Національний університет «Полтавська політехніка імені Юрія Кондратюка»

(повне найменування вищого навчального закладу)

Навчально-науковий інститут інформаційних технологій та робототехніки

(повна назва факультету)

Кафедра комп’ютерних та інформаційних технологій і систем

(повна назва кафедри)

**Пояснювальна записка**

**до дипломного проєкту (роботи)**

бакалавра

(освітньо-кваліфікаційний рівень)

на тему

Онлайн-сервіс продаж для закладів громадського харчування

Виконав: студент 4 курсу, групи 401-ТН

спеціальність

122 Комп’ютерні науки

(код і назва спеціальності)

Мизюра М.С.

(прізвище та ініціали)

Керівник Демиденко М.І.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Полтава – 2021 року**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ**

**КАФЕДРА КОМП’ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ**

**КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА**

**спеціальність 122 «Комп’ютерні науки»**

**на тему**

**«Онлайн-сервіс продаж для закладів громадського харчування»**

**Студента групи 401-ТН Мизюри Михайла Сергійовича**

Керівник роботи

старший викладач

Демиденко М.І.

Завідувач кафедри

кандидат технічних наук,

доцент Головко Г.В.

Полтава – 2021

**РЕФЕРАТ**

Кваліфікаційна робота бакалавра: 101 с., 28 малюнка, 2 додатка, 14 джерел.

**Об’єкт дослідження:** процеси розроблення та функціонування онлайн-сервісу продаж для закладів громадського харчування.

**Мета роботи:** розроблення онлайн-сервісу продаж для закладів громадськогого харчування, який стане посередником та інструментом для клієнтів і закладів громадського харчування, що забезпечує зручну можливість створення замовлень онлайн в Україні.

**Методи:** проєктування бази даних онлайн-сервісу продаж для закладів громадського харчування, створення макету користувацького інтерфейсу, побудова архітектури MVC для серверної частини та архітектури MVVM для клієнтської, розроблення програмного коду, застосування методології Scrum та систем контролю версій.

**Ключові слова:** онлайн-сервіс, JavaScript, Nuxt, компонент, фреймворк, Node, Express, MongoDB, REST API, база даних, модель, контроллер.

**ABSTRACT**

Bachelor's qualification work: 101 pages, 28 drawings, 2 additions, 14 sources.

**Object of research:** processes of development and operation of online sales service for catering establishments.

**Purpose:** development of an online sales service for catering establishments, which will become an intermediary and a tool for customers and public catering establishments, which provides a convenient opportunity to create orders online in Ukraine.

**Methods:** designing an online sales service database for catering establishments, creating a user interface layout, building an MVC architecture for the server part and an MVVM architecture for the client, developing code, applying the Scrum methodology and version control systems.

**Keywords:** онлайн-сервіс, JavaScript, Nuxt, компонент, фреймворк, Node, Express, MongoDB, REST API, база даних, модель, контроллер.

**ЗМІСТ**

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ 4](#_Toc10205090)

[ВСТУП 5](#_Toc10205091)

[РОЗДІЛ 1 ОГЛЯД АВТОМАТИЗОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ УСПІШНОСТІ ЗАКЛАДУ ВИЩОЇ ОСВІТИ 7](#_Toc10205092)

[1.1 Опис предметної галузі 7](#_Toc10205093)  
1.2 Аналіз тенденцій розвитку та актуальності 10

[1.3 Огляд наявних програмних рішень 12](#_Toc10205094)

[1.4 Функціональні вимоги 16](#_Toc10205095)

[РОЗДІЛ 2 ПРОЄКТУВАННЯ АВТОМАТИЗОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ УСПІШНОСТІ ЗАКЛАДУ ВИЩОЇ ОСВІТИ 19](#_Toc10205096)

[2.1 Функціонал та структура системи 19](#_Toc10205097)

[2.2 Проєктування бази даних 21](#_Toc10205098)

[2.3 Розробка макетів 24](#_Toc10205099)

[РОЗДІЛ 3 РОЗРОБКА АВТОМАТИЗОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ УСПІШНОСТІ ЗАКЛАДУ ВИЩОЇ ОСВІТИ 38](#_Toc10205100)

[3.1 Вибір платформи 38](#_Toc10205101)

[3.2 Розробка бази даних 39](#_Toc10205102)

[3.3 Розробка користувацького інтерфейсу 40](#_Toc10205103)

[РОЗДІЛ 4 ТЕСТУВАННЯ АВТОМАТИЗОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ УСПІШНОСТІ ЗАКЛАДУ ВИЩОЇ ОСВІТИ 49](#_Toc10205104)

[4.1 Вибір різновиду тестування 49](#_Toc10205105)

[4.2 Тест план 49](#_Toc10205106)

[4.3. Введення в експлуатацію 53](#_Toc10205107)

[ВИСНОВОК 54](#_Toc10205108)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55](#_Toc10205109)

[ДОДАТОК А ВИХІДНИЙ КОД ОСНОВНИХ ФАЙЛІВ СЕРВЕРА 57](#_Toc10205110)

ДОДАТОК В ВИХІДНИЙ КОД ОСНОВНИХ ФАЙЛІВ КЛІЄНТА [81](#_Toc10205110)

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

**AJAX** – Asynchronous JavaScript And XML

**HTML** – Hypertext Markup Language

**CSS** – Cascading Style Sheets

**SASS** – Syntactically Awesome Stylesheets

**JS** – JavaScript

**DOM** – Document Object Model

**REST** – Representational State Transfer

**API** – Application Programming Interface

**MVC** – Model-View-Controller

**MVVM** – Model-View-ViewModel

**YARN** – Node package manager

**JSON** – JavaScript Object Notation

**БД** – База даних

**СКБД** – Система керування базами даних

**ПЗ** – Програмне забезпечення

**ПК** – Персональний компьютер

**UI** – User Interface

**UX** – User Experience

**SPA** – Single-page application

**GUI** – Graphical user interface

ВСТУП

Основною метою дипломної роботи є розроблення онлайн-сервісу продаж для закладів громадського харчування, що є зручним електронним засобом для покращення якості обслуговування клієнтів і ефективним помічником для бізнесу.

Предмет дослідженя: галузь громадського харчування

Онлайн-сервіс надає перелік інструментів для зручної взаємодії клієнта та заклада громадського харчування. Для клієнта такими інструментами є: наявність рейтинга закладів та можливість оцінювання рівня обслуговування, пошук за кафе та стравами, за знижками та акціями, фільтри за ціною та годинам роботи тощо. Для власника онлайн-сервісу це дає можливість покращити якість обслуговування клієнта.

Особливу хвилю актуальності онлайн-сервіси отримують у сьогоденні, під час пандемії коронавірусної хвороби і карантиних обмежень. У цей період рекомендується зменшити контакт із людьми до мінімума та сидіти вдома, тому у такий час на допомогу приходять сервіси, що мают можливість працювати онлайн і безконтактно.

На даний момент є достатньо електронних сервісів продаж для закладів громадського харчування, які мають інструменти для створення замовлення онлайн, з можливістю доставки їх за адресою. Можна виділити такі: «Uber Eats», «Glovo», «Rocket», «Yandex.Їжа» та «Delivery Club», але перелічені системи мають недоліки, серед яких різна спеціалізація, складність підтримання, вартість, відсутність покриття, бо лише декілька з них працюють в Україні.

Оскільки метою дипломної роботи є розроблення онлайн-сервісу продаж для закладів громадського харчування, що створює інструменти та можливості зручної комунікації між клієнтом та закладом, ми можемо визначити головну вимогу – наявність розповсюдженого, простого та зручного доступу до системи для користувачів. Тому в якості клієнтської частини був обраний вебсервіс, що запускається в будь-якому браузері та дає можливість користуватися сервісом із будь-якого пристрою, що має доступ до мережі інтернет;

Світові тенденції в розробленні та використанні користувальницьких інтерфейсів свідчать про стрімкий розвиток вебтехнологій. За такої умови програмний код інтерфейсів надається користувачам із допомогою звичайного вебсервера, а стандартне програмне клієнтське забезпечення надається практично будь-яким браузером. [1, с. 15]

РОЗДІЛ 1  
ОГЛЯД ОНЛАЙН-СЕРВІСУ ПРОДАЖ ДЛЯ ЗАКЛАДІВ ГРОМАДСЬКОГО ХАРЧУВАННЯ

1.1 Опис предметної галузі

Онлайн-сервіс продаж для закладів громадського харчування – це різновид послуг і сервісу, який надається закладами громадського харчування з допомогою електронних засобів. Онлайн-сервіс надає цілу низку переваг та інструментів для зручної взаємодії клієнта та заклада громадського харчування, на відміність від оффлайн-сервісу. Для клієнта такими перевагами є: наявність рейтинга закладів та можливість оцінювання рівня обслуговування, пошук за кафе та стравами тощо. Для власника онлайн-сервісу продаж це дає можливість насамперед покращити якість обслуговування потенційного клієнта завдяки таких інструментів:

* онлайн сторінка в браузері чи мобільний додаток, завдяки яким заклад громадського харчування може надавати послуги в режимі онлайн. Це забезпечує непреривну роботу закладу та додаткові замовлення, які можуть надходити пізно ввечері тощо. Також, у період обмежень, наприклад, у пандемію, коли клієнти не можуть відвідати ресторан, у них є можливість замовити доставку додому;
* електронне меню, яке дає змогу редагувати в реальному часі наявність знижок чи спеціальних пропонувань, також забезпечує підтримання актуальної інформації для клієнта в меню, наприклад, наявність тієї чи іншої страви, її ціни чи опису;
* збір та аналіз інформації, яка дає змогу слідкувати за замовленнями клієнтів, вести статистику продаж, планувати доходи та витрати.

В Україні онлайн-сервіси продаж громадського харчування поділяються на кілька підвидів.

Агрегатор – це онлайн-ресурс, який дає змогу обирати з багатьох пропонувань найкращу. Розглянемо його роботу на прикладі онлайн-закладів громадського харчування. Агрегатор збирає доступну інформацію про меню ресторанів і кафе, що дає змогу користувачу відфільтрувати, наприклад за ціною чи рейтингу й обрати найкраще пропонування серед доступних. Недоліком таких вебсайтів є те, що агрегатор лише збирає й показує інформації клієнту, не даючи можливості довести замовлення до кінця [2]. Як тільки замовлення було підтверджено, агрегатор передає його до ресторану, і фактичну доставку здійснюють власні кур'єри ресторану.

Оскільки вимоги користувачів зростають разом із технологічним прогресом, люди все більше прагнуть автоматизації, щоб їх не перенаправляли на вебсайт ресторану для завершення замовлення, а щоб усе можна було зробити на одному сайті: і знайти кафе, й обрати страви, і зробити замовлення однією кнопкою, з можливістю сплатити онлайн. Люди очікують, що програма доставки їжі буде керувати повним процесом оформлення замовлення та доставки. Ось чому агрегатори поступово перетворюються на сервіси.

Сервіс – наступна ступінь розвитку агрегатора. Сервіс використовує всі переваги агрегатора, але водночас вирішує його недолік, а саме: дає можливість робити замовлення без переходу на інші сторінки. Також, він може, за вашим дозволом, зберігати різну інформацію про вас, таку як: адресу доставки чи обраний метод оплати, у такий спосіб, полегшуючи та пришвидшуючи оформлення онлайн-замовлення. Крім того, сервіс може нести відповідальність за переказ коштів клієнтів, і в разі виникнення проблеми надати онлайн підтримання чи допомогу.

Платформа – наступна ступінь розвитку сервісу. Поєднує в собі не тільки онлайн-сервіс, а й логістичну компанію для організації доставки. Платформи доставки їжі дають змогу користувачам порівнювати меню в ресторанах та замовляти страви з різних ресторанів через єдиний вебсайт або мобільний додаток так само, як і сервіси, але ключовою відмінністю є те, що на додаток до агрегування і зручного оформлення замовлення, такі платформи також забезпечують логістику.

Надання логістичних послуг відкриває нові горизонти ресторанам, які ніколи не займалися доставкою їжі самостійно. Для ресторанів партнерство з такими платформи може бути корисним, даючи їм змогу пропонувати зручну доставку, не турбуючись про організацію оплати водіїв, відстеження їх руху, покриття витрат на технічне обслуговування автомобілів та сплату страхових полісів тощо. Натомість ресторани можуть зосередитися на більш важливих речах, таких, як:

* покращення якості страв;
* покращення якості і швидкості обслуговування;
* розвиток бізнесу.

Основними перевагами онлайн-сервісів продаж для закладів громадського харчування є зручність використання, що включає в себе:

* економію часу клієнта, який бажає зробити замовлення;
* збільшення конкуренції на ринку закладів громадського харчування, що зі свого боку призводить до покращення якості обслуговування та зниження цін;
* мінімум контакту з людьми в умовах пандемії.

Отже, онлайн-сервіс продаж для закладів громадського харчування це наступна ступінь еволюції розвитку бізнесу, яка використовується власниками закладів, для того, щоби покращити якість послуг, а клієнтами – щоб здобути їх зручно і швидко.

Основною метою дипломної роботи є розроблення такого сервісу. Бізнес-процес системи складається з таких підпроцесів:

* управління електронним меню;
* обробка онлайн замовлень;
* створення онлайн замовлень;
* статистика замовлень;
* оцінювання якості обслуговування.

Саме тому, у результаті має бути розроблений функціонал електронного меню з можливістю його швидкого та простого редагування, функціонал створення та обробки електронного замовлення, функціонал оцінювання якості обслуговування та автоматичний збір статистики замовлень, що є основними функціями, які надають перевагу онлайн сервісам над оффлайн.

1.2 Аналіз тенденцій розвитку

Економічне зростання та глобальне розповсюдження інтернету є рушієм глобальної експансії електронної комерції, а саме онлайн-сервісів продаж громадського харчування.

Електронна комерція – це одна із сфер цифрової економіки, що включає всі фінансові та торгові транзакції, які проводяться з допомогою комп'ютерних мереж, та бізнес-процеси, що пов'язані з проведенням та підтриманням цих транзакцій [3].

За останні десять років ринок електронної комерції зазнав значного зростання, оскільки клієнти все частіше переходять у мережу інтернет. Ця зміна способу покупок споживачів здійснюється завдяки різноманітних чинників, деякі залежать від ринку чи країни, інші виникають у результаті світових змін. Ці зміни включають:

* збільшення доходу і електронної грамотності населення, особливо в країнах, що розвиваються;
* довший робочий день;
* збільшення часу їзди на роботу;
* постійне збільшення проникнення мережі інтернет у повсякденне життя людей;
* покращена безпека електронних платежів;
* збільшення кількості малого та середнього бізнесів, що мають присутність у мережі інтернет;
* зростаюча довіра та обізнаність клієнтів до електронної комерції.

Але так було не завжди, розглянемо це на прикладі, на сьогоді, доволі, розповсюдженої послуги – доставки їжі за адресою. Доставка не була створена за день. Концепція служби доставки їжі датується ще давніми часами Стародавнього Риму. Тогочасні торговці виставляли страви біля прилавка з великих глиняних горщиків, які зберігали та підтримували їжу гарячою, подібно до того, як сьогодні працює фаст-фуд.

Наступним кроком розповсюдження доставки стала доставка молока в Сполучених Штатах Америки. Історично склалося так, що їхні сім’ї майже завжди мали доступ до свіжого молока, оскільки, сім’я мала корову. Але коли Сполучені Штати Америки почали процес індустріалізації, у сільській місцевості почало проживати все менше і меньше людей. Отже, вони почали закуповувати молоко в місцевих виробників. Оскільки молоко швидко псується, і мало хто мав доступ до холодильника в ті часи, щоденні поставки були необхідністю. Як результат, доставка молока стала звичним способом життя для багатьох американців [5].

