Содержание

[Введение 3](#_Toc135377366)

[1 Постановка задачи 4](#_Toc135377367)

[1.1 Обзор аналогов 4](#_Toc135377368)

[2 Проектирование web-приложения 7](#_Toc135377369)

[2.1 Диаграмма использования 7](#_Toc135377370)

[2.2 Структурная схема приложения 7](#_Toc135377371)

[2.3 Проектирование базы данных 8](#_Toc135377372)

[3 Разработка web-приложения 11](#_Toc135377373)

[3.1 Разработка серверной части 11](#_Toc135377374)

[3.2 Разработка клиентской части 15](#_Toc135377375)

[4 Тестирование web-приложения 18](#_Toc135377376)

[5 Руководство пользователя 28](#_Toc135377377)

[5.1 Регистрация пользователя 28](#_Toc135377378)

[5.2 Аутентификация и авторизация пользователя 29](#_Toc135377379)

[5.3 Блюда 30](#_Toc135377380)

[5.4 Оформление предзаказа 31](#_Toc135377381)

[5.5 Страница бронирования 32](#_Toc135377382)

[5.6 Страница оформления предзаказа с бронью 32](#_Toc135377383)

[5.7 Страница с комментариями 33](#_Toc135377384)

[Заключение 34](#_Toc135377385)

[Список используемых источников 35](#_Toc135377386)

[Приложение А 36](#_Toc135377387)

[Приложение Б 38](#_Toc135377388)

[Приложение Г 40](#_Toc135377389)

# Введение

В свете изменяющихся предпочтений и требований потребителей, создание веб-приложения для ресторанов с возможностью бронирования и предзаказа является неотъемлемой частью современного гостеприимства и обеспечивает улучшенный опыт для клиентов.

Основной целью разработки данного веб-приложения является предоставление клиентам удобного и надежного инструмента для бронирования столиков и предзаказа блюд в ресторане. Это позволит клиентам экономить время и получать максимальную отдачу от своего посещения, а также улучшит эффективность работы ресторана.

Для достижения этой цели были определены следующие задачи:

* Разработка интуитивно понятного пользовательского интерфейса: Главным фактором успеха веб-приложения является его интерфейс, который должен быть простым, удобным и интуитивно понятным для клиентов. Мы стремимся создать интерфейс, который позволит пользователям легко осуществлять навигацию по странице и оформлять предзаказ блюд всего лишь несколькими нажатиями;
* Предоставление клиенту возможности оформить предзаказ и забронировать столик, чтобы упростить и ускорить процесс заказа.
* Предоставление клиенту возможности оставлять комментарии и в реальном времени общаться с другими пользователями и взаимодействовать с администратором. Это создаст дополнительную социальную составляющую и позволит клиентам делиться своими впечатлениями, обсуждать ресторан и получать ответы на свои вопросы или запросы от администратора.

1 Постановка задачи

На основе анализа задач, поставленных в данном курсовом проекте, были рассмотрены различные приложения, призванные эти задачи решить.

## 1.1 Обзор аналогов

Основным средством, выбранным для решения поставленных задач, является web-приложение "topbelarus.by". В настоящее время оно является одним из наиболее популярных и успешных в данной сфере, поскольку предоставляет гораздо больше возможностей, чем просто бронирование столов и оформление предзаказов в заведениях. Главным преимуществом этого средства является его расширенные функциональные возможности, поскольку web-приложение "topbelarus.by" предлагает широкий спектр дополнительных функций, которые обогащают пользовательский опыт. К ним относятся возможность ознакомиться с меню ресторанов, оценить отзывы других пользователей, выбрать удобное время и место посещения, а также оставить собственный отзыв. Кроме того, данное web-приложение имеет очень удобный пользовательский интерфейс, в котором сложно запутаться. Главная страница приложения представлена на рисунке 1.1.

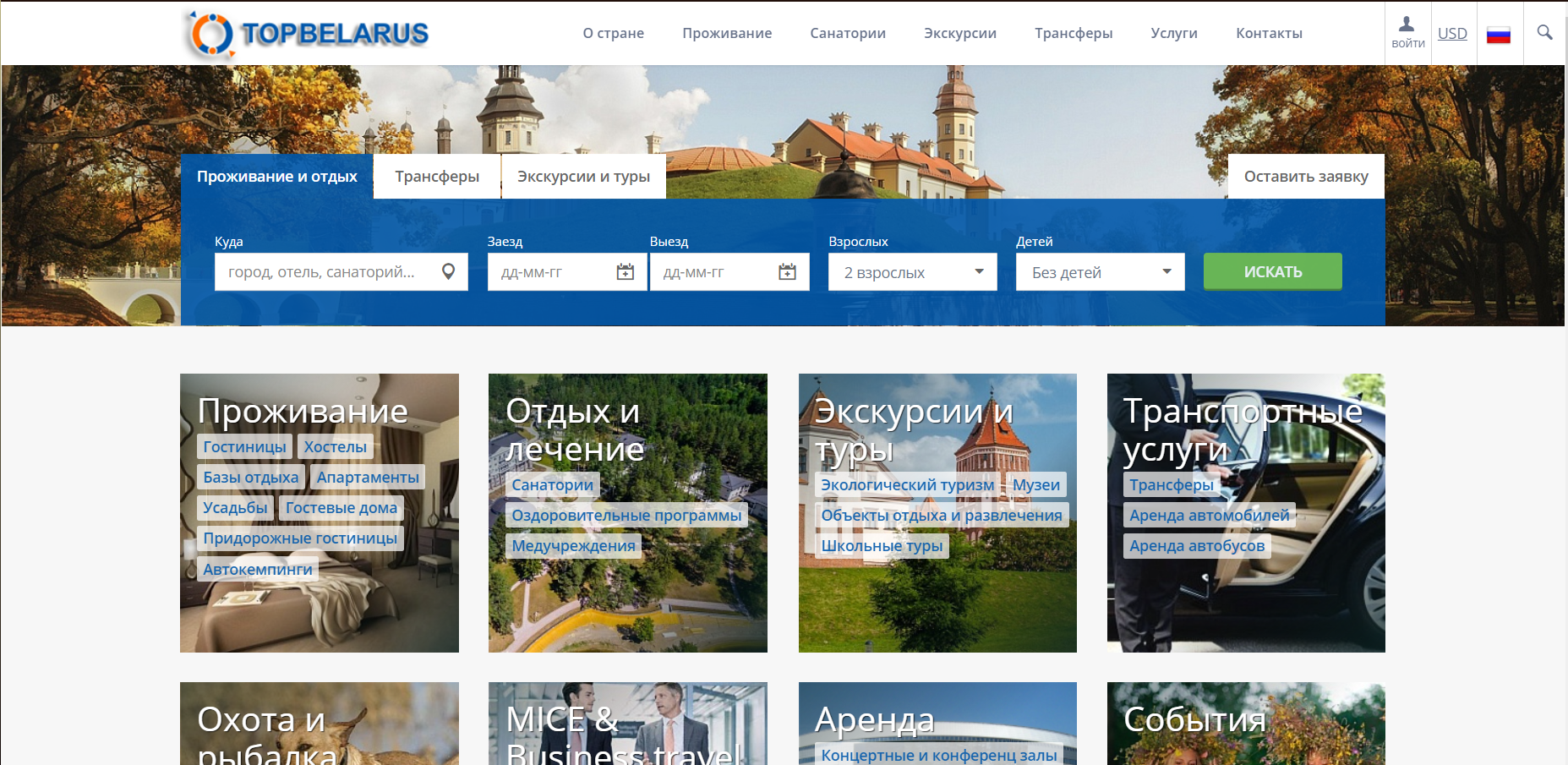


Рисунок 1.1 – Главная страница «topbelarus.by»

Так как web-приложение «topbelarus.by» является одним из лидеров в сфере бронирования и поиска услуг, существенных недостатков выявлено не было. Единственная выявленная проблема заключается в том, что приложение долго загружается.

Также для сравнения был выбран сайт «druzya.by». Его особенностью является то, что он содержит меньше функций, чем «topbelarus.by», поскольку всецело ориентирован на предоставление услуг для собственного ресторана. Кроме того, здесь имеется возможность оформить доставку. Интерфейс этого сервиса представлен на рисунке 1.2.

