|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОЙ РАБОТЕ***

***НА ТЕМУ:***

***Информационный портал «Наставник»***

Студент \_\_ИУ6-52Б\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_Д.В.Лабзунова\_\_\_\_\_\_**

(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

*2021 г.*

Вставить задание на курсовую

**Реферат**

Расчетно-пояснительная записка 32 страницы, 3 части, 16 рисунков, 3 таблицы, 5 источников, 1 приложение.

ВЕБ-СЕРВИС, ВЕБ-ПРИЛОЖЕНИЕ, ДОСТАВКА, ЕДА, РЕСТОРАН, КАФЕ.

Объектом разработки является информационный портал «Наставник».

Цель работы – проектирование и реализация информационного портала, используемого для поиска и общения студентов младших курсов с наставниками, являющимися учащимися старших курсов той же кафедры.

В результате работы был спроектирован и реализован программный комплекс, состоящий из веб-приложения и чат-бота в мессенджере telegram, позволяющий выбрать оптимальный для пользователя формат взаимодействия с информационным порталом. А также было проведено тестирование программного продукта методами белого и черного ящика и юзабилити-тест.

Пользователями данного приложения могут быть студенты ВУЗа, на базе которого развернуто приложение, если данные студенты заинтересованы в поиске наставника, либо же сами хотят быть наставниками.

**Содержание**

[**Введение** 6](#_Toc1)

[1. **Анализ требований и уточнение спецификаций** 7](#_Toc2)

[1.1. **Анализ задания и выбор технологии, языка и среды разработки** 7](#_Toc3)

[1.2. **Разработка диаграммы вариантов использования** 8](#_Toc4)

[1.3. **Анализ хранимой информации и выбор способа ее хранения** 10](#_Toc5)

[1.4. **Разработка диаграмм деятельности** 13](#_Toc6)

[2. **Проектирование структуры и компонентов программного продукта** 18](#_Toc7)

[2.1. **Разработка интерфейса пользователя** 18](#_Toc8)

[2.1.1. **Посторонние диаграммы состояний интерфейса** 18](#_Toc9)

[2.1.2. **Разработка форм интерфейса** 20](#_Toc10)

[2.2. **Разработка структурной схемы программного продукта** 25](#_Toc11)

[3. **Выбор стратегии тестирования и разработка тестов** 26](#_Toc12)

[3.1. **Тестирование структурным контролем** 26](#_Toc13)

[3.2. **Unit-тесты** 28](#_Toc14)

[3.3. **Usability-тест** 28](#_Toc15)

[**Заключение** 30](#_Toc16)

[**Список используемых источников** 31](#_Toc17)

[**Приложение А. Техническое задание.** 32](#_Toc18)

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

ТЗ – техническое задание.

ООП – объектно-ориентированное программирование.

БД – база данных.

СУБД – система управления базами данных.

Gо – Golang, язык программирования, который был разработан корпорацией Google.

JS – JavaScript, язык, обычно используемый как встраиваемый для программного доступа к объектам приложений.

UI – User Interface, (дословно «пользовательский интерфейс») — то, как выглядит интерфейс и то, какие физические характеристики приобретает.

Фреймворк — программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Фронтенд (англ. front-end) — клиентская сторона пользовательского интерфейса к программно-аппаратной части сервиса.

Бэкенд (англ. back-end) — программно-аппаратная часть сервиса, отвечающая за функционирование его внутренней части.

Презентер (англ. presenter) — объект, который управляет отображением данных.

CI/CD – (continuous integration/continuous delivery) -  это комбинация непрерывной интеграции (continuous integration) и непрерывного развертывания программного обеспечения.

Model-View-Controller (MVC, «Модель-Представление-Контроллер») — схема разделения данных приложения, и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель (Model) — компонент схемы MVC, который предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

Представление (View) — компонент схемы MVC, который отвечает за отображение данных модели пользователю, реагируя на изменения модели[1].

Контроллер (Controller) — компонент схемы MVC, который интерпретирует действия пользователя, оповещая модель о необходимости изменений[1].

# **Введение**

Данная работа посвящена проектированию и разработке информационного портала поиска наставников по учебе, представляющего собой программный комплекс из веб-приложение, ориентированное на работу на ПК и ноутбуках (в браузерах с большим размером и альбомной ориентацией окна) и чат-бота, предназначенного для взаимодействия через мобильное приложение “Telegram”. Разрабатываемый информационный портал может быть использован студентами, заинтересованными в участии в программе наставничества. Данный сервис позволяет пользователю выбрать наставника, являющегося студентом более старших курсов той же кафедры, что и пользователь, самому стать наставником, управлять системой заявок в “друзья” и осуществлять поиск верифицированных наставников (т.е. студентов, чья личность подтверждена модератором на основании документов).

В ходе выполнения работы в качестве аналогов были рассмотрены чат-боты, предназначенные для взаимодействия и обучения сотрудников внутри той или иной компании, а также сервисы знакомств. Найти детальное описание более функционально близких аналогов в свободном доступе не удалось.

Актуальность разработки заключается в том, что в рассмотренных аналогах адаптация под использования в целях реализации программы наставничества либо невозможно в принципе, либо крайне затруднительна и малоэффективна.

# **Анализ требований и уточнение спецификаций**

## **Анализ задания и выбор технологии, языка и среды разработки**

Методологией программирования для чат-бота было выбрано событийное программирование и модульный подход. Событийное программирование лежит в основе абсолютного большинства современных чат-ботов, так как оно позволяет реагировать на приход любого нового сообщения или обратного вызова как на атомарное событие, для которого вызывается соответствующий содержанию обработчик. Модульный подход же позволяет разбить обработчики событий на модули в соответствии с иерархией меню, что упрощает проектирование и работу с кодом программы. Для разработки веб-приложения выбрана парадигма Model-View-Controller, являющаяся наиболее популярной и устоявшейся в сфере веб-программирования.

Для разработки как чат-бота, так и веб-приложения был использован язык Python. В случае чат-бота Python был выбран как наиболее популярный язык для написания чат-ботов, имеющий соответствующие библиотеки (в случае данной работы - aiogram) и документацию к ним. В случае веб-приложения был выбран фреймворк Django, как широкоиспользуемый MVC-фреймворк с качественной документацией, позволяющий писать код быстро и компактно в силу динамической типизации и других особенностей языка Python.

В качестве редактора исходного кода был выбран Visual Studio Code, позволяющий установить необходимые расширения как для работы с Python, так и для разработки представлений в веб-приложении, т.е. работы с HTML, CSS, JavaScript. Вместе с необходимыми расширениями VS Code позволяет использовать функции автодополнения кода, автоформатирования, подсказки параметров функций, обнаружение ошибок синтаксиса.

В качестве СУБД был выбран PostgreSQL за ее скорость, свободность распространение, качество документации и наличие библиотек для интеграции с Python.

**1.2. Разработка инфологической модели базы данных**

Разрабатываемая система работает с учетными данными пользователей, а так же с их заявками в “друзья” к потенциальным кураторам. Кроме того, необходимо хранить информацию о количестве кафедр на факультете для контроля вводимых при регистрации значений. Также система работает с фото пользователей, что, в связи со спецификой фреймворка Django, требует отдельной таблицы с информацией о месте хранения фото. На рисунке N приведена инфологическая модель базы данных.

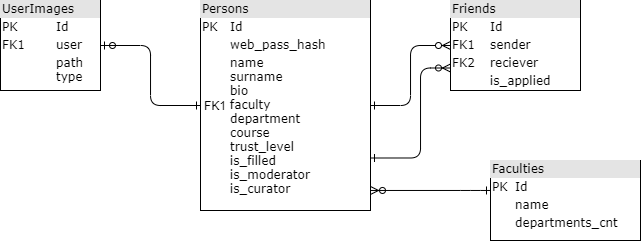


Рисунок N - Инфологическая модель базы данных

В базе данных присутствуют следующие таблицы и поля:

1. Persons - учетные данные пользователей:

* id - идентификатор пользователя;
* web\_pass\_hash - хэши паролей пользователей для web-приложения;
* name - имя пользователя;
* surname - фамилия пользователя;
* bio - поле “о себе”;
* faculty - идентификатор факультета;
* department - кафедра;
* course - год обучения;
* trust\_level - статус верификации;
* is\_filled - флаг завершенности регистрации;
* is\_moderator - флаг модератора;
* is\_curator - флаг куратора;

2) Faculties - данные о факультетах:

* id - идентификатор факультета;
* name - название факультета;
* departments\_cnt - число кафедр на факультете;

3) Friends - таблица заявок в друзья:

* id - номер заявки;
* sender - идентификатор отправителя заявки;
* reciever - идентификатор получателя заявки;
* is\_applied - флаг одобрения заявки;

4) UserImages - служебная информация о картинках:

* id - номер картинки;
* user - идентификатор хозяина картинки;
* path - путь к картинке;
* type - тип картинки;

## **Разработка диаграммы вариантов использования**

Первый шаг проектирования веб-сервиса – определение основных вариантов его использования. Для этого была разработана диаграмма вариантов использования, отображающая основные варианты взаимодействия приложения и пользователя.