Наступний значний крок еволюції служб доставок стався упродовж 1950-х у Сполучених Штатах Америки через те, що сім’ї здебільшого вважали за краще їхати додому на вечерю й що майже в кожній родині був телевізор вдома. Переважна більшість сімей вважали за краще їсти під час перегляду своїх улюблених телевізійних шоу, що призводить до різкого зменшення кількості клієнтів у ресторанах. Ресторани харчування стимулювали свій бізнес, розміщуючи рекламу на телебаченні, це вводило концепцію замовлення їжі додому та створення служб доставлення.

З появою мережі інтернет з’явилась і електронна комерція, яка одразу почала спочатку доповнювати оффлайн бізнес, а потім і витискати його в деяких ринках. Але завоювати довіру в людей, які звикли сплачувати готівкою та довіряти магазинам, у яких вони можуть перебувати фізично й переконатися в тому, що завтра цей магазин не зникне, не легко. Але прогрес не зупинити, тому з розвитком електронних платіжних систем розвивалась їхня захищеність та надійність, а водночас і довіра покупців. Також, не малу роль зіграло те, що все більше бізнесу переходило в онлайн.

Сьогодні ми знайомі з концепцією замовлення їжі в інтернеті та отримання їжі з допомогою сучасних служб доставки. Можна легко замовити їжу чи продукти в інтернеті з допомогою таких програм, як Yandex.Їжа, Rocket, Glovo та Uber Eats. Іноді ресторани можуть співпрацювати з програмами для замовлення страв через інтернет, щоби розмістити свої ресторани у своїй програмі. У цьому випадку ресторани можуть просто зосередитися на підготовці замовлення, розміщеного замовником.

Додатки для замовлення та доставки їжі включають адреси клієнтів доставщикам, які доставляють їжу клієнту. Ми можемо здійснювати операції з купівлі продуктів через інтернет-банкінг, накладеним платежем та негайною оплатою з допомогою карток.

Винос та доставка перетворилися на настільки нормальною частиною нашого життя, що практично неможливо уявити його без цього, особливо в умовах обмежень, наприклад, пандемій тощо.

1.3 Огляд наявних програмних рішень

На сьогодні працює велика кількість електронних сервісів продаж для закладів громадського харчування, які призначені на різні потреби й надають різні можливості для їхніх клієнтів. Розглянемо на прикладі сервісів, що дають змогу створювати онлайн-замовлення, з можливістю доставки за адресою. Серед них можна виділити такі:

* Uber Eats;
* Glovo;
* Rocket;
* Yandex.Їжа;
* Delivery Club.

Вище перечисленні системи мають свої недоліки, серед яких, наприклад, обмежена доступність, бо тільки «Glovo» та «Rocket» працюють в Україні.

Оцінювання наявних програмних рішень буде відбуватися за наступними критеріям:

* плата за доставку. Програми здебільшого вимагають стандартної плати за доставку, але деякі додатки залишають суму оплати на вибір ресторана, у якого ви вирішили зробити замовлення. Відтак потрібно перевірити вартість доставки перед тим, як замовляти;
* мінімальне замовлення. ресторани зазвичай встановлюють мінімальну вартість замовлення, або суму, після якої доставка буде безкоштовною. Тому, потрібно бути уважним при оформленні, та ознайомитися з умовами. Можливо, буде краще додати до замовлення ще одну страву, щоби отримати безкоштовну доставку;
* спосіб оплати. Звичайно, більшість програм для доставки їжі приймають основні кредитні та дебетові картки, готівку, електронні методи платежів тощо. Тому переконайтесь, в тому, що вам підходить спосіб оплати, вашого замовлення;
* рейтинг платформи. Незалежно від того, клієнт ви чи доставщик, важливо взяти до уваги рейтинг ресторана. Це стосується і умов праці, якщо ви плануєте працювати в доставці, і якості обслуговування, якщо ви – потенційний клієнт.
* інтерфейс користувача. Програми доставки їжі призначені для зручності і мають на меті спростити процесс здійснення покупки. Обираючи онлайн-сервіс продаж для закладів громадського харчування, зверніть увагу на зручність користування та простоту інтерфейсу. Користуйтесь фільтрами, такими, як: ціна меню, тип кухні та рейтинги користувачів. Деякі програми навіть дозволяють відстежувати ваші замовлення в режимі реального часу на мапі.

Оцінюючи, яку службу доставки їжі використовувати, подивіться, наскільки широким є охоплення певної служби, де ви проживаєте, і чи може вона приносити вам їжу з ресторанів, де ви любите їсти. Подумайте, яка послуга стягує додаткову плату, хоча це може бути важко розглянути, оскільки більшість зборів за доставку здійснюються на розсуд ресторанів, у яких ви замовляєте.

Якщо ви плануєте широко використовувати послугу, подивіться, чи існує щомісячна програма членства. У багатьох випадках членський внесок може знищити будь-які збори за доставку або зменшити інші внески. У деяких випадках ви навіть можете розраховувати на швидшу доставку, якщо у вас є статус членства.

Те, що ви плануєте замовляти, може визначити, яким сервісом вам краще користуватися. Grubhub, та UberEats спеціалізуються на стравах з ресторанів. Glovo, Rocket та Delivery.com займаються іншими видами доставки їжі.

Далі будуть перечисленні найкращі програми доставки їжі 2020 року, перевірені рейтингами користувачів в App Store, Google Play та онлайн-оглядами по Україні та світу:

* Uber Eats – найкраща служба доставки їжі, якщо ви вже користуєтесь Uber-таксі. Не працює в Україні, але доступна майже в 500 міст у 24 країнах світу. Вступ Uber в ринок доставки їжі здається природним продовженням програми для спільного поїздки, що привносить компанію у світ доставки страв. Користувачі Uber Eats можуть легко шукати улюблений ресторан або відкривати нові місця харчування поблизу. Потім, лише за кілька натискань, користувачі можуть зробити замовлення на негайну або заплановану доставку та відстежувати замовлення в режимі реального часу, точно так само, як якщо б вони забронювали поїздку в Uber-таксі. Оплата може здійснюватися як кредитною карткою або кредитами на своєму рахунку, так і готівкою. Додаток дає можливість перечисляти чайові людині, яка приносить вам замовлення, а також залишати інструкції щодо безконтактної доставки їжі, особливу популярність це здобуло в період пандемії. Ще одним плюсом цієї служби доставки є розповсюдженість, що дає можливість замовляти їжу в багатьох великих містах світу [6];
* Glovo – іспанський стартап з доставки страв, продуктів, непродовольчих товарів впродовж однієї години. Працює з ресторанами, кафе, магазинами, ринками та іншими. Сервіс був заснований в 2015 році в місті Барселона. Він виділяється з ряду великих європейських служб доставки тим, що пропонує привезти будь-який невеликий предмет на вимогу клієнта. Наприклад, забрати речі з хімчистки або привезти забуті ключі тощо. Сервіс доступний в 26 країнах світу. Компанія фокусується на ринках, де не працюють такі гіганти як Deliveroo і Uber Eats, наприклад, в Україні тощо. Замовлення можна зробити у додатку та на вебсайті. Платформа Glovo пов'язує три сторони: користувача, бізнес (магазини, ресторани та інше) і Glovers - незалежних кур'єрів. Додаток стягує комісію, залежну від вартості товару з партнерів і плату за доставку з користувачів [7];
* Rocket – українська компанія-сервіс доставки їжі та продуктів із супермаркетів за допомогою мобільного додатку [8]. Також, пропонує своїм покупцям можливість замовити швидку доставку або вибрати зручний час для клієнта, при компанія гарантує те, що доставка замовлення не займе більше однієї години;
* Grubhub – одна з найрозповсюдженіших назв, яке спадає на думку, коли йде мова, онлайн-сервіс продаж громадського харчування, а саме про послуги доставки їжі. Співпрацює з неймовірною кількістю ресторанів, тому зручно обирати серед великої кількості варіантів;
* Yandex.Їжа – це послуга від широко відомої компанії Yandex. Замовлення доставки страв відбувається через мобільний додаток або вебсайт. Доставку можуть здійснювати і кур'єрські служби, і партнери компанії Yandex, і власні кур'єри ресторану, із якого оформлюється замовлення. Є можливість обирати серед піцерій, пекарень, грузинських та японських ресторанів, фаст-фудів, стейкхаусів та серед інших закладів громадського харчування. Працює більш ніж з 14000 ресторанів та кафе. Працює виключно в містах Росії та Казахстану;
* Delivery Club – велика російська компанія, вебсервіс та мобільна платформа, яка була створена в 2009 році. Станом на 2017 рік, програма об'єднує 5 тисяч 200 ресторанів, які щодня через сайт Delivery Club приймають по 25 тисяч замовлень. Сервіс має логістичну службу з власним парком автомобілів, моторолерів і велосипедів, що дозволяє кур’єрам швидко доставляти замовлення. Використовуючи можливості цієї компанії ви можете вибрати кращі ресторани і кафе вашого міста, порівняти ціни на страви, зробити замовлення і заробити акційні бали, що можуть знадобитися в майбутньому [9];
* Delivery.com – американська онлайн-платформа та набір мобільних додатків, що дозволяє користувачам створювати замовляння на доставку у місцевих кафе, ресторанах та магазинах. Пропонує широкий спектр варіантів доставки, що включає послуги з доставки їжі, послуга таксі, відвезення та забирання одягу з хімчистки тощо, також може привести продовольчі товари з магазину. Крім того, є функція групового замовлення, яка дозволяє розділити замовлення разом із сусідами, якщо ви, наприклад, живите в студентському містечку, або замовляєте обід на роботу з колегами, щоб мінімізувати кількість доставок та їхню ціну [10].

1.4 Функціональні вимоги

Для того, аби визначити функціонал онлайн-сервісу для закладів громадського харчування, розглянемо функції, що можуть виконувати користувачі та адміністратори ресторану.

Таблиця 1.1 – Функції неавторизованиго клієнта

|  |  |  |
| --- | --- | --- |
| №  з/п | Функція | Роль |
|  | ПЕРЕГЛЯД. Перегляд списку доступних ресторанів. | Неавторизований клієнт |
|  | СТВОРЕННЯ. Створення онлайн-замовлення. | Неавторизований клієнт |
|  | РЕЄСТРАЦІЯ. Реєстрація на вебсайті в якості клієнта. | Неавторизований клієнт |
|  | РЕЄСТРАЦІЯ. Реєстрація на вебсайті в якості закладу. | Неавторизований клієнт |

Таблиця 1.2 – Функції авторизованого клієнта

|  |  |  |
| --- | --- | --- |
| №  з/п | Функція | Роль |
|  | ВХІД. Вхід в систему під своїми даними. | Авторизований клієнт |
|  | АКТИВАЦІЯ. Активація аккаунту по пошті. | Авторизований клієнт |
|  | ЗМІНА. Зміна пароля по пошті. | Авторизований клієнт |
|  | СТВОРЕННЯ. Створення онлайн-замовлення. | Авторизований клієнт |
|  | ПЕРЕГЛЯД. Перегляд списку актуальних та онлайн-замовлень та історії. | Авторизований клієнт |
|  | РЕДАГУВАННЯ. Редагування особистих даних адреси доставки. | Авторизований клієнт |
|  | ОЦІНЮВАННЯ. Оцінювання якості замовлення. | Авторизований клієнт |

Таблиця 1.3 – Функції авторизованого закладу

|  |  |  |
| --- | --- | --- |
| №  з/п | Функція | Роль |
|  | ВХІД. Вхід в систему під своїми даними. | Авторизований заклад |
|  | АКТИВАЦІЯ. Активація аккаунту по пошті. | Авторизований заклад |
|  | ЗМІНА. Зміна пароля по пошті. | Авторизований заклад |
|  | РЕДАГУВАННЯ. Встановлення статусу замовлення. | Авторизований заклад |
|  | ПЕРЕГЛЯД. Перегляд списку актуальних та онлайн-замовлень та історії. | Авторизований заклад |
|  | РЕДАГУВАННЯ. Додавання, редагування та видалення ресторану. | Авторизований заклад |
|  | РЕДАГУВАННЯ. Додавання, редагування та видалення категорій. | Авторизований заклад |
|  | РЕДАГУВАННЯ. Додавання, редагування та видалення позицій. | Авторизований заклад |
|  | ПЕРЕГЛЯД. Перегляд списку доступних ресторанів. | Авторизований заклад |

РОЗДІЛ 2  
ПРОЄКТУВАННЯ ОНЛАЙН-СЕРВІСУ ДЛЯ ЗАКЛАДІВ ГРОМАДСЬКОГО ХАРЧУВАННЯ

2.1 Функціонал та структура системи

З огляду на специфіку системи, для якої розроблюються модулі, процес переходу до її використання може виявитися складним і доволі тривалим. Водночас цей процес ускладнюється індивідуальними особливостями структури університету. На рисунку 2.1 зображено архітектуру інформаційної системи.

Diagram

Description automatically generated

Рисунок 2.1 – Архітектура системи

Проаналізуємо функції, які можуть виконувати цільові працівники університету, та зобразимо їх на діаграмі використання (Рисунок 2.2).

Diagram

Description automatically generated with low confidence

Рисунок 2.2 – UML-діаграма онлайн-сервісу для закладів громадського харчування

2.2 Проєктування бази даних

Розпочнемо процесс проектування БД, її структуру, опис моделей, список колекцій і логічні зв’язки між ними.

Визначимо всі схеми онлайн-сервіса для закладів громадського харчування:

Схема «User» зберігає інформацію про аккаунт користувача онлайн-сервісу та має такі параметри:

* id – унікальний ідентифікатор, має тип ObjectID, не може бути пустим;
* info – об’єкт, що включає інформацію про базову адресу замовлення, має тип Object. Містить у собі такі поля:
  + city – назва міста для доставлення замовлення, має тип String, може бути пустим;
  + description – додаткова текстова інформація до замовлення, має тип String, може бути пустим;
  + entranceNumber – номер під’їзда в будівлі, має тип String, може бути пустим;
  + floorNumber – номер поверха, має тип String, може бути пустим;
  + name – ім’я клієнта для подальношо звернення, має тип String, може бути пустим;
  + phone – мобільний номер клієнта, має тип String, може бути пустим;
  + roomNumber – номер кімнати, має тип String, може бути пустим;
* email – пошта користувача, має бути унікальною, має тип String, не може бути пустим;
* password – зашифрований пароль користувача, має тип String, не можу бути пустим;
* role – роль користувача, може бути або клієнтом або рестораном, має тип String, не може бути пустим;
* isActive – чи підтверджена пошта користувача, має тип Boolean, не може бути пустим;
* verificationHash – хеш для підтвердження пошти, має тип String, може бути пустим;
* passwordResetHash – хеш для відновлення паролю, має тип String, може бути пустим.

Схема «Restaurant» містить доступну інформацію про ресторан для клієнтів і має такі параметри:

* id – унікальний ідентифікатор, має тип ObjectID, не може бути пустим;
* description – опис ресторана, має тип String, може бути пустим;
* image – фотографія ресторана, має тип String, не можу бути пустим;
* settlements – масив назв населених пунктів України, має тип Array, не може бути пустим;
* keywords – ключові слова для пошуку ресторана, має тип String, може бути пустим;
* minDeliveryOrderPrice – ціна мінімального замовлення, має тип Number, може бути пустим;
* minDeliveryTimeInMinutes – мінімальний час очікування доставлення замовлення, має тип Number, може бути пустим;
* ownerId – унікальний ідентифікатор користувача, що створив рестран, має тип ObjectID, не може бути пустим;
* categories – список категорій для ресторана, має тип Array, може бути пустим.

Схема «Category» містить інформацію про категорію ресторана і має такі параметри:

* name – назва категорії, має тип String, не може бути пустим;
* restaurantId – унікальний ідентифікатор ресторана, до якого належить категорія, має тип ObjectID, не може бути пустим;
* positions – список позицій категорії для ресторана, має тип Array, може бути пустим.

