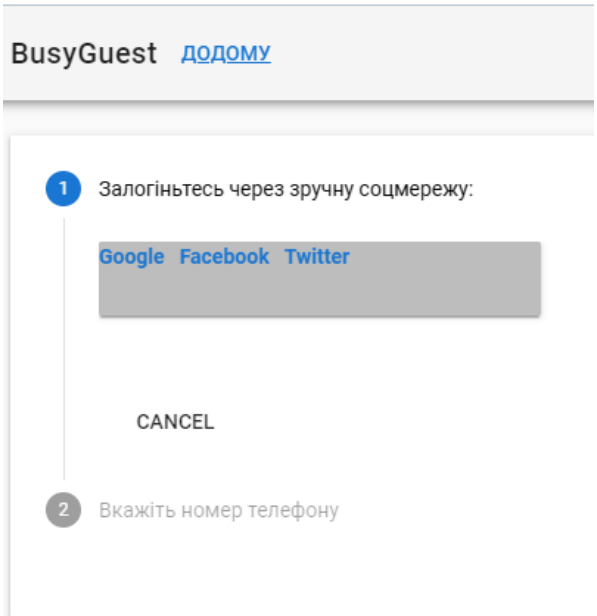

Рисунок 3.2 – Реєстрація через запропоновані соціальні мережі

Реєстрація через соціальні мережі – це ідеальний варіант для веб-сервісу такого типу. Це рішення неодмінно сподобається користувачам та водночас буде дуже надійним, так як всю обробку процесу реєстрації беруть на себе такі гіганти як: Google, Facebook та інші.

3.2 Настанова користувача адміністратора закладу

Після реєстрації користувач одразу ж потрапляє на сторінку створення закладу, де йому пропонується заповнити основну інформацію у форму та відправити заявку на розділ адміністрації, після чого із ним зв'яжуться для підтвердження валідності заявки (рис 3.3).

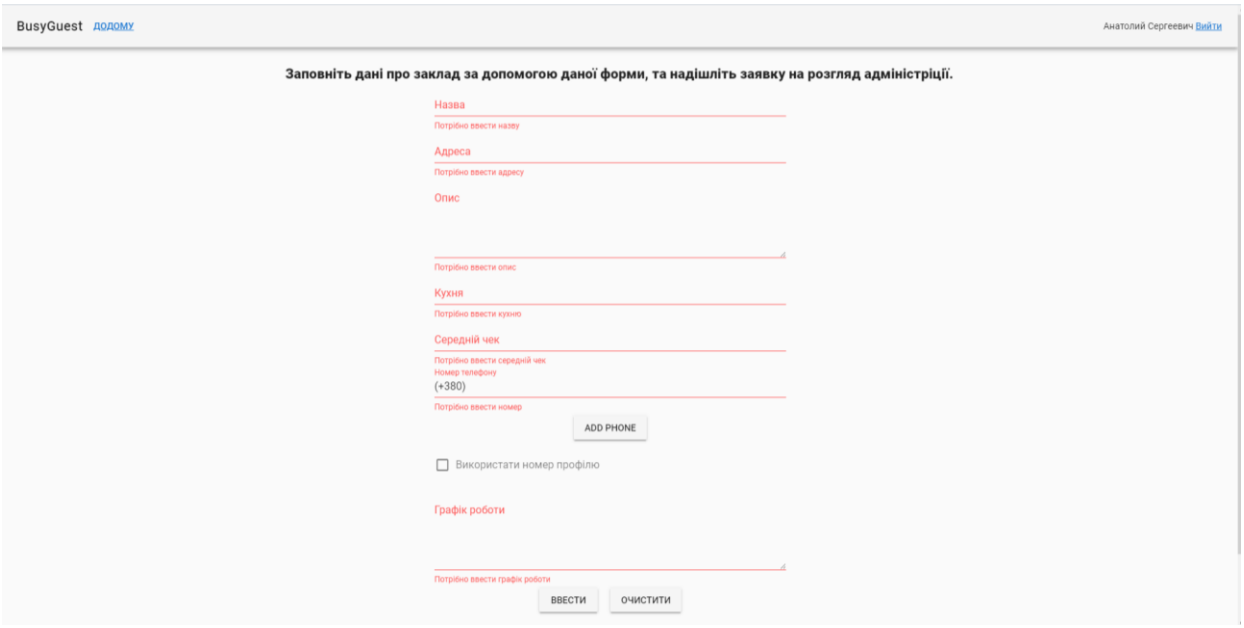

Рисунок 3.3 – Форма заявки на створення закладу

Після відправлення заявки потрібно чекати на рішення адміністрації. Якщо всі перевірки буде пройдено, то користувача обов'язково повідомлять про публікацію закладу. Для редагування даних закладу потрібно біля поля, яке потрібно відредагувати, натиснути на знак олівця, після чого текст заміниться на поле вводу (рис 3.4).

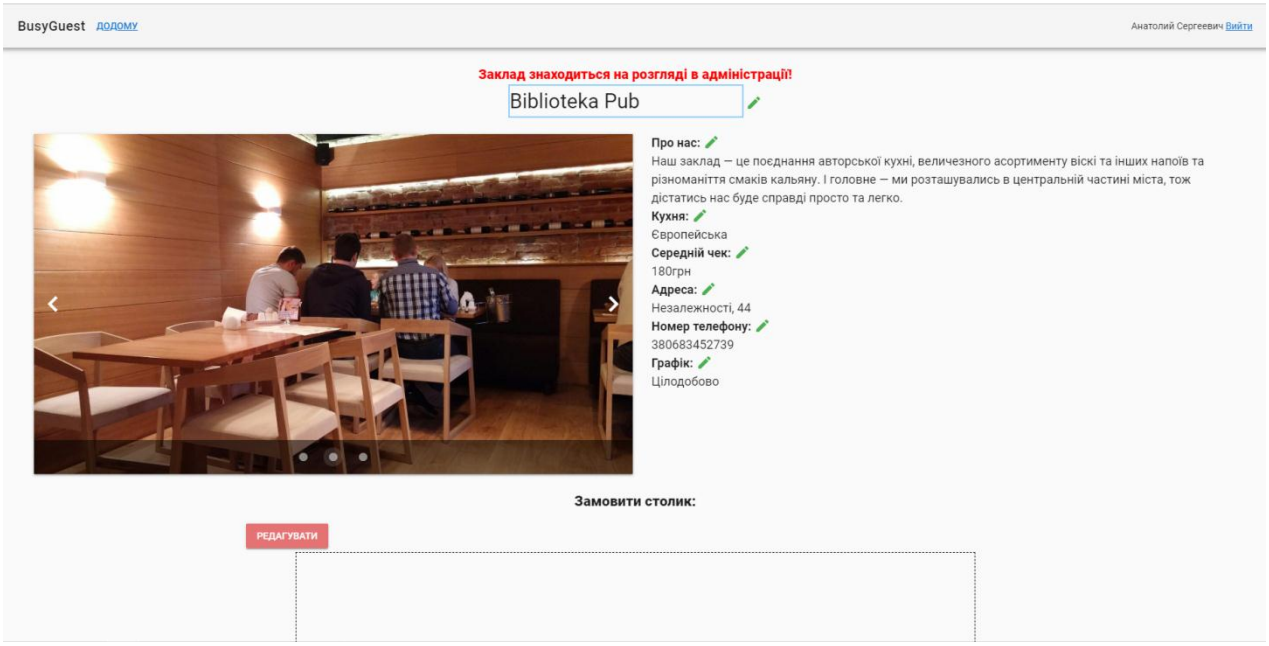

Рисунок 3.4 – Процес редагування назви закладу

Якщо заклад схвалено адміністрацією, то він не показується у списку закладів. З цього моменту заклад може керувати показом закладу у списку самостійно. Відповідна функціональна можливість відображається вверху зліва від назви закладу (рис 3.5).

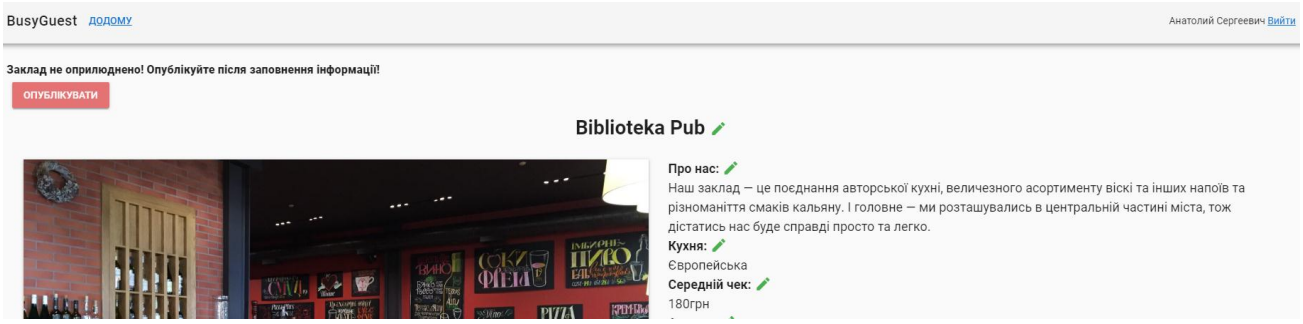

Рисунок 3.5 – Функція публікації закладу у списку

Для додавання столиків та редагування їхньої позиції, а також зміни розміру залу у візуалізаторі залу потрібно натиснути кнопку «Редагувати» у пункті «Замовити столик». Також при додаванні великої кількості столиків, пункт «Номер столика» автоматично інкременується для зручності (рис. 3.6).