После анализа ТЗ были получены следующие варианты взаимодействия пользователя и приложения:

* авторизация;
* выставление фильтров для поиска подходящего заведения;
* добавление в корзину блюд;
* оформление заказа;
* отслеживание информации о заказе;
* общение с рестораном в чате.

Для владельцев ресторанов:

* авторизация;
* добавление блюд;
* управление заказами;
* общение с заказчиками в чате.

Рассмотрим самый распространенный вариант использования сервиса.

Таблица 1 – описание вариантов использования ***Оформление заказа***

|  |  |
| --- | --- |
| Название варианта | Оформление заказа |
| Цель | Оформить заказ на доставку еды с доставкой на дом |
| Действующие лица | Пользователь |
| Краткое описание | Пользователь добавляет в корзину блюда, заполняет данные о себе, делает заказ, отслеживает его статус. |
| Тип | Основной |

Таблица 2 – Вариант использования ***Оформление заказа***

|  |  |
| --- | --- |
| Действие пользователя | Отклик системы |
| 1. Пользователь авторизуется   Пользователь добавляет в корзину блюда.  3. Пользователь добавляет блюда в корзину.  5. Пользователь заполняет данные о себе, оформляет заказ.  7. Пользователь заходит на страницу «мои заказы», следит за их статусом. | 1. Отправляется запрос на авторизацию, пользователю присваивается сессия, записываемая в cookie.   4. Сервис запоминает предметы в корзине пользователя.  6. Заказ сохраняется в базе данных. Теперь он доступен к просмотру рестораном.  8. Сервис предоставляет информацию о текущем статусе заказа и отправляет уведомления о его смене. |

Разработанная диаграмма вариантов использования представлена на рисунке 1.

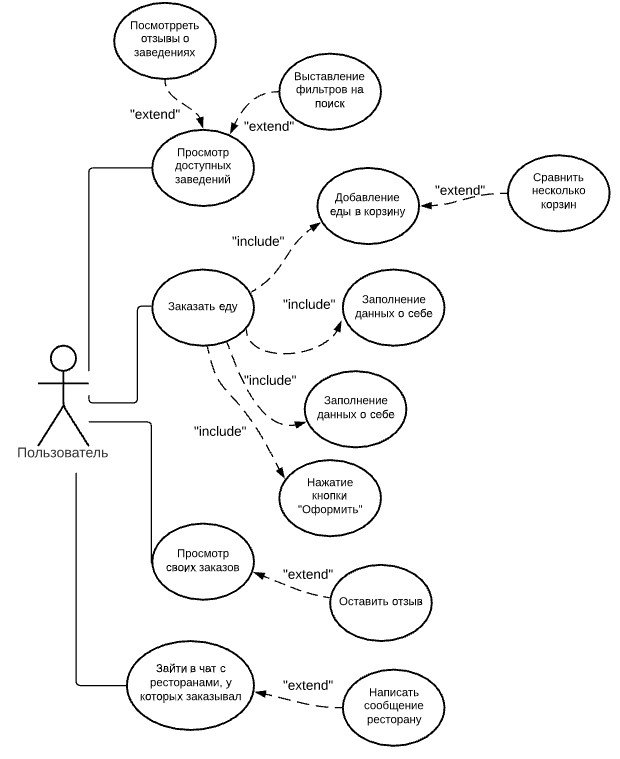


Рисунок 1 – Диаграмма вариантов использования

## **Анализ хранимой информации и выбор способа ее хранения**

При анализе предметной области и технического задания была выявлена следующая информация для хранения:

* пользователь:
  + адрес электронной почты;
  + телефон;
  + пароль;
  + фото;
  + адрес;
  + имя.
* ресторан:
  + описание;
  + стоимость доставки;
  + категория ресторана;
  + адрес электронной почты владельца;
  + телефон владельца;
  + пароль;
  + логотип;
  + адрес;
  + средняя оценка;
  + секции с едой.
* блюдо:
  + название;
  + описание;
  + изображение;
  + цена.
* адрес:
  + текстовый адрес;
  + широта;
  + долгота.
* заказ:
  + ресторан;
  + заказчик;
  + список блюд;
  + статус;
  + сумма;
  + время доставки;
  + время оформления заказа.
* корзина:
  + ресторан;
  + заказчик;
  + список блюд.

В данном списке информация уже сгруппирована по смысловым группам, на основе которых будут строиться таблицы в базе данных.

Для ускорения работы и нормализации базы данных были добавлены также связывающие таблицы блюдо-корзина, корзина-заказ и корзина-пользователь. Когда пользователь набирает блюда в корзину, блюдо добавляется в таблицу блюдо-корзина. Если пользователь сделал заказ, то корзина «открепляется» от него путем удаления ее из строки корзина-пользователь и «прилепляется» к заказу путем добавления ее в таблицу корзина-заказ.

Схема базы данных приведена на рисунке номер 2.

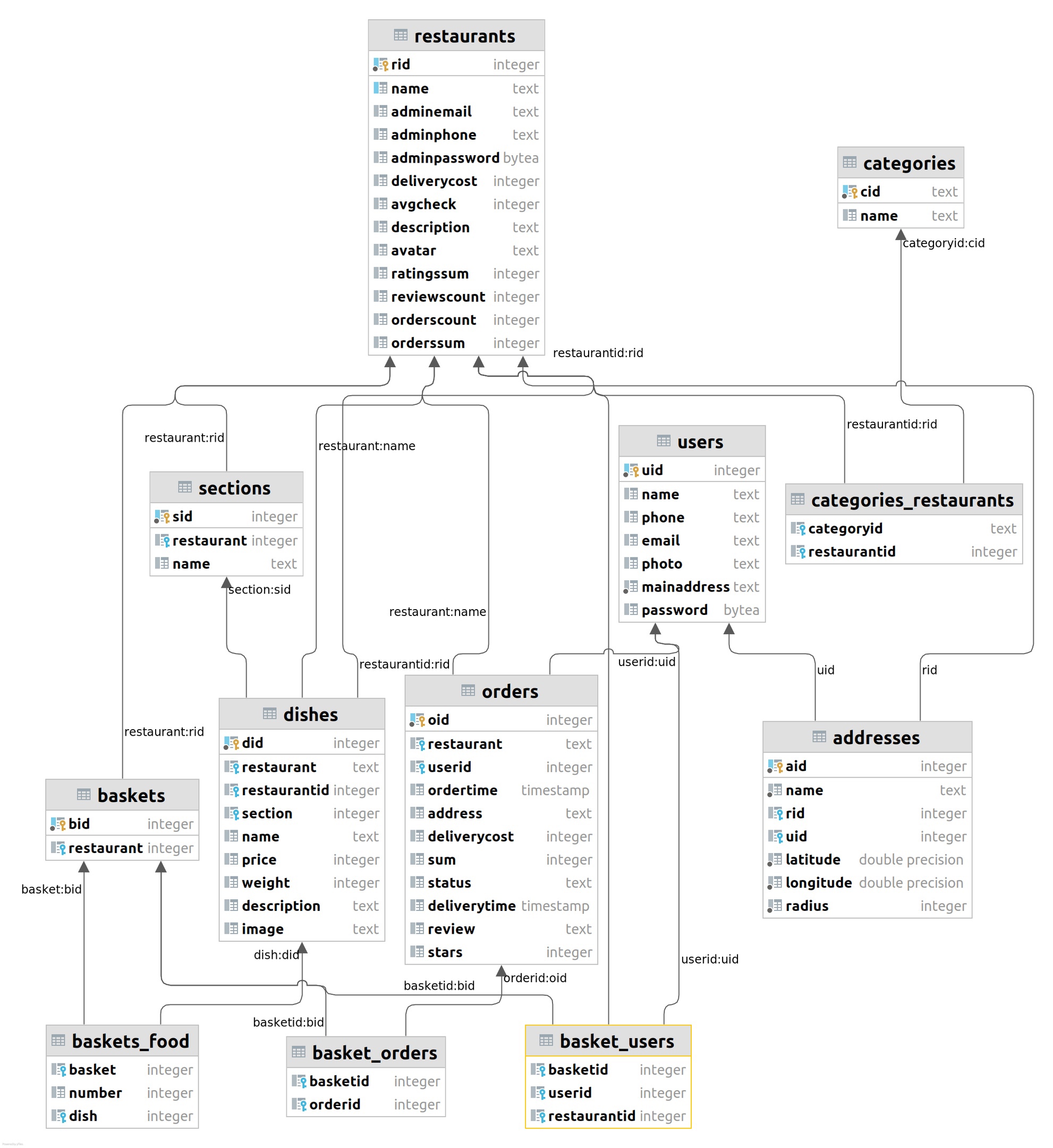


Рисунок 2 – Схема базы данных.