Схема «Position» містить інформацію про позицію категорії для ресторана і має такі параметри:

* name – назва позиції, має тип String, не може бути пустим;
* image – фотографія позиції, має тип String, не можу бути пустим;
* description – текстовий опис позиції, має тип String, може бути пустим;
* price – ціна позиції, має тип Number, не може бути пустим;
* weight – вага позиції, має тип Number, не може бути пустим;
* restaurantId – унікальний ідентифікатор ресторана, до якого належить позиція, має тип ObjectID, не може бути пустим;
* categoryId – унікальний ідентифікатор категорії, до якої належить пазиція, має тип ObjectID, не може бути пустим.

Схема «Order» містить інформацію про онлайн-замовлення ресторана і має такі параметри:

* number – порядковий номер замовлення, має тип Number, не може бути пустим;
* list – об’єкт, що включає інформацію замовлення, має тип Object. Містить у собі такі поля:
  + name – назва позиції, має тип String, не може бути пустим;
  + amount – кількість позицій, має тип Number, не може бути пустим;
  + price – ціна однієї позиції, має тип Number, не може бути пустим;
* isDelivered – признак, що свідчить про те, чи доставленно замовлення, має тип Boolean, не може бути пустим;
* info – об’єкт, що включає інформацію про адресу замовлення, має тип Object. Містить у собі такі поля:
  + city – назва міста для доставлення замовлення, має тип String, може бути пустим;
  + description – додаткова текстова інформація до замовлення, має тип String, може бути пустим;
  + entranceNumber – номер під’їзда в будівлі, має тип String, може бути пустим;
  + floorNumber – номер поверха, має тип String, може бути пустим;
  + name – ім’я клієнта для подальношо звернення, має тип String, може бути пустим;
  + phone – мобільний номер клієнта, має тип String, може бути пустим;
  + roomNumber – номер кімнати, має тип String, може бути пустим;
* userId – унікальний ідентифікатор користувача замовлення, має тип ObjectID, може бути пустим;
* restaurantId – унікальний ідентифікатор ресторана, до якого належить категорія, має тип ObjectID, не може бути пустим;

2.3 Розробка макетів

Розробимо макет вигляду сторінок онлайн-сервісу для закладів громадського харчування, з допомогою векторного онлайн-сервісу розробки інтерфейсів та прототипування Figma. Результат моделювання наведений далі (Рисунок 2.3 – 2.26).

Graphical user interface, application

Description automatically generated

Рисунок 2.3 – Макет сторінки входу в систему

Після успішного входу в аккаунт користувач потраплає на головну сторінку з переліком ресторанів і з’являється можливість перейти на сторінку профіля, де зберігаються актуальні замовлення та історія, а також, інформація про те, чи активовано аккаунт і поля адреси замовлення.

Graphical user interface, application

Description automatically generated

Рисунок 2.4 – Макет сторінки реєстрації в системі

Після успішної реєстрації користувача буде направленно на сторінку з інструкцією активації аккаунта. Активація акаунта робить доступним деякі фунції, наприклад, для користувача – залишати відгук про якість обслуговування, а для ресторана – заповнювати електронне меню.

Graphical user interface, application

Description automatically generated

Рисунок 2.5 – Макет сторінки повідомлення про інструкції для підтвердження пошти

Graphical user interface, text, application, email

Description automatically generated

Рисунок 2.6 – Макет листа підтвердження пошти

Graphical user interface, application

Description automatically generated

Рисунок 2.7 – Макет сторінки повідомлення проуспішне підтвердження пошти

У випадку, якщо користувач забув, або хоче змінити пароль до аккаунта, в нього є така можливість. Макет і процедура зміни паролю наведена нижче.

Graphical user interface, application

Description automatically generated

Рисунок 2.8 – Макет сторінки зміни паролю

Graphical user interface, text, application

Description automatically generated

Рисунок 2.9 – Макет листа зміни паролю

Graphical user interface, application

Description automatically generated

Рисунок 2.10 – Макет сторінки введеня нового паролю

Graphical user interface, application

Description automatically generated

Рисунок 2.11 – Макет сторінки повідомлення про успішну зміну пароля

Якщо користувач баче повідомлення, про успішну зміну пароля, то це означає, що попередній пароль вже недійсний, а потрібно використовувати новий.

Після входу в аккаунт, користувача буде направлено на головну сторінку пошуку ресторанів.

Graphical user interface, website

Description automatically generated

Рисунок 2.12 – Макет головної сторінки

Для того, щоб створити онлайн-замовлення, потрібно спочатку обрати місто, в якому ви знаходитесь, Зробити пошук за бажаним пунктом та обрати ресторан, натиснувши на нього.

Користувач потрапить на сторінку ресторана, де побаче електронне меню, з можливістю переглядати позиції та додавати їх у кошик, змінювати кільксть, а потім і створити замовлення.

Graphical user interface, website

Description automatically generated

Рисунок 2.13 – Макет сторінки ресторана

Graphical user interface, application

Description automatically generated

Рисунок 2.14 – Макет модального вікна кошика

Graphical user interface, application

Description automatically generated

Рисунок 2.15 – Макет модального вікна створення замовлення

Після того, як замовлення створення, воно потраплає в кабінет ресторана, що його опрацьовує і клієнта, якщо він авторизований. Нижче розглянемо макети профіля ресторана.

Graphical user interface, website

Description automatically generated

Рисунок 2.16 – Макет профіля ресторану зі списком ресторанів

Graphical user interface, website

Description automatically generated

Рисунок 2.17 – Макет модального вікна створення та редагування ресторана

Graphical user interface, application, website

Description automatically generated

Рисунок 2.18 – Макет списку категорій обраного ресторану

Graphical user interface, website

Description automatically generated

Рисунок 2.19 – Макет модального вікна створення та редагування категорії

Graphical user interface, website

Description automatically generated

Рисунок 2.20 – Макет списку позицій обраного ресторану і категорії

Graphical user interface, application

Description automatically generated

Рисунок 2.21 – Макет модального вікна створення та редагування позиції

Graphical user interface

Description automatically generated

Рисунок 2.22 – Макет сторінки перегляду ресторанів із замовленнями

Graphical user interface, application

Description automatically generated

Рисунок 2.23 – Макет сторінки перегляду актуальних замовлень за обраним рестораном

Graphical user interface, application

Description automatically generated

Рисунок 2.24 – Макет модального вікна перегляду обраного актуального замовлення

Graphical user interface, application

Description automatically generated

Рисунок 2.25 – Макет сторінки перегляду історії замовлень за рестораном

Кабінет клієнта містить інформації по його актульним онлайн замовлення, історії та сторінку збереження адреси доставки за замовчуванням.

Graphical user interface, application

Description automatically generated

Рисунок 2.26 – Макет сторінки збереження адреси доставки за замовчуванням

Макет онлайн-сервісу продаж для закладів громадського харчування розроблено з дотриманням методології розробки SCRUM, зі збереженням єдиної кольорової палітри та візуального стилю. При розробці макету був використаний перелік найкращих UI та UX практик, що забезпечує зручний, зрозумілий, простий та візуально приємний користувацький інтерфейс.

РОЗДІЛ 3  
ПРОГРАМНА РЕАЛІЗАЦІЯ ОНЛАЙН-СЕРВІСУ ПРОДАЖ ДЛЯ ЗАКЛАДІВ ГРОМАДСЬКОГО ХАРЧУВАННЯ

3.1 Вибір платформи

Розробляючи клієнтську частину онлайн-сервісу продаж для закладів громадського харчування використовується веб платформа, то клієнтська частина використовує HTML, CSS та JavaScript, та рушій Chrome V8. Фреймворк Vue.js та Nuxt.js було використано, щоб спростити та пришвидшити розробку, використовуючи компонентний підхід створення вебзастосунків.

Vue – це швидкий, простий та прогресивний фреймворк для побудови користувальницьких інтерфейсів. На відміну від інших фреймворків, Vue розроблений з нуля, що дозволяє його поступово застосовувати. Основна бібліотека орієнтована лише на процес візуального представлення та його легко підібрати та інтегрувати з іншими бібліотеками або існуючими проектами. З іншого боку, Vue також чудову можливість у застосуванні зі складними SPA, коли використовується у поєднанні з сучасними інструментами розробли та бібліотеками підтримки [11]. У проекті використовується Nuxt, для того, щоб швидше рендерити візуальну частину на стороні клієнта.

Nuxt.js – це фреймворк для створення універсальних програм на Vue.js з використанням Node.js. За допомогою нього можна рендерити UI на сервері, що дозволяє швидше їх відобратизи клієнту [12]. Також, дозволяє генерувати статичні сайти. В ньому використовуються такі основні NPM модулі, як:

* Vuex;
* Vue-Router;
* Vue-Meta;
* vue-loader:
* babel-loader;
* Webpack.

Для того, щоб мати можливість використовувати одну і ту ж мову програмування – JavaScript, як на клієнтській, так і на серверній стороні, Node.js було обрано в якості серверної платформи.

Node.js – це JavaScript–оточення побудоване на JavaScript–рушієві Chrome V8, платформа з відкритим кодом для високопродуктивних серверів, написаних на JavaScript. Якщо раніше Javascript використовувався для обробки даних виключно у браузері і не дозволяв працювати з файловою системою, то з появою новї платформи, з’явилась можливість виконувати серверні сценарії. [13].

У Node є певний перелік властивостей, які стануть у нагоді при розробленні серверної частини, таких, як:

* асинхронність;
* модульна система;
* неблокуючий алгоритм вводу та виводу.

Менеджер пакетів NPM (Node package manager) використовується для управління модулями.

Було обрано MongoDB у якості бази даних у зв’язці з бібліотекою Mongoose ODM, для більш зручної взаємодії з API. У якості GUI, обрано MongoDB Compass. А для перевірки запитів – платформа Postman.

3.2 Розробка бази даних

Після проектування БД ми маємо таку структуру (Рисунок 3.1).

Graphical user interface, application

Description automatically generated

Рисунок 3.1 – Структура створеної бази даних

Створено кластер БД та розміщено його на віддаленому сервері для подальшого використання (Рисунок 3.2).

Table

Description automatically generated

Рисунок 3.2 – База даних у GUI MongoDB Compass

3.3 Розробка користувацького інтерфейсу

При розробці функціоналу було використано спільну мову програмування, як для клієнта, так і для сервервера – JavaScript. Проект поділяється на два підпроекта, перший є клієнтською частиною, реалізований із допомогою архітектури Model-View-ViewModal, другий – серверний, із допомогою Model-View-Controller архітектури. Далі наведено опис основних модулів двох частин онлайн-сервіса для закладів громадського харчування (Таблиця 3.1).

Таблиця 3.1 – Файли та їхній опис серверної частини

|  |  |
| --- | --- |
| **Шлях до файлу** | **Короткий опис** |
| 1 | 2 |
| /index.js | Файл запуску сервера |

Продовження таблиці 3.1

|  |  |
| --- | --- |
| 1 | 2 |
| /db.js | Файл підключення БД |
| /models/auth/User.js | Модель типу даних «Користувач» |
| /models/restaurant/Restaurant.js | Модель типу даних «Ресторан» |
| /models/restaurant/Category.js | Модель типу даних «Категорія» |
| /models/restaurant/Position.js | Модель типу даних «Позиція» |
| /models/restaurant/Order.js | Модель типу даних «Замовлення» |
| /controllers/auth/index.js | Контролер для моделі типу даних «Користувач» |
| /controllers/restaurant/index.js | Контролер для моделі типу даних «Ресторан» |
| /controllers/restaurant/  categories.js | Контролер для моделі типу даних «Категорія» |
| /controllers/restaurant/positions.js | Контролер для моделі типу даних «Позиція» |
| /controllers/restaurant/orders.js | Контролер для моделі типу даних «Замовлення» |
| /middleware/passport.js | Проміжний обробний логічний модуль, що перевіряє чи авторизований користувач, перед тим як віддати обробку запиту далі на контролер |
| /middleware/upload.js | Проміжний обробний логічний модуль, що перевіряє відправленний файл на відповідність, перед тим як віддати обробку запиту далі на контролер |
| /routes/index.js | Маршрутизатор, що з’єднює в собі інші маршрутизатори |
| /routes/auth/index.js | Маршрутизатор, що відповідає за авторизацію |

Продовження таблиці 3.1

|  |  |
| --- | --- |
| 1 | 2 |
| /routes/restaurants/index.js | Маршрутизатор, що відповідає за модуль ресторанів |
| /routes/restaurants/categories.js | Маршрутизатор, що відповідає за модуль категорій |
| /routes/restaurants/positions.js | Маршрутизатор, що відповідає за модуль позицій |
| /routes/restaurants/orders.js | Маршрутизатор, що відповідає за модуль замовлень |
| /utils/error-handler.js | Утиліта, що допомагає стежити за помилками |
| /utils/images.js | Утиліта, для роботи з картинками, що приходять від користувачів |
| /utils/send-email.js | Утиліта, для роботи з відправкою електроних листів |
| /utils/user.js | Список констант для роботи з моделю «Користувач» |
| /package.json | Файл конфігурації для NPM, що керує версіями сторонніх модулів та бібліотек у проєкті |
| /templates/auth/password.pug | Шаблон листа для електроної пошти по зміні паролю |
| /templates/auth/verification.pug | Шаблон листа для електроної пошти по активації аккаунта |

Розробка проєкту за такою серверною архітектурою робить проект гнучким, дає змогу розділити модулі окремо за областю відповідальності, також робить проєкт простішим для підтримки. Весь код модулів серверної частини представлений у додатку А.