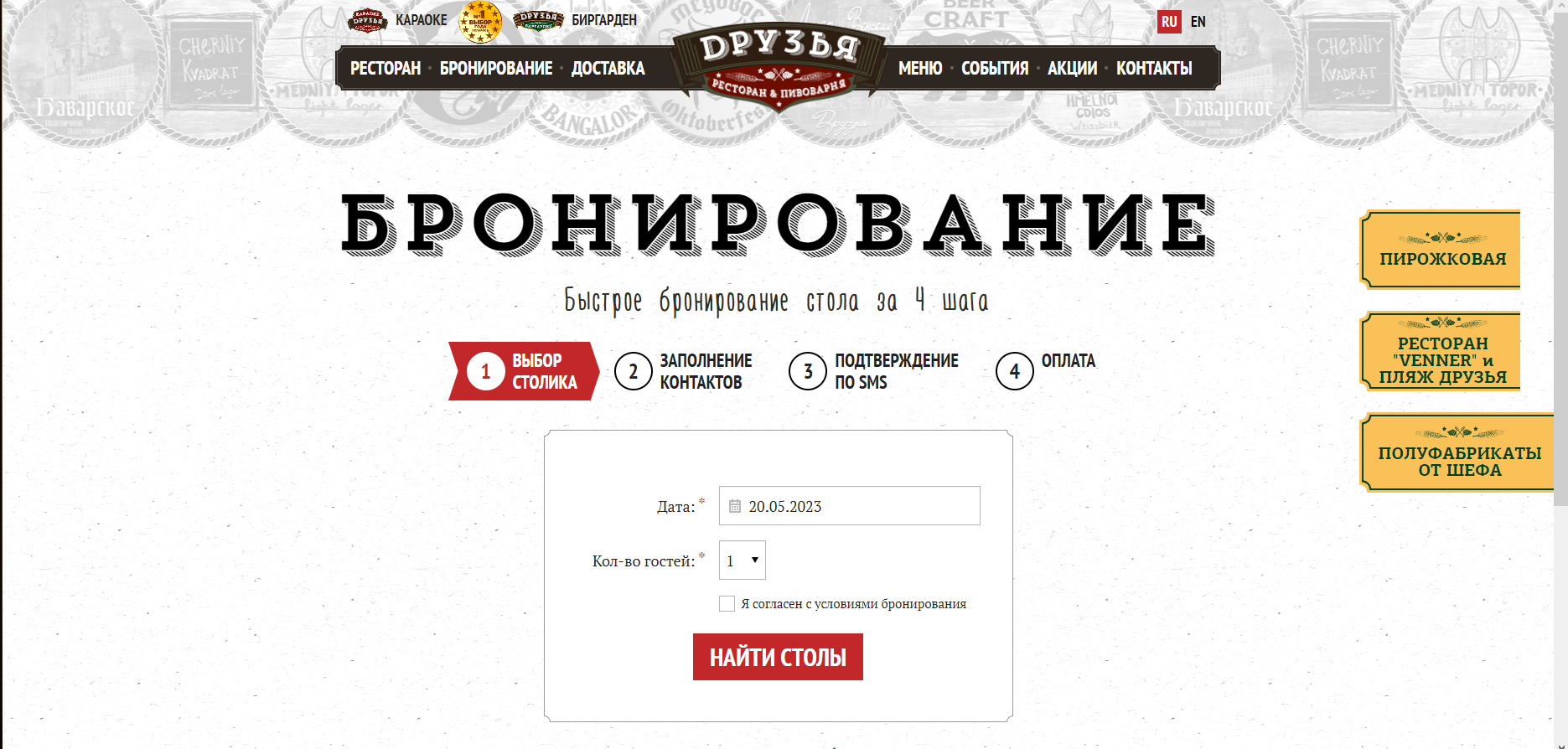


Рисунок 1.2 – Главная страница «druzya.by»

Так как проанализированные приложения уже многие годы являются популярными среди большого количества пользователей, и постоянно активно развиваются и поддерживаются большими командами разработчиков, то реализовывать все возможные функции данных приложений было бы нецелесообразно. Поэтому были выделены лишь основные требования к приложению для ресторана.

Функциональные требования к приложению:

Для администратора:

* обеспечивать возможность авторизации;
* позволять просматривать меню;
* позволять добавлять и удалять пункты меню;
* обеспечивать возможность просматривать брони и предзаказы и подтверждать их с помощью рассылки клиентам на почту;

Для клиента:

* обеспечивать возможность регистрации и авторизации;
* позволять просматривать меню;
* обеспечивать возможность оформлять предзаказ;
* обеспечивать возможность бронирования столов на выбранную дату;
* позволять оставлять комментарии;

Для гостя:

* обеспечивать возможность регистрации;
* позволять просматривать меню.

На основе функциональных требований были выделены технические требования к проектируемому web-приложению.

Требования к системе:

* разработать серверную часть web-приложения на платформе NodeJS;
* при разработке серверной части использовать фреймворк Express;
* для разработки web-socket сервера использовать библиотеку socket.io;
* использовать mongodb в качестве базы данных;
* спроектировать необходимые таблицы в базе данных;
* для взаимодействия серверной части с базой данных использовать драйвер mongoose;
* использовать Firebase Authentication для аутентификации пользователей;
* клиентскую часть приложения разработать с помощью библиотеки ReactJS;
* использовать https в качестве протокола для безопасной передачи информации;

Требования к безопасности:

* для безопасной передачи данных между сервером и клиентом использовать https;
* для защиты от хакерских атак проводить валидацию входных данных.

Требования к дизайну интерфейса:

* дизайн должен быть простым, удобным и интуитивно понятным для пользователей.

2 Проектирование web-приложения

## 2.1 Диаграмма использования

Диаграмма использования для разрабатываемого приложения представлена на рисунке 2.1.

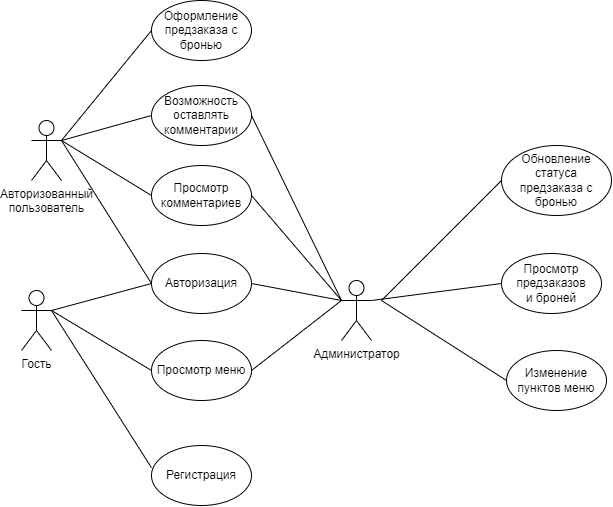


Рисунок 2.1 – Диаграмма использования

Как видно из диаграммы использования, приложение поддерживает три основные роли: гость, пользователь и администратор. Авторизованный администратор, в отличие от пользователя, имеет доступ к списку всех броней с предзаказами и может редактировать меню.

## 2.2 Структурная схема приложения

Структурная схема приложения представлена на рисунке 2.2.

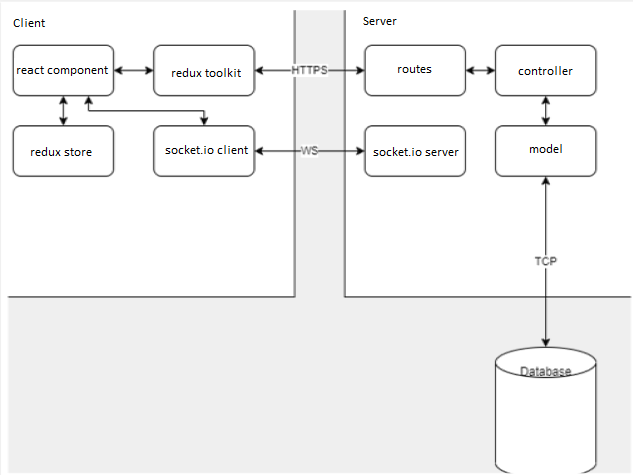


Рисунок 2.2 – Структурная схема приложения

## 2.3 Проектирование базы данных

Хранилище данных является фундаментом приложения, и правильное проектирование позволяет улучшить производительность, обеспечить безопасность и удобство использования. Основная задача проектирования базы данных заключается в создании структуры, которая соответствует требованиям приложения и обеспечивает эффективное хранение и доступ к информации. Это включает определение сущностей и их характеристик, установление связей между сущностями и формирование таблиц, отражающих эту структуру. Кроме того, проектирование базы данных включает разработку правил для обеспечения целостности данных, таких как ограничения, проверки и уникальные индексы. Эти правила помогают предотвратить ошибки при вводе данных и гарантировать точность и надежность информации. Важно также учесть масштабируемость базы данных при планировании. Необходимо предусмотреть возможность расширения базы данных в будущем, чтобы она могла эффективно обрабатывать большие объемы информации. Таким образом, планирование базы данных играет важную роль в разработке приложения, обеспечивая его эффективную работу и надежность. Модель базы данных для приложения представлена на рисунке 2.3.