## **Разработка диаграмм деятельности**

Для подробного описания взаимодействия пользователя и сервиса были разработаны диаграммы деятельности для таких процессов, как:

* заказ пиццы в ресторане со средним чеком до 2000р;
* создание своего ресторана и наполнение его информацией.

Диаграмма деятельности для заказа по заданным параметрам приведена на рисунке 3. Она показывает, что пользователь должен авторизоваться и отметить свой адрес для создания заказа. Ввод данных сопровождается валидацией. Далее пользователь может выбрать категорию кухни и выставить интересующий его фильтр (в данном случае – средний чек). При оформлении заказа, если телефон пользователя и адрес были введены ранее, то они заполнятся автоматически. Если пользователь хочет исправить эти данные, то результат его ввода пройдет валидацию. В случае невалидных данных пользователю выведется уведомление о неверно указанных данных, и кнопка «оформить заказ» будет недоступна. После успешного ввода данных процесс создания заказа завершается.

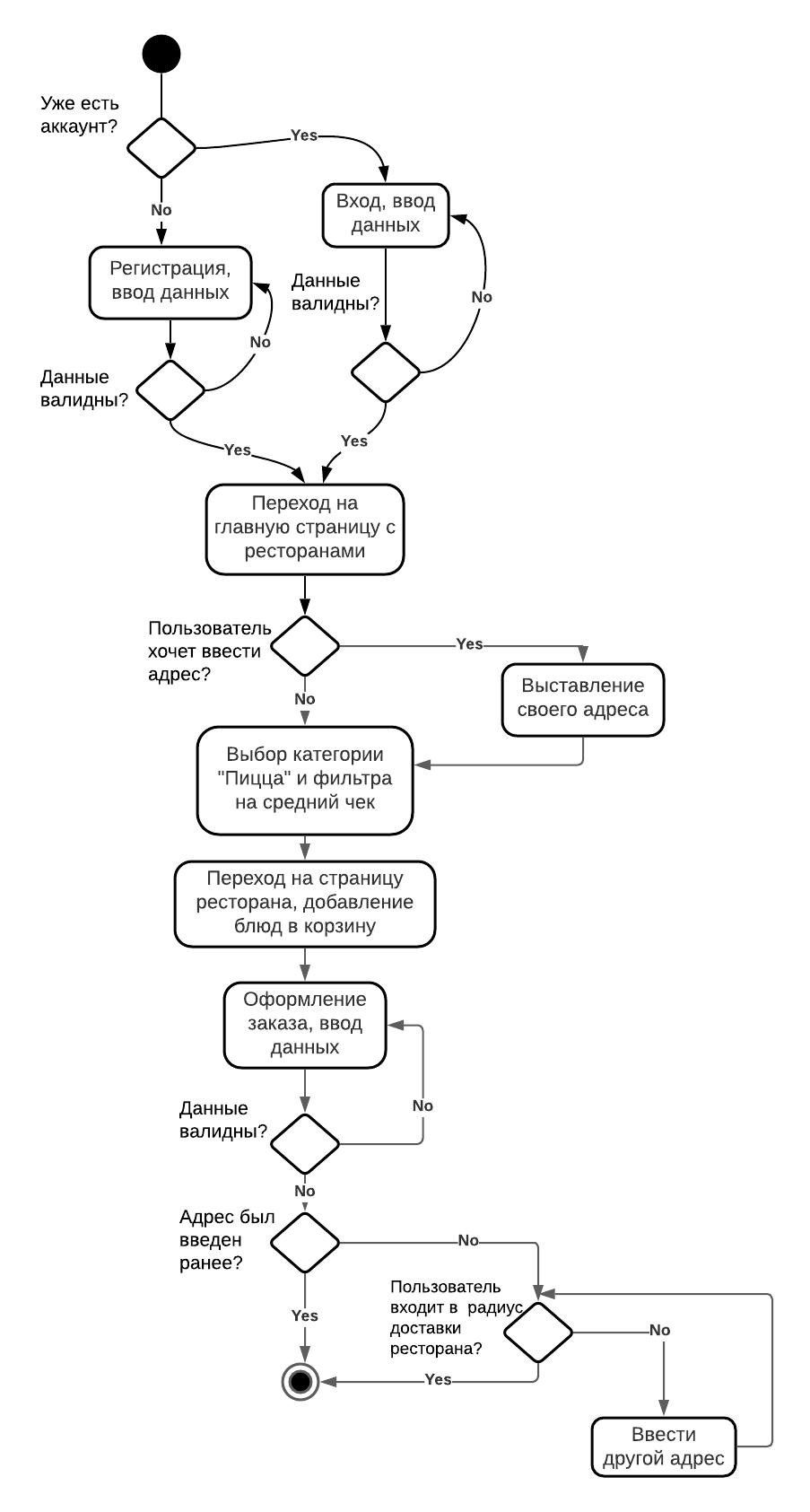


Рисунок 3 – Диаграмма деятельности для создания заказа с параметрами «Пицца из ресторана со средним чеком до 2000р»

Если пользователь не введет адрес до выбора ресторана, то может оказаться так, что он находится вне радиуса доставки ресторана, и тогда ему придется или ввести другой адрес, или выбрать другой ресторан. Радиус доставки ресторана указывает сам ресторан при регистрации.

Диаграмма деятельности по созданию своего ресторана и наполнению его информацией приведена на рисунке 4. Под «наполнением информацией» имеется в виду добавление категорий блюд и самих блюд.

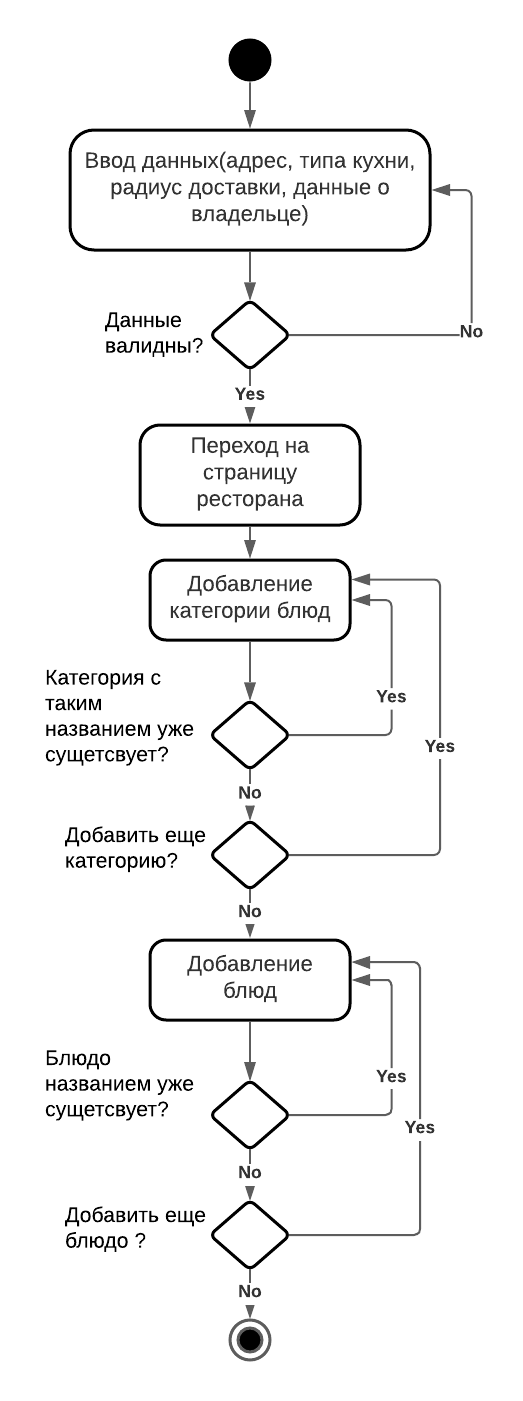


Рисунок 4 – Диаграмма деятельности для создания ресторана

Ввод данных о ресторане сопровождается валидацией и уведомлением о невалидных данных. Секций и блюд можно добавлять неограниченное количество, при этом их названия не должны повторяться в рамках одного ресторана. Для блюд можно добавить также фото и описание.

# **Проектирование структуры и компонентов программного продукта**

## **Разработка интерфейса пользователя**

В ходе разработки интерфейса были соблюдены определенные правила, сформулированные с целью того, чтобы интерфейс был понятным и простым для пользователя.

*Интерфейс должен быть последовательным.* Все возможные пользовательские действия должны быть логичными и уместными, и должны соответствовать диаграммам последовательности действий.

*Должны присутствовать определенные «паттерны», к которым пользователи привыкли в силу того, что они используются повсеместно*. Например, сверху страницы должна быть шапка так называемого «навбара», на которой находится логотип, авторизация и переход в профиль.

*Интерфейс не должен быть «шумным»*. То есть не должно быть слишком много всего, чтобы пользователь с легкостью находил нужный функционал.

*Ошибки валидации, а также любые другие ошибки должны быть понятно описаны* пользователю, чтобы было понятно, что делать, когда возникла ошибка.