Таблиця 3.2 – Файли та їхній опис клієнтської частини

|  |  |
| --- | --- |
| **Шлях до файлу** | **Короткий опис** |
| 1 | 2 |
| /nuxt.congif.js | Файл конфігурації Nuxt.js |
| /package.json | Файл конфігурації для NPM, що керує версіями сторонніх модулів та бібліотек у проєкті |
| /tailwind.config.js | Файл конфігурації бібліотеки TailwindCSS |
| /config/index.js | Файл конфігувації |
| /layouts/auth.vue | Шаблон для сторінок авторизації |
| /layouts/cabinet.vue | Шаблон для сторінок кабінету |
| /layouts/default.vue | Шаблон стандартний |
| /middleware/auth.js | Проміжний обробний логічний модуль, що перевіряє доступність до сторінок авторизації |
| /middleware/cabinet.js | Проміжний обробний логічний модуль, що перевіряє доступність до сторінок кабінету |
| /pages/index.vue | Сторінка «Головна» |
| /pages/auth/index.vue | Сторінка «Вхід/Реєстрація» |
| /pages/auth/password-forgot.vue | Сторінка «Забули пароль» |
| /pages/auth/email-verification/\_hash.vue | Сторінка «Підтвердження пошти» |
| /pages/auth/password-reset/\_hash.vue | Сторінка «Зміна пароля» |
| /pages/restaurants/\_id.vue | Сторінка «Перегляд ресторану» |
| /pages/cabinet/user/account.vue | Сторінка «Аккаунт користувача» |
| /pages/cabinet/user/orders.vue | Сторінка «Замовлення користувача» |
| /pages/cabinet/restaurant/orders  /restaurants/\_restaurantId.vue | Сторінка «Замовлення ресторана» |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| 1 | 2 |
| /pages/cabinet/restaurant/orders  /restaurants/index.vue | Сторінка «Список ресторанів користувача» |
| /pages/cabinet/restaurant/  restaurants/\_restaurantId/  categories/index.vue | Сторінка «Список категорій ресторана користувача» |
| /pages/cabinet/restaurant/  restaurants/\_restaurantId/  categories/\_categoryId/  positions.vue | Сторінка «Список позицій категорії ресторана користувача» |
| /plugins/axios.js | Плагін конфігурації запитів на сервер |
| /plugins/inject-nuxt.js | Плагін для плагінів Nuxt.js |
| /plugins/notify.client.js | Плагін вспливаючих повідомлень |
| /plugins/screen-height.client.js | Плагін для коректного відображення висоти сторінки |
| /store/index.js | Модуль глобального хранилища |
| /store/auth.js | Модуль хранилища авторизації |
| /store/restaurants.js | Модуль хранилища ресторанів |
| /store/categories.js | Модуль хранилища категорій |
| /store/positions.js | Модуль хранилища позицій |
| /store/basket.js | Модуль хранилища кошика |
| /store/orders.js | Модуль хранилища замовлень |
| /utils/auth.js | Утиліта для логіки авторизації |
| /utils/cookies.js | Утиліта для кукі |
| /utils/nuxt-integrations.js | Утиліта для Nuxt.js |
| /utils/api/index.js | Глобальний модуль API |
| /utils/api/auth.api.js | Модуль API для авторизації |
| /utils/api/restaurants.api.js | Модуль API для ресторанів |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| 1 | 2 |
| /utils/api/categories.api.js | Модуль API для категорій |
| /utils/api/positions.api.js | Модуль API для позицій |
| /utils/api/orders.api.js | Модуль API для замовлень |
| /components/TheHeader.vue | Компонент «Хедер» |
| /components/TheLogo.vue | Компонент «Логотип» |
| /components/Auth/  AuthEmailVerificationForm.vue | Компонент «Форма підтвердження пошти» |
| /components/Auth/AuthForm.vue | Компонент «Вхід/реєстрація» |
| /components/Auth/  AuthFormLogin.vue | Компонент «Форма входу» |
| /components/Auth/  AuthFormRegistration.vue | Компонент «Форма реєстрації» |
| /components/Auth/  AuthPasswordForgotForm.vue | Компонент «Забули пароль» |
| /components/Auth/  AuthPasswordResetForm.vue | Компонент «Зміна паролю» |
| /components/Base/  BaseButton.vue | Компонент «Кнопка» |
| /components/Base/  BaseLoading.vue | Компонент «Очікування» |
| /components/Base/Card/Card.vue | Компонент «Картка» |
| /components/Base/Card/  CardCabinet.vue | Компонент «Картка кабінету» |
| /components/Base/Form/  FormAmount.vue | Компонент «Поле вводу кількості» |
| /components/Base/Form/  FormCheckbox.vue | Компонент «Чекбокс» |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| 1 | 2 |
| /components/Base/Form/  FormInput.vue | Компонент «Поле вводу» |
| /components/Base/Form/  FormSettlementSelect.vue | Компонент «Вибір міста» |
| /components/Base/Form/  FormTextarea.vue | Компонент «Поле вводу текста» |
| /components/Base/Form/  FormUpload.vue | Компонент «Поле завантаження» |
| /components/Base/Modal/  Modal.vue | Компонент «Модальне вікно» |
| /components/Basket/  BasketFormModal.vue | Компонент «Кошик» |
| /components/Basket/  BasketPosition.vue | Компонент «Позиція в кошику» |
| /components/Cabinet/  CabinetBreadcrumb.vue | Компонент «Навігація кабінету» |
| /components/Cabinet/  CabinetContent.vue | Компонент «Зміст кабінету» |
| /components/Cabinet/  CabinetHeader.vue | Компонент «Заголовок кабінету» |
| /components/Cabinet/  CabinetSidebar.vue | Компонент «Бокове меню кабінету» |
| /components/Categories/  Categories.vue | Компонент «Категорії» |
| /components/Categories/  CategoriesCabinetCreation  Form.vue | Компонент «Модальне вікно створення категорії» |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| 1 | 2 |
| /components/Categories/  CategoriesCabinetCard.vue | Компонент «Карта категорії в кабінеті» |
| /components/Order/  OrderCard.vue | Компонент «Картка замовлення» |
| /components/Order/  OrderCreationFormModal.vue | Компонент «Модальне вікно створення замовлення» |
| /components/Order/  OrderReviewModal.vue | Компонент «Модальне вікно перегляду замовлення» |
| /components/Order/  OrderHistoryList.vue | Компонент «Список актуальних замовлень» |
| /components/Order/  OrderRelevantList.vue | Компонент «Список історії замовлень» |
| /components/Positions/  PositionsCard.vue | Компонент «Картка позиції» |
| /components/Positions/  PositionsCabinetCard.vue | Компонент «Картка позиції для кабінету» |
| /components/Positions/  PositionsCabinetCreation  FormModal.vue | Компонент «Модальне вікно створення позиції» |
| /components/Restaurants/  RestaurantsCabinetCard.vue | Компонент «Картка ресторана для кабінету» |
| /components/Restaurants/  RestaurantsCard.vue | Компонент «Картка кабінету» |
| /components/Restaurants/  RestaurantSearch.vue | Компонент «Поле пошуку по ресторанам» |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| 1 | 2 |
| /components/Restaurants/  RestaurantsCabinetCreation  Form.vue | Компонент «Модальне вікно створення ресторана» |

Розробка проєкту за такою клієнтською архітектурою робить проект гнучким, дає змогу розділити візуальні компоненти, сторінки і логіку окремо за областю відповідальності, також робить проєкт простішим для підтримки. Код основних файлів клієнтської частини представлений у додатку В.

РОЗДІЛ 4  
ТЕСТУВАННЯ ОНАЙН-СЕРВІСУ ДЛЯ ЗАКЛАДІВ ГРМАДСЬКОГО ХАРЧУВАННЯ

4.1 Вибір методів тестування

Тестування програмного забезпечення є спосібом перевірити, чи відповідає фактичний програмний продукт очікуваним вимогам, і забезпечити відсутність дефектів програмного продукту.

Метою тестування програмного забезпечення є виявлення помилок, прогалин або різницю між заявленими вимогами і фактичними. Для тестування програмного продукта ми будемо використовувати ручне тестування, застосовуючи як негативний, так і позитивний підхіди.

Тестування онлайн-сервіс продаж для закладів громадського харчування буде здійснюватися такими обраними методами та видами:

* функціональне тестування;
* тестування безпеки сайту;
* тестування навантаження;
* ручне тестування;

4.2 Тест план

Для проведення тестування визначемо функціонал, що буде тестуватися:

* вхід;
* реєстрація;
* зміна паролю;
* активації пошти;
* створення замовлення клієнтом;
* редагування електронного меню рестораном;

Розробимо тест-кейси, для перевірки коректної роботи сайта (Таблиця 4.1).

Таблиця 4.1 – Тест-кейси для онлайн-сервісу громадського харчування

|  |  |  |  |
| --- | --- | --- | --- |
| Опис | | Перевірка реєстрації | |
| Передумови | | Відкрита сторінка авторизації на сайті у браузері | |
| № | Дія | | Очікуваний результат |
| 1 | Придумати корректну пошту та пароль для реєстрації | |  |
| 2 | Натиснути на кнопку «Зареєструватися» | | Користувача буде успішно зареєстровано |
| Опис | | Перевірка входу | |
| Передумови | | Відкрита сторінка авторизації на сайті у браузері | |
| № | Дія | | Очікуваний результат |
| 1 | Ввести корректні пошту та пароль для входу | |  |
| 2 | Натиснути на кнопку «Увійти» | | Користувач увійле в аккаунт |
| Опис | | Перевірка зміни паролю | |
| Передумови | | Користувач вийшов із аккаунту | |
| № | Дія | | Очікуваний результат |
| 1 | Перейти на сторінку авторизації на вкладку «Вхід» | |  |
| 2 | Натиснути на кнопку «Забули пароль?» | | З’явиться сторінка з інструкцією для зміни паролю |
| 3 | Ввести пошту до аккаунта для зміни паролю | |  |
| 4 | Натиснути на кнопку «Продовжити» | | Відправиться лист на введену пошту для зміни паролю |
| 5 | Відкрити лист, що прийшов на вказану пошту | |  |

Продовження таблиці 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дія | | Очікуваний результат |
| 6 | Натиснути на посилання «Змінити паролю» | | Відкриється сайт зі сторінкою для введеня нового паролю |
| 7 | Ввести новий пароль і підтвердити його | |  |
| 8 | Натиснути на кнопку «Змінити пароль» | | Пароль буде змінено |
| 9 | Натиснути на кнопку «На головну» | | Відкриється головка сторінка |
| Опис | | Перевірка активації пошти | |
| Передумови | | Користувач уже зареєструваний і ввійшов у аккаунт | |
| № | Дія | | Очікуваний результат |
| 1 | Відкрити лист, що прийшов на вказану, при реєстрації, пошту | |  |
| 2 | Натиснути на кнопку «Активувати аккаунт» | | Відкриється сайт зі сторінкою про активацію пошти |
| Опис | | Перевірка створення замовлення | |
| Передумови | | Клієнт відкрив сайт у браузері | |
| № | Дія | | Очікуваний результат |
| 1 | Натиснути на ресторан зі списку | | Відкриється сторінка ресторану |
| 2 | Додати позиції у кошик, натискаючи на кнопку «До кошику» | | Позиції з’являться в кошику |
| 3 | Натиснути на «Кошик (N грн)» у правій верхній частині екрана | | З’явиться модальне вікно кошика з позиціями |

Продовження таблиці 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дія | | Очікуваний результат |
| 4 | Натиснути на кнопку «Замовити» | | З’явиться модальне вікно продовження створення |
| 5 | Заповнити інформацію для доставки | |  |
| 6 | Натиснути на кнопку «Підтвердити» | | Повідомлення про те, що з клієнтом зв’яжуться |
| Опис | | Редагування електронного меню рестораном | |
| Передумови | | Користувач авторизований у профілі рестрана і зайшов у кабінет на вкладку «Меню» | |
| № | Дія | | Очікуваний результат |
| 1 | Натиснути на кнопку «Додати ресторан» | | Відкриється модальне вікно для додавання |
| 2 | Заповнити поля | |  |
| 3 | Натиснути на кнопку «Додати» | | Список оновиться і з’явиться нова карточка |
| 4 | Натиснути на новій карточці на кнопку «Редагувати» | | Відкриється модальне вікно для редагування |
| 5 | Редагувати поля | |  |
| 6 | Натиснути на кнопку «Зберегти» | | Список і карточка оновиться |
| 7 | Натиснути на кнопку видалення «Х» на будь-якій карточці | | З’явиться алерт і в разі підтвердження карточка видалиться |
| 8 | Повторити попередні дії з категоріями | | Успішне створення, редагування та видалення |
| 9 | Повторити попередні дії з позиціями | | Успішне створення, редагування та видалення |

4.3. Введення в експлуатацію

Для введення в експлуатацію знадобиться три віддалених сервера для:

* клієнта;
* сервера;
* БД.

Оскільки MongoDB вже розміщена на кластері у ввіддаленому сервері, то лишаєтся вирішити питання з клієнтською та серверною частиною. В пригоді знадобиться сервіс «Heroku»

Heroku – хмарна платформа на основі контейнерів. Розробники використовують Heroku для розгортання, управління та масштабування сучасних проєктів та програм. Платформа є гнучкою та проста у використанні, пропонуючи розробникам найпростіший шлях до запуску своїх програм в експлуатацію [14].

Платформа надає можливість просто та швидко розмістити і сервер, що працює на Node.js, і клієнтську частину на Nuxt.js. Для цього, потрібно додати проекти в окремі репозиторії GIT і підключити їх в Heroku.

Оскільки проект вже перенесено на GitHub, то буде простіше налаштувати автоматичне розгортання із сховища проекту. Для цього на інформаційній панелі Heroku натискаємо на назву програми, яка буде розгорнута з GitHub. Потім на сторінці деталей програми вгорі є панель вкладок із опцією «Розгортання». На цій сторінці можна вибрати і налаштувати метод розгортання для інтеграції з GitHub.

Після підключення репозиторія клієнтської частини потрібно прописати команду в терміналі платформи для запуску: npm run dev, а сервера – node index.js. Далі зачекати декілька хвилин, поки хостинг стане доступним і ним можна буде користуватися.

ВИСНОВОК

Мета дипломної роботи полягала в розробці онлайн-сервісу продаж для закладів громадського харчування, складалася з двох підпроєктів: клієнтського і серверного. Знадобився великий об’єм теоретичних та практичних знань і вмінь, що надав можливості створення представленого функціонала.

Розроблений сервіс надає цілу низку інструментів та переваг для зручної взаємодії клієнта та заклада громадського харчування. Для власника онлайн-сервісу продаж це дає можливість насамперед покращити якість обслуговування потенційного клієнта завдяки певних інструментів, а для клієнта – наявність зручний перегляд завжди актульного меню, рейтинг закладів та можливість оцінювання рівня обслуговування, пошук за ресторанами, ключовими словами та стравам тощо.

Серверна частина розроблена за допомогою спільною з клієнтською частиною мовою програмування JavaScript, що забезпечує легкість підтримання та масштабування проєкту за рахунок великої кількісті спеціалістів, що можуть працювати з нею.

Клієнтська частина онлайн-сервісу була розроблена на веб-платформі, яка дає можливість мати доступ клієнту майже з будь-якого пристрою, де є доступ до інтернету та браузера.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методологічні основи створення, впровадження і розвитку інтегрованої інформаційної системи управління університетом : монографія / за ред. канд. фіз.-мат. наук, д-ра біол. наук, проф. С. В. Чернишенка, канд. фіз.-мат. наук, доц. Ю. І. Воротницького. – Суми: Сумський державний університет, 2015. – 333 с.
2. Що таке агрегатор послуг і товарів? [Електронний ресурс] – Режим доступу: <https://timeweb.com/ru/community/articles/chto-takoe-agregator>
3. Вікіпедія. [Електронний ресурс] – Режим доступу: [https://uk.wikipedia.org/wiki/e-commerce](https://uk.wikipedia.org/wiki/%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0_%D0%BA%D0%BE%D0%BC%D0%B5%D1%80%D1%86%D1%96%D1%8F)
4. The History & Evolution Of Food Delivery [Електронний ресурс] – Режим доступу: [https://underscore.factor75.com/history-of-food-delivery](https://underscore.factor75.com/the-history-evolution-of-food-delivery/)
5. Nuxt – The Intuitive Vue Framework [Електронний ресурс] – Режим доступу: <https://nuxtjs.org/>
6. Uber Eats | Food Delivery and Takeout - Order Online [Електронний ресурс] – Режим доступу: <https://www.ubereats.com>
7. Glovo. Доставка їжі і не тільки. [Електронний ресурс] – Режим доступу: [https://about.glovoapp.com/en](https://about.glovoapp.com/en/)
8. ROCKET — Доставка їжі з ресторанів та магазинів. [Електронний ресурс] – Режим доступу: <https://raketaapp.com/index.html>
9. Delivery Club. Пресс-центр. [Електронний ресурс] – Режим доступу: [https://press.delivery-club.ru](https://press.delivery-club.ru/)
10. Delivery.com: Order Food, Alcohol, Laundry & Grocery. [Електронний ресурс] – Режим доступу: [https://www.delivery.com](https://www.delivery.com/)
11. Vue.js. [Електронний ресурс] – Режим доступу: <https://vuejs.org/v2/guide/>
12. Nuxt | Directory Structure. [Електронний ресурс] – Режим доступу: <https://nuxtjs.org/docs/2.x/get-started/directory-structure>
13. Node.js Documentation. [Електронний ресурс] – Режим доступу: <https://nodejs.org/uk/docs/>
14. Heroku Dev Center. [Електронний ресурс] – Режим доступу: <https://devcenter.heroku.com/articles/git>