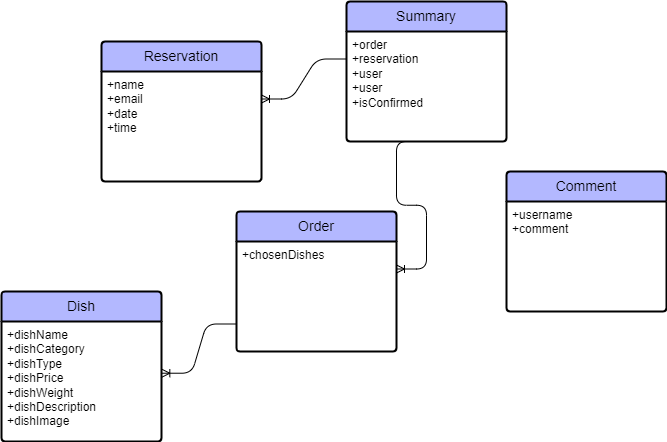


Рисунок 2.3 – Модель базы данных

Далее будет описана каждая таблица базы данных.

В таблице «orders» хранятся предзаказы. Описание структуры таблицы «orders» представлено в таблице 2.2.

Таблица 2.2 – Структура таблицы «orders»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| chosenDishes | Array | Массив, хранящий выбранные блюда |

Таблица «reservations» содержит информацию о бронированиях. Её структура представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы «reservations»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| name | String, required | Имя пользователя |
| email | String, required | Электронная почта |
| date | Date, required | Дата для бронирования |

В таблице «dishes» хранятся блюда. Описание структуры таблицы «users» представлено в таблице 2.1.

Таблица 2.1 – Структура таблицы «dishes»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| dishName | String, required | Название блюда |
| dishCategory | String, required | Категория блюда |
| dishType | String, required | Тип блюда |
| dishPrice | Number, required | Цена блюда |
| dishWeight | Number, required | Вес блюда |
| dishDescription | String, required | Описание блюда |
| dishImage | String, required | Название пути для картинки блюда |

В таблице «summary» содержится информация, включающая себя и предзаказы, и брони. Описание структуры таблицы «summary» представлено в таблице 2.4.

Таблица 2.4 – Структура таблицы «summary»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| order | Schema.Types.ObjectId, ref: "Order" | Идентификатор предзаказа |
| reservation | Schema.Types.ObjectId, ref: "Reservation" | Идентификатор брони |
| user | String, required | Имя пользователя |
| isСonfirmed | Boolean | Проверка на подтверждение заказа |

В таблице «comments» содержатся оставленные пользователями и администратором комментарии. Описание структуры таблицы «comments» представлено в таблице 2.5.

Таблица 2.5 – Структура таблицы «comments»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| comment | String | Текст комментария |

Таким образом при проектировании базы данных используется 5 таблиц, присутствует связь один-ко-многим.

3 Разработка web-приложения

В процессе выполнения курсового проекта были разработаны два отдельных компонента: server, который содержит реализацию серверной части приложения, и client, отвечающий за клиентское представление. Модуль web-socket сервера содержится в папке server. Client представляет собой React-приложение, сгенерированное командой «npx create-react-app».

## 3.1 Разработка серверной части

На рисунке 3.1 представлена структура серверной части приложения, включающая в себя директории, конфигурационные файлы и файлы скриптов.

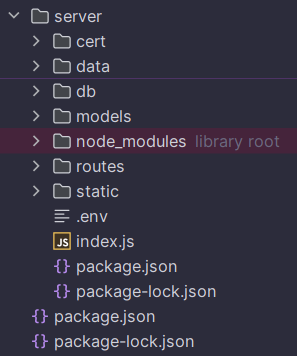


Рисунок 3.1 – Структура серверной части

Входной точкой в приложение является файл index.js. В этом файле запускается прослушивание запросов, которые принимает сервер, и производится создание вебсокет-сервера.

В директории «models» содержатся модели базы данных, которые используются для доступа к данным и для создания новых данных. Листинг моделей представлен в Приложении А.

В директории «routes» находятся роутеры – файлы, содержащие логику маршрутизации и обработки запросов. Файл «dishRouter.js» является примером роутера, который принимает на себя запросы, связанные с блюдами.

Содержимое файла роутера «dishRouter.js» представлено в листинге 3.1.

|  |
| --- |
| const express = require('express')  const router = express.Router()  const Dish = require('../models/dishModel.js')  router.get("/dishes", async (req, res) => {  try {  const result = await Dish.find()  if (!result) {  res.json({  status: "FAILED",  message: "Not found record"  })  } else {  res.json({  status: "SUCCESS",  message: "Records found",  data: result  })  }  } catch (e) {  console.log(e)  }  })  // Получение информации о конкретном блюде  router.get('/dishes/:id', async (req, res) => {  try {  const \_id = req.params.id;  const result = await Dish.findById(\_id);  if (!result) {  res.json({  status: "FAILED",  message: "Record not found on this ID"  })  } else {  res.json({  status: "SUCCESS",  message: "Records found",  data: result  })  }  } catch (e) {  res.send(e)  }  });  router.get('/dishes-by-category', async (req, res) => {  try {  const dishes = await Dish.aggregate([  {$match: {}},  {  $group: {  \_id: '$dishCategory',  dishes: {$push: '$$ROOT'}  }  },  {$project: {name: '$\_id', dishes: 1, \_id: 0}}  ])  res.status(200).send({data: dishes})  } catch (err) {  res.status(400).send({error: err})  }  });  //POST  // Создание нового блюда  router.post("/dishes", async (req, res) => {  try {  console.log(req.body)  const data = new Dish(req.body)  const result = await data.save()  console.log(result)  res.json({  data: result,  status: "SUCCESS",  message: "Dish added successfully.",  })  } catch (error) {  if (error.code === 11000) {  // Обработка ошибки с дубликатом индекса  res.json({  status: "FAILED",  message: "Dish already exists with this name."  })  } else {  // Обработка других ошибок  res.json({  status: "FAILED",  message: "Failed to add dish."  })  }  }  })  // PUT  router.put('/dishes/:id', async (req, res) => {  try {  const \_id = req.params.id;  const result = await Dish.findByIdAndUpdate(\_id, req.body, {new: true});  console.log(result)  if (!result) {  res.json({  status: "FAILED",  message: "Records haven't been updated.",  data: result  })  } else {  res.json({  status: "SUCCESS",  message: "Record was updated.",  data: result  })  }  } catch (e) {  res.send(e)  }  });  // DELETE  router.delete('/dishes/:id', async (req, res) => {  try {  const \_id = req.params.id;  const result = await Dish.findByIdAndDelete(\_id);  if (!result) {  res.json({  status: "FAILED",  message: "Record haven't been deleted."  })  }  else {  res.json({  status: "SUCCESS",  message: "Record was deleted."  })  }  }  catch (e) {  res.send(e)  }  });  module.exports = router |

Листинг 3.1 –Содержимое файла роутера «dishRouter.js»

В папке «cert» находятся сертификаты, необходимые для запуска https-сервера.

В папке «data» содержатся файлы json для вставки в базу данных.

В папке «db» находится файл «index.js» с параметрами подключения к бд.   
Код функции представлен в листинге 3.2.

|  |
| --- |
| const mongoose = require('mongoose')  mongoose  .connect('mongodb://127.0.0.1:27017/SpiceZen', {useNewUrlParser: true})  .catch((e)=>{  console.log(e);  })  const db = mongoose.connection  module.exports = db |

Листинг 3.2 – ./db/index.js

В файле «index.js» создаётся web-socket сервер вместе с https-сервером, который обеспечивает возможность обмена данными между клиентом и сервером через постоянное соединение. Код создания web-socket сервера представлен в Приложении Б.

В файле package.json хранится список зависимостей (библиотек), необходимых приложению NodeJS для работы. Содержимое данного файла представлено в Приложении Г.

## 3.2 Разработка клиентской части

На рисунке 3.2 изображена структура клиентской части разрабатываемого приложения.

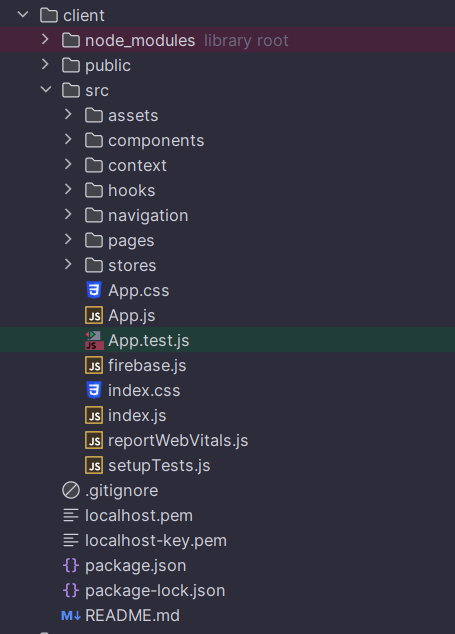


Рисунок 3.2 – Структура клиентской части

Клиентская часть приложения написана на React, JavaScript-библиотеке для создания пользовательских интерфейсов. React является одной из самых популярных библиотек для создания веб-приложений. React используется для создания одностраничных приложений (SPA), где все действия происходят на одной странице, без перезагрузки. Он основан на концепции компонентов, которые могут быть повторно использованы и являются основными строительными блоками приложения. ReactJS также использует виртуальный DOM (Document Object Model), что позволяет обновлять только те части страницы, которые изменились, вместо того чтобы перерисовывать всю страницу целиком. Это делает приложения, созданные с помощью ReactJS, более быстрыми и отзывчивыми. В целом, ReactJS - это мощный инструмент для создания высокопроизводительных и масштабируемых пользовательских интерфейсов для веб-приложений.