*Легкая возможность вернуться назад и отменить свои действия.* Эта функция уменьшает беспокойство, поскольку пользователь знает, что ошибки можно отменить. Легкое изменение действий стимулирует изучение незнакомых вариантов. Единицами обратимости могут быть одно действие, ввод данных или полная группа действий.

### **Посторонние диаграммы состояний интерфейса**

Для описания возможных действий в программном продукте была составлена диаграмма состояний интерфейса, представленная на рисунке 5.



Рисунок 5 – Диаграмма состояний интерфейса

С1 – нажатие на логотип приложения. Переход на главную страницу.

С2 – нажатие кнопки «Вход». Переход на авторизацию.

С3 – нажатие войти/зарегистрироваться. Успешная авторизация.

С4 – нажатие на «я тут впервые». Переход на регистрацию.

С5 – нажатие на «у меня нет ресторана». Переход на авторизацию от имени .пользователя.

С6 – нажатие на «войти как владелец ресторана». Переход на авторизацию от имени ресторатора.

С7 – Нажатие на «Чаты».

С8 – Нажатие на «Заказы».

С9 – Нажатие на «Профиль».

С10 – Нажатие на конкретный чат.

С11 – Нажатие на кнопку добавления секций.

С13 – Нажатие на кнопку добавления блюд.

С14 – Нажатие на корзину.

С15 – Нажатие на «Оформить заказ».

С16 – Нажатие на «Отзывы».

С17 – Нажатие вне зоны окна с отзывами.

С18 – Нажатие на ресторан.

### **Разработка форм интерфейса**

Интерфейс, как было написано выше, должен быть удобным и понятным. Также он не должен содержать слишком маленьких элементов или слишком много цветов. Разработанные страницы показаны на рисунках 6-15.

Главная страница содержит в себе список ресторанов и фильтры для более точного поиска. Также сверху – header, в котором можно указать адрес, перейти в корзину и профиль, если пользователь авторизован, или перейти на авторизацию, если еще нет.



Рисунок 6 – Главная страница

Форма авторизации обычного пользователя и ресторана выглядят аналогично, а формы регистрации пользователя и ресторана отличаются вводимыми параметрами.

