

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования

«Самарский национальный исследовательский университет  
имени академика С.П.Королёва»

Факультет информатики  
Кафедра технической кибернетики

Курсовой проект по дисциплине  
«Технологии сетевого программирования»

**Сервис предзаказа еды**

Выполнили студенты:

Кирилин П. Н.

Наумов М. Е.

Группа:

6409

Проверил:

Белоусов А. А.

Самара 2018

## РЕФЕРАТ

Отчёт: 41 страницы, 16 рисунков, 7 таблицы, 10 источников, 1 приложение.

*ТЕХНОЛОГИИ СЕТЕВОГО ПРОГРАММИРОВАНИЯ, RUBY ON RAILS, VUE, АРХИТЕКТУРА, ВЕБ-ИНТЕРФЕЙС, POSTGRESQL*

Цель работы - создание трехуровневого приложения, работающего с базой данных, описывающей работу «Сервиса предзаказа еды».

## СОДЕРЖАНИЕ

Введение . . . . .	4
1 Обоснование выбора технологий . . . . .	6
2 Сценарии использования приложения . . . . .	7
2.1 Уменьшить время ожидания для покупателей . . . . .	7
2.2 Увеличение дохода ресторана . . . . .	8
3 Структура базы данных . . . . .	10
3.1 Схемы базы данных . . . . .	10
3.2 Описание сущностей . . . . .	11
4 CI интеграция . . . . .	15
5 Архитектура приложения . . . . .	16
5.1 Серверная сторона . . . . .	16
5.2 Клиентская сторона . . . . .	17
5.3 База данных . . . . .	18
6 Пользовательский интерфейс . . . . .	18
Заключение . . . . .	27
Список использованных источников . . . . .	28
Скрипт генерации баз данных . . . . .	29
CI скрипты . . . . .	36
Пример контроллера Ruby on Rails . . . . .	39
Пример модели Ruby on Rails . . . . .	41

## **ВВЕДЕНИЕ**

Данное приложение позволит вам совершать предзаказ и получать еду к назначенному времени.

Пользователю предоставляется выбор ресторана, где он может совершить заказ. В каждом ресторане имеется меню товаров, доступных для заказа. После составления заказа, пользователю предлагается сообщается о времени, к которому заказ должен быть приготовлен.

Сервис «feedEm» не имеет очередей ожидания. Всё, что понадобится для заказа - соединение с сетью интернет.

Заказ пользователя готовится к назначенному времени и вместо ожидания в очереди нужно только забрать заказ.

Обычным пользователям системы предоставляются следующие возможности:

1. Регистрация аккаунта;
2. Поиск интересующего товара по названию и описанию;
3. Добавление понравившихся товаров в корзину;

Зарегистрированные пользователи имеют права на:

1. Добавление кредитных карт, с помощью которых производится оплата;
2. Оплата товаров, находящихся в корзине;
3. Отменить заказ;
4. Поставить оценку продавцу.

Продавцы зарегистрированные в системе «feedEm» могут:

1. Редактировать меню товаров, предоставляемое пользователям;

2. Отменить заказ;
3. Сообщить пользователю о готовности заказа;
4. Запросить новый токен для аутентификации.

Администраторы имеют следующие возможности:

1. Изменять аккаунты клиентов;
2. Изменять аккаунты продавцов;
3. Изменять аккаунты администраторов;
4. Изменять товары;
5. Изменять совершенные заказы;

## **1 Обоснование выбора технологий**

Разработка приложения велась на языке Ruby[1], с использованием фреймворка Rails[2] с использованием JavaScript фреймворка Vue[3]. Выбор обусловлен тем, что разработка приложений с использованием фреймворка «Ruby on rails» существенно ускоряет и упрощает разработку серверной части приложения и настройку окружения для разрабатываемого приложения, в которое входит взаимодействие с системами управления базами данных, а также всё необходимое для разработки клиентской части приложения.

## **2 Сценарии использования приложения**

### **2.1 Уменьшить время ожидания для покупателей**

Данное приложение позволит вам совершать предзаказ и получать еду к назначенному времени.

Пользователю предоставляется выбор ресторана, где он может совершить заказ. В каждом ресторане имеется меню возможных товаров. После составления заказа, пользователю показывается время, к которому заказ должен быть приготовлен.

Сервис feeEm не имеет очередей ожидания. Всё, что понадобится для заказа - соединение с сетью интернет.

Заказ пользователя готовится к назначенному времени и вместо ожидания в очереди нужно только забрать заказ.

#### **1. Покупатель может просматривать каталоги ресторанов.**

Для просмотра меню ресторана нужно выполнить следующую последовательность действий:

1.1. Зайти на сайт;

1.2. Зарегистрироваться (Если нужно);

1.3. Пройти авторизацию;

1.4. Осуществить поиск интересующего ресторана (Поиск осуществляется по описанию, названию, по товарам);

1.5. Выбрать ресторан.

После выполненных действий пользователю будет доступно меню выбранного ресторана.

#### **2. Покупатель может осуществить заказ онлайн.**

Для совершения заказа нужно выполнить следующую последовательность действий:

- 2.1. Открыть меню интересующего ресторана;
- 2.2. Добавить интересующие пункты меню в корзину;
- 2.3. Добавить кредитную карту (Если нужно);
- 2.4. Проверить примерное время заказа (Если устраивает, то продолжить);
- 2.5. Заказать и оплатить заказ.

### 3. Рейтинговая система для выбора лучшего магазина.

Также, пользователи могут поставить оценку ресторану, что поможет другим пользователям выбрать продавца с наилучшим сервисом. Чтобы это сделать нужно:

- 3.1. Открыть страницу ресторана;
- 3.2. Поставить оценку (Нравится или не нравится).

## **2.2 Увеличение дохода ресторана**

FeedEm создан не для доставки, а для предзаказов. Цель продавцов - подготовка еды к нужному для клиентов времени. Покупатели сами придут, чтобы получить заказ.

### 1. Составление меню для продажи онлайн.

Для составления меню нужно выполнить следующие действия:

- 1.1. Открыть страницу управления рестораном.
- 1.2. Открыть страницу редактирования меню.
- 1.3. Добавить/отредактировать/удалить товары.



## 2. Прием заказов онлайн.

Для приема доступных заказов:

- 2.1. Открыть страницу управления рестораном.
- 2.2. Открыть вкладку заказов.
- 2.3. Принять/отредактировать примерное время/отклонить заказы.

## 3. Просмотр рейтинга ресторанов.

Просмотр наиболее популярных ресторанов может помочь узнать, что сейчас интересно покупателям.

Для просмотра рейтинга рестораноа:

- 3.1. Открыть страницу выбора ресторана. На данной странице представлены рестораны, отсортированные по рейтингу.

Изменение рейтинга ресторана доступно только для авторизованных покупателей.

### 3 Структура базы данных

#### 3.1 Схемы базы данных

На рисунках 1 и 2 изображены логическая и физическая схемы базы данных.