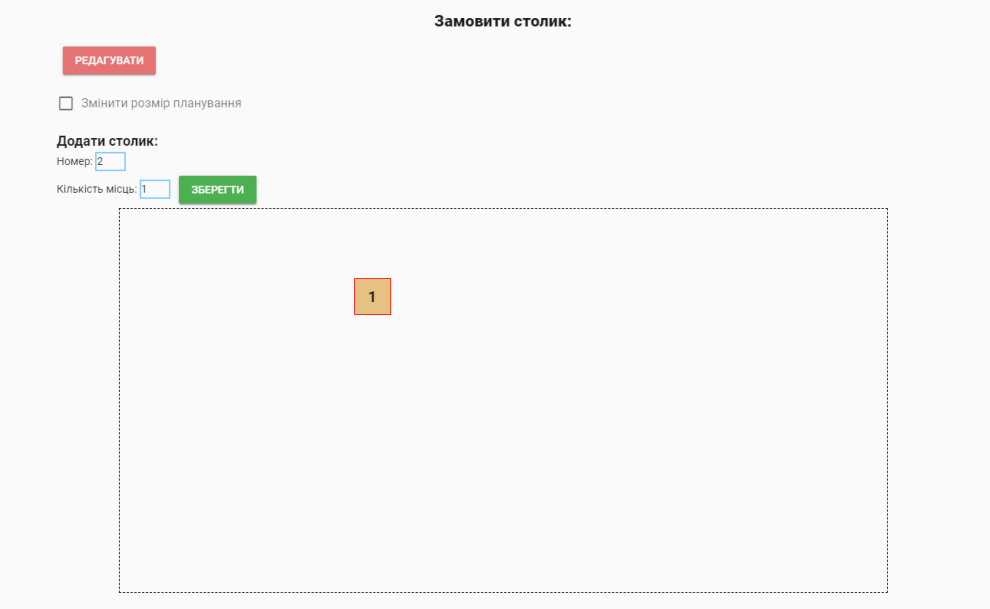

Рисунок 3.6 – Функції доступні для маніпуляції із залом

Створені столики можна розміщувати та змінювати їхній розмір відповідно до плану закладу. Для зміни розміру столика потрібно натиснути на нього і появляться відповідні кнопки. Розміщення столика відбувається за допомогою перетягування «Drag and Drop» (рис 3.7).

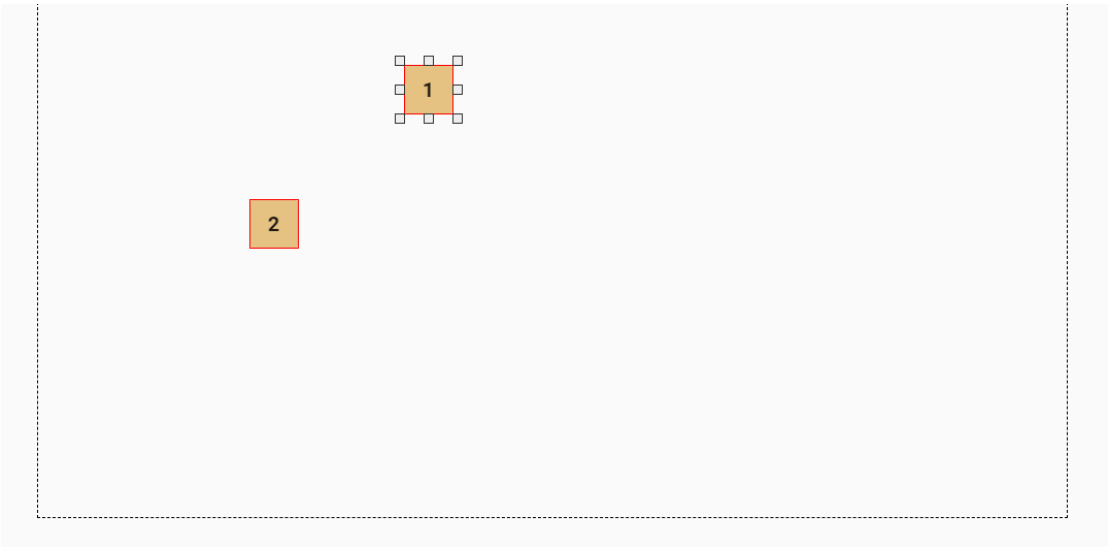

Рисунок 3.7 – Маніпуляції із столиком закладу

Для зміни розміру плану закладу потрібно активувати функцію «Змінити розмір закладу», після чого користувачу стане доступною функція зміни плану (рис 3.8).

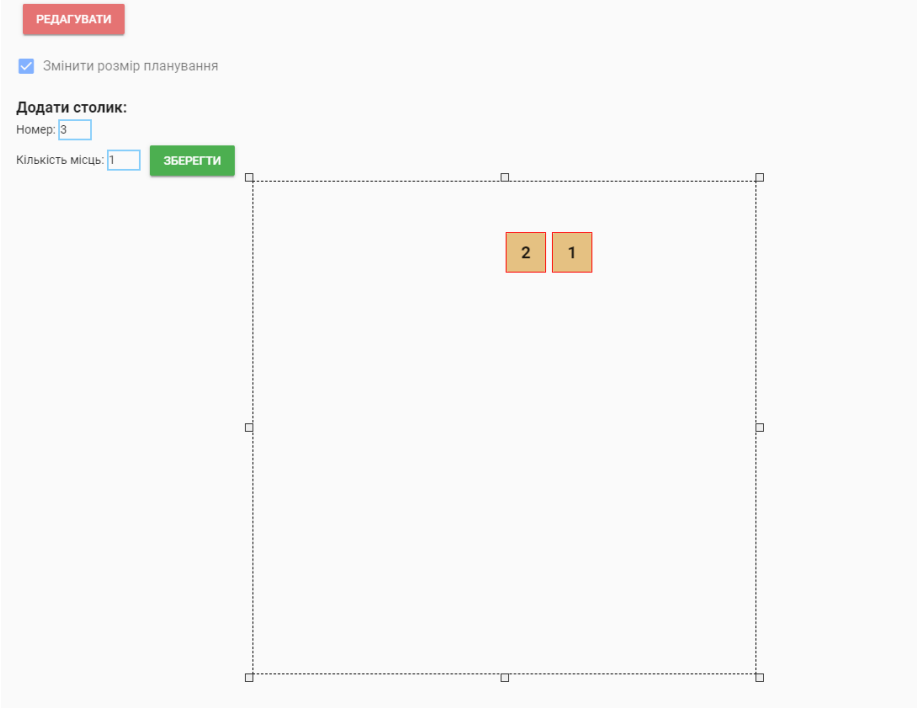

Рисунок 3.8 – Зміна розміру плану закладу

Додавши та розмістивши столики згідно плану, для зберігання даних потрібно знову натиснути кнопку «Редагувати», після чого функції маніпулювання планом закладу заховаються. Тепер заклад може маніпулювати створеними столами. Для цього на кожному столі є значки олівця та сміттєвого відра (рис 3.9).

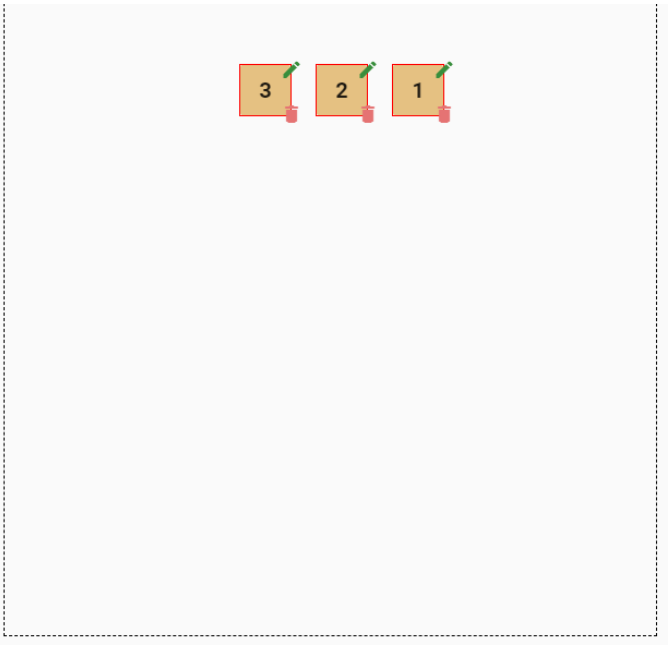

Рисунок 3.9 – Значки для модифікування столика

При взаємодії із іконкою видалення появляється відповідне вікно в якому потрібно підтвердити, або ж скасувати дію. Таким ж чином працює іконка редагування даних столика. У відповідному вікні розміщена форма, яка валідується і не дозволяє вводити порожні значення. У разі, якщо дані не були змінені, але кнопка вводу буде нажатою, то запит на сервер не буде відправлено. Якщо на столик прийде нове замовлення то колір його тла зміниться на червоний. Натиснувши на нього, появиться вікно зі списком замовлень. Ті, котрі не підтверджені будуть виділятися червоним кольором (рис. 3.10).

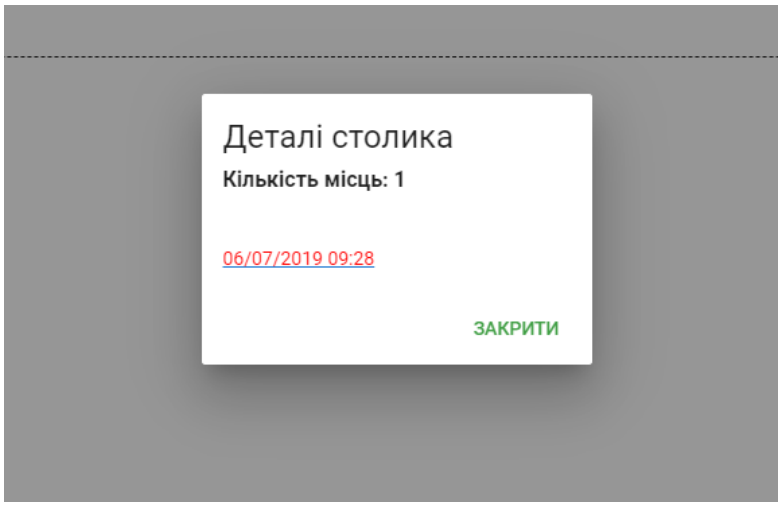

Рисунок 3.10 – Вікно зі списком замовлень

Натиснувши на непідтверджене замовлення зі списку користувача перенаправлює на сторінку замовлення, де можна переглянути деталі, такі як: ім'я замовника, номер телефону, дату, час, стан замовлення та репутацію. Адміністратор закладу може підтвердити замовлення (рис. 3.11). Якщо замовлення не буде підтверджене протягом 15 хвилин – воно автоматично відміниться.