Директория «src» содержит исходный код клиентской части приложения.

Папка «assets» содержит стили и изображения.

Папка «components» содержит React-компоненты, которые могут повторно использоваться во всем приложении.

Папка «context» содержит функциональность аутентификации пользователей в React-приложении с использованием Firebase.

Папка «hooks» содержит пользовательскую хук-функцию для управления переключением вкладок в React-приложении.

Папка «navigation» содержит файл «index.js», который используется для определения маршрутов страниц приложения.

Папка «pages» содержит компоненты-страницы приложения, каждый из которых имеет свой маршрут и отображает определенный контент на странице. Компоненты-страницы используют компоненты из папки «components» для построения пользовательского интерфейса.

Папка «stores» содержит модули Redux, которые определяют срез (slice) для хранения информации.

Файл «firebase.js» представляет собой модуль, который инициализирует приложение Firebase и экспортирует объект auth для работы с аутентификацией.

Файл «index.js» является точкой входа в React-приложение.

Файл «App.js» представляет компонент App, который является корневым компонентом вашего React-приложения.

Файлы «localhost.pem» и «localhost-key.pem» нужны для запуска клиентского приложения на https. Они указываются в файле «package.json».

4 Тестирование web-приложения

В данном разделе будет рассмотрен процесс тестирования готового, целостного приложения. Тестирование будет проводиться с использованием клиентской части.

При регистрации и авторизации пользователя в случае ввода некорректных данных, будет выведено сообщение об ошибке. Пример такого сообщения представлен на рисунке 4.1

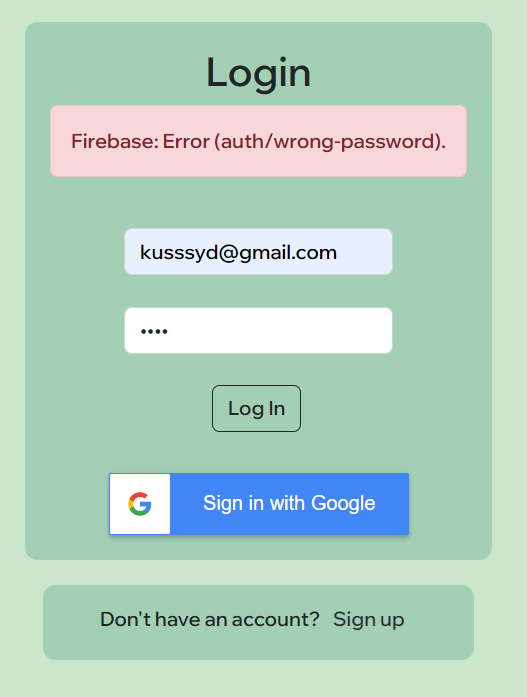


Рисунок 4.1 – Ошибка некорректных данных при авторизации

Кроме того, при регистрации также выполняется проверка на соответствие вводимого пароля. Пример ошибки короткого пароля представлен на рисунке 4.2.

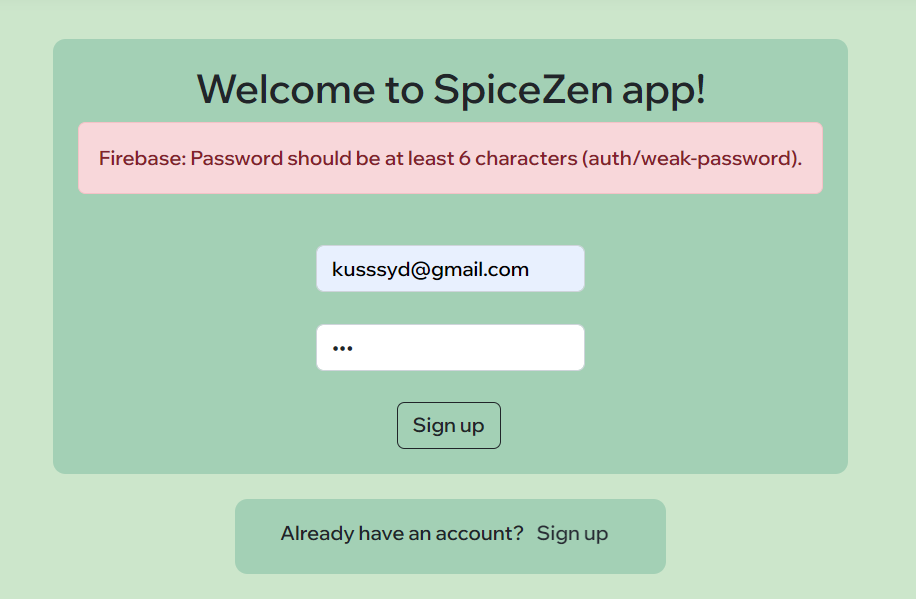


Рисунок 4.2 – Ошибка, связанная со слишком коротким паролем

Если пользователь попытается зарегистрировать существующую почту, то получит соответствующую ошибку. Случай такой ошибки представлен на рисунке 4.3.

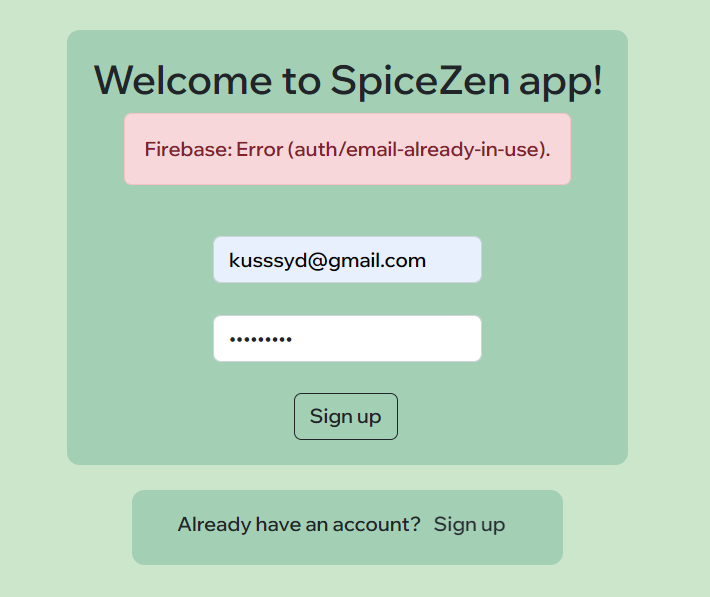


Рисунок 4.3 – Ошибка регистрации существующего пользователя

Валидация выполняется на стороне платформы Firebase, и в случае, если она не пройдена, возвращается ошибка, которая выводится на стороне клиента.

Также если пользователь попытается войти под несуществующей почтой, то получит соответствующую ошибку, которая представлена на рисунке 4.4.

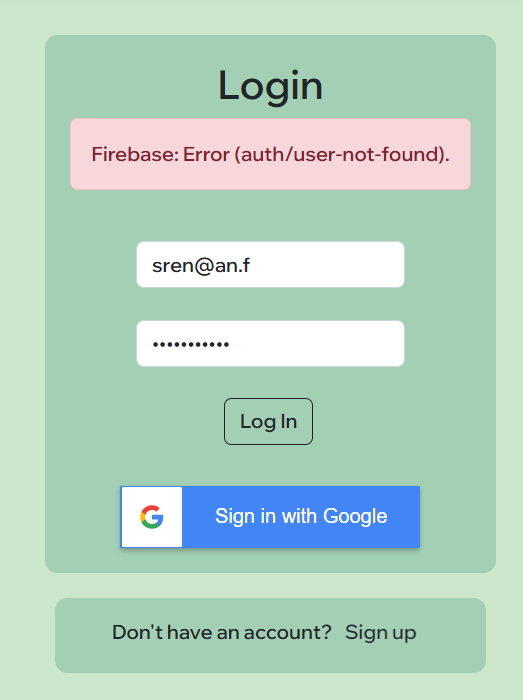


Рисунок 4.4 – Вход под несуществующим пользователем

На странице добавления блюда каждое поле валидируется. Это можно увидеть на рисунке 4.5.