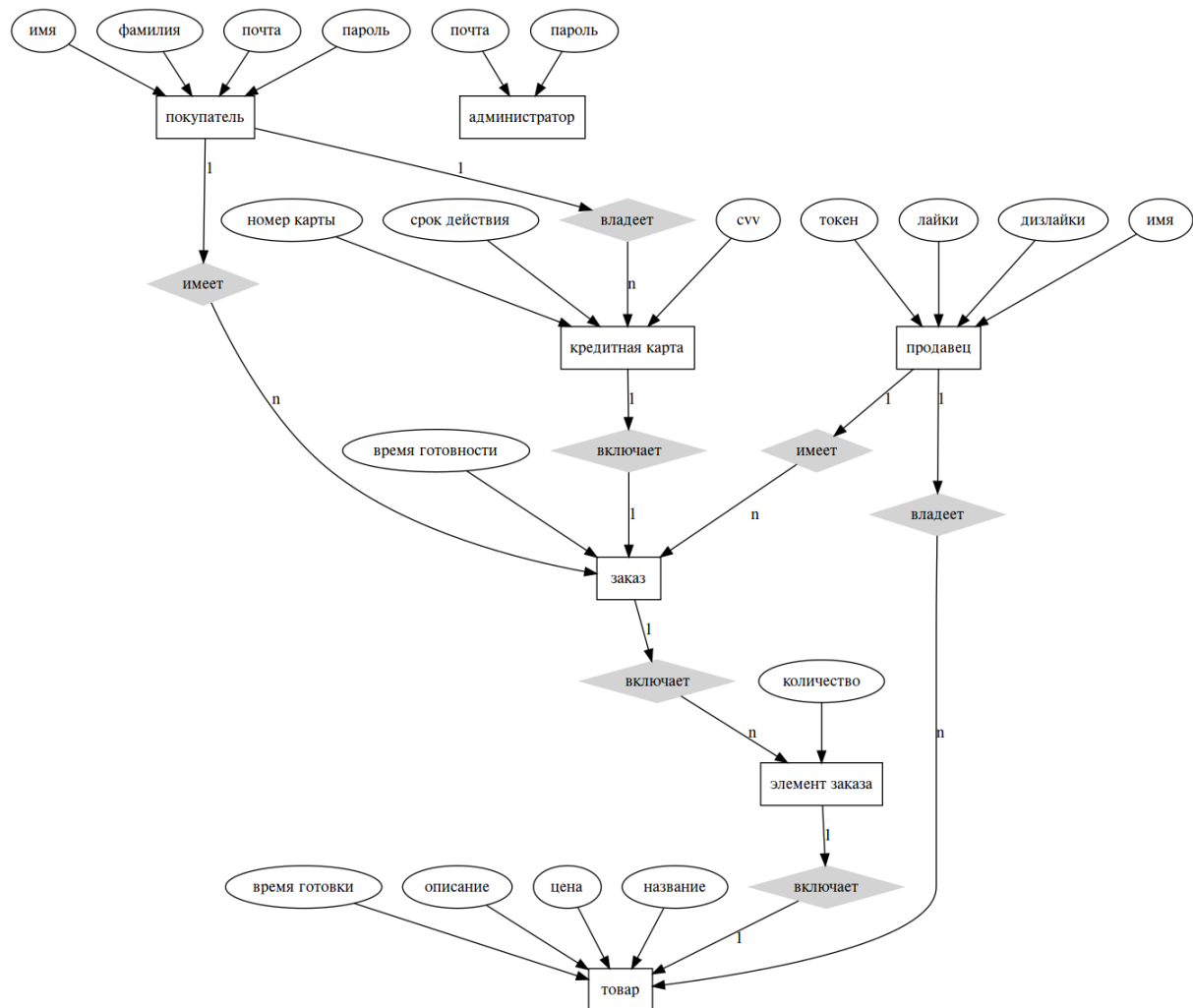
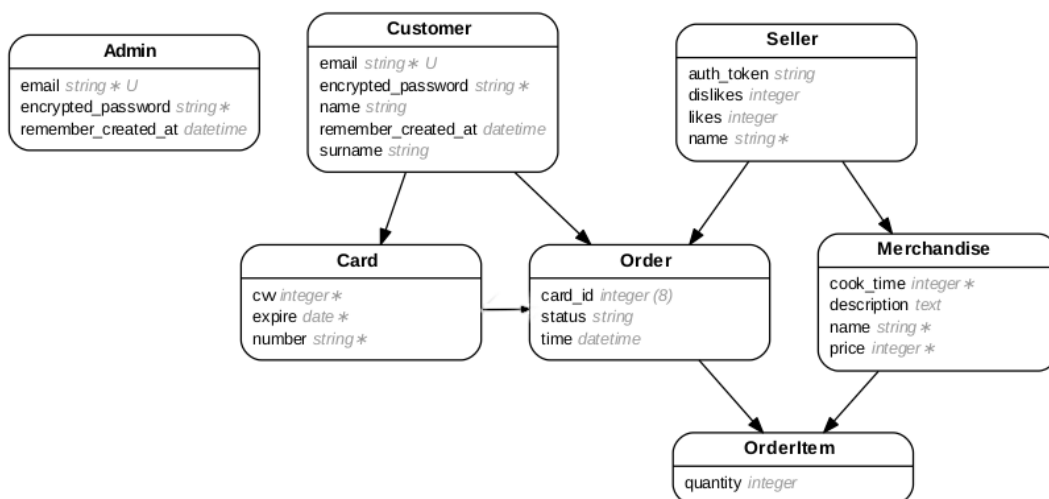


Рисунок 1 – Логическая схема базы данных



*Рисунок 2 – Физическая схема базы данных*

В приложении А представлены скрипты генерации и заполнения баз данных.

### 3.2 Описание сущностей

Краткое описание сущностей приведено в таблицах 1, 2, 3, 4, 5, 6, 7.

*Таблица 1 – Описание сущности «покупатель»*

Таблица	Атрибут	Описание
customers	Содержит информацию о покупателях.	
[Первичный ключ]	id	Уникальный идентификатор покупателя.
	name	Имя покупателя.
	surname	Фамилия покупателя.
[Вторичный ключ]	email	Электронная почта покупателя.
	password	Зашифрованный пароль покупателя.

*Таблица 2 – Описание сущности «продавец»*

Таблица	Атрибут	Описание
sellers	Содержит информацию о продавцах.	
[Первичный ключ]	id	Уникальный идентификатор продавца.
[Вторичный ключ]	token	Зашифрованный пароль продавца.
	likes	Количество положительных оценок.
	dislikes	Количество отрицательных оценок.

*Таблица 3 – Описание сущности «администратор»*

Таблица	Атрибут	Описание
admins	Содержит информацию об администраторах.	
[Первичный ключ]	id	Уникальный идентификатор администратора.
[Вторичный ключ]	email	Электронная почта администратора.
	password	Зашифрованный пароль администратора.

Таблица 4 – Описание сущности «товар»

Таблица	Атрибут	Описание
merchandises	Содержит информацию о товарах.	
[Первичный ключ]	id	Уникальный идентификатор товара.
[Вторичный ключ]	seller_id	Ключ продавца товара.
	name	Название товара.
	description	Описание товара.
	cook_time	Время приготовления товара.
	price	Цена товара.

Таблица 5 – Описание сущности «заказ»

Таблица	Атрибут	Описание
orders	Содержит информацию о заказах.	
[Первичный ключ]	id	Уникальный идентификатор заказа.
	name	Название зак.
[Вторичный ключ]	customer_id	Ключ покупателя.
[Вторичный ключ]	seller_id	Ключ продавца.
[Вторичный ключ]	card_id	id карты, которой был оплачен заказ.
	status	Текущий статус заказа.

*Таблица 6 – Описание сущности «кредитная карта»*

Таблица	Атрибут	Описание
cards	Содержит информацию о кредитных картах.	
[Первичный ключ]	id	Уникальный идентификатор кредитной карты.
[Вторичный ключ]	customer_id	Ключ покупателя.
	cvv	Защитный код cvv.
	expire	Дата истечения срока действия кредитной карты.
	number	Номер кредитной карты.

*Таблица 7 – Описание сущности «элемент заказа»*

Таблица	Атрибут	Описание
order_items	Содержит информацию об элементах заказа.	
[Первичный ключ]	id	Уникальный идентификатор кредитной карты.
[Вторичный ключ]	order_id	Ключ заказа.
[Вторичный ключ]	merch_id	Ключ товара.
	quantity	Количество товара в заказе.

## 4 CI интеграция

Данный проект проходит набор тестов для проверки работоспособности на каждом изменении в системе контроля версий, с помощью TravisCI[6]. Само приложение, как и база данных, изолированы в контейнеры и объединены системой композиции контейнеров «docker-compose»[4]. Также, с помощью сервиса Codacy[5] проводится автоматическая оценка качества написанного кода. Увидеть текущие статусы сборок и анализа написанного кода можно по следующим ссылкам:

1. <https://travis-ci.org/s3rius/feedEm>
2. <https://app.codacy.com/project/win10/feedEm/dashboard>

Скрипты генерации контейнеров и их композиции представлены в приложении Б.

## **5 Архитектура приложения**

Представляемая информационная система взаимодействует с базой данных, редактирует, добавляет, удаляет и выполняет параметризованные запросы.

Приложение состоит из трёх основных компонент:

1. Backend (Серверная сторона)
2. Frontend (Клиентская сторона)
3. База данных

### **5.1 Серверная сторона**

Backend написан на языке Ruby, с применением фреймворка Ruby On Rails. В основе данного фреймворка лежит архитектурный шаблон проектирования MVC (Model-view-controller). Ruby on Rails базируется на следующих принципах разработки:

1. Максимальное переиспользование кода, что позволяет снизить дублирование кода в приложении (принцип DRY);
2. Явная спецификация требуется только в нестандартных случаях, по умолчанию используются соглашения по конфигурации (принцип Convention over configuration).

Кроме уже перечисленных преимуществ, Ruby on Rails предоставляет удобное окружение для разработчика. В комплекте с фреймворком поставляется командная утилита, которая позволяет:

1. создать новый проект из шаблона;
2. применить миграции к базе данных;



3. откатить миграции к базе данных;
4. сгенерировать заглушки для различных элементов системы (контроллеры, модели, представления, миграции);
5. очистить базу данных;
6. заполнить базу данных заранее подготовленными данными;
7. запустить тестирование проекта;
8. запустить простой веб-сервер;
9. вывести пути маршрутизации приложения.

Также стоит отметить большое количество готовых и отлаженных модулей, которые значительно ускоряют разработку приложений.

Пример контроллера, реализованного на фреймворке Ruby on Rails, представлен в приложении В.

В приложении Г показан пример модели, реализованной на Ruby on Rails.

## **5.2 Клиентская сторона**

За последние 10 лет веб-страницы стали более динамическими и сложными, благодаря использованию языка JavaScript. Много кода, который раньше исполнялся на сервере был перенесен на клиентскую сторону, из-за чего тысячи строк JavaScript-кода, соединенного с html и CSS работали без какой либо организации именно поэтому в данной работе был использован JavaScript фреймворк VueJs, который позволяет создавать более организованные, поддерживаемые и тестируемые приложения на JavaScript.