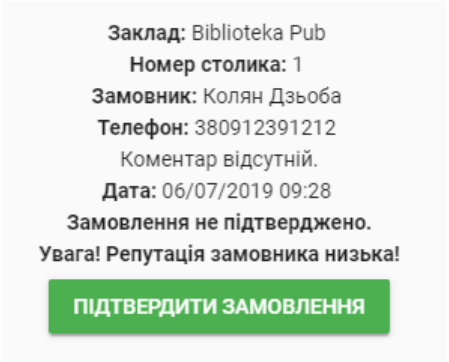

Рисунок 3.11 – Фрагмент сторінки замовлення

У разі підтвердження замовлення клієнту прийде лист на емейл-адресу (рис. 3.12) та колір замовлення у списку стане зеленим (рис 3.13).

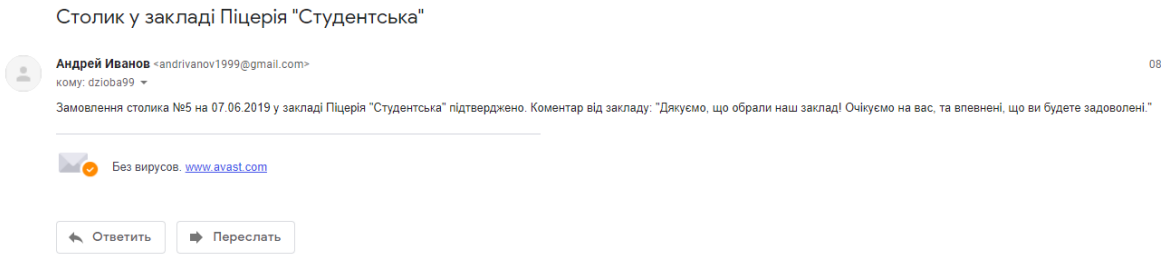

Рисунок 3.12 – Лист із підтвердженням замовлення на емейл-адресі

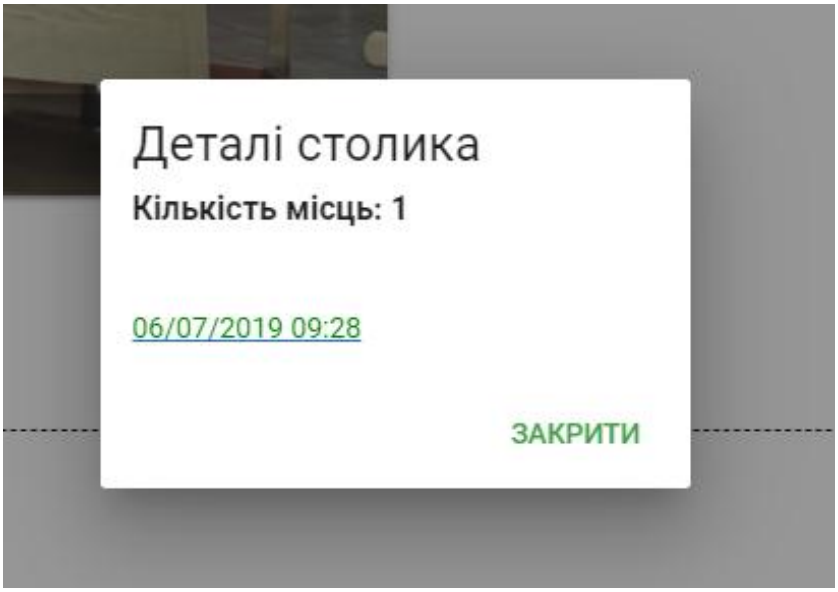

Рисунок 3.13 – Підтверджене замовлення у списку

Перелічені вище функції є основними для адміністратора закладу. Менш значущі функції було опущено.

3.3 Настанова користувача клієнта закладу

Одразу на головній сторінці клієнта закладу чекає список закладів відсортований за популярністю із логотипом, назвою та коротким описом (рис. 3.14). Якщо опис закладу не поміщається, то навівши на нього мишею, можна побачити його повністю у спливаючій підказці.

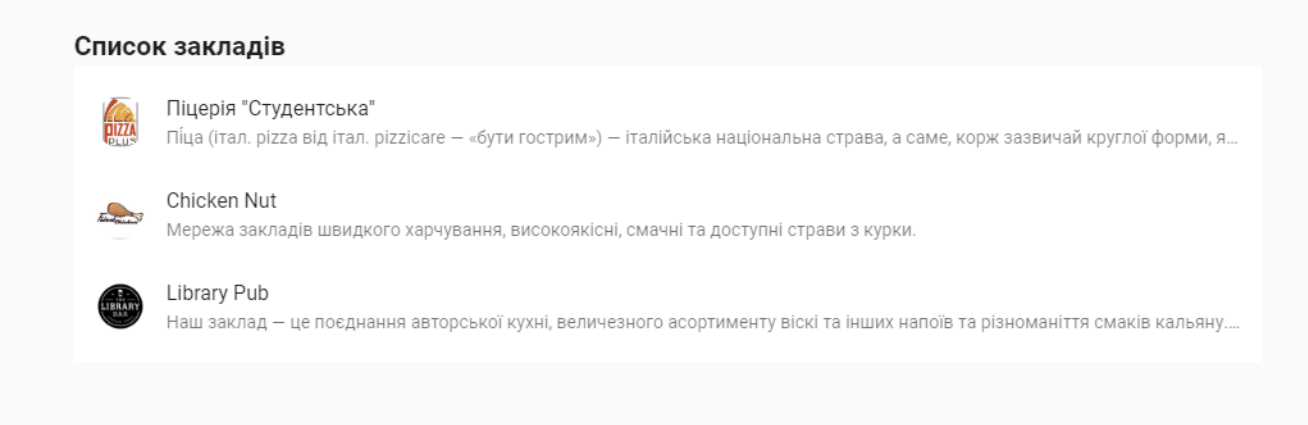

Рисунок 3.14 – Список закладів на головній сторінці

Потрапивши на сторінку закладу, користувачу показується вся інформація про заклад, наприклад, назву, опис, тип кухні, середній чек, адресу і тд. Зліва від списку інформації розміщений слайдер із фотографіями закладу, який в автоматичному режимі змінює фото кожні 7 секунд. В самому низу розміщений блок «Замовити столик» у якому показуються всі столики закладу, відповідно до візувалізації плану залу.

Приклад сторінки закладу проілюстрований на рисунку 3.15.

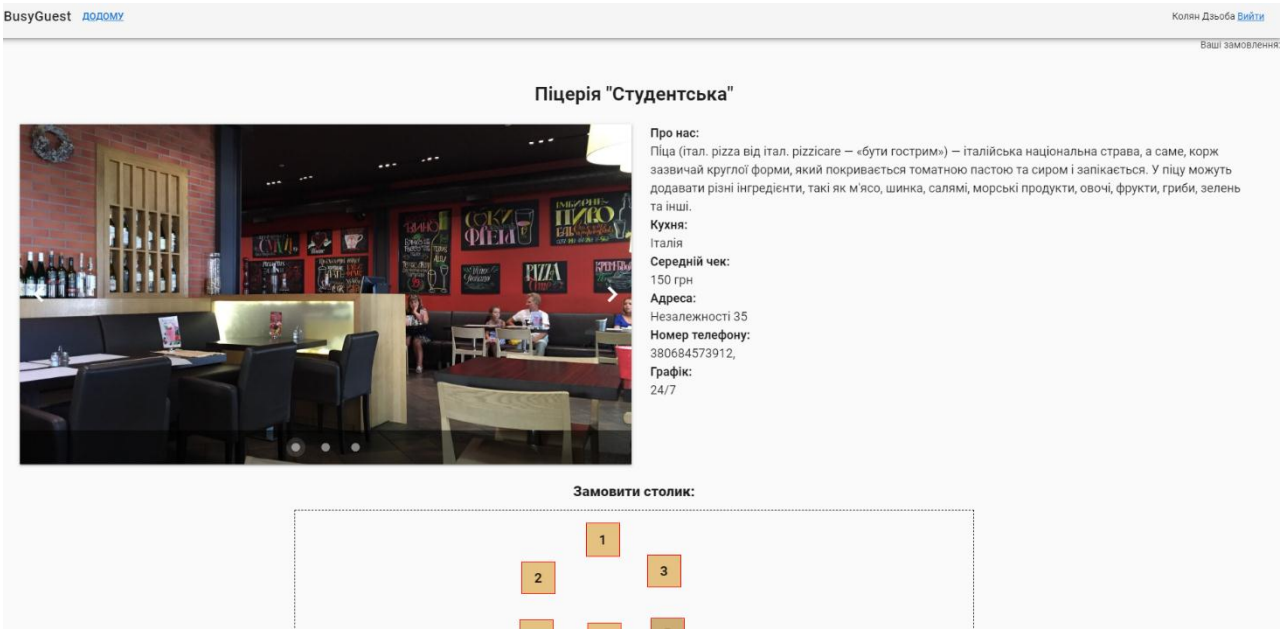

Рисунок 3.15 – Приклад сторінки закладу

Натиснувши на столик мишею, відкривається вікно із формою замовлення столика. В ній присутні пункти: «Особливі побажання», «Дата», «Час». Пункт «Особливі побажання» є необов'язковим і використовується для вказування клієнтом додаткової інформації. Якщо буде введена дата менша за поточний час або менша ніж на пів години, то у формі виведеться відповідне повідомлення (рис. 3.16). Також повідомлення виведеться, якщо на такий час цей столик вже замовлено.