ДОДАТОК А  
ВИХІДНИЙ КОД ОСНОВНИХ ФАЙЛІВ СЕРВЕРА

1. Файл запуску сервера (/index.js):

import express from 'express'

import http from 'http'

import cors from 'cors'

import bodyParser from 'body-parser'

import passport from 'passport'

import '@/db'

import { PORT } from '@/config/keys'

import routes from '@/routes'

const app = express()

const server = http.createServer(app)

app.use('/uploads', express.static('uploads'))

app.use(passport.initialize())

app.use(bodyParser.urlencoded({ extended: true }))

app.use(bodyParser.json())

app.use(

cors({

methods: ['GET', 'POST', 'PATCH', 'DELETE'],

allowedHeaders: ['Content-Type', 'Authorization', 'clienturl']

})

)

require('@/middleware/passport')(passport)

routes(app)

server.listen(PORT, () => {

console.log(`Server has been started on port: ${PORT}`)

})

2. Файл підключення БД (/db.js):

import mongoose from 'mongoose'

import keys from '@/config/keys'

mongoose

.connect(keys.MONGO\_URI, {

useCreateIndex: true,

useNewUrlParser: true,

useUnifiedTopology: true,

useFindAndModify: false

})

.then(() => console.log('MongoDB connected'))

.catch(e => console.error(e))

3. Модель типу даних «Користувач» (/models/auth/User.js):

import { Schema, model } from 'mongoose'

import { ROLES } from '@/utils/user'

const schema = new Schema({

email: {

type: String,

unique: true,

lowercase: true,

required: true

},

password: {

type: String

},

role: {

type: String,

enum: Object.values(ROLES),

default: 'user'

},

info: {

name: { type: String, default: '' },

phone: { type: String, default: '' },

city: { type: String, default: '' },

street: { type: String, default: '' },

entranceNumber: { type: String, default: '' },

floorNumber: { type: String, default: '' },

roomNumber: { type: String, default: '' },

description: { type: String, default: '' }

},

isActive: {

type: Boolean,

default: false

},

verificationHash: {

type: String,

default: null

},

passwordResetHash: {

type: String,

default: null

}

})

schema.methods.getPublicProfile = function () {

const user = this.toObject()

delete user.\_\_v

delete user.password

delete user.verificationHash

delete user.passwordResetHash

return user

}

module.exports = model('User', schema)

4. Модель типу даних «Ресторан» (/models/restaurant/Restaurant.js):

import { Schema, model } from 'mongoose'

import { defineImageLink, deleteOneImage } from '@/utils/images'

const schema = new Schema({

name: {

type: String,

required: true,

trim: true

},

description: {

type: String,

default: '',

trim: true

},

image: {

type: String,

default: '',

required: true,

trim: true

},

// rating: {

// type: Number,

// default: 0

// },

settlements: {

type: String,

required: true

},

keywords: {

type: String,

default: '',

trim: true

},

minDeliveryOrderPrice: {

type: Number,

default: 0

},

minDeliveryTimeInMinutes: {

type: Number,

default: 0

},

ownerId: {

type: Schema.Types.ObjectId,

ref: 'User',

required: true

},

categories: [

{

type: Schema.Types.ObjectId,

ref: 'Category'

}

]

})

schema.methods.toJSON = function () {

const restaurant = this.toObject()

restaurant.image = defineImageLink(restaurant.image)

return restaurant

}

schema.pre('deleteOne', { document: true }, async function (next) {

await deleteOneImage(this.image)

const Category = model('Category')

await Category.deleteMany({ \_id: { $in: this.categories } })

next()

})

module.exports = model('Restaurant', schema)

5. Модель типу даних «Категорія» (/models/restaurant/Category.js):

import { Schema, model } from 'mongoose'

const schema = new Schema({

name: {

type: String,

required: true

},

restaurantId: {

type: Schema.Types.ObjectId,

ref: 'Restaurant',

required: true

},

positions: [

{

type: Schema.Types.ObjectId,

ref: 'Position'

}

]

})

schema.pre('save', async function (next) {

const Restaurant = model('Restaurant')

await Restaurant.findByIdAndUpdate(this.restaurantId, {

$push: { categories: { $each: [this.\_id], $position: 0 } }

})

next()

})

schema.pre('deleteOne', { document: true }, async function (next) {

const Position = model('Position')

await Position.deleteMany({ \_id: { $in: this.positions } })

const Restaurant = model('Restaurant')

const restaurant = await Restaurant.findById(this.restaurantId)

const { categories } = restaurant

categories.splice(categories.indexOf(this.\_id), 1)

await restaurant.save()

next()

})

schema.pre('deleteMany', async function (next) {

const Category = model('Category')

const categoriesToDelete = await Category.find(this.\_conditions)

const positionsToDelete = categoriesToDelete.flatMap(

category => category.positions

)

const Position = model('Position')

await Position.deleteMany({ \_id: { $in: positionsToDelete } })

next()

})

module.exports = model('Category', schema)

6. Модель типу даних «Позиція» (/models/restaurant/Position.js):

import { Schema, model } from 'mongoose'

import { defineImageLink, deleteManyImages } from '@/utils/images'

const schema = new Schema({

name: {

type: String,

required: true,

trim: true

},

image: {

type: String,

default: '',

trim: true

},

description: {

type: String,

default: '',

trim: true

},

price: {

type: Number,

required: true

},

weight: {

type: Number,

required: true

},

restaurantId: {

type: Schema.Types.ObjectId,

ref: 'Restaurant',

required: true

},

categoryId: {

type: Schema.Types.ObjectId,

ref: 'Category',

required: true

}

})

const toObject = function () {

const position = this.\_doc

position.image = defineImageLink(position.image)

return position

}

schema.method('toObject', toObject, { suppressWarning: true })

schema.pre('save', async function (next) {

const Category = model('Category')

await Category.findByIdAndUpdate(this.categoryId, {

$push: { positions: { $each: [this.\_id], $position: 0 } }

})

next()

})

schema.pre('deleteOne', { document: true }, async function (next) {

const Category = model('Category')

const category = await Category.findById(this.categoryId)

const { positions } = category

positions.splice(positions.indexOf(this.\_id), 1)

await category.save()

next()

})

schema.pre('deleteMany', async function (next) {

const Position = model('Position')

const positionsToDelete = await Position.find(this.\_conditions)

const imagesToDelete = positionsToDelete.map(position => position.image)

await deleteManyImages(imagesToDelete)

next()

})

module.exports = model('Position', schema)

7. Модель типу даних «Замовлення» (/models/restaurant/Order.js):

import { Schema, model } from 'mongoose'

const schema = new Schema(

{

number: {

type: Number,

required: true

},

list: [

{

name: { type: String, required: true },

amount: { type: Number, required: true },

price: { type: Number, required: true }

}

],

isDelivered: {

type: Boolean,

default: false

},

info: {

name: { type: String, required: true },

phone: { type: String, required: true },

city: { type: String, required: true },

street: { type: String, required: true },

entranceNumber: { type: String, default: '' },

floorNumber: { type: String, default: '' },

roomNumber: { type: String, default: '' },

personAmount: { type: Number, default: '' },

description: { type: String, default: '' }

},

userId: {

type: String,

default: ''

},

restaurantId: {

type: Schema.Types.ObjectId,

ref: 'Restaurant',

required: true

}

},

{ timestamps: true }

)

module.exports = model('Order', schema)

8. Контролер для моделі типу даних «Користувач» (/controllers/auth/index.js):

import bcrypt from 'bcryptjs'

import jwt from 'jsonwebtoken'

import pug from 'pug'

import User from '@/models/auth/User'

import keys from '@/config/keys'

import errorHandler from '@/utils/error-handler'

import sendEmail from '@/utils/send-email'

const generateHash = () =>

Math.random().toString(36).substring(2, 20) +

Math.random().toString(36).substring(2, 20)

const companyName = '«Україна Їсть»'

const emailOptionsForEmailVerification = ({ email }, link) => ({

subject: `Активація аккаунта ${companyName}`,

html: pug.compileFile('templates/auth/verification.pug')({ email, link })

})

const emailOptionsForPasswordReset = ({ email }, link) => ({

subject: `Зміна пароля ${companyName}`,

html: pug.compileFile('templates/auth/passwordReset.pug')({ email, link })

})

const createVerifyEmail = async (user, clienturl) => {

const hash = generateHash()

user.verificationHash = hash

await user.save()

const link = `${clienturl}/auth/email-verification/${hash}`

const emailOptions = emailOptionsForEmailVerification(user, link)

await sendEmail(emailOptions)

}

// Basic

module.exports.login = async (req, res) => {

const { email, password } = req.body

const candidate = await User.findOne({ email })

if (!candidate) res.json({ isSuccess: false })

const compareResult = bcrypt.compareSync(password, candidate.password)

if (!compareResult) res.json({ isSuccess: false })

const expiresIn = 604800 // one week

const token = jwt.sign({ userId: candidate.\_id }, keys.JWT, { expiresIn })

res.json({ isSuccess: true, token: `Bearer ${token}` })

}

module.exports.registration = async (req, res) => {

const { email, password, role, restaurantName } = req.body

const { clienturl } = req.headers

const candidate = await User.findOne({ email })

if (candidate) {

return res.json({ isSuccess: false, message: 'Such user already exists' })

}

const salt = bcrypt.genSaltSync(10)

const isRestaurant = role === 'restaurant'

const payload = {

email,

password: bcrypt.hashSync(password, salt),

...(isRestaurant && { role, restaurantName })

}

const user = new User(payload)

try {

await user.save()

await createVerifyEmail(user, clienturl)

res.json({ isSuccess: true, user: user.getPublicProfile() })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.user = async (req, res) => {

const { id } = req.user

try {

const user = await User.findById(id)

res.json({ isSuccess: true, user: user.getPublicProfile() })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.updateInfo = async (req, res) => {

const { id } = req.user

const toUpdate = req.body

try {

await User.findByIdAndUpdate(id, { info: toUpdate })

const user = await User.findById(id)

res.json({ isSuccess: true, user: user.getPublicProfile() })

} catch (e) {

errorHandler(res, e)

}

}

// Verification

module.exports.sendVerificationEmail = async (req, res) => {

const { id } = req.user

const { clienturl } = req.headers

try {

const user = await User.findById(id)

const { isActive, verificationHash } = user

if (isActive)

res.json({ isSuccess: false, message: 'Аккаунт вже підтверджено' })

else {

const link = `${clienturl}/auth/email-verification/${verificationHash}`

const mailOptions = emailOptionsForEmailVerification(user, link)

await sendEmail(mailOptions)

res.json({ isSuccess: true })

}

} catch (e) {

errorHandler(res, e)

}

}

module.exports.verifyEmail = async (req, res) => {

const { hash } = req.query

try {

await User.findOneAndUpdate(

{ verificationHash: hash },

{

$set: { isActive: true, verificationHash: null }

}

)

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

// Password reset

module.exports.sendPasswordResetEmail = async (req, res) => {

const { clienturl } = req.headers

const { email } = req.body

try {

const user = await User.findOne({ email: email.toLowerCase() })

if (!user) {

return res.json({ isSuccess: false, message: 'No such user' })

}

const hash = user.passwordResetHash || generateHash()

user.passwordResetHash = hash

await user.save()

const link = `${clienturl}/auth/password-reset/${hash}`

const mailOptions = emailOptionsForPasswordReset(user, link)

await sendEmail(mailOptions)

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.resetPassword = async (req, res) => {

const { hash, password } = req.body

const salt = bcrypt.genSaltSync(10)

try {

await User.findOneAndUpdate(

{ passwordResetHash: hash },

{

$set: {

password: bcrypt.hashSync(password, salt),

passwordResetHash: null

}

}

)

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

9. Контролер для моделі типу даних «Ресторан» (/controllers/restaurant/index.js):

import Restaurant from '@/models/restaurant/Restaurant'

import errorHandler from '@/utils/error-handler'

import { saveImage } from '@/utils/images'

const populateRestaurantOptions = Object.freeze({

path: 'categories',

populate: {

path: 'positions'

}

})

module.exports.getAll = async (req, res) => {

const { settlement, search = '' } = req.query

let filter = {}

try {

filter = {

$or: [

{ name: { $regex: search, $options: 'i' } },

{ keywords: { $regex: search, $options: 'i' } },

{ description: { $regex: search, $options: 'i' } }

],

settlements: { $regex: settlement, $options: 'i' }

}

const restaurants = await Restaurant.find(filter).populate(

populateRestaurantOptions

)

res.json({ isSuccess: true, restaurants })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.getById = async (req, res) => {

const { id } = req.params

try {

const restaurant = await Restaurant.findById(id).populate(

populateRestaurantOptions

)

res.json({ isSuccess: true, restaurant })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.getByOwnerId = async (req, res) => {

const { ownerId } = req.params

try {

const restaurants = await Restaurant.find({ ownerId }).populate(

populateRestaurantOptions

)

res.json({ isSuccess: true, restaurants })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.create = async (req, res) => {

const { \_id: ownerId, email: ownerEmail } = req.user

const { name, description, settlements, keywords } = req.body

const { minDeliveryOrderPrice, minDeliveryTimeInMinutes } = req.body

const image = await saveImage(req.file)

const restaurant = new Restaurant({

name,

description,

image,

settlements,

keywords,

minDeliveryOrderPrice,

minDeliveryTimeInMinutes,

ownerId

})

try {

await restaurant.save()

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.delete = async (req, res) => {

const { id } = req.params

try {

const restaurant = await Restaurant.findById(id)

await restaurant.deleteOne()

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

10. Контролер для моделі типу даних «Категорія» (/controllers/restaurant/categories.js):

import Category from '@/models/restaurant/Category'

import errorHandler from '@/utils/error-handler'

module.exports.create = async (req, res) => {

const { restaurantId } = req.params

const { name } = req.body

const category = new Category({ name, restaurantId })

try {

await category.save()

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.delete = async (req, res) => {

const { id } = req.params

try {

const category = await Category.findById(id)

await category.deleteOne()

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

11. Контролер для моделі типу даних «Позиція» (/controllers/restaurant/positions.js):

import Category from '@/models/restaurant/Category'

import Position from '@/models/restaurant/Position'

import errorHandler from '@/utils/error-handler'

import { saveImage } from '@/utils/images'

module.exports.create = async (req, res) => {

const { categoryId } = req.params

const { name, description, price, weight } = req.body

const category = await Category.findById(categoryId)

const restaurantId = category.restaurantId

const image = await saveImage(req.file)

const position = new Position({

name,

image,

description,

price,

weight,

restaurantId,

categoryId

})

try {

await position.save()

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.delete = async (req, res) => {

const { id } = req.params

try {

const position = await Position.findById(id)

await position.deleteOne()

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

12. Контролер для моделі типу даних «Замовлення» (/controllers/restaurant/orders.js):

import Order from '@/models/restaurant/Order'

import errorHandler from '@/utils/error-handler'

module.exports.getAll = async (req, res) => {

const { restaurantId, userId } = req.query

const filter = { restaurantId, userId }

Object.keys(filter).forEach(key => {

if (!filter[key]) delete filter[key]

})

try {

const orders = await Order.find(filter)

res.json({ isSuccess: true, orders })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.create = async (req, res) => {

const { restaurantId } = req.query

const { list, info, userId } = req.body

const orders = await Order.find()

const orderNumber = orders.length + 1

const order = new Order({

number: orderNumber,

list,

info,

userId,

restaurantId

})

try {

await order.save()

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

module.exports.update = async (req, res) => {

const { orderId } = req.params

const toUpdate = req.body

try {

const order = await Order.findById(orderId)

await Order.findByIdAndUpdate(orderId, toUpdate)

res.json({ isSuccess: true })

} catch (e) {

errorHandler(res, e)

}

}

13. Проміжний обробний логічний модуль, що перевіряє чи авторизований користувач, перед тим як віддати обробку запиту далі на контролер (/middleware/passport.js):

import passportJwt from 'passport-jwt'

import keys from '@/config/keys'

import User from '@/models/auth/User'

const JwtStrategy = passportJwt.Strategy

const ExtractJwt = passportJwt.ExtractJwt

const options = {

jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),

secretOrKey: keys.JWT

}

module.exports = passport => {

passport.use(

new JwtStrategy(options, async (payload, done) => {

try {

const user = await User.findById(payload.userId).select('id email')

if (user) done(null, user)

else done(null, false)