Клиентская сторона сервиса «feedEm» полностью построена на фреймворке VueJs с использованием turbolinks. Данный подход позволяет разбить

построение графических интерфейсов на компоненты, что повышает переиспользование кода и ускоряет общую скорость верстки html страниц в дальнейшем.

Turbolinks в свою очередь позволяет сделать динамическую подгрузку страниц более плавной и оптимизированной. Данную технологию сейчас используют такие сайты-гиганты как YouTube, VK, Github и другие.

### **5.3 База данных**

В качестве системы управления базами данных в данной работе была выбрана СУБД PostgreSQL. Данный выбор обусловлен бесплатностью этой СУБД, а также её широкими возможностями и хорошей стабильностью.

Одной из таких возможностей является поддержка полнотекстового поиска, который, в данной работе, используется для поиска товаров и продавцов.

## **6 Пользовательский интерфейс**

При открытии главной страницы сервиса, пользователю предоставляется информация о поступивших товарах. Строка для поиска и каталог доступных продавцов. Неавторизованному пользователю предоставляется возможность добавления товаров в корзину и просмотр рейтинга продавцов, без возможности изменить его. Также, неавторизованные пользователи могут начать поиск по товарам и продавцам.

На рисунках 3 и 4 показана главная страница, открытая для просмотра всех товаров и продавцов соответственно.

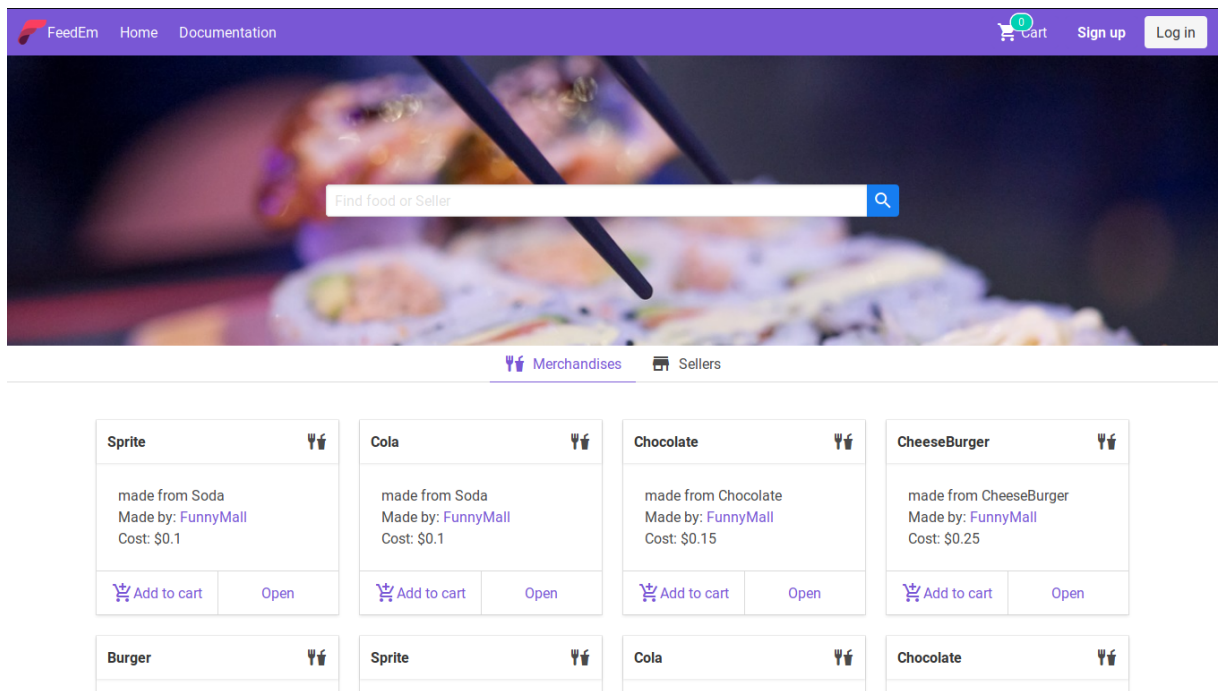


Рисунок 3 – Главная страница на вкладке товаров

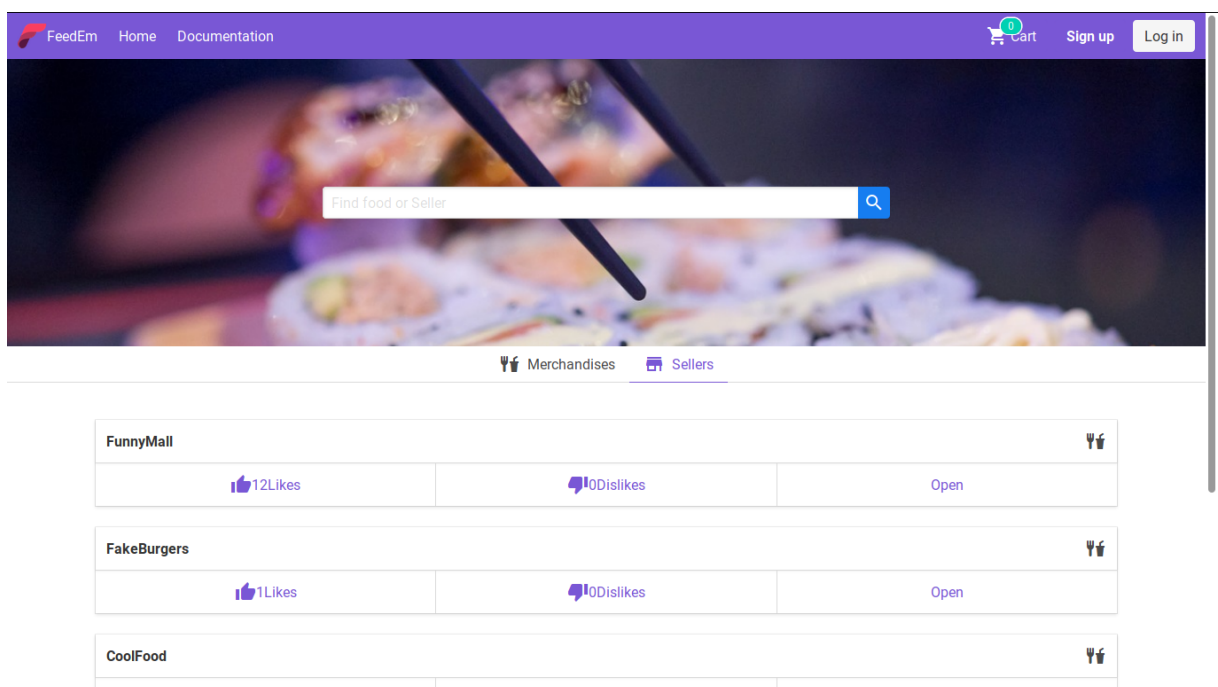


Рисунок 4 – Главная страница на вкладке продавцов

Все продавцы, показанные на главной странице, отсортированы по рейтингу.

На рисунке 5 показана открытая корзина покупок.

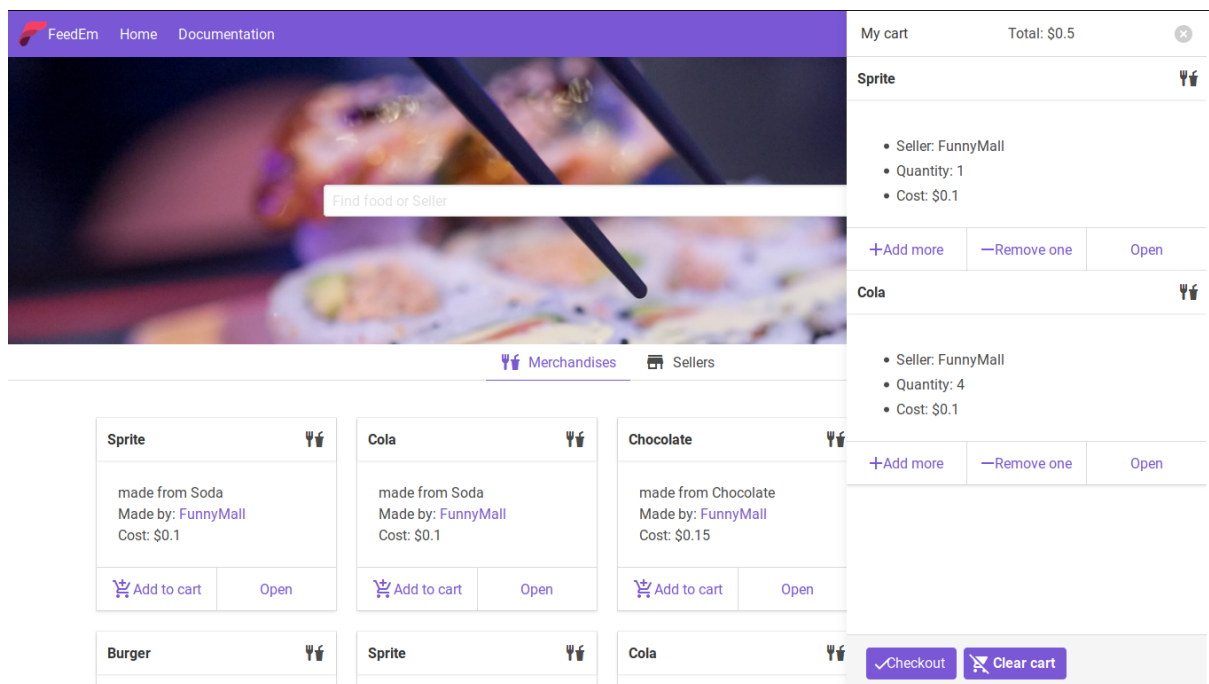


Рисунок 5 – Открытая корзина

На рисунке 6 показана страница входа покупателя.