Після введення коректних даних замовлення буде зроблено і в ту ж мить адміністратор закладу буде повідомлений про нього. А замовнику виведеться сповіщення та прийде лист на електронну пошту. Лист подібного формату прийде також у разі прийняття чи відміни заявки.

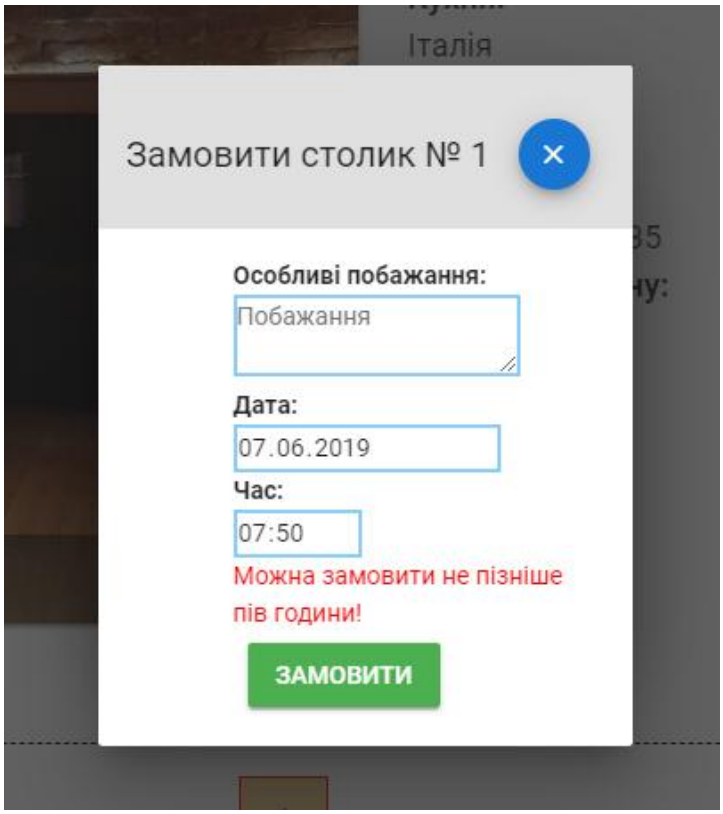

Рисунок 3.16 – Форма замовлення столу із повідомленням про помилку

В правому верхньому куті відображатиметься список замовлень (рис. 3.17), котрі зроблені даним користувачем. На нього можна перейти та потрапити на сторінку замовлення, а також відмінити, за допомогою кнопки видалення. Ця дія негативно впливає на репутацію. На сторінці замовлення буде доступна функція редагування даних замовлення, у тому випадку, якщо адміністратор закладу ще не встиг його переглянути (рис. 3.18). У формі дата та чай об'єднуються в одне поле localdatetime, що є зручнішим для сприйняття для адміністратора. Дана форма також валідується, некоректні дані прийматися не будуть. У випадку не зміни старих даних і натискання кнопки «Зберегти» запит на сервер не відправиться.

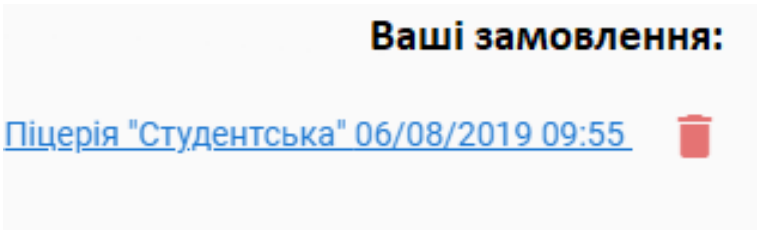

Рисунок 3.17 – Список замовлень користувача

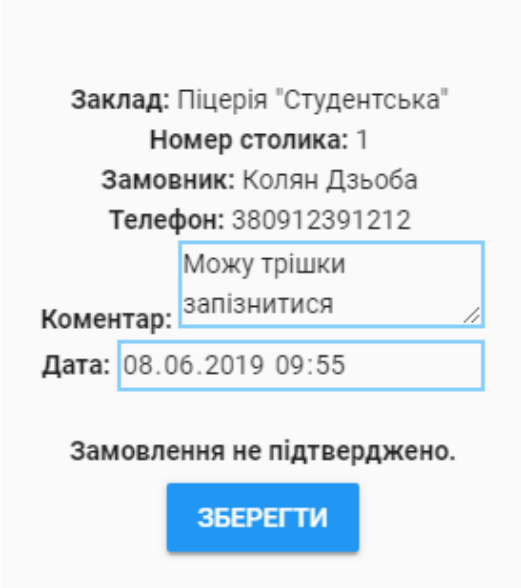

Рисунок 3.18 – Редагування деталей замовлення

Перелічені в даному підрозділі функції є необхідні для клієнта закладу, щоб активно взаємодіяти із закладом та мати широкий набір функцій.

3.4 Функціональні можливості адміністратора веб-сервісу

Адміністратор веб-сервісу – це дуже важлива та відповідальна роль, яку слід реалізовувати зваживши всі ризики. Тому для даного сервісу було вирішено наділіти адміністратора усіма наявними можливостями адміністратора закладу. Це зроблено для того, щоб у випадку виникнення проблем у адміністратора закладу, можна було швидко їх вирішити через підтримку. У базі даних роль у кожного типу користувачів вказана одна для запобігання зв'язаних із цим помилок, та для адміністратора доступні усі функції закладу, але не клієнта. Також для цього типу користучів доступні функції розгляду та підтвердження заявки на долучення до сервісу (рис. 3.19), а також вилучення закладу харчування за поручення правил чи низьку репутацію, пов'язану із частою відміною замовлень у сервісі, або низькі відгуки клієнтів.

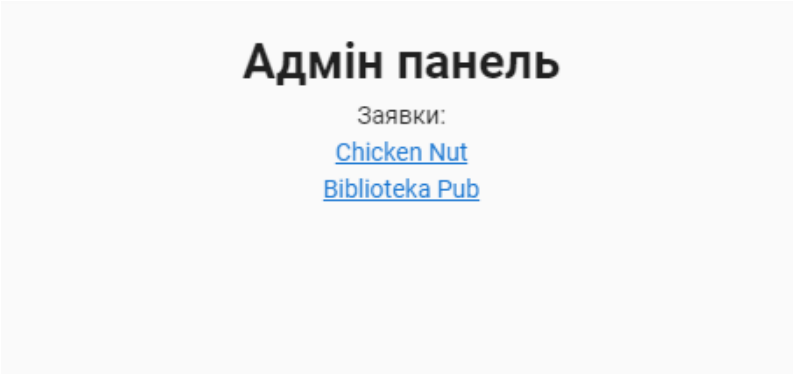

Рисунок 3.19 – Список заявок на долучення до сервісу

Ще однією особливістю ролі адміністратора веб-сервісу є можливість блокування користувачів через порушення внутрішніх правил, таких як, неправдивий відгук про заклад чи постійне відмінення замовлень (для запобігання спаму).

ВИСНОВКИ

Під час виконання даного дипломного проекту було розроблено веб-додаток під назвою BusyGuest для резервування місць у закладах харчування.

Веб-додаток реалізований за допомогою останніх новітніх веб технологій таких як - VueJS (JavaScript фреймворк), MongoDB (база даних), Spring (Java фреймворк). Проект розроблений з розрахунком на подальше вдосконалення і додавання нових функціональних можливостей. Тому інтеграція нових функцій займатиме мінімум часу і зусиль.

Результатом роботи є повністю функціональний, протестований і готовий до використання веб-додаток.

Тестування сервісу відбувалося в два етапи. Перший етап полягав в тестуванні додатку в різних браузерах (Google Chrome, Mozila, Opera, Internet Explorer) з різних типів пристроїв (телефони, планшети, комп'ютери), які мають різну роздільну здатність екранів. Необхідність такого тестування полягає в тому, щоб виявити і виправити помилки, допущенні під час розробки. Другий етап тестування необхідний для опрацювання всіх можливих сценаріїв дій користувача, для того, щоб переконатися, що все працює правильно і ніяких помилок в проектуванні не допущено.

Дипломний проект включає в себе вичерпний теоретичний і практичний опис проекту та кожної його частини. Для зручності користування створено додаток, у вигляді посібника для користувачів, який містить інструкцію з описом всіх сторінок, компонентів і кнопок.

Дипломний проект виконано у повній відповідності до завдання і всіма нормативними вимогами.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Именование ресурсов. Матеріал з restapitutorial. URL: http://www.restapitutorial.ru/lessons/restfulresourcenaming.html (дата звернення: 23.05.2019).
2. Spring. Матеріал з javastudy. URL: http://javastudy.ru/frameworks/spring/ (дата звернення: 23.05.2019).
3. MongoDB. Матеріал з Вікіпедії — вільної енциклопедії. URL: https://ru.wikipedia.org/wiki/MongoDB (дата звернення: 23.05.2019).
4. Обзор Vue.js. Матеріал з Timeweb. URL: https://timeweb.com/ru/community/articles/obzor-vue-js-1 (дата звернення: 23.05.2019).
5. WebSocket. Матеріал з learn.javascript.ru/. URL: https://learn.javascript.ru/websockets (дата звернення: 24.05.2019).
6. Spring Framework. Матеріал з Вікіпедії — вільної енциклопедії. URL: https://ru.wikipedia.org/wiki/Spring_Framework (дата звернення: 24.05.2019).
7. Введение. Что такое Vue.js?. Матеріал з офіційної документації Vue.js. URL: https://ru.vuejs.org/v2/guide/index.html (дата звернення: 24.05.2019).
8. Разработка веб-структуры сайта и приложения. Матеріал з WebForMyself. URL: https://webformyself.com/razrabotka-veb-struktury-sajta-i-prilozheniya/ (дата звернення: 25.05.2019).
9. Создание архитектуры программы или как проектировать табуретку. Матеріал з Habr. URL: https://habr.com/ru/post/276593/ (дата звернення: 25.05.2019).
10. Что такое MongoDB. Матеріал з Metanit. URL: https://metanit.com/nosql/mongodb/1.1.php (дата звернення: 25.05.2019).
11. Что такое Юзкейс (Use Case) или "Сценарий Использования" в Тестировании ПО? Матеріал з Software Testing. URL: https://software-testing.org/testing/chto-takoe-yuzkeys-use-case-ili-scenariy-ispolzovaniya-v-testirovanii-po.html (дата звернення: 26.05.2019).