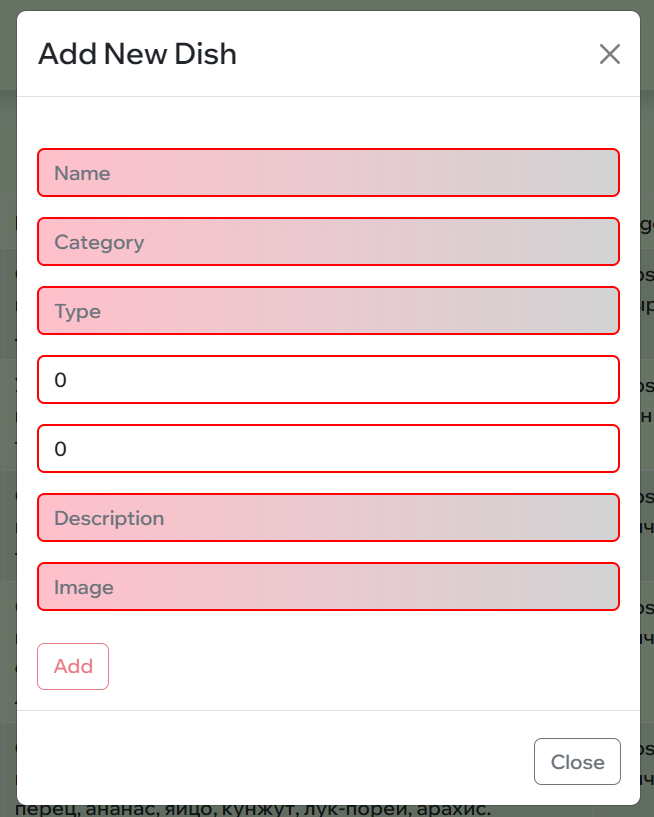


Рисунок 4.5 – Ошибка повторного создания блюда

На рисунке 4.6 представлен пример правильного заполнения информации о блюдах:

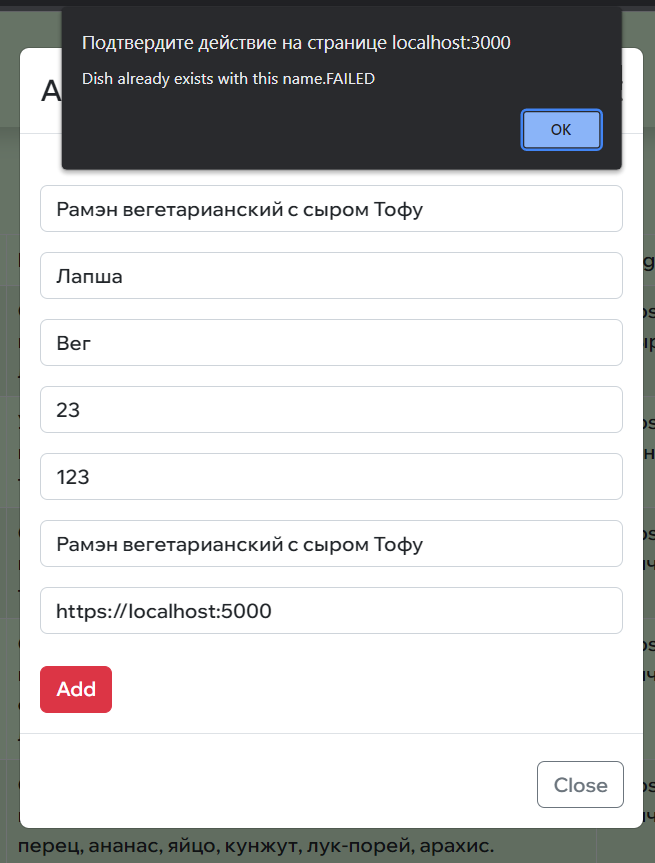


Рисунок 4.6 – Правильное заполнение информации о блюдах

В случае, если на странице с созданием представления с названием, уже существующим в базе данных, получим ошибку, которая представлена на рисунке 4.7.

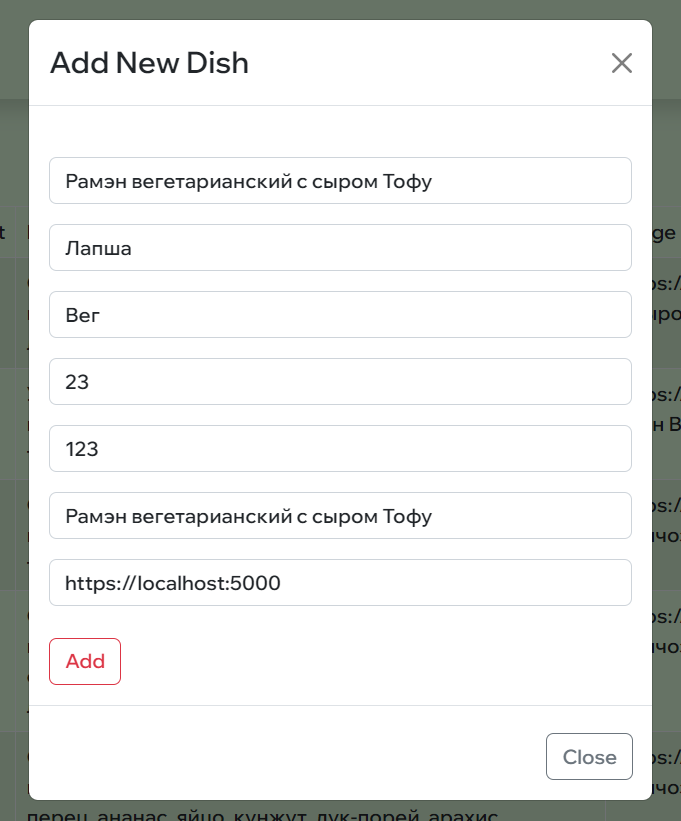


Рисунок 4.7 – Корректное заполнение полей для блюда

То же самое происходит с редактированием блюда. Примеры представлены на рисунках 4.8 – 4.9.

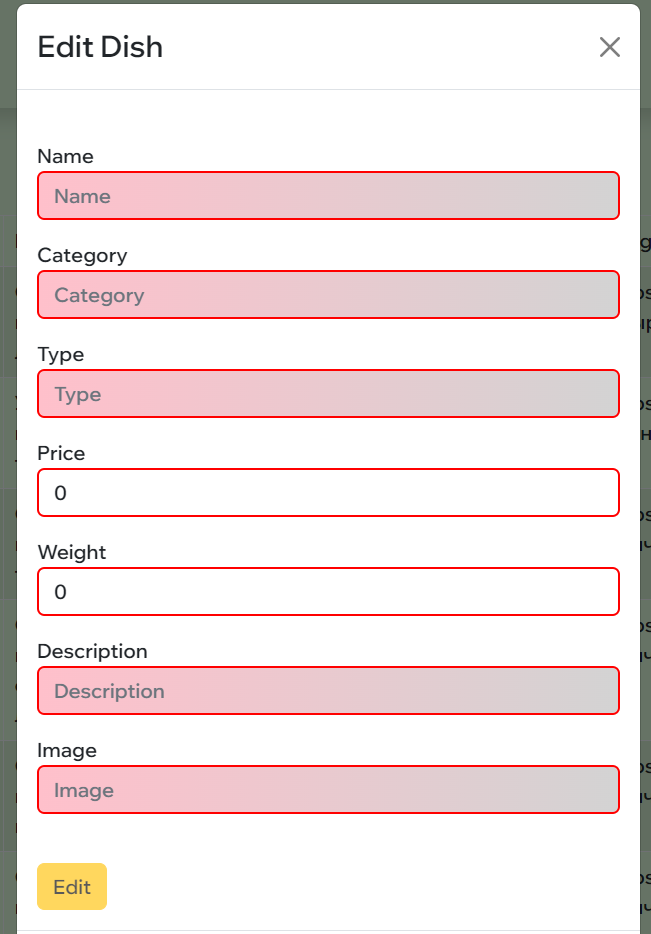


Рисунок 4.8 – Ошибки при заполнении полей при редактировании блюд

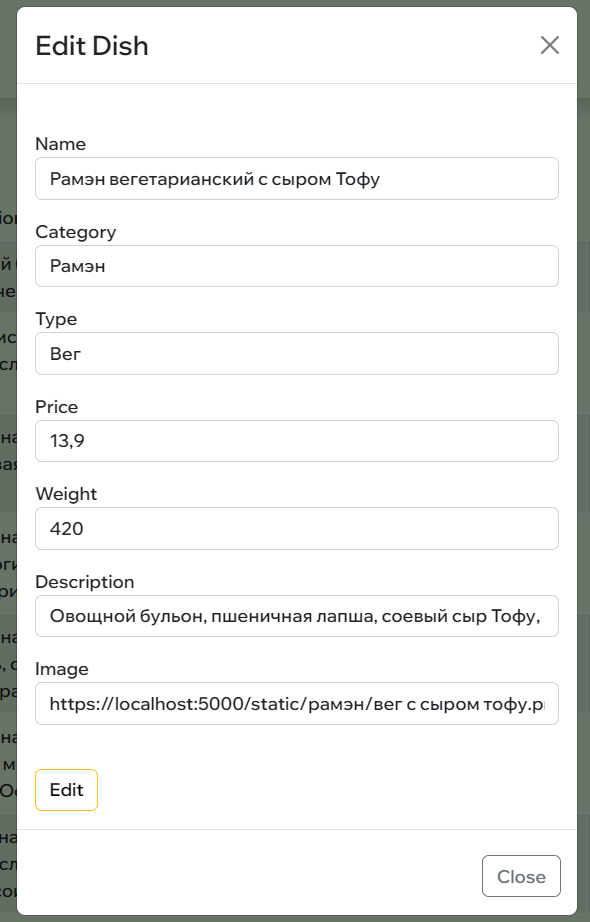


Рисунок 4.9 – Пример обновления данных о блюде с помощью корректных значений

Таким образом было проведено тестирование разрабатываемого приложения. Были продемонстрированы типичные ситуации, вызывающие ошибки и реакция на них. Результаты тестирования позволили выявить и исправить множество ошибок и проблем, связанных с функциональностью и производительностью приложения.