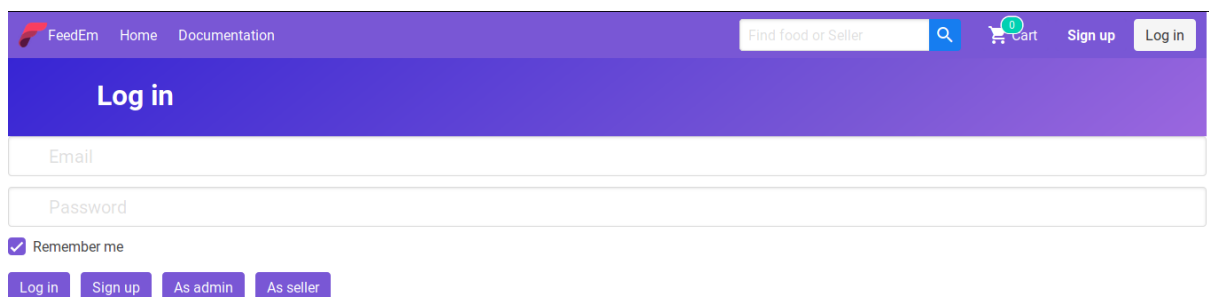
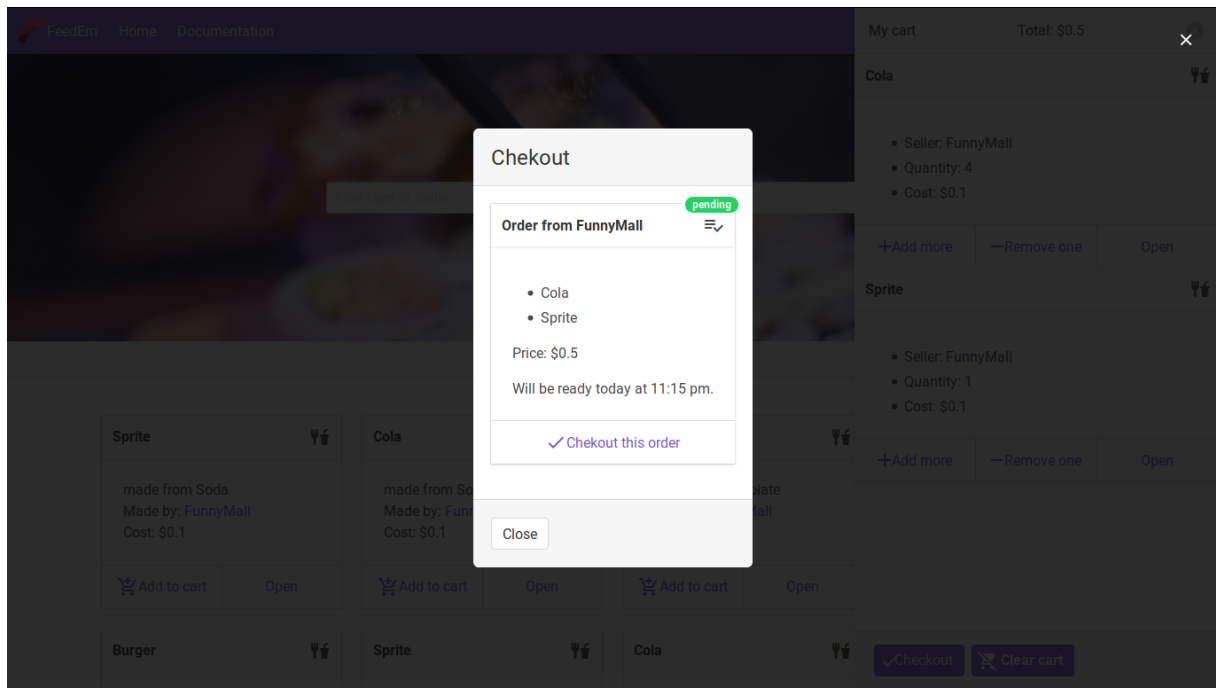


Рисунок 6 – Страница входа

На данной странице можно зайти в аккаунт не только покупателя, но и администратора или продавца.

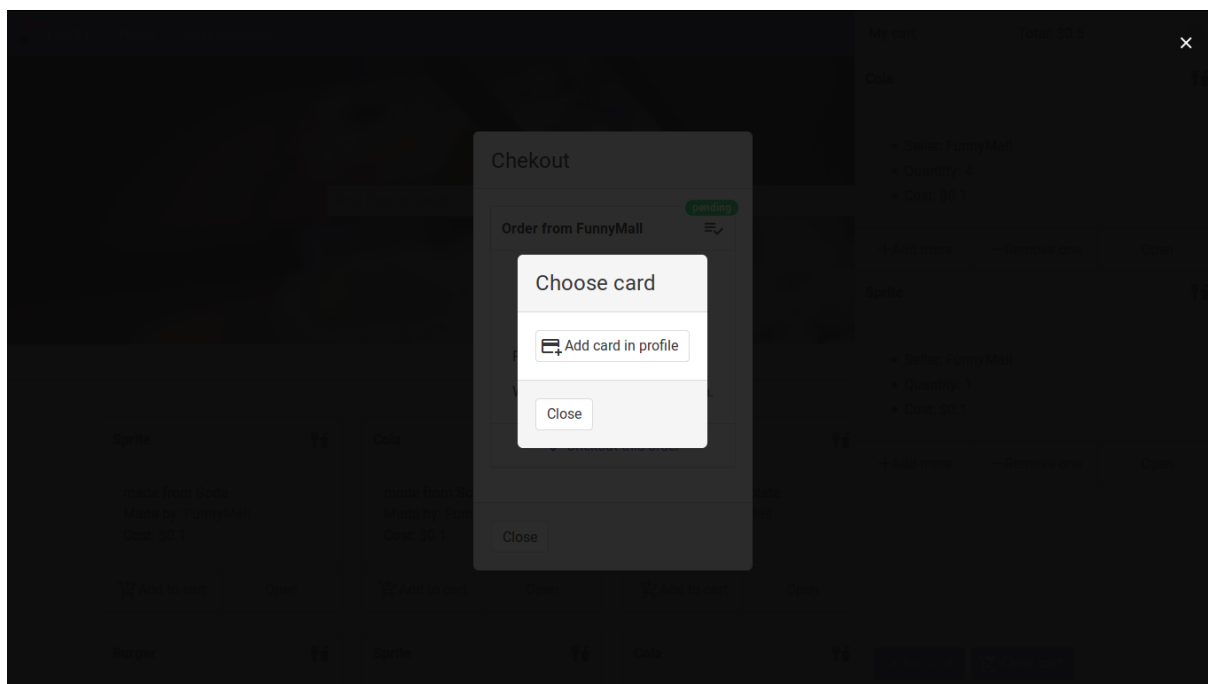
При попытке оплатить товар появляется всплывающее окно, пока-

занное на рисунке 7, которое позволяет выбрать какой именно из сформированных заказов подлежит к оплате. Все заказы формируются исходя из того, что в каждый из потенциальных заказов помещаются все товары одного продавца.