12. Архитектурное проектирование. Матеріал з Studbooks. URL: https://studbooks.net/2047820/informatika/proektirovanie_informatsionnoy_sistemy (дата звернення: 26.05.2019).

13. Компоненты сетевого приложения. Клиент-серверное взаимодействие и роли серверов. Матеріал з 4stud. URL: http://www.4stud.info/networking/lecture5.html (дата звернення: 26.05.2019).

14. Model-View-Controller. Матеріал з Вікіпедії — вільної енциклопедії. URL: https://ru.wikipedia.org/wiki/Model-View-Controller (дата звернення: 26.05.2019).

15. Аналіз протоколів взаємодії у клієнт-серверній архітектурі. Матеріал з Inter-Nauka. URL: https://www.inter-nauka.com/uploads/public/14673574977414.pdf (дата звернення: 27.05.2019).

16. Блок-схема алгоритма. Матеріал з Shkolo. URL: http://shkolo.ru/blok-shema-algoritma/ (дата звернення: 27.05.2019).

17. @Controller. Матеріал з Fandom. URL: https://java.fandom.com/ru/wiki/@Controller (дата звернення: 28.05.2019).

18. Что такое websocket. Матеріал з BlogProgrammista. URL: http://blog-programmista.ru/post/33-cto-takoe-websocket.html (дата звернення: 28.05.2019).

19. Spring Data JPA. Пишем DAO и Services. Матеріал з Devcolibri. URL: https://devcolibri.com/spring-data-jpa-пишем-dao-и-services/ (дата звернення: 28.05.2019).

20. Spring Services. Матеріал з Spring Documentation. URL: https://docs.spring.io/spring-boot/docs/2.1.5.RELEASE (дата звернення: 29.05.2019).

21. Что такое OAuth? Определение термина OAuth. Матеріал з Animatika. URL: https://animatika.ru/info/gloss/oauth.html (дата звернення: 29.05.2019).

22. OAuth 2.0 простым и понятным языком. Матеріал з Habr. URL: https://habr.com/ru/company/mailru/blog/115163/ (дата звернення: 30.05.2019).

23. Баженова И.Ю. Язык программирования Java. АО «Диалог-МИФИ», 1997. 82с.

24. Крис Шефер Кларенс Хо. Спринг от А до Я, 2017.

25. Брюс, А. Тейт JavaScript. Быстрая веб-разработка : БХВ-Петербург, 2008. - 224 с.

26. Фленов М. Transact-SQL, Санкт-Петербург: БХВ Петербург, 2006. – 250 с.

27. Липпман С., Лажойе Ж. Язык программирования JavaScript. Вводный курс. : С-Пб, Невский проспект, 2006. – 1406 с.

28. Marijn Haverbeke-Вячеслав Голованов -Выразительный JavaScript: Введение Пер. с англ. — «Питер», 2016. — 433 с.

29. Дэвид Флэнаган, JavaScript. Карманный справочник. Пер. з англ. – «Питер», 2015. – 600с.

30. Дэвид Флэнаган, JavaScript – Подробное руководство 6-ое издание Пер. с англ. — М.: Издательский дом «Вильямс», 2015. — 300 с.

Додаток А

Програмний код клієнтської частини

index.html:

```
<!DOC
TYPE
html>
        <html lang="en" xmlns:th="http://www.thymeleaf.org">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width,
        initial-scale=1, maximum-scale=1, user-scalable=no,
        minimal-ui">
            <link
        href='https://fonts.googleapis.com/css?family=Roboto:100,30
        0,400,500,700,900|Material+Icons' rel="stylesheet">
            <title>Test</title>
            <script th:inline="javascript">
                var frontendData = [[${frontendData}]];
            </script>


        </head>
        <body>
        <div id="app"></div>


        <script th:src="${isDevMode} ?
        'http://localhost:3000/main.js' : '/js/main.js'"></script>
        </body>
        </html>
```

App.vue:

```
<template>
                <v-app>
                     <v-toolbar app>
                      <v-toolbar-title>BusyGuest</v-toolbar-title>
```

```
          <v-toolbar-items class="hidden-sm-and-down">
                <v-btn flat>
                    <router-link to="/">Додому</router-
link>
                </v-btn>
              </v-toolbar-items>

          <v-spacer></v-spacer>
          <v-toolbar-items class="hidden-sm-and-down">
          <div style="width: 500px; margin-left: 75%;">
                      <v-alert
                         v-model="alert"
                         dismissible
                         type="success"
                       >
                         Столик замовлено!
                       </v-alert>
                   </div>
                   </v-toolbar-items>
              <div v-if="profile">
                <span>{{profile.name}}</span>
                   <a href="/logout">Вийти</a>
              </div>

          </v-toolbar>


          <v-content>

              <div v-if="role=='VISITOR'">
                <books></books>
              </div>
              <v-container fluid>
                <sign-up-stepper v-if="profile &&
profile.phone=='' || showStepper"></sign-up-stepper>
                    <div v-else>
                        <div v-if="profile">
                            <router-view></router-view>
                        </div>
                        <div v-else style="text-align:
center;">
                        <h2>Для користування сервісом
потрібно увійти в систему. </h2>
                              <v-btn color="primary"
href="/login">Вхід</v-btn>
                              <v-btn color="primary"
@click="createEstb">Вхід для підприємців</v-btn>
                        </div>
```

```
                    </div>
                </v-container>

            </v-content>


        </v-app>
    </template>


    <script>
        import EstablishmentsList from
    'components/establishments/EstablishmentsList.vue'
        import Books from 'components/profile/Books.vue'
        import SignUpStepper from
    'components/UI/SignUpStepper.vue'
        import { mapState} from 'vuex'
        import { mapActions } from 'vuex'
        import {PhoneConstants} from
    'constants/PhoneConstants'
        import EventBus from 'eventBus/event-bus.js'

        export default {
            components: {
                EstablishmentsList,
                Books,
                SignUpStepper
            },
            data: function () {
                return {
                    showStepper: false,
                    step: 0,
                    alert: false
                }
            },
            created(){
              if(this.profile && this.profile.phone==''){
                    showStepper: true
              }
              this.checkRole()
            },
            watch:{
                $route (to, from){
                    this.checkRole()
                }
            },
            mounted () {
                EventBus.$on('close_stepper', () => {
```

```
                        this.showStepper=false
                        this.checkRole()
                    }),
                    EventBus.$on('booked', () => {
                            this.alert = true
                    })
                },
            computed: mapState(['profile',
'role','establishments']),
            methods:{
                createEstb(){
                 this.showStepper = true
                 this.$cookie.set('bussines', 'true',
'1h')
                },
                checkRole(){
                  /* if(this.role=='ADMIN'){
                       this.$router.push('/admin')
                  }else  */if(this.role=='ESTB'&&
this.profile.phone != '') {
                        const estb =
this.establishments.find(element =>
element.email==this.profile.email)
                        if(estb==null){

     this.$router.push('/establishment')
                        }
                 }
                }

            }
        }
</script>


<style>
</style>
```

AdminPanel.vue:

```
<template>
            <div >
             <div v-if="role=='ADMIN'" style="text-align:
center;">
                  <h1>Адмін панель</h1>
                  <span>Заявки:</span>
```

```
          <div v-for="establishment in establishments"
:key="establishment._id">
                <router-link
:to="'/establishment/'+`${establishment._id}`">
                  {{establishment.name}}
              </router-link>
            </div>
           </div>
    </div>
</template>


<script>
    import { mapState} from 'vuex'
    import establishmentApi from 'api/establishmentApi'
    export default {
        data: function () {
            return {
              establishments: []
            }
        },
        created(){
          this.loadData()
        },
        computed: mapState(['profile', 'role']),
        methods:{
          async loadData(){
                const result = await
establishmentApi.getAll()
                const data = await result.json()
                const notAcceptedEstablishments = []
                data.map(function(item) {
                       if(!item.accepted)

notAcceptedEstablishments.push(item)
                });
                this.establishments =
notAcceptedEstablishments

          }
        }
    }

</script>


<style>
</style>
```