Однако, как и любая система, приложение нуждается в постоянном обновлении и совершенствовании. Для этого необходимо проводить регулярные тестирования, анализировать данные и отзывы пользователей, чтобы улучшать функциональность и повышать качество приложения.

5 Руководство пользователя

## 5.1 Регистрация пользователя

Для регистрации пользователя, необходимо ввести электронную почту и пароль. Форма регистрации представлена на рисунке 5.1.

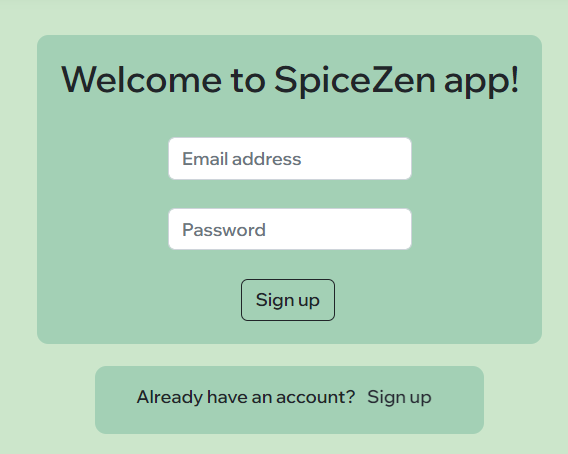


Рисунок 5.1 – Форма регистрации

После того, как пользователь нажмет на кнопку «Sign up», будет отправлен запрос на регистрацию, на стороне сервера выполнится валидация всех полей. В случае неудачной валидации, пользователь увидит сообщение об ошибке. Если данные введены верно, произойдет переадресация на главную страницу приложения. Если пользователь перейдет по ссылке «Login» в правом верхнем углу, выполнится переадресация на станицу авторизации.

## 5.2 Аутентификация и авторизация пользователя

На странице «Login» пользователь может произвести аутентификацию, заполнив все необходимые поля формы. Произойдет проверка существования данного пользователя и проверка правильности ввода пароля.

В случае успешной аутентификации пользователя, он будет перенаправлен на главную страницу приложения и у него станут доступными другие страницы приложения, такие как полное меню с категориями и возможностью добавлять блюда к предзаказу, бронирование, комментарии. Таким образом, это означает, что авторизация пользователя прошла успешно. Форма авторизации представлена на рисунке 5.2.

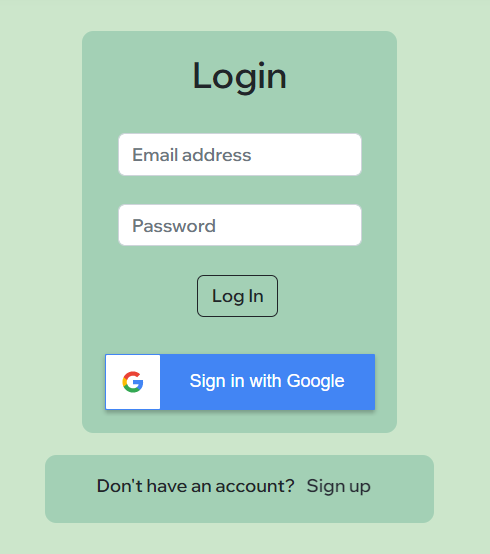


Рисунок 5.2 – Форма авторизации

Все неавторизованные пользователи – гости, авторизованные – пользователи. Администратор может быть только один и проверка, является ли пользователь администратором, осуществляется с помощью контекста Firebase. Таким образом ограничиваются возможности пользователей в приложении.

У каждого пользователя определена роль. Всего их три: гость, пользователь, администратор. Роль обычного пользователя позволяет использовать все возможности приложения. Администратору, в отличие от пользователя, дополнительно разрешено добавлять, изменять и удалять данные о блюдах, подтверждать предзаказы с бронями.

## 5.3 Блюда

При успешной авторизации пользователь попадает на главную страницу приложения, на которой находится ознакомительное содержимое меню. Однако в отличие от неавторизованного пользователя, он может добавлять блюда в корзину по кнопке. Ознакомительное меню представлено на рисунке 5.3.

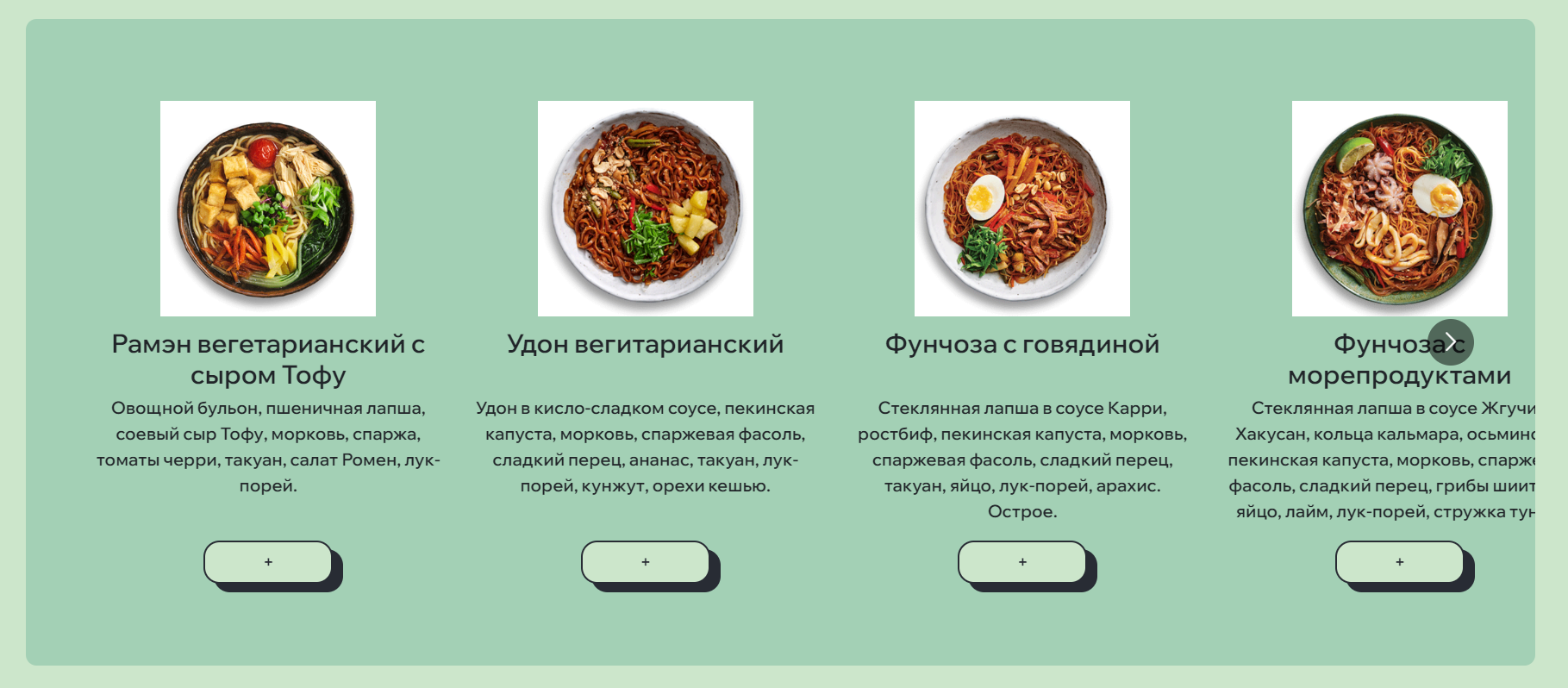


Рисунок 5.3.1 – Ознакомительное меню внизу страницы

Кроме того, теперь пользователь может увидеть полное меню с делением на категории. Полное меню представлено на рисунке 5.3.2.