} catch (e) {

console.error(e.message)

}

})

)

}

14. Проміжний обробний логічний модуль, що перевіряє відправленний файл на відповідність, перед тим як віддати обробку запиту далі на контролер (/middleware/upload.js):

import multer from 'multer'

import dayjs from 'dayjs'

const storage = multer.diskStorage({

destination(req, file, cb) {

cb(null, 'uploads/')

},

filename(req, file, cb) {

const date = dayjs().format('YYYY-MM-DD-HH:mm:ss-SSS')

const filename = `${date}.${file.mimetype.split('/')[1]}`

cb(null, filename)

}

})

const fileFilter = (req, file, cb) => {

if (!file.originalname.match(/\.(jpg|jpeg|png|webp)$/)) {

cb(new Error('Invalid file format. Use jpg/jpeg/png/webp'))

return

}

cb(null, true)

}

// Size limit 5MB

const limits = { fileSize: 1024 \* 1024 \* 5 }

module.exports = multer({ storage, fileFilter, limits })

15. Маршрутизатор, що з’єднює в собі інші маршрутизатори (/routes/index.js):

import authRoutes from '@/routes/auth'

import restaurantsRoutes from '@/routes/restaurants'

import categoriesRoutes from '@/routes/restaurants/categories'

import positionsRoutes from '@/routes/restaurants/positions'

import ordersRoutes from '@/routes/restaurants/orders'

module.exports = function (app) {

app.use('/api/auth', authRoutes)

app.use('/api/restaurants', restaurantsRoutes)

app.use('/api/restaurants', categoriesRoutes)

app.use('/api/restaurants', positionsRoutes)

app.use('/api/orders', ordersRoutes)

}

16. Маршрутизатор, що відповідає за авторизацію (/routes/auth/index.js):

import Router from 'express/lib/router'

const passport = require('passport').authenticate('jwt', { session: false })

import controller from '@/controllers/auth'

const router = Router()

// Basic

router.post('/login', controller.login)

router.post('/registration', controller.registration)

router.get('/user', passport, controller.user)

router.patch('/user', passport, controller.updateInfo)

// Verification

router.get(

'/send-verification-email',

passport,

controller.sendVerificationEmail

)

router.get('/verify-email', controller.verifyEmail)

// Password reset

router.post('/send-password-reset-email', controller.sendPasswordResetEmail)

router.post('/reset-password', controller.resetPassword)

module.exports = router

17. Маршрутизатор, що відповідає за модуль ресторанів (/routes/restaurants/index.js):

import Router from 'express/lib/router'

const passport = require('passport').authenticate('jwt', { session: false })

const upload = require('@/middleware/upload').single('image')

import controller from '@/controllers/restaurants'

const router = Router()

router.get('/', controller.getAll)

router.get('/:id', controller.getById)

router.get('/owner/:ownerId', passport, controller.getByOwnerId)

router.post('/', passport, upload, controller.create)

router.delete('/:id', passport, controller.delete)

module.exports = router

18. Маршрутизатор, що відповідає за модуль категорій (/routes/restaurants/categories.js):

import Router from 'express/lib/router'

const passport = require('passport').authenticate('jwt', { session: false })

import controller from '@/controllers/restaurants/categories'

const router = Router()

const BASE\_ROUTE = '/:restaurantId/categories'

router.post(`${BASE\_ROUTE}/`, passport, controller.create)

router.delete(`/categories/:id`, passport, controller.delete)

module.exports = router

19. Маршрутизатор, що відповідає за модуль позицій (/routes/restaurants/positions.js):

import Router from 'express/lib/router'

const passport = require('passport').authenticate('jwt', { session: false })

const upload = require('@/middleware/upload').single('image')

import controller from '@/controllers/restaurants/positions'

const router = Router()

const BASE\_ROUTE = '/categories/:categoryId/positions'

router.post(`${BASE\_ROUTE}/`, passport, upload, controller.create)

router.delete(`/categories/positions/:id`, passport, controller.delete)

module.exports = router

20. Маршрутизатор, що відповідає за модуль замовлень (/routes/restaurants/orders.js):

import Router from 'express/lib/router'

const passport = require('passport').authenticate('jwt', { session: false })

import controller from '@/controllers/restaurants/orders'

const router = Router()

router.get('/', passport, controller.getAll)

router.post('/', passport, controller.create)

router.patch(`/:orderId`, passport, controller.update)

module.exports = router

21. Утиліта, що допомагає стежити за помилками (/utils/error-handler.js):

module.exports = (res, error) => {

console.trace()

console.log('❌ ERROR', error.message)

res.json({

isSuccess: false,

message: error.message ? error.message : error

})

}

22. Утиліта, для роботи з картинками, що приходять від користувачів (/utils/images.js):

import fs from 'fs'

import jimp from 'jimp'

const keys = require('@/config/keys')

const defineImagePath = image => {

const imageName = image.split('uploads')[1]

return `./uploads/${imageName}`

}

const defineImageLink = image => {

if (image.includes('http') || image.includes('localhost')) {

return image

}

if (process.env.NODE\_ENV === 'production') {

image = `//${keys.DOMEN}:${keys.PORT}${image}`

} else {

image = `//localhost:${keys.PORT}${image}`

}

return image

}

const saveImage = file => {

return new Promise((resolve, reject) => {

const { destination, filename } = file

const path = `./${destination}${filename}`

jimp

.read(path)

.then(image => {

const newFilename = filename.replace(/(.png|.jpg|.jpeg|.gif|.webp)/, '')

const newPath = `/${destination}${newFilename}.jpg`

image.cover(1024, 500).quality(60).write(`.${newPath}`)

try {

fs.unlink(path, () => resolve(newPath))

} catch (error) {

reject(error.message)

}

})

.catch(reject)

})

}

const deleteOneImage = image => {

return new Promise(resolve => {

try {

const imagePath = defineImagePath(image)

const isImageExist = fs.existsSync(imagePath)

if (isImageExist) {

fs.unlinkSync(imagePath)

resolve(true)

}

resolve(true)

} catch {

resolve(false)

}

})

}

const deleteManyImages = images => {

return new Promise((resolve, reject) => {

const promises = images.map(deleteOneImage)

return Promise.all(promises)

.then(result => {

const isSuccess = result.every(Boolean)

resolve(isSuccess)

})

.catch(reject)

})

}

export { defineImageLink, saveImage, deleteOneImage, deleteManyImages }

23. Утиліта, для роботи з відправкою електроних листів (/utils/send-email.js):

import nodemailer from 'nodemailer'

import keys from '@/config/keys'

module.exports = options => {

const transporter = nodemailer.createTransport({

service: 'gmail',

auth: {

user: 'email.tester.misha@gmail.com',

pass: keys.NODEMAILER\_PASS

}

})

const mailOptions = {

from: 'email.tester.misha@gmail.com',

to: 'ILikePoltava@gmail.com',

...options

}

transporter.sendMail(mailOptions, (error, info) => {

if (error) console.log(error)

else console.log('Email sent: ' + info.response)

})

}

24. Список констант для роботи з моделю «Користувач» (/utils/user.js):

export const ROLES = Object.freeze({

USER: 'user',

RESTAURANT: 'restaurant'

})

25. Файл конфігурації для NPM, що керує версіями сторонніх модулів та бібліотек у проєкті (/package.json):

{

"name": "qFood-server",

"version": "1.0.0",

"main": "dist/index.js",

"author": "myziura.work@gmail.com",

"license": "MIT",

"private": true,

"scripts": {

"dev": "nodemon --exec babel-node index.js",

"clean": "rm -rf dist",

"prod": "node ./dist/index.js"

},

"dependencies": {

"@babel/runtime": "^7.12.5",

"bcryptjs": "^2.4.3",

"body-parser": "^1.19.0",

"cors": "^2.8.5",

"dayjs": "^1.10.4",

"express": "^4.17.1",

"jimp": "^0.16.1",

"jsonwebtoken": "^8.5.1",

"lodash": "^4.17.20",

"mongoose": "^5.10.13",

"multer": "^1.4.2",

"nodemailer": "^6.5.0",

"passport": "^0.4.1",

"passport-jwt": "^4.0.0",

"pug": "^3.0.2"

},

"devDependencies": {

"@babel/cli": "^7.12.10",

"@babel/core": "^7.12.3",

"@babel/node": "^7.12.6",

"@babel/plugin-transform-runtime": "^7.12.10",

"@babel/preset-env": "^7.12.1",

"babel-plugin-root-import": "^6.6.0",

"morgan": "^1.10.0",

"nodemon": "^2.0.6",

"prettier": "2.1.2"

},

"engines": {

"node": "14.x"

}

}

26. Шаблон листа для електроної пошти по зміні паролю (/templates/auth/password.pug):

span Вітаємо, #{email} !

br

span Ви подали заявку на зміну пароля.

br

span Будь ласка натисніть на посилання, щоб змінити пароль:

br

<a href="#{link}">Змінити пароль</a>

27. Шаблон листа для електроної пошти по активації аккаунта (/templates/auth/verification.pug):

span Вітаємо, #{email} !

br

span Дякуємо за реєстрацію.

br

span Будь ласка, натисніть на посилання, щоб активувати свій аккаунт:

br

<a href="#{link}">Активувати аккаунт</a>

ДОДАТОК B  
ВИХІДНИЙ КОД ОСНОВНИХ ФАЙЛІВ КЛІЄНТА

1. Файл конфігурації Nuxt.js (/nuxt.congif.js):

export default {

head: {

title: 'Україна Їсть',

htmlAttrs: {

lang: 'uk'

},

meta: [

{ charset: 'utf-8' },

{

name: 'viewport',

content:

'width=device-width, initial-scale=1, maximum-scale=1.0, user-scalable=no'

},

{ hid: 'description', name: 'description', content: '' }

],

script: [

{

src: 'https://kit.fontawesome.com/faf1716791.js',

crossorigin: 'anonymous'

}

],

link: [

{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },

{

rel: 'stylesheet',

href:

'https://fonts.googleapis.com/css2?family=Montserrat:wght@400;600;700&display=swap'

}

]

},

plugins: [

'@/plugins/axios.js',

'@/plugins/inject-nuxt.js',

'@/plugins/directives.client.js',

'@/plugins/screen-height.client.js',

'@/plugins/notify.client.js'

],

components: true,

buildModules: [

'@nuxtjs/eslint-module',

'@nuxtjs/tailwindcss',

'@nuxtjs/style-resources'

],

modules: ['@nuxtjs/axios'],

css: ['@/assets/styles/global.scss'],

styleResources: {

scss: ['@/assets/styles/scss/transitions.scss']

},

build: {

extend(config, ctx) {

config.module.rules.push({

enforce: 'pre',

test: /\.(js|vue)$/,

loader: 'eslint-loader',

exclude: /(node\_modules)/,

options: {

fix: true

}

})

}

},

loading: {

name: 'circle',

color: '#39b981',

height: '0.5rem',

continuous: true

},

server: {

host: '0.0.0.0'

}

}

2. Файл конфігурації для NPM, що керує версіями сторонніх модулів та бібліотек у проєкті (/package.json):

{

"name": "board-game",

"version": "1.0.0",

"private": true,

"scripts": {

"dev": "nuxt",

"build": "nuxt build",

"start": "nuxt start",

"generate": "nuxt generate"

},

"dependencies": {

"@nuxtjs/axios": "^5.12.5",

"cookie": "^0.4.1",

"core-js": "^3.8.2",

"js-cookie": "^2.2.1",

"nuxt": "^2.15.4",

"vue-toast-notification": "^0.6.1"

},

"devDependencies": {

"@nuxtjs/eslint-config": "^5.0.0",

"@nuxtjs/eslint-module": "^3.0.2",

"@nuxtjs/style-resources": "^1.0.0",

"@nuxtjs/tailwindcss": "^4.0.3",

"babel-eslint": "^10.1.0",

"eslint": "^7.19.0",

"eslint-config-prettier": "^7.2.0",

"eslint-loader": "^4.0.2",

"eslint-plugin-nuxt": "^2.0.0",

"eslint-plugin-prettier": "^3.3.1",

"eslint-plugin-vue": "^7.5.0",

"node-sass": "^5.0.0",

"prettier": "^2.2.1",

"sass-loader": "^10.1.1"

}

}

3. Файл конфігурації бібліотеки TailwindCSS (/tailwind.config.js):

const COLORS = Object.freeze({

MASK: 'rgba(0, 0, 0, .2)',

MASK\_LIGHT: 'rgba(255, 255, 255, .8)'

})

module.exports = {

theme: {

backgroundColor: theme => ({

...theme('colors'),

mask: `${COLORS.MASK}; backdrop-filter: blur(5px);`,

'mask-light': `${COLORS.MASK\_LIGHT}; backdrop-filter: blur(5px);`

}),

extend: {

height: {

14: '3.5rem',

80: '20rem',

128: '32rem',

192: '48rem',

screen: '100vh; height: calc(var(--vh, 1vh) \* 100)'

},

width: {

14: '3.5rem'

},

minWidth: {

10: '2.5rem'

},

maxWidth: {

14: '3.5rem',

96: '24rem',

128: '32rem'

}

}

},

variants: {

margin: ['last'],

backgroundColor: ['hover', 'checked'],

borderColor: ['hover', 'checked', 'focus']

}

}

4. Файл конфігувації (/config/index.js):

export const API\_URL = 'http://192.168.0.115:5000'

5. Шаблон для сторінок авторизації (/layouts/auth.vue):

<template>

<div class="h-screen flex flex-col bg-gray-100">

<div class="w-full max-w-96 m-auto flex flex-col items-center">

<Logo class="mb-8" />

<div class="w-full flex flex-col rounded-md shadow-xl overflow-hidden">

<nuxt />

</div>

</div>

</div>

</template>

<script>

import Logo from '@/components/TheLogo'

export default {

components: { Logo },

middleware: 'auth'

}

</script>

6. Шаблон для сторінок кабінету (/layouts/cabinet.vue):

<template>

<div class="h-screen w-full pt-0 flex flex-col text-gray-700 bg-gray-100">

<Header />

<div class="h-full flex overflow-hidden">

<Sidebar class="h-full w-96" />

<Content class="h-full w-full">

<nuxt />

</Content>

</div>

</div>

</template>

<script>

import Header from '@/components/Cabinet/CabinetHeader'

import Sidebar from '@/components/Cabinet/CabinetSidebar'

import Content from '@/components/Cabinet/CabinetContent'

export default {

components: { Header, Sidebar, Content },

middleware: 'cabinet'

}

</script>

7. Шаблон стандартний (/layouts/default.vue):

<template>

<div class="w-full pt-0 flex flex-col text-gray-700 bg-gray-100">

<Header />

<div class="flex">

<nuxt />

</div>

</div>

</template>

<script>

import Header from '@/components/TheHeader'

export default {

components: { Header }

}

</script>

8. Проміжний обробний логічний модуль, що перевіряє доступність до сторінок авторизації (/middleware/auth.js):

export default function ({ store, redirect, route }) {

const isAuthenticated = store.getters['auth/isAuthenticated']

const allowedPages = ['auth-email-verification-hash']

const isForbiddenPage = !allowedPages.includes(route.name)

const isRedirect = isForbiddenPage && isAuthenticated

if (isRedirect) {

redirect('/')

}

}

9. Проміжний обробний логічний модуль, що перевіряє доступність до сторінок кабінету (/middleware/cabinet.js):

export default function ({ store, route, redirect }) {

const isAuthenticated = store.getters['auth/isAuthenticated']

const isRestaurant = store.getters['auth/isRestaurant']

const isUser = store.getters['auth/isUser']

const isAuthRequired = ['cabinet'].includes(route.name)

const isOnlyRestaurantRoutes = route.name.includes('cabinet-restaurant')

const isOnlyUserRoutes = route.name.includes('cabinet-user')

if (isAuthRequired && !isAuthenticated) {

redirect({ name: 'auth-login' })

