МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных

технологий

Специализация 1 –40 01 01 Программное обеспечение информационных

технологий (программирование интернет – изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Социальная платформа для программистов»

Выполнил студент Чернявский Алексей Леонидович

(Ф.И.О.)

Руководитель проекта пр.-стаж. Бурмакова А.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: пр.-стаж. Бурмакова А.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: пр.-стаж. Бурмакова А.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2020

Реферат

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Чернявский А.Л.

Провер.

Бурмакова А.В.

Н. контр.

Бурмакова А.В.

Утверд.

Бурмакова А.В.

Реферат

Лит.

Листов

1

БГТУ 71171075, 2020

Пояснительная записка курсового проекта содержит 30 страниц пояснительной записки, 23 иллюстраций, 8 источников литературы, 2 приложения.

NODE.JS, JAVASCRIPT, MONGODB, EXPRESS.JS, MONGOOSE, MVC

Основная цель курсового проекта: разработка кроссплатформенной социальной платформы для программистов.

В первой главе проводится аналитический обзор литературы по тематике курсового проекта и содержит анализ методов решения поставленных задач.

Вторая глава посвящена процессу проектирования системы.

В третьей главе приведено обоснование технических приёмов.

В четвертой главе описано тестирование, позволяющее подробно понять интерфейс программного средства.

В заключении приведены результаты проделанной работы.

Abstract

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Чернявский А. Л.

Провер.

Бурмакова А. В

Н. контр.

Бурмакова А.В.

Утверд.

Бурмакова А.В.

Abstract

Лит.

Листов

1

БГТУ 71171075, 2020

The explanatory note of the course project contains 30 pages of the explanatory note, 23 illustrations, 8 sources of literature, 3 appendices.

NODE.JS, JAVASCRIPT, MICROSOFT SQL SERVER, EXPRESS.JS, TEDIOUS, SEQUELIZE, MVC

The main goal of the course project: development of cross-platform software for order list management.

The first chapter provides an analytical review of the literature on the subject of the course project and contains an analysis of methods for solving the tasks.

The second chapter is devoted to the system design process.

The third chapter provides the rationale for technical techniques.

The fourth chapter describes testing, which allows you to understand in detail the interface of a software tool.

In conclusion, the results of the work done are presented.

Содержание

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Чернявский А.Л.

Провер.

Бурмакова А.В.

Н. контр.

Бурмакова А.В.

Утверд.

Бурмакова А.В.

Содержание

Лит.

Листов

1

БГТУ 71171075, 2020

[Введение 7](#_Toc41336698)

[1 Аналитический обзор литературы 8](#_Toc41336699)

[2 Разработка программного средства 11](#_Toc41336700)

[2.1 Проектирование структуры базы данных 11](#_Toc41336701)

[2.2 Проектирование модели базы данных 12](#_Toc41336702)

[2.3 Проектирование структуры сервера 13](#_Toc41336703)

[2.4 Проектирование приложения клиента 14](#_Toc41336704)

[3 Обоснование технических приемов программирования 16](#_Toc41336705)

[4 Тестирование, проверка работоспособности 18](#_Toc41336706)

[Заключение 29](#_Toc41336707)

[Список литературы 30](#_Toc41336708)

[Приложение А 31](#_Toc41336709)

[Приложение Б 32](#_Toc41336710)

Введение

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Чернявский А.Л.

Провер.

Бурмакова А.В.

Н. контр.

Бурмакова А.В.

Утверд.

Бурмакова А.В.

Введение

Лит.

Листов

1

БГТУ 71171075, 2020

Целью данной работы была разработка кроссплатформенного приложения на тему «Социальная платформа для программистов». Приложение должно быть разработано с помощью языка JavaScript и программной платформы Node.js реализуя клиент-серверную архитектуру.

Node.js – программная платформа, основанная на движке Chrome V8. Node.js асинхронен и событийно-ориентирован [1]. Данная платформа позволяет JavaScript взаимодействовать с устройствами ввода-вывода через свой API, подключать другие внешние библиотеки. Node.js предназначен для построения масштабируемых сетевых приложений, преимущественно серверов.

Npm – менеджер пакетов, входящий в состав Node.js, позволяющий расширить возможности приложения.

Для работы с Node.js были выбраны следующий пакеты:

* Express.js – предоставляет собой слой фундаментальных функций для упрощения разработки веб-приложений;
* Body-parser – пакет для парсинга тела запросов;
* Mongoose – ODM-библиотека (Object Data Modelling) для работы с MongoDB;

В качестве СУБД для базы данных была выбрана MongoDB. Для приложения создан кластер базы данных с помощью MongoDB Atlas.

Написанное приложение строится на клиент-серверной архитектуре. Для создания пользовательского интерфейса использовалась библиотека React. Для сервера был выбрана платформа Node.js, в связи с кроссплатформенностью, простотой развертки сервера и гибкости работы за счёт npm.

Постановка задачи и обзор аналогов

ФИО

Подпись

Дата

Лист

1

КП 01.00.ПЗ

Разраб.

Чернявский А.Л.

Провер.

Бурмакова А.В.

Н. контр.

Бурмакова А.В.

Утверд.

Бурмакова А.В.