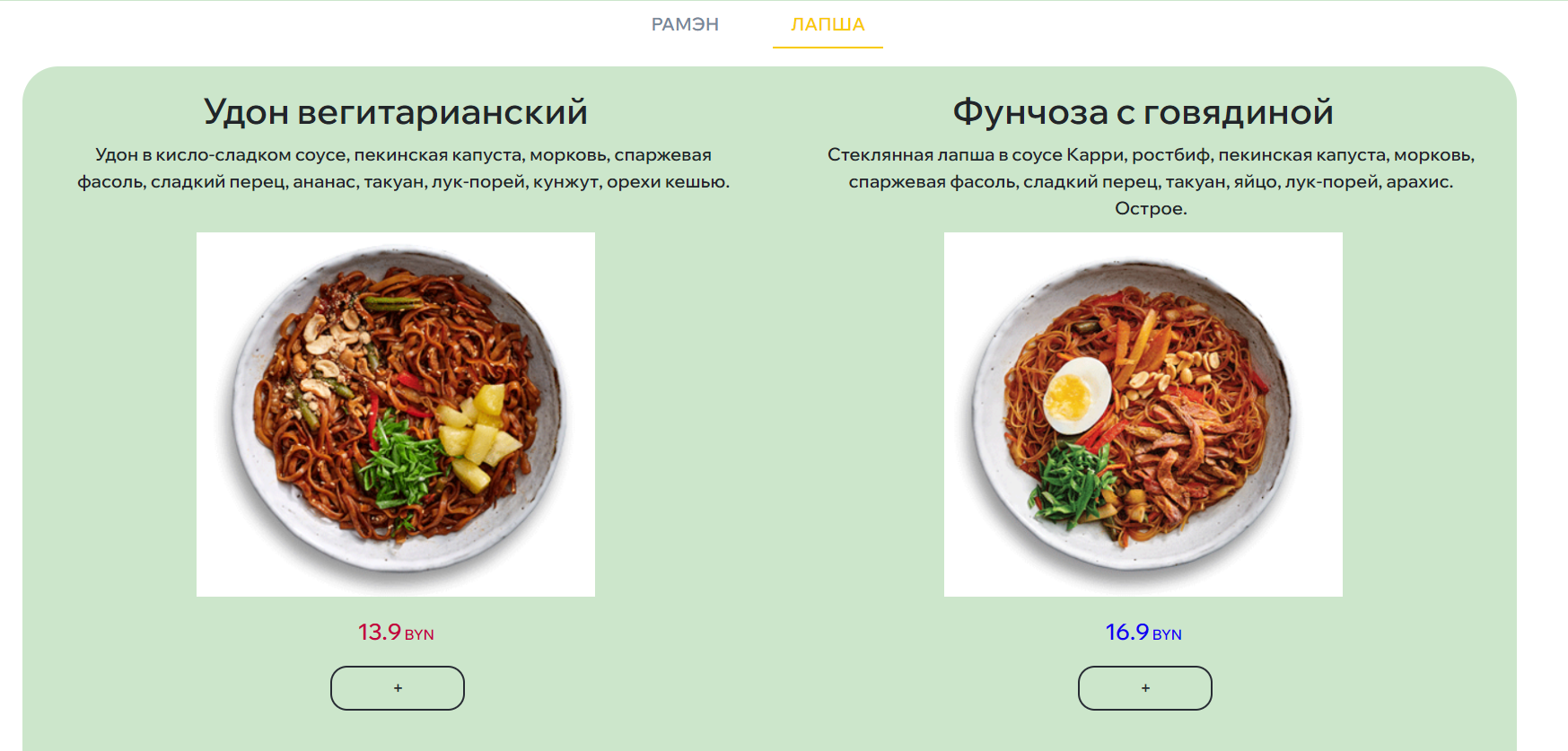


Рисунок 5.3.2 – Полное меню

## 5.4 Оформление предзаказа

Страница предзаказа содержит подробную информацию о выбранных блюдах и их количестве. Однако если ничего выбрано не будет, то выведется соответствующее сообщение.

Страница предзаказа с выбранными блюдами и с пустой корзиной представлена на рисунках 5.4.1 и 5.4.2.

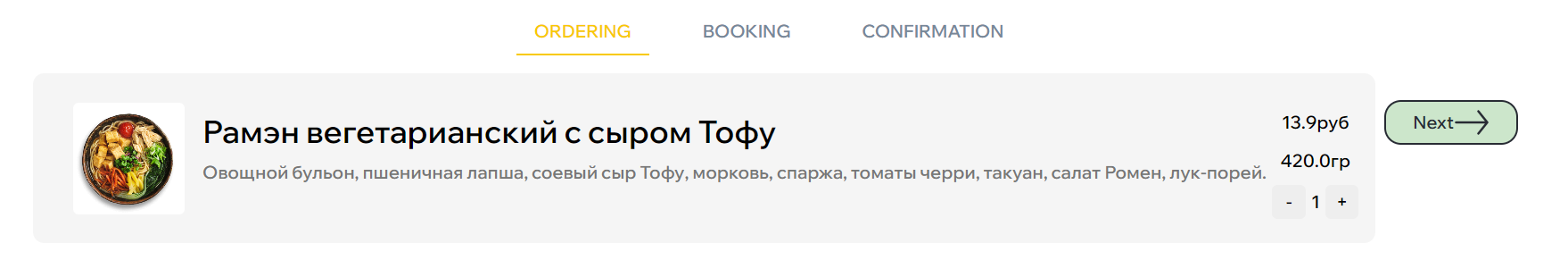


Рисунок 5.4.1 – Страница предзаказа с заполненной корзиной

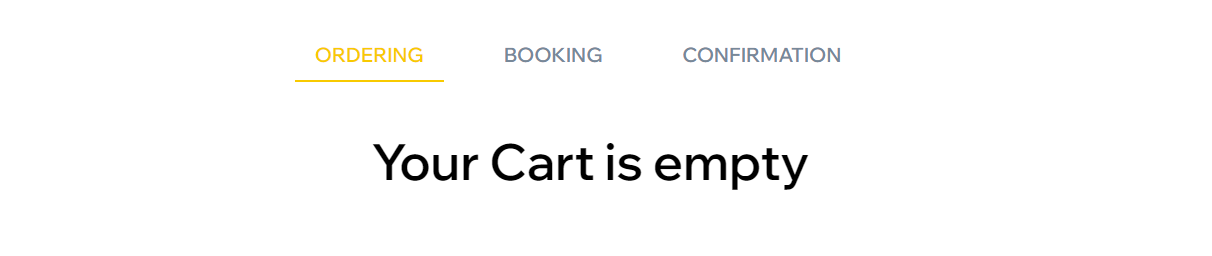


Рисунок 5.4.2 – Страница предзаказа с пустой корзиной

## 5.5 Страница бронирования

Данная страница предоставляет пользователю заполнить данные, необходимые для бронирования. Страница бронирования представлена на рисунке 5.5.

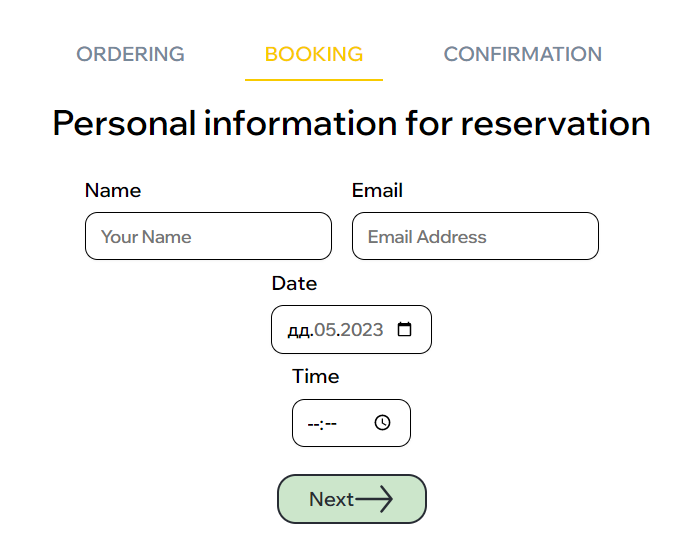


Рисунок 5.5 – Страница бронирования

## 5.6 Страница оформления предзаказа с бронью

На этой странице пользователь может нажать кнопку, чтобы оформить предзаказ и бронь. Однако если он не уверен в введённых данных, то может переключиться назад на другие страницы. Страница оформления предзаказа с бронью представлена на рисунке 5.6.1.

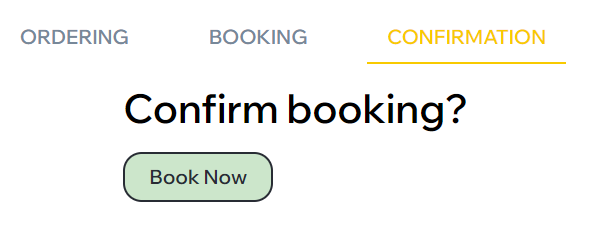


Рисунок 5.6.1 – Страница оформления предзаказа с бронью

После нажатия на кнопку пользователь будет перенаправлен на страницу, представленную на рисунке 5.6.2.