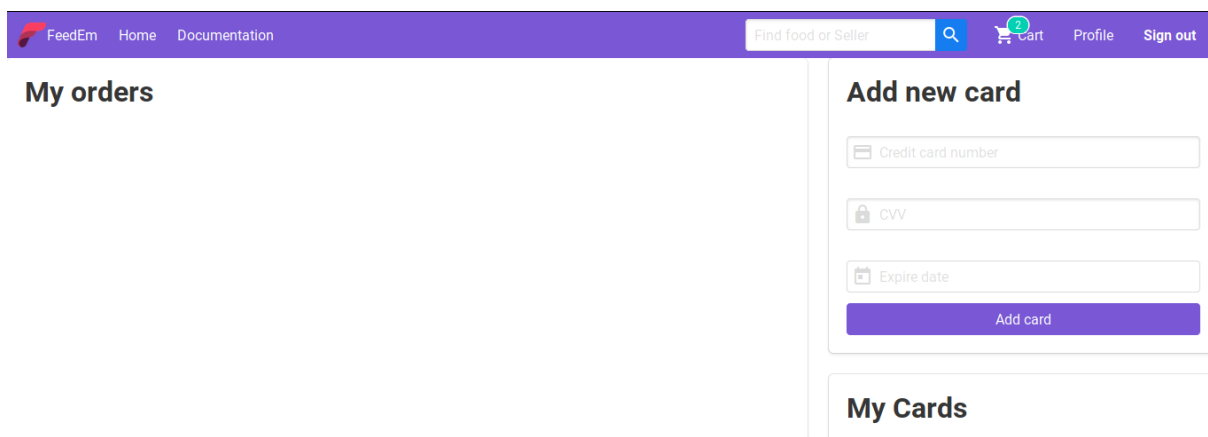
*Рисунок 7 – Выбор заказа пр попытке оплаты*

При отсутствии карт, зарегистрированных на аккаунт текущего покупателя, появляется предложение добавить карту на странице пользователя. Оповещение показано на рисунке 8.



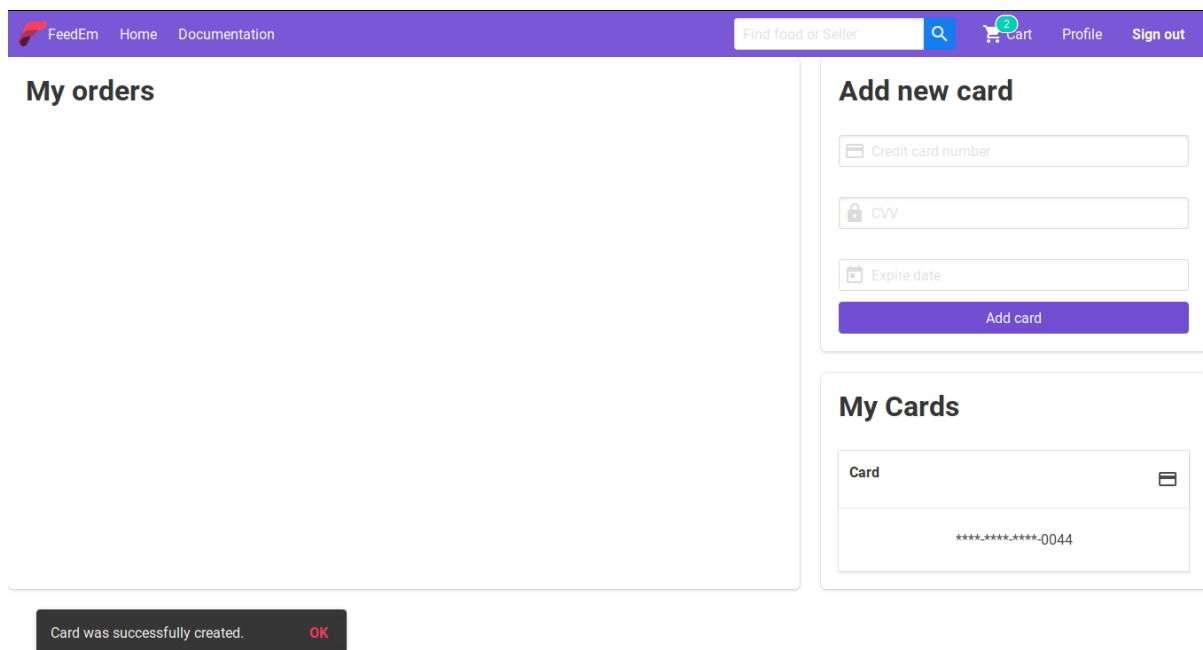
*Рисунок 8 – Оповещение об отсутствии зарегистрированных карт*

Переходя на страницу профиля будет доступна форма для добавления карты, как показано на рисунке 9.



*Рисунок 9 – Страница пользователя*

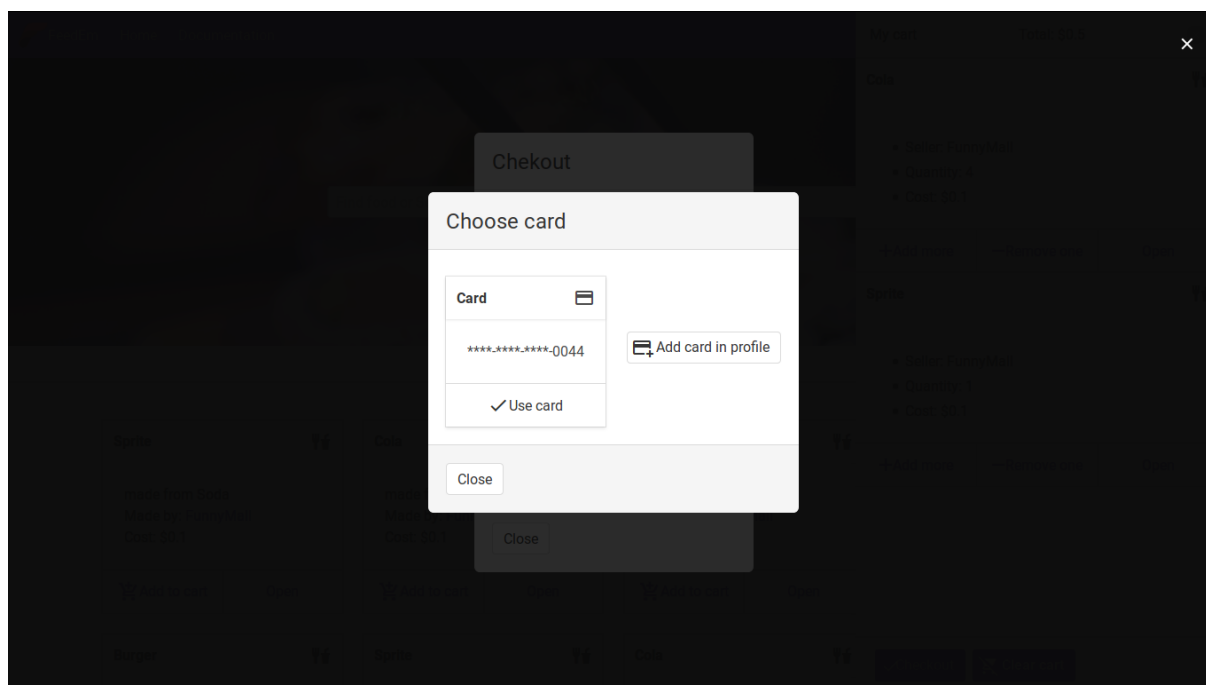
После добавления карты, новая карта сразу появится в соответствующем разделе на странице профиля. Как показано на рисунке 10.



*Рисунок 10 – Страница с добавленной картой*

После этого карта сразу станет доступна для покупки товаров.

На рисунке 11 показано окно выбора карты для оплаты заказа.



*Рисунок 11 – Выбор карт при оплате товара*

После совершения покупки, на странице заказов пользователя появится соответствующая запись, как показано на рисунке 12, с помощью

которой можно определить статус готовности заказа.

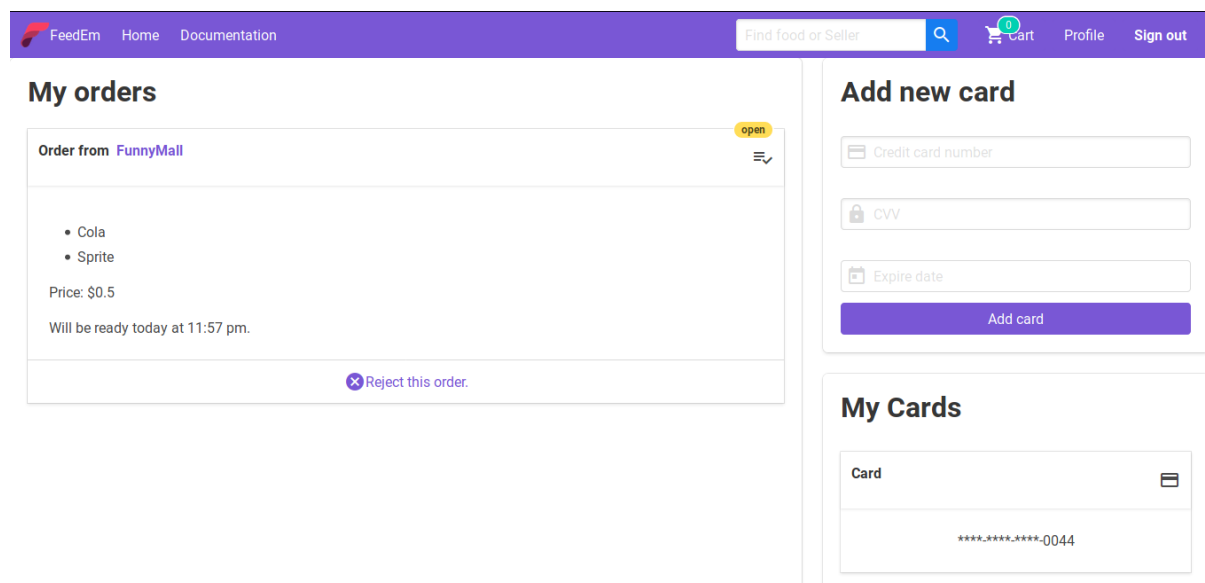


Рисунок 12 – Отображение заказа на странице пользователя

Со стороны продавца данный заказ выглядит как показано на рисунке 13.