}

if (isOnlyRestaurantRoutes) {

if (!isRestaurant) redirect('/')

const isRestaurants = Boolean(

store.state.restaurants.restaurants.list.length

)

if (!isRestaurants) {

redirect({ name: 'cabinet-restaurant-restaurants' })

}

}

if (isOnlyUserRoutes && !isUser) {

redirect('/')

}

}

10. Сторінка «Головна» (/pages/index.vue):

<template>

<div

class="w-full min-h-screen px-8 py-32 mx-auto flex flex-col"

style="max-width: 1280px"

>

<Search class="mt-16" />

<span class="mb-16 text-5xl font-semibold leading-none">Ресторани</span>

<div class="grid grid-cols-3 gap-8">

<Card

v-for="restaurant of restaurants.list"

:key="`restaurant-${restaurant.\_id}`"

class="col-span-1"

:restaurant="restaurant"

/>

</div>

</div>

</template>

<script>

import { mapState } from 'vuex'

import Search from '@/components/Restaurants/RestaurantsSearch'

import Card from '@/components/Restaurants/RestaurantsCard'

export default {

components: { Search, Card },

async fetch({ store }) {

await store.dispatch('restaurants/FETCH\_ALL')

},

computed: {

...mapState('restaurants', ['restaurants'])

}

}

</script>

11. Сторінка «Вхід/Реєстрація» (/pages/auth/index.vue):

<template>

<AuthForm />

</template>

<script>

import AuthForm from '@/components/Auth/AuthForm'

export default {

components: { AuthForm },

layout: 'auth'

}

</script>

12 Сторінка «Забули пароль» (/pages/auth/password-forgot.vue):

<template>

<AuthPasswordForgotForm />

</template>

<script>

import AuthPasswordForgotForm from '@/components/Auth/AuthPasswordForgotForm'

export default {

components: { AuthPasswordForgotForm },

layout: 'auth'

}

</script>

13. Сторінка «Підтвердження пошти» (/pages/auth/email-verification/\_hash.vue):

<template>

<AuthEmailVerificationForm />

</template>

<script>

import AuthEmailVerificationForm from '@/components/Auth/AuthEmailVerificationForm'

export default {

components: { AuthEmailVerificationForm },

layout: 'auth'

}

</script>

14. Сторінка «Зміна пароля» (/pages/auth/password-reset/\_hash.vue):

<template>

<AuthPasswordResetForm />

</template>

<script>

import AuthPasswordResetForm from '@/components/Auth/AuthPasswordResetForm'

export default {

components: { AuthPasswordResetForm },

layout: 'auth'

}

</script>

15. Сторінка «Перегляд ресторану» (/pages/restaurants/\_id.vue):

<template>

<div class="w-full flex flex-col">

<div

class="relative h-192 w-full bg-gray-400 bg-cover"

:style="`background-image: url(${selected.image})`"

>

<div

class="absolute left-0 bottom-0 h-192 w-full p-4 pt-20 flex flex-col justify-center items-center bg-gradient-to-t from-gray-800"

>

<span class="text-6xl font-semibold text-white">

{{ selected.name }}

</span>

<div class="my-8 w-full max-w-128 text-center">

<span class="text-white text-lg">{{ selected.description }}</span>

</div>

<div class="flex">

<div

class="mr-4 flex p-2 px-4 bg-gray-800 text-white font-semibold rounded-xl"

>

<span>{{ selected.minDeliveryTimeInMinutes }} хв</span>

</div>

<div

class="mr-4 flex p-2 px-4 bg-gray-800 text-white font-semibold rounded-xl"

>

<span>від {{ selected.minDeliveryOrderPrice }} грн</span>

</div>

<!-- <div

class="mr-4 flex p-2 px-4 bg-gray-800 text-white font-semibold rounded-xl"

>

<span>⭐️ {{ selected.rating }}</span>

</div> -->

</div>

</div>

</div>

<div class="p-8 bg-gray-100">

<Categories :restaurant="selected" />

</div>

</div>

</template>

<script>

import { mapState } from 'vuex'

import Categories from '@/components/Categories/Categories'

export default {

components: { Categories },

async fetch({ store, params }) {

await store.dispatch('restaurants/FETCH\_BY\_ID', params.id)

},

computed: {

...mapState('restaurants', ['restaurants']),

...mapState('basket', ['basket']),

selected() {

return this.restaurants?.selected || {}

}

}

}

</script>

16. Сторінка «Аккаунт користувача» (/pages/cabinet/user/account.vue):

<template>

<div class="h-full w-full flex flex-col">

<div class="h-full flex flex-col">

<div class="p-8 flex justify-center border-b">

<div class="flex justify-between" style="width: 768px">

<h2 class="pr-8 text-xl"></h2>

<div class="w-96 flex-col">

<div v-if="auth.user.isActive" class="flex">

<div class="flex items-center">

<div

class="h-6 w-6 mr-4 flex items-center justify-center bg-green-500 rounded-full"

>

<i class="fas fa-check text-white"></i>

</div>

<span>Аккаунт активовано</span>

</div>

</div>

<template v-else>

<div class="mb-4 flex items-center">

<div

class="h-6 w-6 mr-4 flex items-center justify-center bg-red-500 rounded-full"

>

<i class="fas fa-times text-white"></i>

</div>

<span>Аккаунт не активовано</span>

</div>

<div

class="underline cursor-pointer"

:class="{

'text-green-500': isVisibleSendEmailButton,

'text-gray-300 pointer-events-none': !isVisibleSendEmailButton

}"

@click="handleSendEmail"

>

<span>Відправити лист активації</span>

<span v-if="countdown.isVisible">

({{ countdown.duration / 1000 }})

</span>

</div>

</template>

</div>

</div>

</div>

<div class="p-8 flex justify-center border-b">

<div class="flex justify-between" style="width: 768px">

<h2 class="pr-8 text-xl">Контактна інформація</h2>

<div class="w-96 flex-col">

<Input

class="mb-4"

label="Ім'я"

placeholder="Тарас"

:is-disabled="auth.isLoading"

:value.sync="form.name"

/>

<Input

class="mb-4"

label="Телефон"

placeholder="0996239448"

:is-disabled="auth.isLoading"

:value.sync="form.phone"

/>

</div>

</div>

</div>

<div class="p-8 flex justify-center border-b">

<div class="flex justify-between" style="width: 768px">

<h2 class="pr-8 text-xl">Адреса доставки</h2>

<div class="w-96 flex-col">

<Input

class="mb-4"

label="Місто"

placeholder="Назва міста"

:is-disabled="auth.isLoading"

:value="form.city"

/>

<Input

class="mb-4"

label="Вулиця, номер"

placeholder="Назва вулиці, номер"

:is-disabled="auth.isLoading"

:value.sync="form.street"

/>

<div class="mb-4 flex justify-between items-end">

<Input

class="mr-2"

label="Під'їзд"

placeholder="1"

:is-disabled="auth.isLoading"

:value.sync="form.entranceNumber"

/>

<Input

class="mx-2"

label="Поверх"

placeholder="5"

:is-disabled="auth.isLoading"

:value.sync="form.floorNumber"

/>

<Input

class="ml-2"

label="Квартира"

placeholder="49"

:is-disabled="auth.isLoading"

:value.sync="form.roomNumber"

/>

</div>

</div>

</div>

</div>

<div class="p-8 flex justify-center">

<div class="flex justify-between" style="width: 768px">

<h2 class="pr-8 text-xl">Додаткова інформація</h2>

<div class="w-96 flex-col">

<Texarea

label="Додаткова інформація"

placeholder="Без цибулі, будь ласка"

:is-disabled="auth.isLoading"

:value.sync="form.description"

/>

</div>

</div>

</div>

</div>

<div

class="sticky bottom-0 px-8 py-4 flex justify-center bg-gray-100 border-t"

>

<div class="flex justify-end" style="width: 768px">

<Button

class="w-64"

text="Зберегти"

:is-loading="auth.isLoading"

@click="handleSubmit"

/>

</div>

</div>

</div>

</template>

<script>

import { mapState, mapActions } from 'vuex'

import Button from '@/components/Base/BaseButton'

import Input from '@/components/Base/Form/FormInput'

import Texarea from '@/components/Base/Form/FormTextarea'

export default {

components: { Button, Input, Texarea },

layout: 'cabinet',

data: () => ({

form: {

// name: 'Сергій',

// phone: '380996239448',

// city: 'Харків',

// street: 'Шевченка, 12',

// entranceNumber: '1',

// floorNumber: '6',

// roomNumber: '49',

// personAmount: 1,

// description: 'Алергія на цибулю. Код дверей – 1234'

},

isVisibleSendEmailButton: true,

countdown: {

isVisible: false,

duration: 30000,

interval: 1000,

step: 1000

}

}),

computed: {

...mapState('auth', ['auth'])

},

created() {

this.form = { ...this.auth.user.info }

},

methods: {

...mapActions('auth', {

updateUserInfo: 'UPDATE\_USER\_INFO',

sendVerificationEmail: 'SEND\_VERIFICATION\_EMAIL'

}),

async handleSendEmail() {

this.isVisibleSendEmailButton = false

await this.sendVerificationEmail()

this.startCountdown(() => {

this.isVisibleSendEmailButton = true

})

this.$toast.success('Лист надіслано на пошту')

},

async handleSubmit() {

await this.updateUserInfo(this.form)

this.$toast.success('Дані оновлено')

},

startCountdown(cb) {

this.countdown.isVisible = true

const countdown = setInterval(() => {

this.countdown.duration -= this.countdown.step

if (this.countdown.duration <= 0) {

clearInterval(countdown)

this.countdown.isVisible = false

cb()

}

}, this.countdown.interval)

}

}

}

</script>

17. Сторінка «Замовлення користувача» (/pages/cabinet/user/orders.vue):

<template>

<div class="w-full p-8 flex flex-col">

<div class="mb-8 flex text-2xl">

<span>Замовлення</span>

</div>

<div class="w-full flex justify-between">

<div class="w-full flex">

<div

class="w-64 p-4 flex items-center justify-center text-center rounded-t-md cursor-pointer"

:class="{

'bg-gray-50 shadow-inner': isHistoryTab,

'bg-white border border-b-0': !isHistoryTab

}"

@click="handleClickRelevantTab"

>

<span

class="font-semibold text-lg select-none"

:class="{

'text-gray-400': isHistoryTab,

'text-green-500': !isHistoryTab

}"

>

Актуальні ({{ orders.relevant.length }})

</span>

</div>

<div

class="w-64 p-4 flex items-center justify-center text-center rounded-t-md cursor-pointer"

:class="{

'bg-gray-50 shadow-inner': isRelevantTab,

'bg-white border border-b-0': !isRelevantTab

}"

@click="handleClickHistoryTab"

>

<span

class="font-semibold text-lg select-none"

:class="{

'text-gray-400': isRelevantTab,

'text-green-500': !isRelevantTab

}"

>

Історія ({{ orders.history.length }})

</span>

</div>

</div>

<div

class="h-full px-8 flex items-center cursor-pointer transform hover:text-green-500 hover:scale-110"

@click="handleRefresh"

>

<i class="fas fa-sync-alt text-xl"></i>

</div>

</div>

<div class="h-full w-full">

<OrdersRelevantList

v-if="isRelevantTab"

:list="orders.relevant"

@select="handleOpenReviewModal"

/>

<OrdersHistoryList

v-else

:list="orders.history"

@select="handleOpenReviewModal"

/>

</div>

<OrderReviewModal

v-if="order.isVisibleReviewModal"

:order="order.selected"

@close="handleCloseReviewModal"

/>

</div>

</template>

<script>

import { mapState, mapActions } from 'vuex'

import OrdersRelevantList from '@/components/Order/OrdersRelevantList'

import OrdersHistoryList from '@/components/Order/OrdersHistoryList'

import OrderReviewModal from '@/components/Order/OrderReviewModal'

export default {

components: { OrdersRelevantList, OrdersHistoryList, OrderReviewModal },

layout: 'cabinet',

data: () => ({

isRelevantTab: true,

order: {

selected: null,

isVisibleReviewModal: false

}

}),

computed: {

...mapState('auth', ['auth']),

...mapState('restaurants', ['restaurants']),

...mapState('orders', ['orders', 'isLoading']),

isHistoryTab() {

return !this.isRelevantTab

},

restaurantId() {

return this.$route.params.restaurantId

},

restaurant() {

const restaurant = this.restaurants.list.find(

item => item.\_id === this.restaurantId

)

return restaurant

}

},

created() {

this.handleRefresh()

},

methods: {

...mapActions('orders', {

fetchAll: 'FETCH\_ALL'

}),

handleRefresh() {

this.fetchAll({ userId: this.auth.user.\_id })

},

handleClickRelevantTab() {

this.isRelevantTab = true

},

handleClickHistoryTab() {

this.isRelevantTab = false

},

handleOpenReviewModal(order) {

this.order = {

selected: order,

isVisibleReviewModal: true

}

},

handleCloseReviewModal() {

this.order = {

selected: null,

isVisibleReviewModal: false

}

}

}

}

</script>

18. Сторінка «Замовлення ресторана» (/pages/cabinet/restaurant/orders/restaurants/\_restaurantId.vue):

<template>

<div class="w-full p-8 flex flex-col">

<div class="mb-8 flex text-2xl">

<nuxt-link

class="text-green-500 hover:underline"

:to="{ name: 'cabinet-restaurant-orders-restaurants' }"

>

Ресторани

</nuxt-link>

<span class="mx-1">/</span>

<span>{{ restaurant.name }}</span>

<span class="mx-1">/</span>

<span>Замовлення</span>

</div>

<div class="w-full flex justify-between">

<div class="w-full flex">

<div

class="w-64 p-4 flex items-center justify-center text-center rounded-t-md cursor-pointer"

:class="{

'bg-gray-50 shadow-inner': isHistoryTab,

'bg-white border border-b-0': !isHistoryTab

}"

@click="handleClickRelevantTab"

>

<span

class="font-semibold text-lg select-none"

:class="{

'text-gray-400': isHistoryTab,

'text-green-500': !isHistoryTab

}"

>

Актуальні ({{ orders.relevant.length }})

</span>

</div>

<div

class="w-64 p-4 flex items-center justify-center text-center rounded-t-md cursor-pointer"

:class="{

'bg-gray-50 shadow-inner': isRelevantTab,

'bg-white border border-b-0': !isRelevantTab

}"

@click="handleClickHistoryTab"

>

<span

class="font-semibold text-lg select-none"

:class="{

'text-gray-400': isRelevantTab,

'text-green-500': !isRelevantTab

}"

>

Історія ({{ orders.history.length }})

</span>

</div>

</div>

<div

class="h-full px-8 flex items-center cursor-pointer transform hover:text-green-500 hover:scale-110"

@click="handleRefresh"

>

<i class="fas fa-sync-alt text-xl"></i>

</div>

</div>

<div class="h-full w-full">

<OrdersRelevantList

v-if="isRelevantTab"

:list="orders.relevant"

@select="handleOpenReviewModal"

/>

<OrdersHistoryList

v-else

:list="orders.history"

@select="handleOpenReviewModal"

/>

</div>

<OrderReviewModal

v-if="order.isVisibleReviewModal"

:order="order.selected"

@close="handleCloseReviewModal"