BookedTable.vue:

```
<templat
e>
                <v-app>
                     <div style="text-align: center;">
                          <span><b>Заклад:</b>
{{bookedTable.estb.name}}</span> <br>
                          <span><b>Номер столика:
</b>{{bookedTable.table.tableNum}}</span> <br>
                          <span><b>Замовник:</b>
{{bookedTable.user.name}}</span> <br>
                          <span><b>Телефон:</b>
{{bookedTable.user.phone}}</span> <br>
                          <div v-if="this.bookedTable.comment">
                          <b>Коментар:</b>
                               <span id="date" v-show="!showEdit">
{{bookedTable.comment}} </span>
                               <textarea v-show="showEdit"
style="border: 2px solid #87CEFA;" v-
model="bookedTable.comment"
placeholder="Коментар"></textarea>
                          </div>
                          <div v-else v-show="!showEdit">Коментар
відсутній.</div>


                          <div v-show="showEdit">
                               <b>Дата:</b> <input style="border:
2px solid #87CEFA;" type="datetime-local" v-
model="bookedTable.bookedOn">
                          </div>
                          <span v-show="!showEdit" id="date">
<b>Дата:</b> {{bookedTable.bookedOn | moment("MM/DD/YYYY
HH:mm ") }}</span> <br>
                          <span><b>Замовлення {{bookedTable.accepted
? '':'не'}} підтверджено.</b> </span>
                          <div v-if="role===`VISITOR`">
                               <v-btn color="info"
@click="edit()">{{showEdit ?
'Зберегти':'Редагувати'}}</v-btn>
                          </div>
                          <div v-if="role===`ESTB`">
                               <div v-
```

```
if="this.bookedTable.user.reputation.name =='UNRELIABLE'
"><b>Увага! Репутація замовника низька!</b></div>
                    <v-btn color="success"
@click="changeBookStatus()">{{bookedTable.accepted ?
'Відмінити' : 'Підтвердити'}} замовлення</v-btn>
                </div>
            </div>
        </v-app>
</template>


<script>
    import EstablishmentsList from
'components/establishments/EstablishmentsList.vue'
    import { mapState} from 'vuex'
    import EventBus from 'eventBus/event-bus.js'
    import bookedTablesApi from 'api/bookedTableApi'

    export default {
        computed: mapState(['role']),
        data(){
            return{
                bookedTable:null,
                oldBookedTable:null,
                showEdit:false
            }
        },
        components: {
            EstablishmentsList
        },
        methods:{
          async loadbookedTable(id){
                const result = await
bookedTablesApi.get(id)
                const data = await result.json()
                this.bookedTable = data
            },
            edit() {
              if(!this.showEdit )
                    this.oldBookedTable =
{...this.bookedTable}

                if(this.showEdit &&
JSON.stringify(this.oldBookedTable) !=
JSON.stringify(this.bookedTable))

bookedTablesApi.update(this.bookedTable)
```

```
                        this.showEdit=!this.showEdit
                    },
                    changeBookStatus(){
                      this.bookedTable.accepted =
!this.bookedTable.accepted
                      bookedTablesApi.update(this.bookedTable)

        this.$router.push(`establishment/`+this.bookedTable.
estb._id)
                    }

                },
                created: function(){
                    this.loadbookedTable(this.$route.params.id)
                },
                mounted () {
                    EventBus.$on('reload-booked-table-
component', (id) => {
                        this.loadbookedTable(id)
                    })
                }
            }
</script>
<style>
</style>
```

CreateEstablishment.vue:

```
<template>
                <v-app >
                     <div v-if="role=='ESTB'">
                      <div v-if="profileEstb == null">
                              <h2><b>Заповніть дані про заклад за
допомогою даної форми, та надішліть заявку на розгляд
адміністріції.</b></h2>

                              <establishment-form></establishment-
form>
                    </div>
                    <div v-else>
                          Ви вже створили заклад!
                     </div>
                  </div>
                  <div v-else>
                       Ви не можете створювати заклад!
```

```
                    </div>
                </v-app>
            </template>


            <script>
                 import EstablishmentForm from
            'components/establishments/EstablishmentForm.vue'
                 import { mapState} from 'vuex'
                export default {
                     components:{
                      EstablishmentForm,
                     },
                     data() {
                         return {
                          profileEstb: null
                         }
                     },
                     created(){
                      this.profileEstb =
            this.establishments.find(item =>
            item.email==this.profile.email)
                     },
                     computed: {
                      ...mapState(['profile', 'role',
            'establishments'])
                     }
                }
            </script>


            <style>
            </style>
```

Establishment.vue:

```
<templ
ate>
             <v-app>
              <div v-if="role=='ADMIN'">
                   <h2><b>Заклад {{establishment.accepted ? '' :
        'не'}} опубліковано! </b></h2>
                   <v-btn color="red lighten-2"
        @click="acceptEstablishment" dark> {{establishment.accepted
        ? 'Відмінити підтвердження' : 'Підтвердити'}}</v-btn>
              </div>
```

```
     <div v-if="establishment.show || role!='VISITOR'">
                <div v-if="role==='ESTB' &&
profile.email!=establishment.email">
                     <b>Не ваш заклад!</b>
                </div>
                 <div v-else>
                     <div v-if="role==='ESTB' &&
establishment.accepted">

                          <div v-if="!establishment.show">
                          <h3><b>Заклад не оприлюднено!
Опублікуйте після заповнення інформації!</b></h3>
                          <v-btn color="red lighten-2"
@click="changeShowStatus()" dark>Опублікувати</v-btn>
                     </div>
                     <div v-else>
                          <h3><b>Заклад оприлюднено.</b></h3>
                          <v-btn color="red lighten-2"
@click="changeShowStatus()" dark>Заховати</v-btn>
                     </div>
                </div>

                <div v-if="role==='ESTB' &&
!establishment.accepted">
                     <h3><b>Заклад знаходиться на розгляді в
адміністрації!</b></h3>
                </div>
                    <div>
                     <div style="text-align: center;">
                          <h1 style="display: inline;" v-
show="editElementId != 'name'" id="name">
                               {{establishment.name}}
                          </h1>
                          <span v-show="editElementId ===
'name'">
                               <input style="border: 2px solid
#87CEFA; font-size: 30px;" type="text" v-
model="establishment.name">
                          </span>
                          <button v-if="role==`ESTB`" v-
on:click="edit('name')"><v-icon color="green">edit</v-
icon></button>
                     </div>

                          <v-container fluid grid-list-md
>
                               <v-layout align-start
justify-space-around fill-height wrap>
```

```
                                        <v-flex xs6 order-lg2>
                                          <v-carousel>
                                                <v-carousel-item
                                                  v-for="(item,i)
in items"
                                                  :key="i"
                                                  :src="item.src"
                                                ></v-carousel-
item>
                                              </v-carousel>
                                        </v-flex>
                                        <v-flex xs6 order-lg2>
                                          <establishment-info
:establishment="establishment"></establishment-info>
                                        </v-flex>
                                      </v-layout >
                                    </v-container>

                        </div>
                        <h2 style="text-align:
center;"><b>Замовити столик:</b></h2>


                      <tables-list
:establishment="establishment"></tables-list>
                    </div>
                </div>

                <div v-else>
                 Заклад не оприлюднено!
                </div>
        </v-app>
</template>


<script>
    import { mapState, mapMutations} from 'vuex'
    import TablesList from
'components/tables/TablesList.vue'
    import EstablishmentInfo from
'components/establishments/EstablishmentInfo.vue'
    import establishmentApi from 'api/establishmentApi'
    export default {
        components: {
            TablesList,
            EstablishmentInfo
        },
        data () {
```

```
            return {
                editElementId: '',
                editClickCount: 0,
                editElementValue:'',
               establishment: null,
               items: [
                   {
                     src: 'https://media-
cdn.tripadvisor.com/media/photo-
o/0c/9f/ce/c5/photo0jpg.jpg'
                   },
                   {
                     src: 'https://media-
cdn.tripadvisor.com/media/photo-
o/07/b8/1a/21/extraordinary-pizza-atmosphere.jpg'
                   },
                   {
                     src: 'https://media-
cdn.tripadvisor.com/media/photo-
o/07/b8/1a/35/extraordinary-pizza-atmosphere.jpg'
                   }
                 ]
            }
        },
        computed:
mapState(['profile','establishments','role']),
        created: function() {
          this.loadData()
          this.loadCoordinates()

        },
        methods: {
          ...mapMutations(['changeDragTablesMutation',
'addEstbCoordinatesMutation']),
            loadCoordinates(){
              var estbCoordinates


    this.$http.get('/estb/establishment/'+this.establishme
nt._id+'/coordinates').then((response) => {

    this.addEstbCoordinatesMutation(response.data)
               }, (response) => {
                  estbCoordinates = {
                            estbId: this.establishment._id,
                            width: 1000,
                            height: 500,
                            tables: {},
```

```
                                signs: {}
                            }

        this.addEstbCoordinatesMutation(estbCoordinates)
                    })
                },
                loadData(){
                  const index =
this.establishments.findIndex(item => item._id ===
this.$route.params.id)
                  this.establishment =
this.establishments[index]
                },
                async changeShowStatus(){
                  const establishment = {...this.establishment}
                  establishment.show = !establishment.show
                  const result = await
establishmentApi.update(establishment)
                  const data = await result.json()
                  this.establishment = data
                },
                edit(id) {
                    if(this.editClickCount==0){
                        this.editElementValue =
this.establishment.name
                        this.editElementId=id
                        this.editClickCount++
                    }else if(this.editClickCount!=0 &&
this.editElementId != id){
                       this.editElementId=id
                    }else{
                        this.editElementId=''
                        this.editClickCount=0;


if(this.editElementValue!=this.establishment.name){

establishmentApi.update(this.establishment)
                        }
                        this.editElementValue=''
                    }
                },
                acceptEstablishment(){
                  const estb = {
                       ...this.establishment,
                       accepted: !this.establishment.accepted
                  }
```

```
        this.$resource('/estb/establishment/{id}').update({id:
        estb._id}, estb).then(result =>
                        result.json().then(data =>
        this.establishment = data))
                    }
                }
            }
        </script>


        <style>
        </style>
```

store.js:

```
import
Vue
from
'vue'
        import Vuex from 'vuex'
        import profileApi from 'api/profile'
        import tablesApi from 'api/tablesApi'
        import bookedTablesApi from 'api/bookedTableApi'


        Vue.use(Vuex)


        export default new Vuex.Store({
            state: {
             establishments: frontendData.establishments,
                profile: frontendData.profile,
                role: frontendData.role,
                userBookedTables:[],
                dragTables: false,
                estbCoordinates:{
                  estbId: null,
                  width: null,
                  height: null,
                  tables: null,
                  signs: null
                },
            },
            mutations: {
                addPhoneMutation(state, profile) {
                    state.profile = profile
```

```
            },
            getUserBookedTablesMutation(state,
        userBookedTables){
              state.userBookedTables = userBookedTables
            },
            addEstablishmentMutation(state, establishment) {
                state.establishments.push(establishment)
            },
            changeDragTablesMutation(state, dragTables) {
                state.dragTables = !dragTables
            },
            addEstbCoordinatesMutation(state, estbCoordinates)
        {
              state.estbCoordinates = estbCoordinates
            },
            setTablesCoordinates(state, [tableId,
        tablesCoordinatesObj]) {
              state.estbCoordinates.tables[tableId] =
        tablesCoordinatesObj
            },
            setSignsCoordinates(state, [signId, signsObj]) {
              state.estbCoordinates.signs[signId] = signsObj
            }
        },
        actions: {
         async addPhoneAction({commit}, profile) {
              const result = await profileApi.update(profile)
              const data = await result
                 commit('addPhoneMutation', data)
            },
            async getUserBookedTablesAction({commit}, userId){
              const result = await
        bookedTablesApi.getByUser(userId)
              const data = await result.json()
              commit('getUserBookedTablesMutation', data);
            }
        }
        })
```

router.js:

```
import
Vue
from
'vue'
```

```
import VueRouter from 'vue-router'
import Establishment from 'pages/Establishment.vue'
import CreateEstablishment from
'pages/CreateEstablishment.vue'
import EstablishmentsList from
'components/establishments/EstablishmentsList.vue'
import TablesList from 'components/tables/TablesList.vue'
import BookedTable from 'pages/BookedTable.vue'
import AdminPanel from 'pages/AdminPanel.vue'


Vue.use(VueRouter)



 const routes = [
   {path: '/establishment/:id', component: Establishment,
props: true},
   {path: '/establishment', component: CreateEstablishment,
props: true},
   {path: '/', component: EstablishmentsList, props: true},
   {path: '/bookedTable/:id', component: BookedTable,
props: true},
   {path: '/admin', component: AdminPanel, props: true},


   { path: '*', component: EstablishmentsList}
 ]


export default new VueRouter({
    mode: 'history',
    routes
})
```