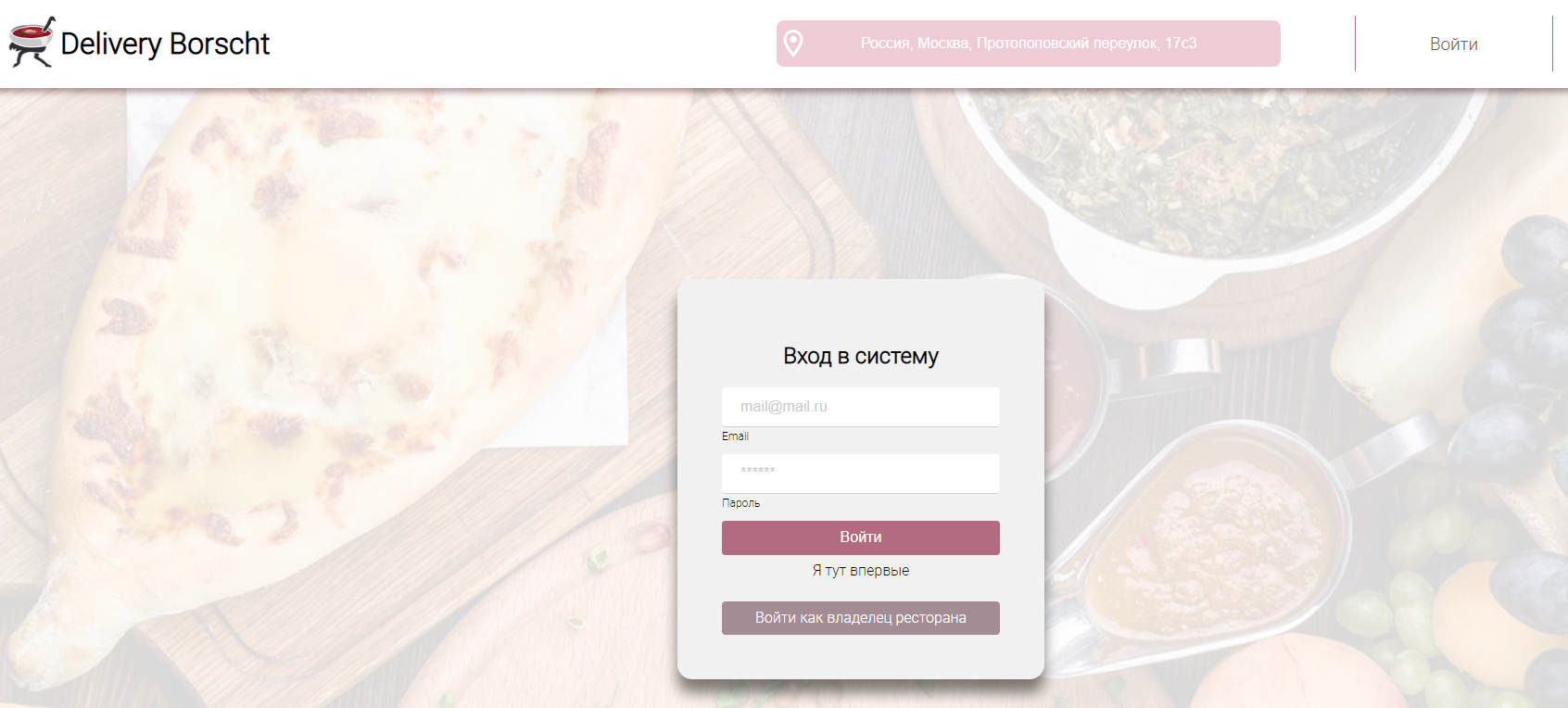


Рисунок 7 – Форма входа

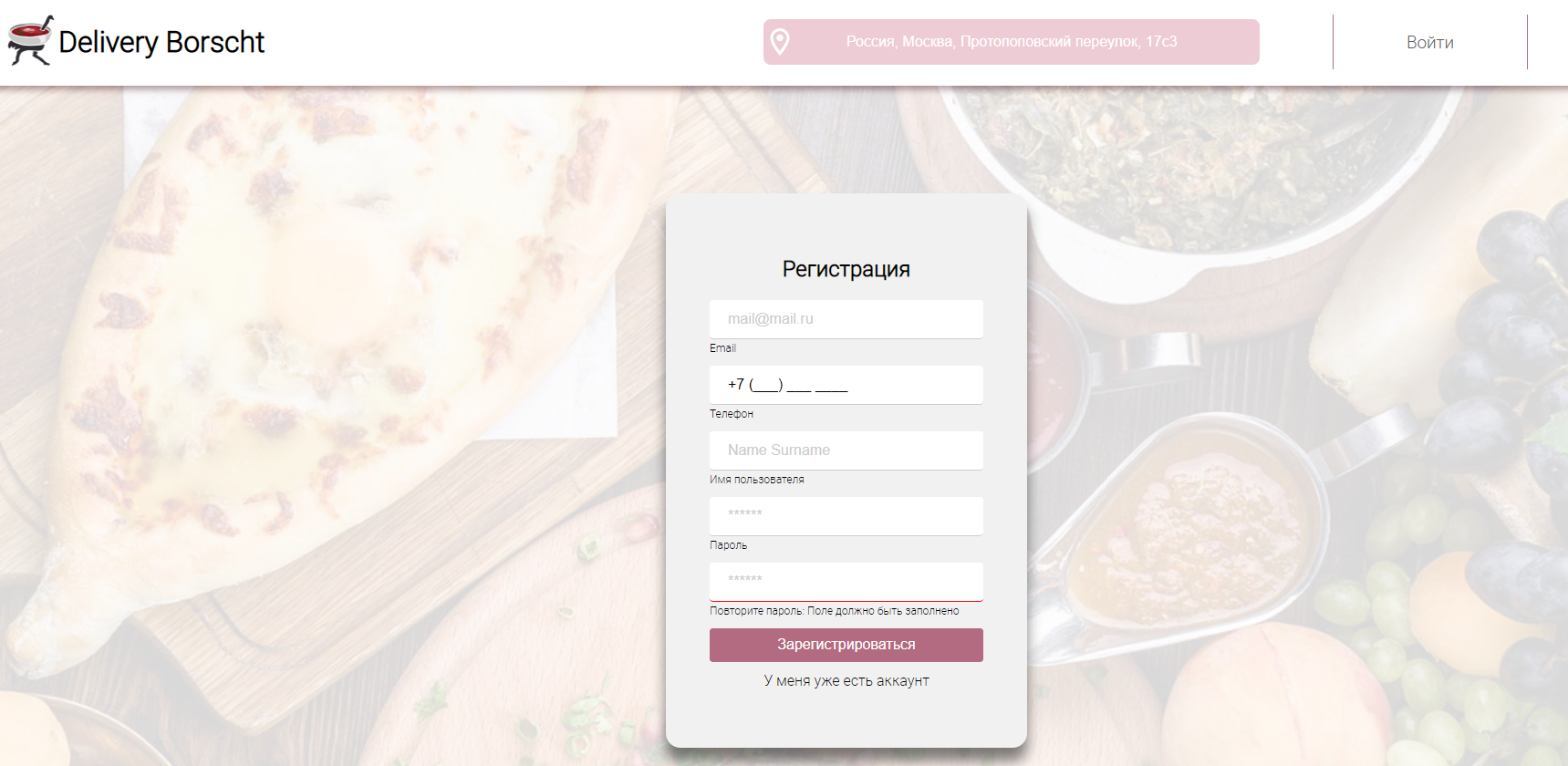


Рисунок 8 – Форма регистрации пользователя

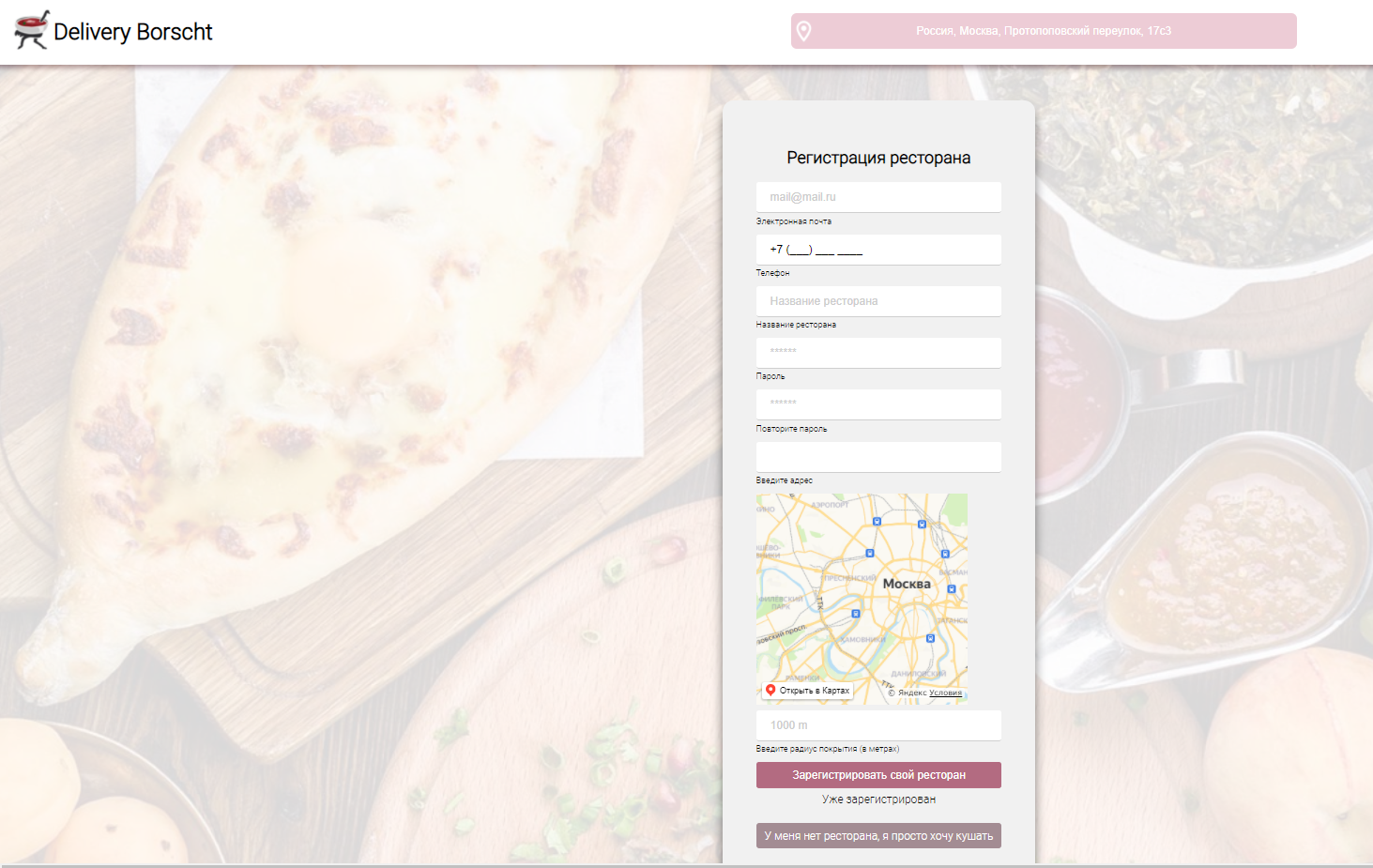


Рисунок 9 – Форма регистрации ресторана

Страница ресторана содержит название ресторана, его рейтинг, секции меню, само меню, корзину, стоимость доставки.

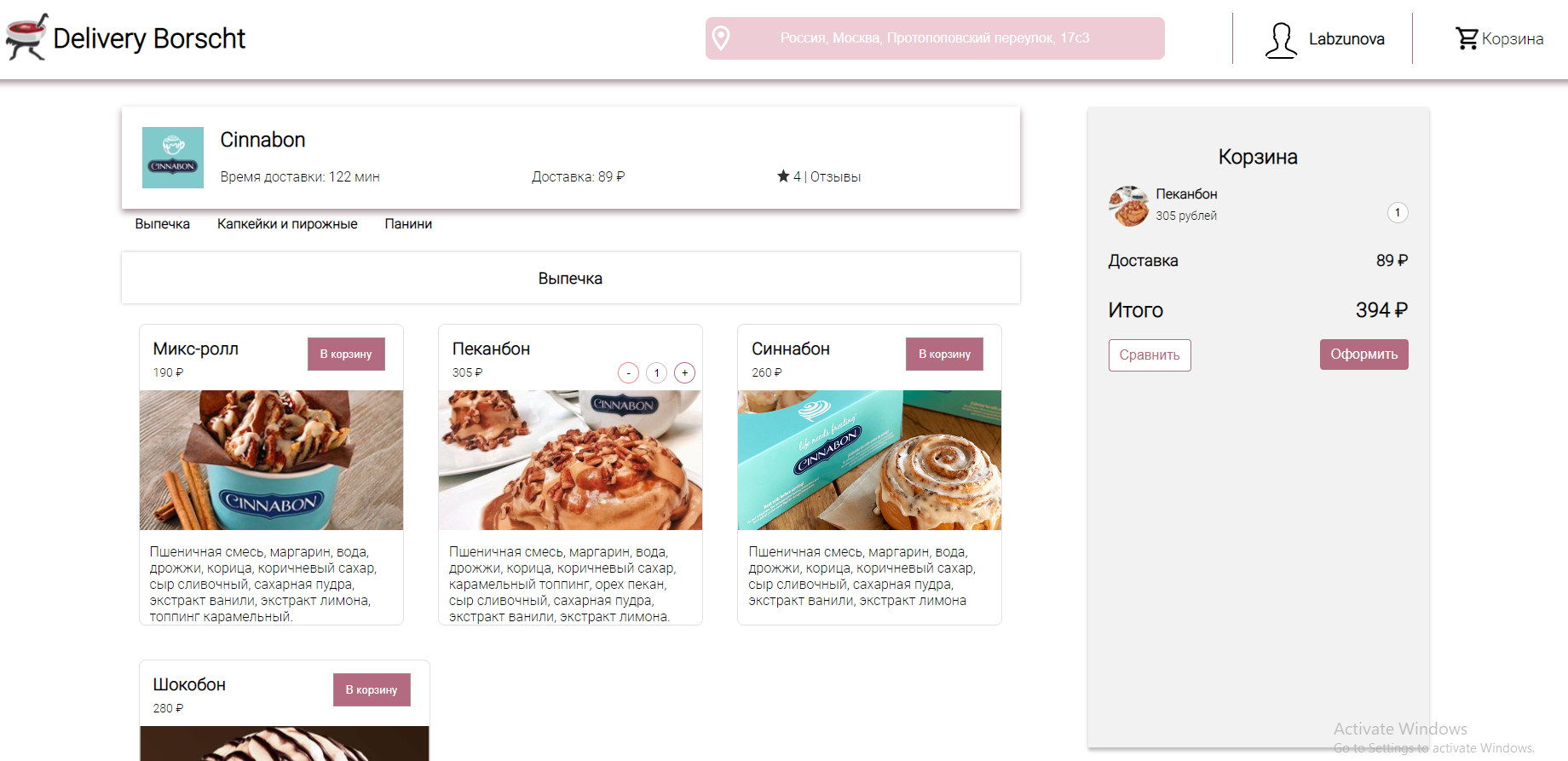


Рисунок 10 – Страница ресторана

При нажатии на «Отзывы» можно почитать отзывы.