/>

</div>

</template>

<script>

import { mapState, mapActions } from 'vuex'

import OrdersRelevantList from '@/components/Order/OrdersRelevantList'

import OrdersHistoryList from '@/components/Order/OrdersHistoryList'

import OrderReviewModal from '@/components/Order/OrderReviewModal'

export default {

components: { OrdersRelevantList, OrdersHistoryList, OrderReviewModal },

layout: 'cabinet',

data: () => ({

isRelevantTab: true,

order: {

selected: null,

isVisibleReviewModal: false

}

}),

computed: {

...mapState('restaurants', ['restaurants']),

...mapState('orders', ['orders', 'isLoading']),

isHistoryTab() {

return !this.isRelevantTab

},

restaurantId() {

return this.$route.params.restaurantId

},

restaurant() {

const restaurant = this.restaurants.list.find(

item => item.\_id === this.restaurantId

)

return restaurant

}

},

created() {

this.handleRefresh()

},

methods: {

...mapActions('orders', {

fetchAll: 'FETCH\_ALL'

}),

handleRefresh() {

this.fetchAll({ restaurantId: this.restaurantId })

},

handleClickRelevantTab() {

this.isRelevantTab = true

},

handleClickHistoryTab() {

this.isRelevantTab = false

},

handleOpenReviewModal(order) {

this.order = {

selected: order,

isVisibleReviewModal: true

}

},

handleCloseReviewModal() {

this.order = {

selected: null,

isVisibleReviewModal: false

}

}

}

}

</script>

19. Сторінка «Список ресторанів користувача» (/pages/cabinet/restaurant/orders/restaurants/index.vue):

<template>

<div class="w-full p-8 flex flex-col">

<div class="mb-8 flex text-2xl">

<span>Ресторани</span>

</div>

<div class="grid grid-cols-3 gap-8">

<Card

v-for="restaurant of restaurants.list"

:key="`order-restaurant-${restaurant.\_id}`"

class="col-span-1"

:is-buttons="false"

:restaurant="restaurant"

@select="handleSelect(restaurant)"

/>

</div>

</div>

</template>

<script>

import { mapState, mapActions } from 'vuex'

import Card from '@/components/Restaurants/RestaurantsCabinetCard'

export default {

components: { Card },

layout: 'cabinet',

computed: {

...mapState('restaurants', ['restaurants'])

},

async created() {

await this.fetchAll()

},

methods: {

...mapActions('restaurants', {

fetchAll: 'FETCH\_ALL',

removeById: 'REMOVE\_BY\_ID'

}),

handleSelect(restaurant) {

this.$router.push({

name: `cabinet-restaurant-orders-restaurants-restaurantId`,

params: { restaurantId: restaurant.\_id }

})

}

}

}

</script>

20. Сторінка «Список категорій ресторана користувача» (/pages/cabinet/restaurant/restaurants/\_restaurantId/categories/index.vue):

<template>

<div class="w-full p-8 flex flex-col">

<div class="mb-8 flex text-2xl">

<nuxt-link

class="text-green-500 hover:underline"

:to="{ name: 'cabinet-restaurant-restaurants' }"

>

Ресторани

</nuxt-link>

<span class="mx-1">/</span>

<span>{{ restaurant.name }}</span>

<span class="mx-1">/</span>

<span>Категорії</span>

</div>

<div class="w-auto mb-8 flex">

<Button text="Додати категорію" @click="handleOpenCreationFormModal" />

</div>

<div class="grid grid-cols-3 gap-8">

<CategoryCard

v-for="category of restaurant.categories"

:key="`category-${category.\_id}`"

class="col-span-1"

:category="category"

@select="handleSelect(category)"

@remove="handleRemove(category)"

/>

</div>

<CreationFormModal

v-if="isVisibleCreationFormModal"

@close="handleCloseCreationFormModal"

/>

</div>

</template>

<script>

import { mapState, mapActions } from 'vuex'

import Button from '@/components/Base/BaseButton'

import CategoryCard from '@/components/Categories/CategoriesCabinetCard'

import CreationFormModal from '@/components/Categories/CategoriesCabinetCreationFormModal'

export default {

components: { Button, CategoryCard, CreationFormModal },

layout: 'cabinet',

data: () => ({

isVisibleCreationFormModal: false

}),

computed: {

...mapState('restaurants', ['restaurants']),

restaurant() {

const restaurantId = this.$route.params.restaurantId

const restaurant = this.restaurants.list.find(

item => item.\_id === restaurantId

)

return restaurant

}

},

methods: {

...mapActions('categories', {

removeById: 'REMOVE\_BY\_ID'

}),

handleSelect(category) {

this.$router.push({

name: `cabinet-restaurant-restaurants-restaurantId-categories-categoryId-positions`,

params: { restaurantId: this.restaurant.\_id, categoryId: category.\_id }

})

},

async handleRemove(category) {

const isRemove = confirm('Видалити категорію?')

if (!isRemove) return

const isSuccess = await this.removeById(category.\_id)

if (isSuccess) {

this.$toast.success('Категорію видалено')

}

},

handleOpenCreationFormModal() {

this.isVisibleCreationFormModal = true

},

handleCloseCreationFormModal() {

this.isVisibleCreationFormModal = false

}

}

}

</script>

21. Сторінка «Список позицій категорії ресторана користувача» (/pages/cabinet/restaurant/restaurants/\_restaurantId/categories/\_categoryId/positions.vue):

<template>

<div class="w-full p-8 flex flex-col">

<div class="mb-8 flex text-2xl">

<nuxt-link

class="text-green-500 hover:underline"

:to="{ name: 'cabinet-restaurant-restaurants' }"

>

Ресторани

</nuxt-link>

<span class="mx-1">/</span>

<span>{{ restaurant.name }}</span>

<span class="mx-1">/</span>

<nuxt-link

class="text-green-500 hover:underline"

:to="{

name: `cabinet-restaurant-restaurants-restaurantId-categories`,

params: { restaurantId: restaurant.\_id }

}"

>

Категорії

</nuxt-link>

<span class="mx-1">/</span>

<span>{{ category.name }}</span>

</div>

<div class="w-auto mb-8 flex">

<Button text="Додати позицію" @click="handleOpenCreationFormModal" />

</div>

<div class="grid grid-cols-3 gap-8">

<PositionCard

v-for="position of category.positions"

:key="`position-${position.\_id}`"

:position="position"

class="col-span-1"

@remove="handleRemove(position)"

/>

</div>

<CreationFormModal

v-if="isVisibleCreationFormModal"

@close="handleCloseCreationFormModal"

/>

</div>

</template>

<script>

import { mapState, mapActions } from 'vuex'

import Button from '@/components/Base/BaseButton'

import PositionCard from '@/components/Positions/PositionsCabinetCard'

import CreationFormModal from '@/components/Positions/PositionsCabinetCreationFormModal'

export default {

components: { Button, PositionCard, CreationFormModal },

layout: 'cabinet',

data: () => ({

isVisibleCreationFormModal: false

}),

computed: {

...mapState('restaurants', ['restaurants']),

restaurant() {

const restaurantId = this.$route.params.restaurantId

const restaurant = this.restaurants.list.find(

item => item.\_id === restaurantId

)

return restaurant

},

category() {

const categoryId = this.$route.params.categoryId

const category = this.restaurant.categories.find(

item => item.\_id === categoryId

)

return category

}

},

methods: {

...mapActions('positions', {

removeById: 'REMOVE\_BY\_ID'

}),

async handleRemove(category) {

const isRemove = confirm('Видалити категорію?')

if (!isRemove) return

const isSuccess = await this.removeById(category.\_id)

if (isSuccess) {

this.$toast.success('Позицію видалено')

}

},

handleOpenCreationFormModal() {

this.isVisibleCreationFormModal = true

},

handleCloseCreationFormModal() {

this.isVisibleCreationFormModal = false

}

}

}

</script>

22. Плагін конфігурації запитів на сервер (/plugins/axios.js):

import { API\_URL } from '@/config'

let clientUrl = ''

if (process.client) {

clientUrl = location.protocol + '//' + location.host

}

export default function ({ $axios, store }) {

$axios.defaults.baseURL = API\_URL

$axios.defaults.browserBaseURL = API\_URL

$axios.onRequest(config => {

config.params = config.params || {}

config.headers = {

...config.headers,

clientUrl,

authorization: store.state.auth.auth.token || ''

}

})

$axios.onError(error => {

// eslint-disable-next-line no-console

console.error(error)

})

return $axios

}

23. Плагін для плагінів Nuxt.js (/plugins/inject-nuxt.js):

import generateAPI from '@/utils/api'

export default (context, inject) => {

inject('api', generateAPI(context))

}

24. Плагін для коректного відображення висоти сторінки (/plugins/screen-height.client.js):

if (process.browser) {

const onResize = () => {

const vh = window.innerHeight \* 0.01

document.documentElement.style.setProperty('--vh', `${vh}px`)

}

window.addEventListener('resize', onResize)

onResize()

}

25. Модуль глобального хранилища (/store/index.js):

import \* as cookies from '@/utils/cookies'

export const actions = {

async nuxtServerInit({ commit, dispatch }, context) {

const { req } = context

const token = cookies.getAuthToken(req)

await commit('auth/SET\_TOKEN', token)

await dispatch('auth/FETCH\_USER')

}

}

26. Модуль хранилища авторизації (/store/auth.js):

/\* eslint-disable no-undef \*/

import \* as cookies from '@/utils/cookies'

import { ROLES } from '@/utils/auth'

export const state = () => ({

auth: {

user: null,

token: null,

isAuthenticated: null,

isLoading: false

}

})

export const getters = {

isAuthenticated: state => Boolean(state.auth.user),

isRestaurant: state => state.auth.user?.role === ROLES.RESTAURANT,

isUser: state => state.auth.user?.role === ROLES.USER

}

export const mutations = {

SET\_USER(state, user) {

state.auth.user = user

},

SET\_TOKEN(state, token) {

state.auth.token = token

},

SET\_LOADING(state, isLoading) {

state.auth.isLoading = isLoading

}

}

export const actions = {

async LOGIN({ commit, dispatch }, payload) {

let result

commit('SET\_LOADING', true)

try {

result = await this.$api.auth.login(payload)

const { isSuccess, token } = result

if (isSuccess) {

await commit('SET\_TOKEN', token)

cookies.setAuthToken(token)

await dispatch('FETCH\_USER')

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return result

},

async REGISTRATION({ commit, dispatch }, payload) {

let result

commit('SET\_LOADING', true)

try {

result = await this.$api.auth.registration(payload)

const { isSuccess } = result

if (isSuccess) {

dispatch('LOGIN', payload)

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return result

},

async FETCH\_USER({ commit, dispatch, state }) {

const isToken = Boolean(state.auth.token)

if (!isToken) {

await dispatch('LOGOUT')

return

}

try {

const { isSuccess, user } = await this.$api.auth.getUser()

if (isSuccess) {

await commit('SET\_USER', user)

}

} catch (e) {

console.error(e.message)

}

},

async UPDATE\_USER\_INFO({ commit, dispatch, state }, toUpdate) {

const isToken = Boolean(state.auth.token)

if (!isToken) {

await dispatch('LOGOUT')

return

}

try {

const { isSuccess, user } = await this.$api.auth.updateUserInfo(toUpdate)

if (isSuccess) {

await commit('SET\_USER', user)

}

} catch (e) {

console.error(e.message)

}

},

async LOGOUT({ commit }) {

if (process.client) {

const pages = ['cabinet'].join(' ')

const currentPage = $nuxt.$route.name

if (currentPage.includes(pages)) {

$nuxt.$router.push('/')

}

}

cookies.removeAuthToken()

await commit('SET\_USER', null)

},

async SEND\_VERIFICATION\_EMAIL() {

let isSuccess = false

try {

isSuccess = await this.$api.auth.sendVerificationEmail()

console.log(isSuccess)

} catch (e) {

console.error(e.message)

}

return isSuccess

},

async VERIFY\_EMAIL({ commit, dispatch }, hash) {

let isSuccess = false

commit('SET\_LOADING', true)

try {

isSuccess = await this.$api.auth.verifyEmail(hash)

if (isSuccess) {

await dispatch('FETCH\_USER')

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return isSuccess

},

async SEND\_PASSWORD\_RESET\_EMAIL({ commit }, email) {

commit('SET\_LOADING', true)

let isSuccess = false

try {

isSuccess = await this.$api.auth.sendPasswordResetEmail(email)

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return isSuccess

},

async RESET\_PASSWORD({ commit }, { password, hash }) {

commit('SET\_LOADING', true)

let isSuccess = false

try {

isSuccess = await this.$api.auth.resetPassword({ password, hash })

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return isSuccess

}

}

27. Модуль хранилища ресторанів (/store/restaurants.js):

export const state = () => ({

restaurants: {

list: [],

selected: null,

isLoading: false

},

settlement: 'Полтава'

})

export const mutations = {

SET\_LIST(state, list) {

state.restaurants.list = list

},

SET\_SELECTED(state, selected) {

state.restaurants.selected = selected

},

SET\_LOADING(state, isLoading) {

state.restaurants.isLoading = isLoading

},

SET\_SETTLEMENT(state, settlement) {

state.settlement = settlement

}

}

export const actions = {

async FETCH\_ALL({ state, commit }, filter) {

let result

commit('SET\_LOADING', false)

filter = {

...filter,

settlement: state.settlement

}

try {

result = await this.$api.restaurants.getAll(filter)

const { isSuccess, restaurants } = result

if (isSuccess) {

await commit('SET\_LIST', restaurants)

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return result

},

async FETCH\_BY\_ID({ commit }, id) {

let result

commit('SET\_LOADING', true)

try {

result = await this.$api.restaurants.getById(id)

const { isSuccess, restaurant } = result

if (isSuccess) {

await commit('SET\_SELECTED', restaurant)

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return result

},

async CREATE({ commit, dispatch }, payload) {

let isSuccess

commit('SET\_LOADING', true)

try {

isSuccess = await this.$api.restaurants.create(payload)

if (isSuccess) {

dispatch('FETCH\_ALL')

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return isSuccess

},

async REMOVE\_BY\_ID({ commit, dispatch }, id) {

let isSuccess

commit('SET\_LOADING', true)

try {

isSuccess = await this.$api.restaurants.removeById(id)

if (isSuccess) {

dispatch('FETCH\_ALL')

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return isSuccess

}

}

28. Модуль хранилища категорій (/store/categories.js):

export const state = () => ({

categories: {

isLoading: false

}

})

export const mutations = {

SET\_LOADING(state, isLoading) {

state.categories.isLoading = isLoading

}

}

export const actions = {

async CREATE({ commit, dispatch }, { restaurantId, form }) {

let isSuccess

commit('SET\_LOADING', true)

try {

isSuccess = await this.$api.categories.create({ restaurantId, form })

if (isSuccess) {

dispatch('restaurants/FETCH\_ALL', {}, { root: true })

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return isSuccess

},

async REMOVE\_BY\_ID({ commit, dispatch }, id) {

let isSuccess

commit('SET\_LOADING', true)

try {

isSuccess = await this.$api.categories.removeById(id)

if (isSuccess) {

dispatch('restaurants/FETCH\_ALL', {}, { root: true })

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return isSuccess

}

}

29. Модуль хранилища позицій (/store/positions.js):

export const state = () => ({

positions: {

isLoading: false

}

})

export const mutations = {

SET\_LOADING(state, isLoading) {

state.positions.isLoading = isLoading

}

}

export const actions = {

async CREATE({ commit, dispatch }, { categoryId, form }) {

let isSuccess

commit('SET\_LOADING', true)

try {

isSuccess = await this.$api.positions.create({ categoryId, form })

if (isSuccess) {

dispatch('restaurants/FETCH\_ALL', {}, { root: true })

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return isSuccess

},

async REMOVE\_BY\_ID({ commit, dispatch }, id) {

let isSuccess

commit('SET\_LOADING', true)

try {

isSuccess = await this.$api.positions.removeById(id)

if (isSuccess) {

dispatch('restaurants/FETCH\_ALL', {}, { root: true })

}

} catch (e) {

console.error(e.message)

}

commit('SET\_LOADING', false)

return isSuccess

}

}