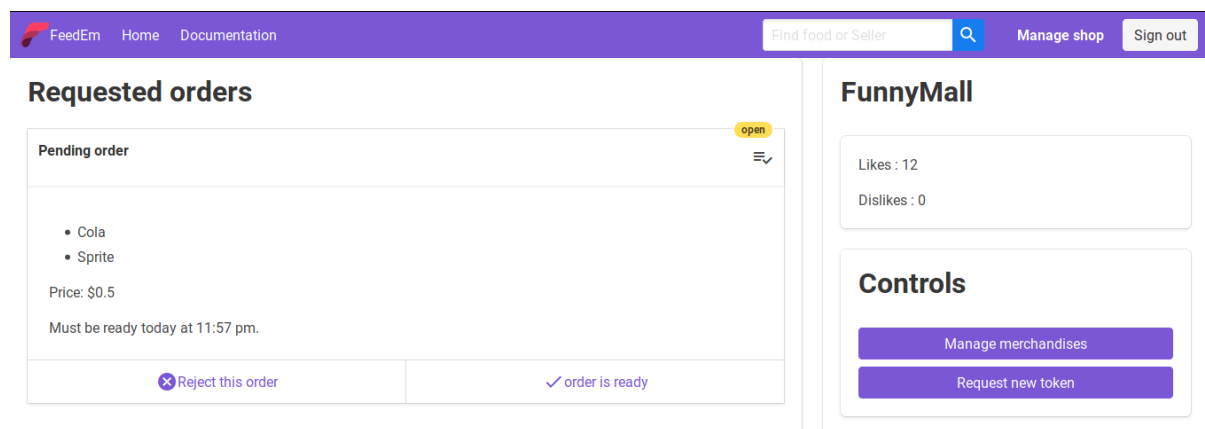


Рисунок 13 – Страница заказов, запрошенных у продавца

После того как продавец поменял статус заказа на «готово», отобра-



жение статуса меняется как у продавца, так и у покупателя, как показано на рисунках 14 и 15 соответственно.

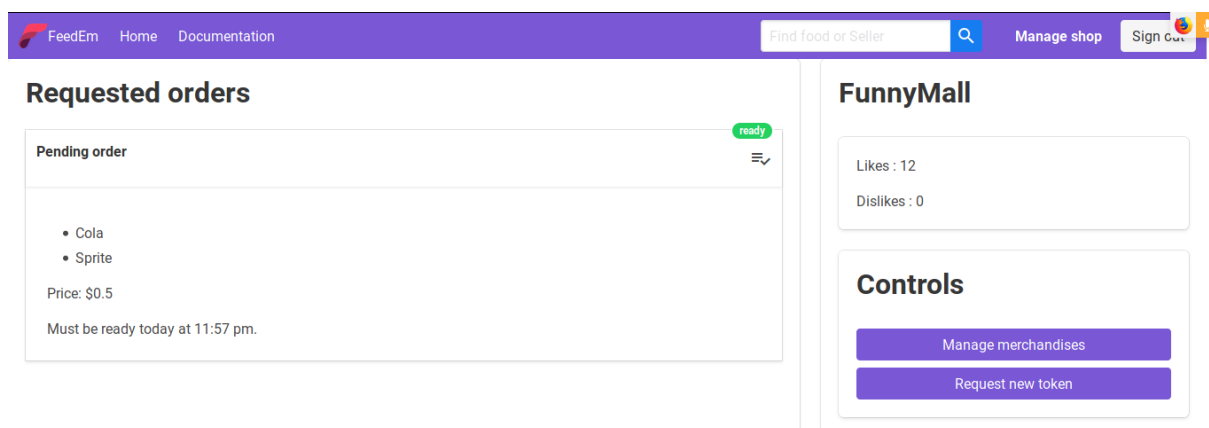


Рисунок 14 – Смена статуса готовности заказа

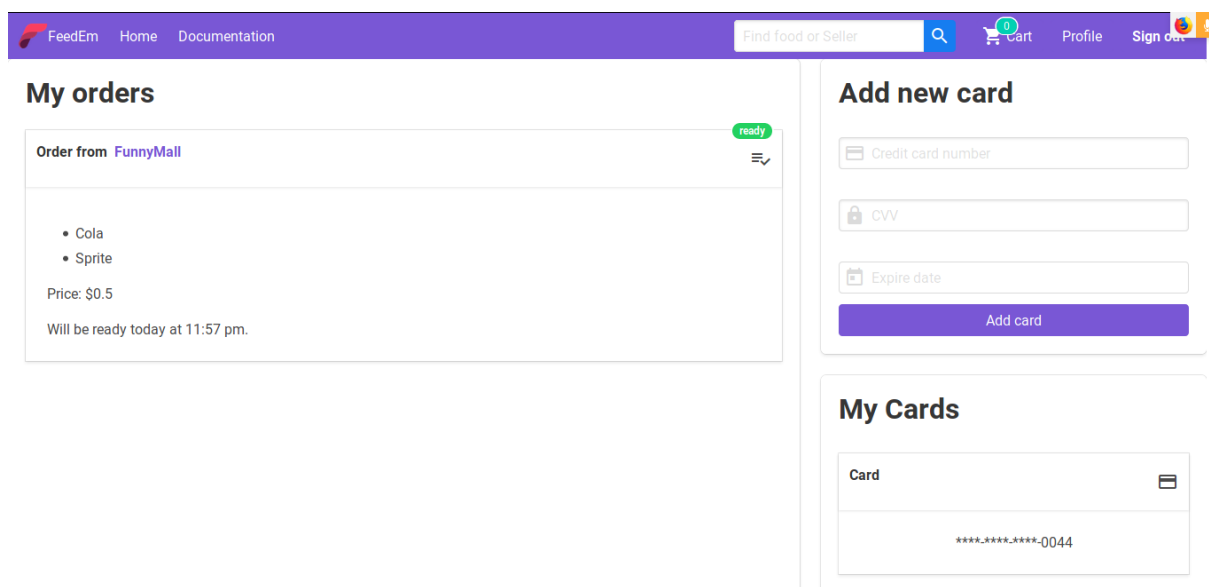



Рисунок 15 – Смена статуса готовности заказа

Страница управления товарами у продавца выглядит как показано на рисунке 16:

 [Home](#) [Documentation](#)

[Manage shop](#) [Sign out](#)

Merchandises

New Merchandise

*Рисунок 16 – Страница редактирования меню*

## ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был спроектирован и реализован сервис предзаказа еды, предоставляющий покупателям и продавцам удобный веб-интерфейс и широкие возможности. Можно сделать вывод что использование технологий Ruby on Rails и VueJs, а также сопутствующих средств разработки облегчает создание веб-приложений, за счёт использования модульной архитектуры. Кроме того, шаблон проектирования MVC позволяет значительно упростить добавление новой функциональности, поддержку старой, а также тестирование проекта.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Ruby Documentation[Электронный ресурс]: Официальный сайт документации языка программирования Ruby. - URL: <https://www.ruby-lang.org/en/docu>
- 2 Ruby on Rails documentation[Электронный ресурс]: Официальная документация фреймворка Ruby on rails. - URL: <https://guides.rubyonrails.org/>
- 3 VueJs documentation[Электронный ресурс]: Официальная документация фреймворка VueJs. - URL: <https://vuejs.org/v2/guide/>
- 4 Docker documentation[Электронный ресурс]: Официальная документация системы Docker. - URL: <https://docs.docker.com/>
- 5 Codacy documentation[Электронный ресурс]: Официальная документация системы Codacy. - URL: <https://support.codacy.com/hc/en-us>
- 6 TravisCI documentation[Электронный ресурс]: Официальная документация системы TravisCI. - URL: <https://docs.travis-ci.com/>