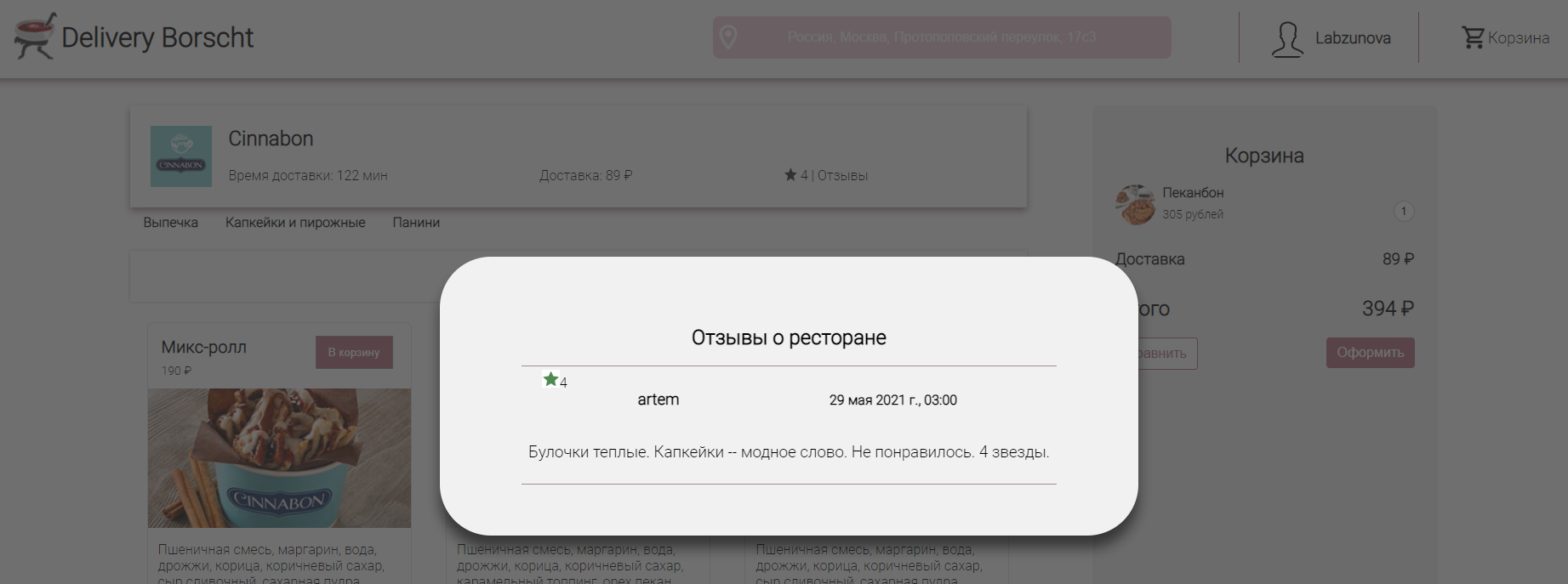


Рисунок 11 – Отзывы о ресторане

При нажатии «оформить» в корзине пользователь попадает на страницу оформления заказа.

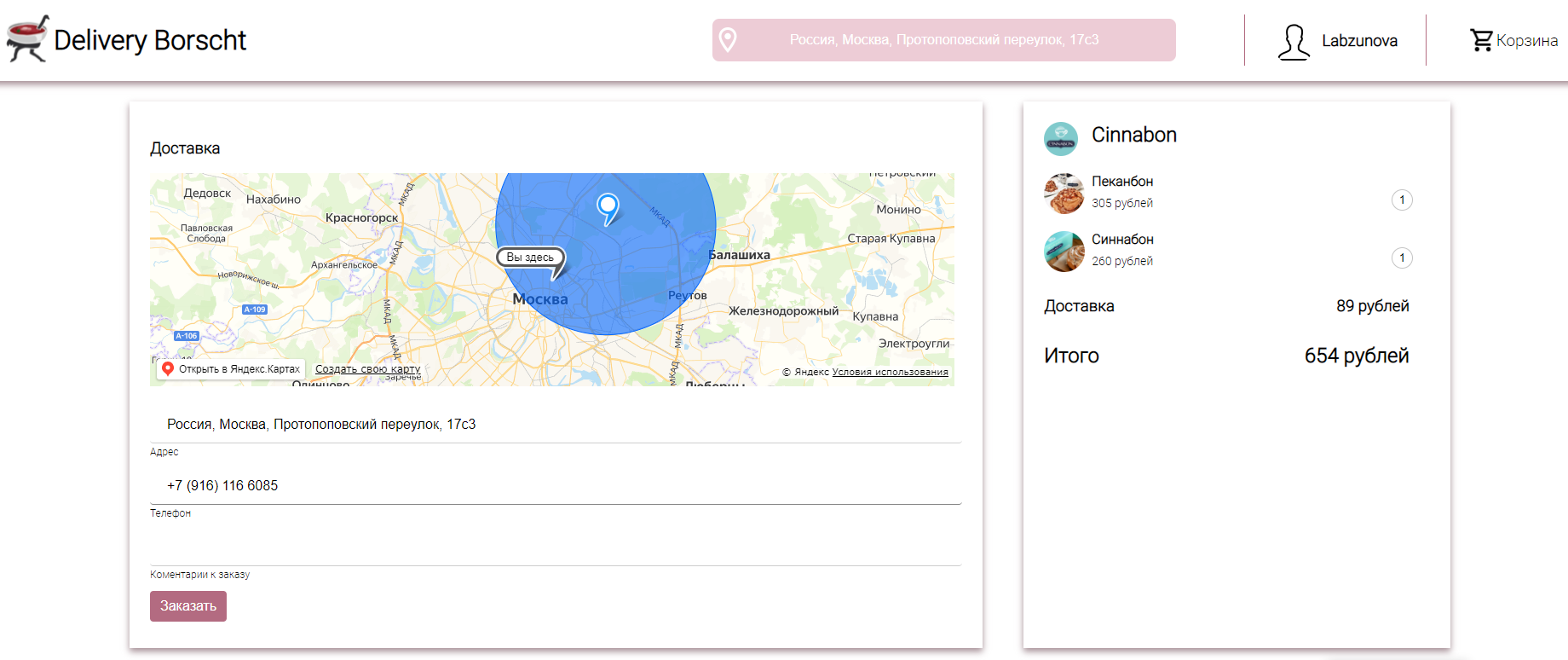


Рисунок 12 – Оформление заказа

После оформления пользователь попадает на страницу со своими заказами, где может отслеживать их статус.

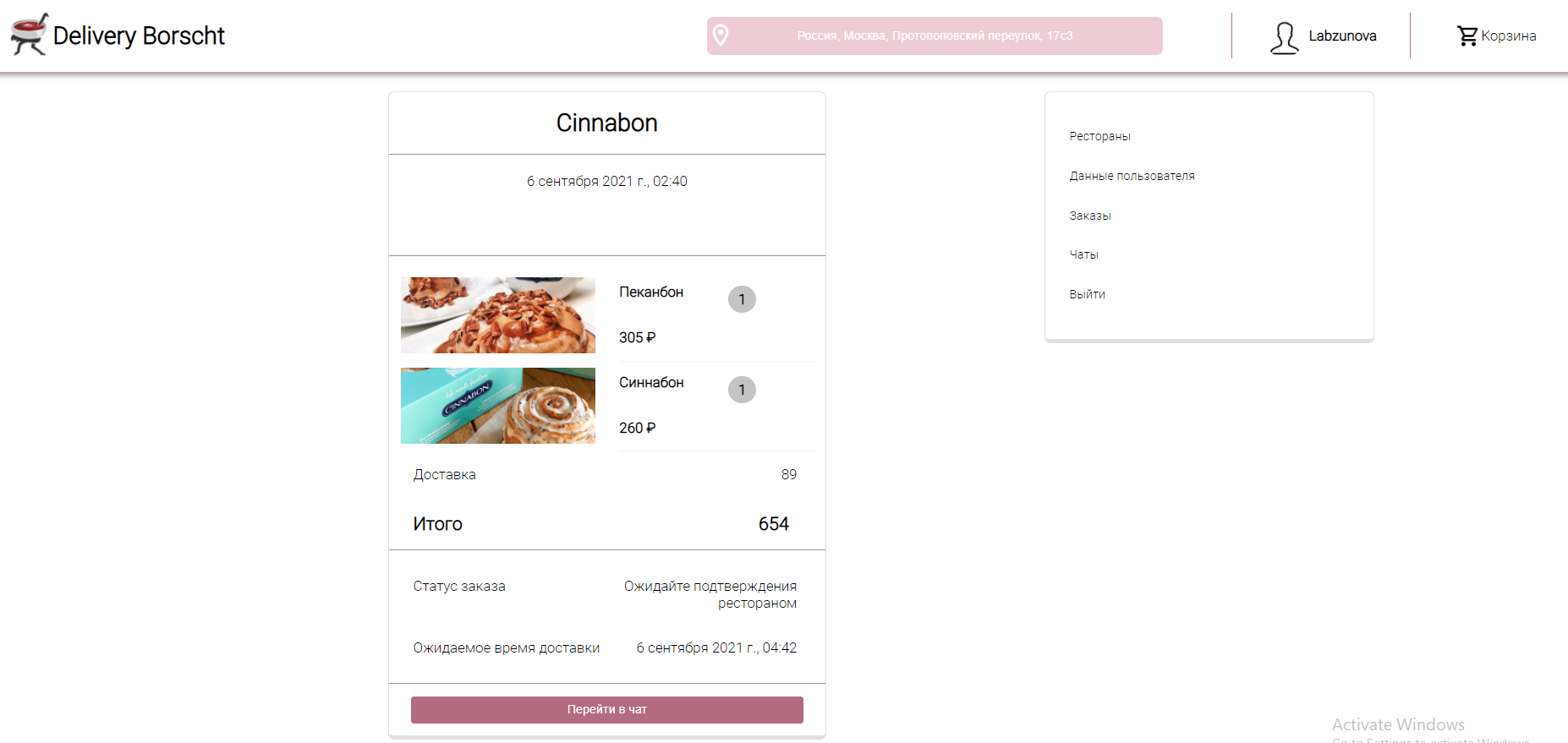


Рисунок 13 – Страница с заказами пользователя

При нажатии на «Данные пользователя» в правом меню можно посмотреть и изменить свои данные.