Додаток Б

Програмний код серверної частини

MainController.java:

```
@Controller
@RequestMapping("/")
public class MainController {

    @Autowired
    private EstablishmentServiceImpl establishmentServiceImpl;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private TableRepository tableRepository;

    @Value("${spring.profiles.active}")
```

```
    private String profile;



    @GetMapping

    public String main(Model model, @AuthenticationPrincipal User user) throws
Exception {

        HashMap<Object, Object> data = new HashMap<>();



        if (user != null) {

            Optional<User> usse = userRepository.findById(user.getId());

            user.setPhone(usse.get().getPhone());




            data.put("profile", user);

            data.put("role", user.getRole().getName());

            data.put("establishments",
establishmentServiceImpl.getEstablishmentList());

        }



        model.addAttribute("frontendData", data);
```

```
        model.addAttribute("isDevMode", "dev".equals(profile));

        return "index";

    }



    @GetMapping("/gs-guide-websocket/info")

    public String redirect(Model model, @AuthenticationPrincipal User user)
throws Exception {

        HashMap<Object, Object> data = new HashMap<>();



        if (user != null) {

            Optional<User> usse = userRepository.findById(user.getId());

            user.setPhone(usse.get().getPhone());



            data.put("profile", user);

            data.put("role", user.getRole().getName());

            data.put("establishments",
establishmentServiceImpl.getEstablishmentList());

        }
```

```
        model.addAttribute("frontendData", data);

        model.addAttribute("isDevMode", "dev".equals(profile));

        return "index";

    }
```

}**VisitorController.java:**

```
    @RestController
    @RequestMapping("/visitor")
    public class VisitorController {
        private final static double RELIABLE_ORDERS_PROCENT=80;
        private final static int RELIABLE_ORDERS_NUMBER=80;

        private final EstablishmentServiceImpl
establishmentServiceImpl;
        private final  TableServiceImpl tableServiceImpl;

        private final BookedTableServiceImpl
bookedTableServiceImpl;

        private final  UserRepository userRepository;

        private final  ReputationRepository reputationRepository;

        private final BiConsumer<EventType, BookedTable> wsSender;

        public VisitorController(EstablishmentServiceImpl
establishmentServiceImpl, TableServiceImpl tableServiceImpl,
BookedTableServiceImpl bookedTableServiceImpl, UserRepository
userRepository, ReputationRepository reputationRepository,
                WsSender wsSender) {
```

```
            super();
            this.establishmentServiceImpl                                =
establishmentServiceImpl;
            this.tableServiceImpl = tableServiceImpl;
            this.bookedTableServiceImpl = bookedTableServiceImpl;
            this.userRepository = userRepository;
            this.reputationRepository = reputationRepository;
            this.wsSender                                                =
wsSender.getSender(ObjectType.BOOKEDTABLE, Views.Id.class);
        }


        @GetMapping("bookedTable/{id}")
        public    BookedTable    getBookedTable(@PathVariable("id")
String id) throws Exception {
            return bookedTableServiceImpl.getBookedTableById(id);
        }

        @GetMapping("{tableId}/bookedTable")
        public                                    List<BookedTable>
getEstablishmentBookedTable(@PathVariable("tableId")    String    id)
throws Exception {
            return
bookedTableServiceImpl.getActualBookedTableByTableId(id);
        }

        @GetMapping("{userId}/userBookedTable")
        public                                    List<BookedTable>
getUserBookedTables(@PathVariable("userId")    String    id)    throws
Exception {
            return
bookedTableServiceImpl.getBookedTableByUserId(id);
        }

        @PostMapping("bookedTable")
```

```
        public BookedTable createBookedTable(@Valid @RequestBody
BookedTable bookedTable) throws Exception {
            Reputation                    rep                    =
bookedTable.getUser().getReputation();
            if(rep.getOrders()!=0) {
                double                  procent                  =
(rep.getSuccessfulOrders()/rep.getOrders())*100;
                if(procent>=RELIABLE_ORDERS_PROCENT            &&
rep.getOrders()>=RELIABLE_ORDERS_NUMBER)
                    rep.setName(Reputations.RELIABLE);
                else
                    rep.setName(Reputations.UNRELIABLE);
            }

            rep.setOrders(rep.getOrders()+1);
            reputationRepository.save(rep);

            BookedTable           createdBookedTable           =
bookedTableServiceImpl.createBookedTable(bookedTable);
            wsSender.accept(EventType.CREATE,
createdBookedTable);

                return createdBookedTable;
        }

        @DeleteMapping("bookedTable/{id}")
        public     void     deleteBooledTables(@PathVariable("id")
BookedTable bookedTable) throws Exception {

    bookedTableServiceImpl.deleteBookedTable(bookedTable.get_id());
            wsSender.accept(EventType.REMOVE, bookedTable);
        }

        @PutMapping("/bookedTable/{id}")
```

```
        public BookedTable updateBookedTable(@PathVariable("id")
BookedTable bookedTableFromDB, @RequestBody BookedTable
bookedTable) throws Exception {
             BeanUtils.copyProperties(bookedTable,
bookedTableFromDB, "id");
             BookedTable updatedBookedTable =
bookedTableServiceImpl.createBookedTable(bookedTableFromDB);
            wsSender.accept(EventType.UPDATE, updatedBookedTable);

            return updatedBookedTable;
        }



        @PutMapping("profile")
        public User updateProfile(@Valid @RequestBody User user)
throws Exception {
             return userRepository.save(user);
        }


        @GetMapping("checkUserBookedTable/{userId}/{estbId}")
        public boolean
checkActualUserBookedTable(@PathVariable("userId") String userId,
@PathVariable("estbId") String estbId) throws Exception {
             return
bookedTableServiceImpl.checkActualBookedTableByEstablishmentId(user
Id, estbId);
        }

        @GetMapping("checkBookedTable/{table}")
        public List<BookedTable>
checkActualBookedTable(@PathVariable("table") String estbId) throws
Exception {
```

```
            return
bookedTableServiceImpl.getActualBookedTableByTableId(estbId);
        }


    }
```

EstablishmentController.java:

```
    @RestController
    @RequestMapping("/estb")
    public class EstablishmentController {

        @Autowired
        private EstablishmentServiceImpl establishmentServiceImpl;

        @Autowired
        private TableServiceImpl tableServiceImpl;

        @Autowired
        private BookedTableServiceImpl bookedTableServiceImpl;
        @Autowired
        private                  EstablishmentCoordinatesRepository
establishmentCoordinatesRepository;

        @GetMapping("establishment/{id}")
        public                                          Establishment
getEstablishmentById(@PathVariable("id")    String    id)    throws
Exception {
            return establishmentServiceImpl.findById(id);
        }

        @GetMapping("establishment")
        public  List<Establishment>  getAllEstablishment()  throws
Exception {
```

```
            return
establishmentServiceImpl.getEstablishmentList();
        }

        @PostMapping("establishment")
        public Establishment createEstablishment( @RequestBody
Establishment establishment) throws Exception {
            return
establishmentServiceImpl.createEstablishment(establishment);
        }

        @PutMapping("establishment/{id}")
        public                                    Establishment
updateEstablishment(@PathVariable("id")           Establishment
establishmentFromDB, @RequestBody Establishment establishment) {
            return
establishmentServiceImpl.updateEstablishment(establishmentFromDB,
establishment);
        }

        @GetMapping("establishment/{estbId}/table")
        public                                      List<Table>
getEstablishmentTables(@PathVariable("estbId") String id) throws
Exception {
            return tableServiceImpl.getTableListByEstb(id);
        }

        @GetMapping("establishment/table/{id}")
        public Table getTableById(@PathVariable("id") String id)
throws Exception {
            return tableServiceImpl.getTableById(id);
        }

        @GetMapping("establishment/{id}/coordinates")
```

```
        public EstablishmentCoordinates getCoordinates(@PathVariable("id") String id) throws Exception {
            Optional<EstablishmentCoordinates> ec = establishmentCoordinatesRepository.findById(id);
                if(ec.isPresent())
                    return ec.get();
                else throw new NotFoundException("Not Found");
        }

        @PostMapping("establishment/{id}/coordinates")
        public EstablishmentCoordinates setCoordinates(@PathVariable("id") String id, @Valid@RequestBody EstablishmentCoordinates establishmentCoordinates) throws Exception {
            return establishmentCoordinatesRepository.save(establishmentCoordinates);
        }

        @PostMapping("establishment/table")
        public Table createTable(@RequestBody Table table) throws Exception {
            return tableServiceImpl.createTable(table);
        }

        @PutMapping("establishment/table/{id}")
        public Table updateTable(@PathVariable("id") Table tableFromDB, @RequestBody Table table) throws Exception {
            return tableServiceImpl.updateTable(tableFromDB, table);
        }

        @DeleteMapping("establishment/table/{id}")
        public Table deteleTable(@PathVariable("id") Table table) throws Exception {
            tableServiceImpl.deleteTable(table);
```

```
            return table;
        }

    }
```