# ПРИЛОЖЕНИЕ А

## Скрипт генерации баз данных

```
\connect "app_development";

DROP TABLE IF EXISTS "admins";
DROP SEQUENCE IF EXISTS admins_id_seq;
CREATE SEQUENCE admins_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE 9223372036854775807
    → START 1 CACHE 1;

CREATE TABLE "public"."admins" (
    "id" bigint DEFAULT nextval('admins_id_seq') NOT NULL,
    "created_at" timestamp NOT NULL,
    "updated_at" timestamp NOT NULL,
    "email" character varying DEFAULT '' NOT NULL,
    "encrypted_password" character varying DEFAULT '' NOT NULL,
    "remember_created_at" timestamp,
    CONSTRAINT "admins_pkey" PRIMARY KEY ("id"),
    CONSTRAINT "index_admins_on_email" UNIQUE ("email")
) WITH (oids = false);

INSERT INTO "admins" ("id", "created_at", "updated_at", "email", "encrypted_password",
    → "remember_created_at") VALUES
(1, '2018-12-25 08:59:05.660659', '2018-12-25 08:59:05.660659', 's3rius@emial.mail', '
    → $2a$11$53.5QT9QnhlAPShy3i4cqOU7H05i3L9v7Nxx.FdZRX5brjIVHkMxK', NULL),
(2, '2018-12-25 08:59:05.964432', '2018-12-25 08:59:05.964432', 'kolaer@emial.mail', '
    → $2a$11$YRkxGSXwaNGJWgE2GgFEE.1UjSz4BHjcWWDscp4qHSIPdqU532pqG', NULL),
(3, '2018-12-25 08:59:06.249906', '2018-12-25 08:59:06.249906', 'animethug@emial.mail'
    → , '$2a$11$7tPgWtNiJUbrCwqiNvovIe1Fy6WM7n6F5Co3k/p00WLTE14QzC1va', NULL);

DROP TABLE IF EXISTS "ar_internal_metadata";
CREATE TABLE "public"."ar_internal_metadata" (
    "key" character varying NOT NULL,
    "value" character varying,
    "created_at" timestamp NOT NULL,
    "updated_at" timestamp NOT NULL,
    CONSTRAINT "ar_internal_metadata_pkey" PRIMARY KEY ("key")
) WITH (oids = false);
```

```
INSERT INTO "ar_internal_metadata" ("key", "value", "created_at", "updated_at") VALUES
('environment', 'development', '2018-12-25 08:59:02.754075', '2018-12-25
→ 08:59:02.754075');
```

```
DROP TABLE IF EXISTS "cards";
DROP SEQUENCE IF EXISTS cards_id_seq;
```

```
CREATE SEQUENCE cards_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE 9223372036854775807 START
→ 1 CACHE 1;
```

```
CREATE TABLE "public"."cards" (
  "id" bigint DEFAULT nextval('cards_id_seq') NOT NULL,
  "customer_id" bigint,
  "number" character varying,
  "cvv" integer,
  "expire" date,
  "created_at" timestamp NOT NULL,
  "updated_at" timestamp NOT NULL,
  CONSTRAINT "cards_pkey" PRIMARY KEY ("id"),
  CONSTRAINT "index_cards_on_number" UNIQUE ("number"),
  CONSTRAINT "fk_rails_778182f614" FOREIGN KEY (customer_id) REFERENCES customers(id)
→ NOT DEFERRABLE
) WITH (oids = false);
```

```
CREATE INDEX "index_cards_on_customer_id" ON "public"."cards" USING btree ("
→ customer_id");
```

```
DROP TABLE IF EXISTS "customers";
DROP SEQUENCE IF EXISTS customers_id_seq;
```

```
CREATE SEQUENCE customers_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE 9223372036854775807
→ START 1 CACHE 1;
```

```
CREATE TABLE "public"."customers" (
  "id" bigint DEFAULT nextval('customers_id_seq') NOT NULL,
  "name" character varying,
  "surname" character varying,
  "created_at" timestamp NOT NULL,
  "updated_at" timestamp NOT NULL,
  "email" character varying DEFAULT '' NOT NULL,
  "encrypted_password" character varying DEFAULT '' NOT NULL,
```

```

    "remember_created_at" timestamp,
    CONSTRAINT "customers_pkey" PRIMARY KEY ("id"),
    CONSTRAINT "index_customers_on_email" UNIQUE ("email")
) WITH (oids = false);

INSERT INTO "customers" ("id", "name", "surname", "created_at", "updated_at", "email",
    ↪ "encrypted_password", "remember_created_at") VALUES
(1, 'Pavel', 'Kirilin', '2018-12-25 08:59:05.513541', '2018-12-25 08:59:05.513541', '
    ↪ s3rius@emial.mail', '
    ↪ $2a$11$8hXx4K1CVs0ZgEVt0T5Uae3zDCm1FCWgQQbx8qDei5GMqVYC1fbea', NULL),
(2, 'Maxim', 'Naumov', '2018-12-25 08:59:05.809281', '2018-12-25 08:59:05.809281', '
    ↪ kolaer@emial.mail', '$2a$11$.9
    ↪ fNOUVqSjSEOAYb5Sw97eb9NuSy1zaNXp5EkdTevR44i6KevvH7G', NULL),
(3, 'Andrei', 'Belousov', '2018-12-25 08:59:06.108177', '2018-12-25 08:59:06.108177',
    ↪ 'animethug@emial.mail', '$2a$11$qU1C1V508dhX5bFqzubbeeM7pdMN9RC0lZosg/zTb3/
    ↪ TIJoiVu5yC', NULL);

DROP TABLE IF EXISTS "merchandises";
DROP SEQUENCE IF EXISTS merchandises_id_seq;
CREATE SEQUENCE merchandises_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE
    ↪ 9223372036854775807 START 1 CACHE 1;

CREATE TABLE "public"."merchandises" (
    "id" bigint DEFAULT nextval('merchandises_id_seq') NOT NULL,
    "seller_id" bigint,
    "name" character varying,
    "description" text,
    "price" integer,
    "cook_time" integer,
    "created_at" timestamp NOT NULL,
    "updated_at" timestamp NOT NULL,
    CONSTRAINT "merchandises_pkey" PRIMARY KEY ("id"),
    CONSTRAINT "fk_rails_63736ada83" FOREIGN KEY (seller_id) REFERENCES sellers(id) NOT
    ↪ DEFERRABLE
) WITH (oids = false);

CREATE INDEX "index_merchandises_on_seller_id" ON "public"."merchandises" USING btree
    ↪ ("seller_id");

INSERT INTO "merchandises" ("id", "seller_id", "name", "description", "price", "

```

```

    ↪ cook_time", "created_at", "updated_at") VALUES
(1, 1, 'Burger', 'made from Burger', 20, 20, '2018-12-25 08:59:05.179458', '2018-12-25
    ↪ 08:59:05.179458'),
(2, 1, 'CheeseBurger', 'made from CheeseBurger', 25, 20, '2018-12-25 08:59:05.187246',
    ↪ '2018-12-25 08:59:05.187246'),
(3, 1, 'Chocolate', 'made from Chocolate', 15, 20, '2018-12-25 08:59:05.198679', '
    ↪ 2018-12-25 08:59:05.198679'),
(4, 1, 'Cola', 'made from Soda', 10, 20, '2018-12-25 08:59:05.210367', '2018-12-25
    ↪ 08:59:05.210367'),
(5, 1, 'Sprite', 'made from Soda', 10, 20, '2018-12-25 08:59:05.221047', '2018-12-25
    ↪ 08:59:05.221047'),
(6, 2, 'Burger', 'made from Burger', 20, 20, '2018-12-25 08:59:05.243941', '2018-12-25
    ↪ 08:59:05.243941'),
(7, 2, 'CheeseBurger', 'made from CheeseBurger', 25, 20, '2018-12-25 08:59:05.255126',
    ↪ '2018-12-25 08:59:05.255126'),
(8, 2, 'Chocolate', 'made from Chocolate', 15, 20, '2018-12-25 08:59:05.265752', '
    ↪ 2018-12-25 08:59:05.265752'),
(9, 2, 'Cola', 'made from Soda', 10, 20, '2018-12-25 08:59:05.27669', '2018-12-25
    ↪ 08:59:05.27669'),
(10, 2, 'Sprite', 'made from Soda', 10, 20, '2018-12-25 08:59:05.287987', '2018-12-25
    ↪ 08:59:05.287987'),
(11, 3, 'Burger', 'made from Burger', 20, 20, '2018-12-25 08:59:05.309411', '
    ↪ 2018-12-25 08:59:05.309411'),
(12, 3, 'CheeseBurger', 'made from CheeseBurger', 25, 20, '2018-12-25 08:59:05.32063',
    ↪ '2018-12-25 08:59:05.32063'),
(13, 3, 'Chocolate', 'made from Chocolate', 15, 20, '2018-12-25 08:59:05.331886', '
    ↪ 2018-12-25 08:59:05.331886'),
(14, 3, 'Cola', 'made from Soda', 10, 20, '2018-12-25 08:59:05.343125', '2018-12-25
    ↪ 08:59:05.343125'),
(15, 3, 'Sprite', 'made from Soda', 10, 20, '2018-12-25 08:59:05.354625', '2018-12-25
    ↪ 08:59:05.354625');

```

```

DROP TABLE IF EXISTS "order_items";

```

```

DROP SEQUENCE IF EXISTS order_items_id_seq;

```

```

CREATE SEQUENCE order_items_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE 9223372036854775807
    ↪ START 1 CACHE 1;

```

```

CREATE TABLE "public"."order_items" (
    "id" bigint DEFAULT nextval('order_items_id_seq') NOT NULL,
    "order_id" bigint,