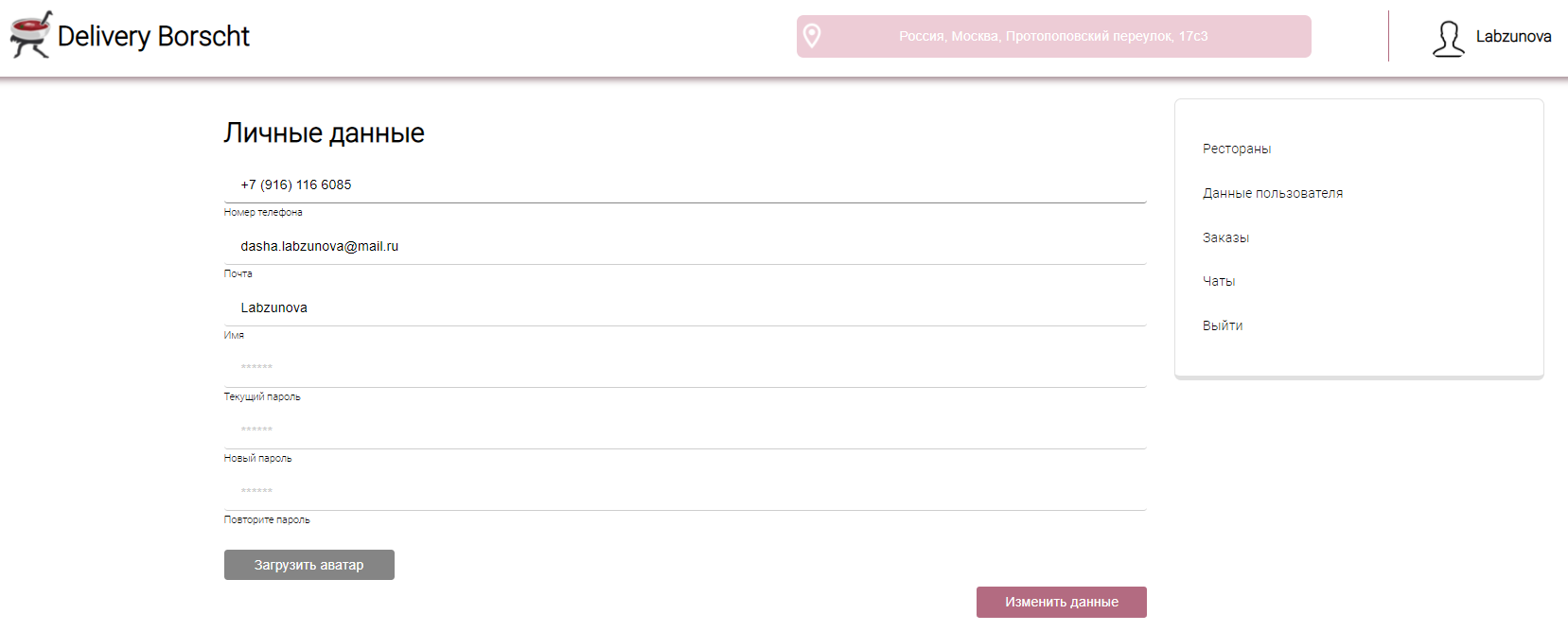


Рисунок 14 – Страница с данными пользователя

Со стороны ресторана можно добавлять и редактировать секции меню и блюда.

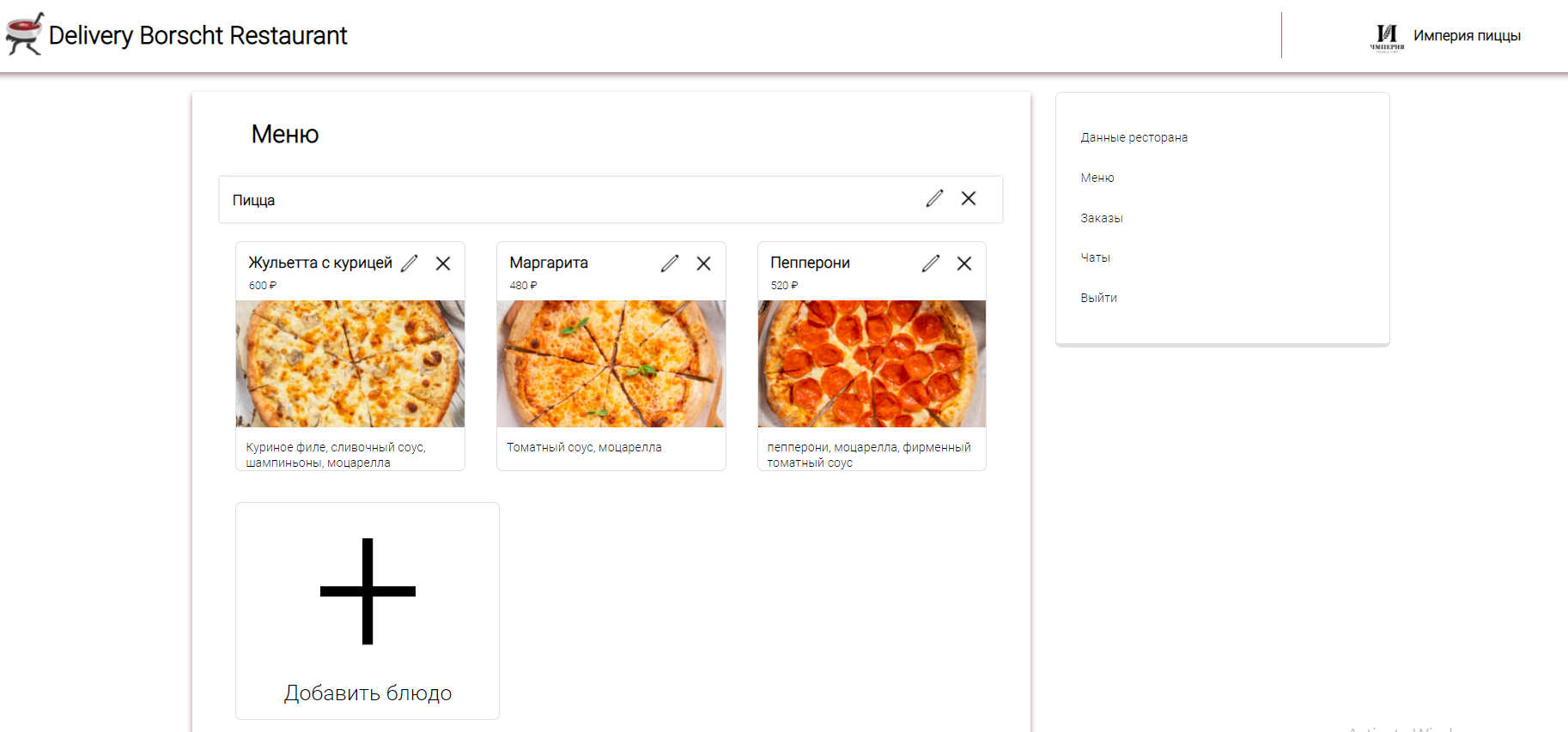


Рисунок 15 – Меню со стороны ресторана

## **Разработка структурной схемы программного продукта**

С целью разбить программу на микросервисы была составлена структурная схема. В ходе анализа процессов в приложении было выявлено четыре подсистемы.

*Главный сервис.* В него приходят изначально все запросы. Он обрабатывает запросы на отдачу списка ресторанов, заказов, профиля пользователя, изменение профиля пользователя и ресторана. Остальные запросы он распределяет по микросервисам, описанным ниже;

*Микросервис авторизации.* В нем происходит обработка запросов авторизации: входа и регистрации.

*Микросервис корзин.* В нем обрабатываются запросы, связанные с корзиной: добавление в корзину, удаление из корзины, запрос на отдачу корзины.