WebSecurityConfig.java:

```
    @Configuration
    @EnableWebSecurity
    @EnableOAuth2Sso
    public class WebSecurityConfig extends
WebSecurityConfigurerAdapter {
        @Autowired
        private RoleRepository roleRepository;
        @Autowired
        private ReputationRepository reputationRepository;

        @Override
        protected void configure(HttpSecurity http) throws
Exception {

    http.antMatcher("/**").authorizeRequests().antMatchers("/",
"/login**", "/js/**", "/error**").permitAll().
                    antMatchers("/estb/establishment/{estbId}/table",
"/visitor/{tableId}/bookedTable",
"/visitor/bookedTable/{id}","/visitor/profile").
                    hasAnyAuthority("VISITOR", "ESTB").

    antMatchers("/admin/user/{id}/bussines").hasAnyAuthority("VISIT
OR","ADMIN").
                    antMatchers("/estb/establishment/**",
"/estb/establishment/{id}/coordinates").hasAnyAuthority("VISITOR","
ADMIN", "ESTB").

    antMatchers("/visitor/**").hasAnyAuthority("VISITOR").
```

```
                antMatchers("/admin/**").hasAuthority("ADMIN").
                antMatchers("/estb/**").hasAuthority("ESTB").
                anyRequest().authenticated().and()
               .logout().logoutSuccessUrl("/").permitAll().
                and().csrf().disable();
        }

        @Bean
        public PrincipalExtractor principalExtractor(UserRepository userRepository, HttpServletRequest req) {
            return map -> {
                String id = (String) map.get("sub");

                User user = userRepository.findById(id).orElseGet(() -> {
                    Role visitorRole = null;
                    for (Cookie c : req.getCookies()) {
                        if (c.getName().equals("bussines")) {
                            visitorRole = roleRepository.findByName(Roles.ESTB);
                            break;
                        }
                        else visitorRole = roleRepository.findByName(Roles.VISITOR);
                    }

                    User newUser = new User();


    if(visitorRole.getName().equals(Roles.VISITOR)) {
                        Reputation reputation = new Reputation(null, Reputations.UNRELIABLE, 0, 0);

    newUser.setReputation(reputationRepository.save(reputation));
```

```
                    }

                    newUser.setRole(visitorRole);
                    newUser.setId(id);
                    newUser.setName((String) map.get("name"));
                    newUser.setEmail((String) map.get("email"));
                    newUser.setUserpic((String)
map.get("picture"));
                    newUser.setPhone("");

                    return newUser;
                });
                return userRepository.save(user);
            };
        }

        @Bean
        public                                AuthoritiesExtractor
AuthoritiesExtractor(UserRepository userRepository) {
            return map -> {

                String id = (String) map.get("sub");
                Optional<User>              user              =
userRepository.findById(id);
                if(!user.isPresent())
                 return            Collections.<GrantedAuthority>
emptyList();
                return
AuthorityUtils.createAuthorityList(user.get().getRole().getName().t
oString());

            };
        }
```

```
}
```

WebSocketConfig.java:

```
package
iful.edu.bg.
config;


                import
                org.springframework.context.annotation.Configuration;
                import
                org.springframework.messaging.simp.config.MessageBrok
                erRegistry;
                import
                org.springframework.web.socket.config.annotation.Enab
                leWebSocketMessageBroker;
                import
                org.springframework.web.socket.config.annotation.Stom
                pEndpointRegistry;
                import
                org.springframework.web.socket.config.annotation.WebS
                ocketMessageBrokerConfigurer;


                @Configuration
                @EnableWebSocketMessageBroker
                public class WebSocketConfig implements
                WebSocketMessageBrokerConfigurer {


                    @Override
                    public void
                configureMessageBroker(MessageBrokerRegistry config)
                {
                        config.enableSimpleBroker("/topic");

                config.setApplicationDestinationPrefixes("/app");
                    }


                    @Override
                    public void
                registerStompEndpoints(StompEndpointRegistry
                registry) {
                        registry.addEndpoint("/gs-guide-
```

```
                websocket").withSockJS();
                    }


                }
```

BookedTableServiceImpl.java:

```
@Service
public class BookedTableServiceImpl implements BookedTableService
{

     @Autowired
     private BookedTableRepository bookedTableRepository;

     @Autowired
     private TableServiceImpl tableServiceImpl;

     @Autowired
     private EstablishmentServiceImpl establishmentServiceImpl;

     @Override
     public BookedTable createBookedTable(BookedTable bookedTable)
{
          return bookedTableRepository.save(bookedTable);
     }

     @Override
     public List<BookedTable> getActualBookedTableByTableId(String
id) throws Exception {
          Table table = tableServiceImpl.getEstbTableById(id);
          return
bookedTableRepository.findActualBookedTables(table.get_id(), new
Date(System.currentTimeMillis()));
     }

     @Override
     public BookedTable getBookedTableById(String id) throws
Exception {
          Optional<BookedTable> bookedTable =
bookedTableRepository.findById(id);
          if(bookedTable.isPresent())
               return bookedTable.get();
          else throw new Exception("Not found");
     }
```

```
    @Override
    public List<BookedTable> getBookedTableByUserId(String id)
throws Exception {
        // TODO Auto-generated method stub
        return
bookedTableRepository.findBookedTablesByUserId(id, new
Date(System.currentTimeMillis()));
    }

    @Override
    public void deleteBookedTable(String id) {
        bookedTableRepository.deleteById(id);
    }

    @Override
    public boolean checkActualBookedTableByEstablishmentId(String
userId, String estbId) throws Exception {
        Optional<BookedTable> bt =
bookedTableRepository.findActualBookedTablesByEstablishmentId(use
rId, estbId, new Date(System.currentTimeMillis()));
        return bt.isPresent();
    }
```

EstablishmentServiceImpl.java:

```
package iful.edu.bg.service;


      import java.util.List;
      import java.util.Optional;


      import org.springframework.beans.BeanUtils;
      import
    ingframework.beans.factory.annotation.Autowired;
      import org.springframework.stereotype.Service;


      import iful.edu.bg.entity.Establishment;
      import iful.edu.bg.repository.EstablishmentRepository;
      import javassist.NotFoundException;


      @Service
```

```
  public class EstablishmentServiceImpl implements
shmentService {

      @Autowired
      private EstablishmentRepository establishmentRepository;


      @Override
      public List<Establishment> getEstablishmentList() {
           return establishmentRepository.findAll();
      }


      @Override
      public Establishment findById(String id) throws Exception

           Optional<Establishment> establishment =
shmentRepository.findById(id);
           if (establishment.isPresent())
                return establishment.get();
           else
                throw new NotFoundException("Establishment Not
;
      }


      @Override
      public Establishment createEstablishment(Establishment
shment) {
           establishment.setAccepted(false);
           return establishmentRepository.save(establishment);
      }


      @Override
      public Establishment updateEstablishment(Establishment
shmentFromDB, Establishment establishment) {
           BeanUtils.copyProperties(establishment,
shmentFromDB, "id");
           return
shmentRepository.save(establishmentFromDB);
      }


  }
```

TableServiceImpl.java:

```
package iful.edu.bg.service;


	import java.security.Principal;
	import java.util.List;
	import java.util.Optional;


	import org.springframework.beans.BeanUtils;
	import org.springframework.beans.factory.annotation.Autowire
	import org.springframework.stereotype.Service;


	import iful.edu.bg.entity.Establishment;
	import iful.edu.bg.entity.Table;
	import iful.edu.bg.enums.TableStatuses;
	import iful.edu.bg.repository.TableRepository;


	@Service
	public class TableServiceImpl implements TableService {


	    @Autowired
	    private TableRepository tableRepository;


	    @Autowired
	    private EstablishmentServiceImpl establishmentServiceIm


	    @Override
	    public Table getTableById(String id) throws Exception {
	        Optional<Table> table = tableRepository.findById(i
	        if (table.isPresent())
	            return table.get();
	        else
	            throw new Exception("Not found");
	    }


	    @Override
	    public List<Table> getTableListByEstb(String id) throws
	        Establishment estb = establishmentServiceImpl.findI
	        return tableRepository.findAllByEstb(estb);
```

```
    }


    @Override
    public Table getEstbTableById(String id) throws Excepti
        Optional<Table> table = tableRepository.findById(i
        if(table.isPresent())
            return table.get();
        else throw new Exception("Not found");
    }


    @Override
    public Table getEstbTableBySeats(Establishment estb, in
        return tableRepository.findByEstbAndSeats(estb, se
    }


    @Override
    public List<Table> getEstbTableListByStatus(Principal p
        Establishment establishment = establishmentService
        return tableRepository.findAllByEstbAndStatus(esta
    }


    @Override
    public Table createTable(Table table) throws Exception
        table.setStatus(TableStatuses.FREE);
        return tableRepository.save(table);
    }


    @Override
    public Table updateTable(Table tableFromDB, Table table
        BeanUtils.copyProperties(table, tableFromDB, "id")
        return tableRepository.save(tableFromDB);
    }


    @Override
    public void deleteTable(Table table) throws Exception {
        tableRepository.delete(table);
    }
}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: Розробка веб-сервісу резервування місць в закладах харчування. Розробка клієнтської частини за допомогою Vue.js, а серверної – за допомогою Java.

Обсяг пояснювальної записки:. ___69___аркушів

Перелік графічних матеріалів:

- рисунків_____37_____
- додатків_____2 на 34_____аркушів.

Дата закінчення дипломного проекту: “__”_____2019 р.

Студент – дипломник _____

(підпис) (розшифровка підпису)