```



```

"merchandise_id" bigint,
"quantity" integer DEFAULT '1',
"created_at" timestamp NOT NULL,
"updated_at" timestamp NOT NULL,
CONSTRAINT "order_items_pkey" PRIMARY KEY ("id"),
CONSTRAINT "fk_rails_4b64fa6392" FOREIGN KEY (merchandise_id) REFERENCES
    ↳ merchandises(id) NOT DEFERRABLE,
CONSTRAINT "fk_rails_e3cb28f071" FOREIGN KEY (order_id) REFERENCES orders(id) NOT
    ↳ DEFERRABLE
) WITH (oids = false);

CREATE INDEX "index_order_items_on_merchandise_id" ON "public"."order_items" USING
    ↳ btree ("merchandise_id");

CREATE INDEX "index_order_items_on_order_id" ON "public"."order_items" USING btree ("
    ↳ order_id");

DROP TABLE IF EXISTS "orders";
DROP SEQUENCE IF EXISTS orders_id_seq;
CREATE SEQUENCE orders_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE 9223372036854775807
    ↳ START 1 CACHE 1;

CREATE TABLE "public"."orders" (
    "id" bigint DEFAULT nextval('orders_id_seq') NOT NULL,
    "customer_id" bigint,
    "time" timestamp,
    "created_at" timestamp NOT NULL,
    "updated_at" timestamp NOT NULL,
    "card_id" bigint,
    "status" character varying,
    "seller_id" bigint,
    CONSTRAINT "orders_pkey" PRIMARY KEY ("id"),
    CONSTRAINT "fk_rails_3dad120da9" FOREIGN KEY (customer_id) REFERENCES customers(id)
        ↳ NOT DEFERRABLE,
    CONSTRAINT "fk_rails_5ee51a3f64" FOREIGN KEY (card_id) REFERENCES cards(id) NOT
        ↳ DEFERRABLE,
    CONSTRAINT "fk_rails_f569184c98" FOREIGN KEY (seller_id) REFERENCES sellers(id) NOT
        ↳ DEFERRABLE
) WITH (oids = false);

```

```

CREATE INDEX "index_orders_on_card_id" ON "public"."orders" USING btree ("card_id");

CREATE INDEX "index_orders_on_customer_id" ON "public"."orders" USING btree ("
    ↪ customer_id");

CREATE INDEX "index_orders_on_seller_id" ON "public"."orders" USING btree ("seller_id"
    ↪ );

DROP TABLE IF EXISTS "pg_search_documents";
DROP SEQUENCE IF EXISTS pg_search_documents_id_seq;
CREATE SEQUENCE pg_search_documents_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE
    ↪ 9223372036854775807 START 1 CACHE 1;

CREATE TABLE "public"."pg_search_documents" (
    "id" bigint DEFAULT nextval('pg_search_documents_id_seq') NOT NULL,
    "content" text,
    "searchable_type" character varying,
    "searchable_id" bigint,
    "created_at" timestamp NOT NULL,
    "updated_at" timestamp NOT NULL,
    CONSTRAINT "pg_search_documents_pkey" PRIMARY KEY ("id")
) WITH (oids = false);

CREATE INDEX "index_pg_search_documents_on_searchable_type_and_searchable_id" ON "
    ↪ public"."pg_search_documents" USING btree ("searchable_type", "searchable_id");

DROP TABLE IF EXISTS "schema_migrations";
CREATE TABLE "public"."schema_migrations" (
    "version" character varying NOT NULL,
    CONSTRAINT "schema_migrations_pkey" PRIMARY KEY ("version")
) WITH (oids = false);

INSERT INTO "schema_migrations" ("version") VALUES
('20181219173022'),
('20181108165412'),
('20181209102221'),
('20181209102156'),

```

```

('20181108165429'),
('20181108164659'),
('20181219131334'),
('20181219172103'),
('20181107183125'),
('20181204145632'),
('20181107182554'),
('20181107180845'),
('20181128154124'),
('20181107184320');

```

```

DROP TABLE IF EXISTS "sellers";
DROP SEQUENCE IF EXISTS sellers_id_seq;
CREATE SEQUENCE sellers_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE 9223372036854775807
    ↪ START 1 CACHE 1;

```

```

CREATE TABLE "public"."sellers" (
    "id" bigint DEFAULT nextval('sellers_id_seq') NOT NULL,
    "name" character varying,
    "likes" integer DEFAULT '0',
    "dislikes" integer DEFAULT '0',
    "auth_token" character varying,
    "created_at" timestamp NOT NULL,
    "updated_at" timestamp NOT NULL,
    CONSTRAINT "index_sellers_on_auth_token" UNIQUE ("auth_token"),
    CONSTRAINT "index_sellers_on_name" UNIQUE ("name"),
    CONSTRAINT "sellers_pkey" PRIMARY KEY ("id")
) WITH (oids = false);

```

```

INSERT INTO "sellers" ("id", "name", "likes", "dislikes", "auth_token", "created_at",
    ↪ "updated_at") VALUES
(1, 'CoolFood', '0', '0', 'Y8BYWrzNKAUSfLUF8n1T2PMs', '2018-12-25 08:59:05.13454', '
    ↪ 2018-12-25 08:59:05.13454'),
(2, 'FakeBurgers', '0', '0', 'fTqC8A6Ab79PyQfLGR9YUyEz', '2018-12-25 08:59:05.232299',
    ↪ '2018-12-25 08:59:05.232299'),
(3, 'FunnyMall', '0', '0', 'SLBm3zKzsJfCHGTa7KUrjKir', '2018-12-25 08:59:05.298075', '
    ↪ 2018-12-25 08:59:05.298075');

```

# ПРИЛОЖЕНИЕ Б

## CI скрипты

Файл генерации контейнера ruby on rails:

```
FROM ruby:2.5
RUN apt-get update -qq && apt-get install -y --no-install-recommends build-essential
    ↪ libpq-dev apt-transport-https \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*
# Install capybara-webkit deps
RUN apt-get update \
    && apt-get install -y --no-install-recommends xvfb qt5-default libqt5webkit5-dev \
    gstreamer1.0-plugins-base gstreamer1.0-tools gstreamer1.0-x \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

# Node.js
RUN curl -sL https://deb.nodesource.com/setup_11.x | bash - \
    && apt-get install -y nodejs

# yarn
RUN curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | apt-key add -\
    && echo "deb https://dl.yarnpkg.com/debian/ stable main" | tee /etc/apt/sources.list.d
    ↪ /yarn.list \
    && apt-get update \
    && apt-get install -y --no-install-recommends yarn \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

RUN mkdir /myapp
WORKDIR /myapp

COPY Gemfile /myapp/Gemfile
COPY Gemfile.lock /myapp/Gemfile.lock
RUN bundle install

COPY package.json /myapp/package.json
COPY yarn.lock /myapp/yarn.lock
```

```
RUN yarn install
```

```
RUN yarn upgrade
```

## Файл композиции контейнеров приложения:

```
version: '3'

services:
  db:
    image: postgres
    volumes:
      - ./app/tmp/db:/var/lib/postgresql/data
  web:
    image: kolaer/feedem_web:backend
    command: bundle exec rails s -p 3000 -b '0.0.0.0'
    volumes:
      - ./app:/myapp
    ports:
      - "3000:3000"
    depends_on:
      - db
      - webpacker
  webpacker:
    image: kolaer/feedem_web:backend
    command: bundle exec bin/webpack
    volumes:
      - ./app:/myapp
    ports:
      - "3050:8080"
  adminer:
    image: adminer
    restart: always
    ports:
      - "4352:8080"
```

# ПРИЛОЖЕНИЕ В

## Пример контроллера Ruby on Rails

```
class WelcomeController < ApplicationController
  def index
    @merchandises = Merchandise.includes(:seller)

    @merchandises = @merchandises.map { |merch|
      res = merch.attributes
      res[:seller] = merch.seller
      res
    }
    @merchandises.reverse!

    @sellers = []

    Seller.all.sort_by { |seller|
      (seller.likes - seller.dislikes + 4)/(seller.likes + seller.dislikes + 2)
    }.each do |seller|
      @sellers << {
        id: seller.id,
        name: seller.name,
        likes: seller.likes,
        dislikes: seller.dislikes
      }
    end
  end

  def merch
    @merchandises = Merchandise.all
  end

  def sellers
    @sellers = []

    Seller.all.sort_by { |seller|
      (seller.likes - seller.dislikes + 4)/(seller.likes + seller.dislikes + 2)
    }.reverse.each do |seller|
      @sellers << {
```

```
      id: seller.id,  
      name: seller.name,  
      likes: seller.likes,  
      dislikes: seller.dislikes  
    }  
  end  
end  
end
```



# ПРИЛОЖЕНИЕ Г

## Пример модели Ruby on Rails

```
class Seller < ApplicationRecord
  include PgSearch

  has_secure_token :auth_token
  has_many :merchandise, dependent: :destroy
  has_many :order, dependent: :destroy

  pg_search_scope :search_by_name,
    against: :name,
    using: { tsearch: { prefix: true } },
    ranked_by: ':trigram'

  validates :name, presence: true

  validates :name, length: { minimum: 3 }
end
```