Рисунок 5.6.2 – Страница после оформления предзаказа с бронью

## 5.7 Страница с комментариями

Кроме всего прочего, на главной странице, в самом низу, есть поле, которое позволяет вписать комментарий или же вопрос, на который администратор или другие пользователи могут ответить. Это продемонстрировано на рисунке 5.7.

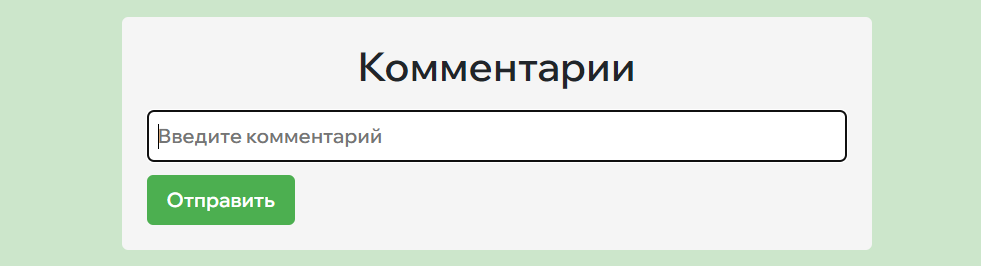


Рисунок 5.7 – Страница с комментариями

# Заключение

В ходе данного курсового проекта было разработано удобное, безопасное web-приложение ресторана.

Перед началом разработки был произведен аналитический обзор прототипов приложений подобной тематики и определение функциональных возможностей разрабатываемого приложения.

Был разработан удобный и понятный интерфейс, в котором могут разобраться даже те пользователи, которые ни разу не посещали подобные сайты.

Приложение предоставляет возможность выбрать просмотреть меню, выбрать блюда для предзаказа и оформить предзаказ с бронью.

В процессе разработки использовались современные и популярные библиотеки, модули и фреймворки, которые позволили полностью реализовать задуманный функционал. Среди них: Express, mongoose, socket.io, React, Redux и другие.

Для обеспечения безопасного обмена информации между клиентом и сервером был сгенерирован ssl-сертификат.

Кроме того, приложение взаимодействует с платформой Firebase и осуществляет с помощью него авторизацию, регистрацию.

Важным этапом разработки было тестирование программного продукта, которое позволило выявить и исправить ошибки и недостатки в работе приложения. Тестирование позволило убедиться, что программа работает верно и соответствует требованиям технического задания.

В результате разработки были выполнены все поставленные задачи, в том числе:

Для администратора:

* обеспечивать возможность авторизации;
* позволять просматривать меню;
* позволять добавлять и удалять пункты меню;
* обеспечивать возможность просматривать брони и предзаказы и подтверждать их с помощью рассылки клиентам на почту;

Для клиента:

* обеспечивать возможность регистрации и авторизации;
* позволять просматривать меню;
* обеспечивать возможность оформлять предзаказ;
* обеспечивать возможность бронирования столов на выбранную дату;
* позволять оставлять комментарии;

Для гостя:

* обеспечивать возможность регистрации;
* позволять просматривать меню.

# Список используемых источников

1. MongoDB документация [Электронный ресурс]. – Режим доступа: <https://www.mongodb.com> – Дата доступа: 25.02.2023.
2. Mongoose документация [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express\_Nodejs/mongoose#учебник\_по\_mongoose – Дата доступа: 09.03.2023
3. Firebase документация [Электронный ресурс]. – Режим доступа: <https://firebase.google.com> – Дата доступа: 15.03.2023.
4. Создание самоподписанного SSL-сертификата [Электронный ресурс]. – Режим доступа: https://support.kaspersky.com/KWTS/6.1/ru-RU/193662.htm Дата доступа: 25.03.2023.
5. Bootstrap документация [Электронный ресурс]. – Режим доступа: <https://getbootstrap.com> – Дата доступа: 20.03.2023.

# Приложение А

|  |
| --- |
| const mongoose = require("mongoose");  const Schema = mongoose.Schema;  const dishSchema = new Schema({  dishName: {type: String, required: true},  dishCategory: {type: String, required: true}, // Ссылка на категорию  dishType: {type: String, required: true}, // Ссылка на тип  dishPrice: {type: Number, required: true},  dishWeight: {type: Number, required: true},  dishDescription: {type: String, required: true},  dishImage: {type: String, required: true}  }, {  versionKey: false  });  const dishModel = mongoose.model('Dish', dishSchema);  }  const orderSchema = mongoose.Schema(  {  chosenDishes: [  {  name: {type: String, required: true},  amount: {type: Number, required: true},  price: {type: Number, required: true},  weight: {type: Number, required: true},  imageUrl: {type: String, required: true},  dish: {  type: mongoose.Schema.Types.ObjectId,  required: true,  ref: "Dish",  },  },  ],  }, {  versionKey: false  })  const orderModel = mongoose.model('Order', orderSchema);  const reservationSchema = new mongoose.Schema({  name: {type: String, required: true},  email: {type: String, required: true},  date: {type: Date, required: true},  time: {type: String, required: true}  }, {  versionKey: false  });  const reservationModel = mongoose.model('Reservation', reservationSchema);  const summarySchema = new Schema({  order: {type: Schema.Types.ObjectId, ref: "Order", default: null}, // Ссылка на категорию  reservation: {type: Schema.Types.ObjectId, ref: "Reservation", required: true},  user: {type: String, required: true},  isConfirmed:{type: Boolean, default: false},  }, {  versionKey: false  })  const summaryModel = mongoose.model('Summary', summarySchema); |

Листинг «Модели базы данных»

# Приложение Б

const express = require('express')

const bodyParser = require('body-parser')

const cors = require('cors')

const https = require('https')

const fs = require('fs')

require('socket.io');

require('dotenv').config()

const db = require('./db/index.js');

const dishRouter = require('./routes/dishRouter.js');

const bookingRouter = require('./routes/bookingRouter.js');

const summaryRouter = require('./routes/summaryRouter.js');

const app = express();

const corsOptions = {

origin: "https://localhost:3000"

};

// Создаём HTTPS-сервер с помощью сертификатов SSL

const options = {

key: fs.readFileSync('./cert/localhost-key.pem'),

cert: fs.readFileSync('./cert/localhost.pem')

}

app.use(cors(corsOptions));

app.use(bodyParser.json());

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

let httpsServer = https.createServer(options, app);

const io = require("socket.io")(httpsServer, {

cors: {

origin: "\*",

methods: ["GET", "POST"],

}

});

// Настройка маршрута для загрузки статических файлов

app.use('/static', express.static('D:\\1POIT\\3\\CW\\SpiceZen restaurant\\server\\static'));

db.on("error", console.error.bind(console, 'MongoDB connection error'));

db.once("open", () => {

console.log('Connected to DB!');

})

//console.log( io.engine.port);

// Обработчик подключения нового клиента

io.on('connection', (socket) => {

console.log(`Пользователь подключился`);

// Обработчик получения нового комментария от клиента

socket.on('newComment', (comment) => {

// Сохранение комментария в базе данных или выполнение другой логики

console.log('Новый комментарий:', comment);

// Отправка нового комментария всем клиентам, подключенным к сокету

io.emit('newComment', `${comment}`);

let comm = {

comment: comment,

}

db.collection('comments').insertOne(comm)

});

});

app.get('/', (req, res) => {

res.json({message: 'Welcome'});

});

// ROUTERS

app.use('/api/', bookingRouter);

app.use('/api/', dishRouter);

app.use('/api/', summaryRouter);

const PORT = process.env.PORT || 5000;

// app.listen(PORT, () => {

// console.log(`Server listening on port ${PORT}`);

// })

httpsServer.listen(PORT, () => {

const {address, port} = httpsServer.address();

console.log(`Socket is listening on ${address}:${port}`);

})

Файл index.js

# Приложение Г

|  |
| --- |
| {  "name": "server",  "version": "1.0.0",  "description": "",  "main": "index.js",  "scripts": {  "test": "echo \"Error: no test specified\" && exit 1",  "start": "node index.js",  "create-data": "node data/generateFakeData.js"  },  "author": "",  "license": "ISC",  "dependencies": {  "body-parser": "^1.20.2",  "cookie-parser": "^1.4.6",  "cors": "^2.8.5",  "dotenv": "^16.0.3",  "express": "^4.18.2",  "lodash": "^4.17.21",  "mongodb": "^5.4.0",  "mongoose": "^7.1.1",  "nodemailer": "^6.9.2",  "nodemon": "^2.0.22",  "socket.io": "^4.6.1"  }  } |

Файл package.json