*Микросервис чатов.*  В нем обрабатываются запросы, связанные с чатами: получение всех чатов, получение сообщений конкретного чата, отправка сообщения.

На основе выделенных микросервисов была составлена структурная схема программного продукта.

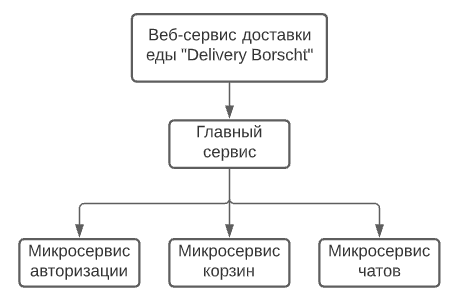


Рисунок 16 – Структурная схема программного продукта

# **Выбор стратегии тестирования и разработка тестов**

## **Тестирование структурным контролем**

Первым методом тестирования было выбрано тестирование структурным контролем. Данное тестирование позволяет определить типовые ошибки, часто встречающиеся в коде. Список вопросов для структурного контроля пополняется с годами и опытом программистов. Результаты тестирования приведены в таблице 3.

Таблица 3 – Структурный контроль

|  |  |  |
| --- | --- | --- |
| Вопрос | Результат тестирования | Вывод |
| Все ли переменные инициализированы? | Да, иначе IDE показал бы ошибку. | Все переменные проинициализированы. |
| Присутствуют ли переменные со сходными именами? | Нет, иначе IDE показал бы ошибку. | Переменных с одинаковыми именами нет |
| Не выходят ли индексы за границы массивов? | Все обращения к массивам происходят в рамках длины массива. | Индексы не выходят за границы массивов. |
| Корректно ли выполнены вычисления с переменными различных типов (в том числе с использованием целочисленной арифметики)? | Да, иначе IDE показал бы ошибку. | Вычисления с переменными различных типов выполнены корректно. |
| Будут ли корректно завершены циклы? | Да, так как возможные ошибки обработаны, и выход из цикла будет либо по окончании счетчика, либо в ходе ошибки. | Циклы будут завершены корректно. |
| Будет ли завершена программа? | Каждый запрос пройдет полностью и отдаст ответ. Либо полностью пройдет весь алгоритм, либо обнаружит ошибку и сообщит о ней в ответе. | Программа завершена будет. |
| Существуют ли циклы, которые не будут выполняться из-за нарушения условия входа? Корректно ли продолжатся вычисления? | Таких циклов не существует. Если цикл не зайдет в свое тело – значит, так и нужно. В случае ошибок они обрабатываются и отдаются в ответе. | Все циклы выполняются и завершаются. Ошибки обработаны. |
| Существуют ли поисковые циклы? Корректно ли отрабатываются ситуации «элемент найден» и «элемент не найден»? | Все ситуации обработаны. В случае «элемент найден» и «элемент не найден» программа отдаст адекватный ответ. | Ситуации обрабатываются корректно. |
| Не изменяет ли подпрограмма аргументов, которые не должны изменяться? | Нет. Переменные, которые не должны меняться, отмечены как константы. | Программа не меняет аргументов, которые не должны меняться. |
| Не происходит ли нарушения области действия глобальных и локальных переменных с одинаковыми именами? | Переменных с одинаковыми названиями нет. | Нарушения области действия глобальных и локальных переменных с одинаковыми именами не происходит. |

Вывод из структурного контроля: программа работает корректно и типовых ошибок не обнаружено.

## **Unit-тесты**

Unit-тесты — модульные тесты, применяемые в различных слоях приложения. Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

В ходе работы были написаны Unit-тесты для большой части функционала. Окончательное покрытие кода тестами – 60%. Тестировалось как корректное исполнение программы, так и ситуации, в которых должны возникать ошибки. Тестировались все уровни архитектуры – handlers, usecase, repository.

Для имитации работы нижних слоев архитектуры использовались моки. Моки(mock) - это такие классы-заглушки (и соответсвенно объекты), которые позволяют избавиться от внешних зависимостей при модульном (unit) тестировании. Для создания моков использовался фреймворк GoMock. Для имитации работы базы данных в тестах самого нижнего слоя архитектуры – repository использовался фреймворк go-sqlmock. Все запросы в базу данных «ходили» не в реальную базу, а их обрабатывали написанные ранее заглушки на sqlmock.

Перед развертыванием веб-приложения обязательно запускаются тесты. И деплой приложения происходит только с условием того, что все тесты прошли. Добиться этого позвонила технология Continuous Integration(CI). Когда код приложения оказывается в главной ветке репозитория, технологии Github запускают тесты и начинают собирать и развертывать приложение.

## **Usability-тест**

Для проверки удобности и понятности интерфейса был проведен usability-тест. Юзабилити-тестирование — это метод оценки интерфейса со стороны удобства и эффективности его использования. Чтобы получить ее, нужно привлечь представителей целевой аудитории программного продукта. Был приглашен независимый пользователь, и для начала ему было задано несколько вопросов, а затем было предложено пройти на сайте два задания. Сценарий теста:

1. Представиться, рассказать, что разрабатываем
2. Сколько вам лет?
3. Как часто вы заказываете еду?
4. С помощью каких сервисов вы это делаете?
5. В каких ситуация вы заказываете еду?
6. Опишите процесс заказа еды
7. Как часто возникают сложности при заказе еды? Какие это сложности?
8. Вы обычно заказываете, конкретно зная, что хотите? Или принимаете решения, просматривая сайт?
9. Сколько времени занимает принятие решения о том, в каком ресторане заказать?
10. Сравниваете ли вы еду из разных ресторанов? Как вы это делаете?
11. Задание: Если вы сейчас хотите что-то поесть - закажите еду в соответствии со своими пожеланиями.
12. Задание: Вспомните один из своих заказов еды и попробуйте повторить его.
13. Задание: Вы хотите заказать азиатской еды на сумму до 1500 рублей, из ресторана рейтингом 4+

Тестирование проводилось с целью определить удобство продукта и выявить места, в которых интерфейс или функционал можно изменить. В ходе тестирования было выявлено, что в целом, веб-сервис удобный и понятный, но работа с корзиной неудобна для пользователей ввиду того, что блюдо нельзя удалить из корзины в окне самой корзины. Это было взято на заметку к исправлению.

# **Заключение**

В результате выполнения курсовой работы был получен опыт в разработке на языке Golang и JavaScript, а также опыт проектирования баз данных и архитектуры веб-сервисов в целом. Как итог – разработан веб-сервис для заказа еды «Delivery Borscht», позволяющее заказывать еду с доставкой на дом, а также зарегистрировать на сервисе свой ресторан и принимать заказы с данного веб-сервиса.

В ходе разработки была использована система контроля версий Git и платформа для хранения репозиториев Github, на которой хранится исходный код программы. Для тестирования и развертывания была использована методология CI/CD.

Также в процессе разработки были созданы различные диаграммы и схемы с целью получения более продуманного продукта и понимания всех выкладок его работы.

По окончании разработки приложение было протестировано для того, чтобы удостовериться, что оно работает корректно и правильно выполняет предполагаемые функции.

Итоговый сервис соответствует заявленному ТЗ и выполняет все требования.

# **Список используемых источников**

1. Официальная документация по языку Golang [Электронный ресурс]. – URL: https://golang.org/doc/
2. Официальная документация СУБД PostgreSQL, версия 13.4 [Электронный ресурс]. – URL: https://www.postgresql.org/docs/13/index.html
3. Конспекты и видеозаписи лекций по курсу «Разработка веб-сервисов на Golang» образовательной программы «Технопарк»
4. Иванова Г.С., Ничушкина Т.Н. Тестирование программного обеспечения: Методические указания по выполнению лабораторной работы по дисциплине "Технология разработки программных систем" [Текст]. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2019. – 13 с.
5. Методы обработки данных и оценки программ : учебное пособие / Г. С. Иванова, Т. Н. Ничушкина, Е. К. Пугачев. — Москва : Издательство МГТУ им. Н. Э. Баумана, 2020. — 72, [2] с. : ил.
6. Роберт Мартин. Чистая архитектура. Искусство разработки программного обеспечения – Издательство Питер СПБ, 2018. – 352 с.
7. Backend-репозиторий проекта 2021\_1\_Borscht\_With\_Cabbage – URL: https://github.com/go-park-mail-ru/2021\_1\_Borscht\_With\_Cabbage
8. Frontend-репозиторий проекта 2021\_1\_Borscht\_With\_Cabbage – URL: https://github.com/frontend-park-mail-ru/2021\_1\_Borscht\_With\_Cabbage

# **Приложение А. Техническое задание.**

Исходный код веб-приложения для заказа еды «Delivery Borscht» находится на Github.com:

Backend-часть приложения: https://github.com/go-park-mail-ru/2021\_1\_Borscht\_With\_Cabbage

Frontend-часть приложения: https://github.com/frontend-park-mail-ru/2021\_1\_Borscht\_With\_Cabbage