1 Постановка задачи и обзор аналогов

Лит.

Листов

2

БГТУ 71171075, 2020

1.1 Постановка задачи

Для описания функциональных требований, определяющих функции, которые должно выполнять программное средство, и эксплуатационные требование, определяющие характеристики его функционирования следует сначала дать определение, что есть «социальная платформа» или «социальная веб-платформа».

Социальная веб-платформа – это программный продукт на базе и(возможно) с помощью которого пользователь или группа пользователей могут создать управляемое им (ими) же социальное веб-пространство. Это целевое пространство, в свою очередь, является основой для образования собственной (целевой) социальной сети.

Кратко можно сделать вывод, что приложение должно быть направлено на пользователей и взаимоотношения между ними.

Предполагается, что пользователями этой платформы будут являться программисты, QA-инженеры, менеджеры – люди, которые связаны с работой над IT-продуктами. Исходя из целевой аудитории программного продукта можно сделать вывод: аналогами будут являться различные IT-форумы.

1.2 Обзор аналогов

Одним из аналогов социальной платформы для программистов является «Форум программистов и сисадминов Киберфорум». Главная страница веб-приложения представлена на рисунке 1.1.

Функционал приложения включает в себя:

* регистрация и авторизация;
* блоги и форумы программистов и сисадминов;
* посты и комментарии к ним.

Стоит отметить, что форум был создан еще в 2000 году и с тех пор дизайн не менялся. Из-за этого лояльность к приложению у молодого поколения довольно низкая.

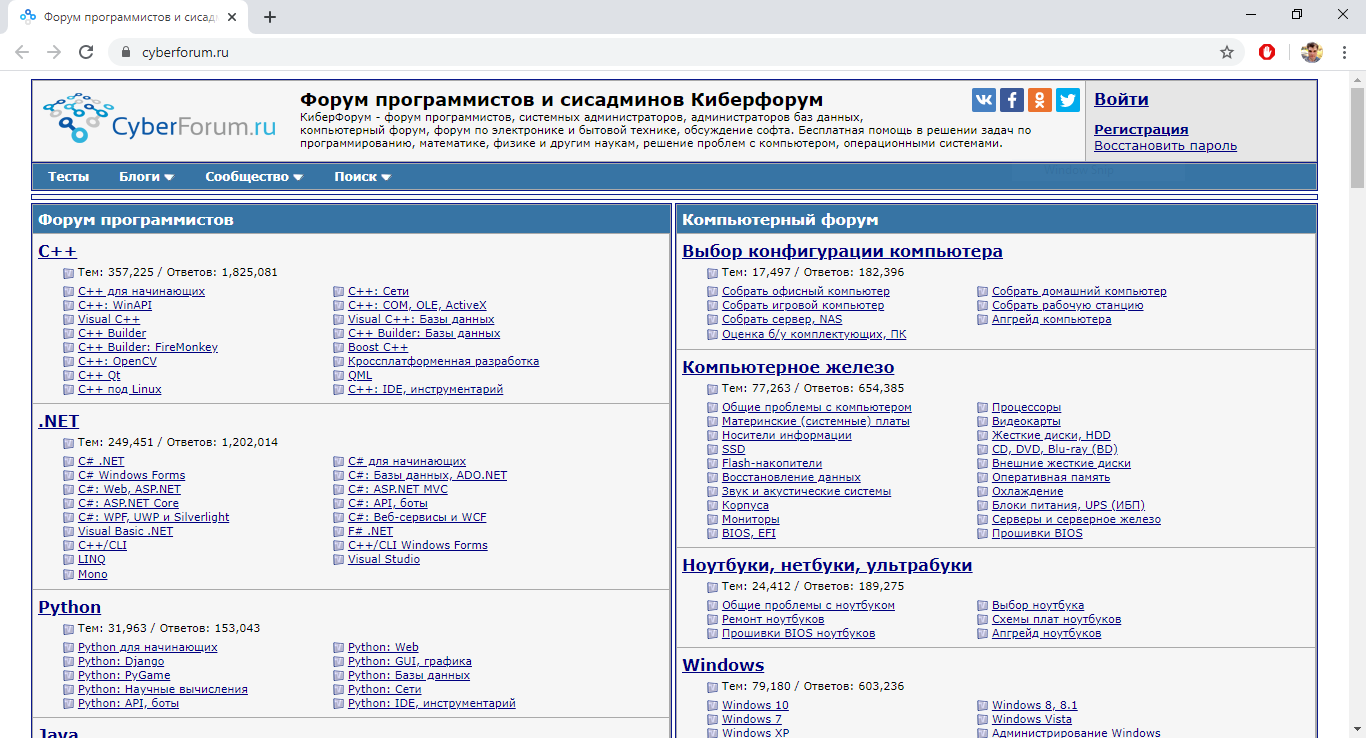


Рисунок 1.1 – «Форум программистов и сисадминов Киберфорум»

Еще один из ресурсов, на которые бы стоило обратить внимание – это «Хабр». Главную страницу можно увидеть на рисунке 1.2.

В самом начале работы это была небольшая новостная лента, но за 12 лет она превратилась в крупную площадку с многомиллионной аудиторией и необычными сервисами.

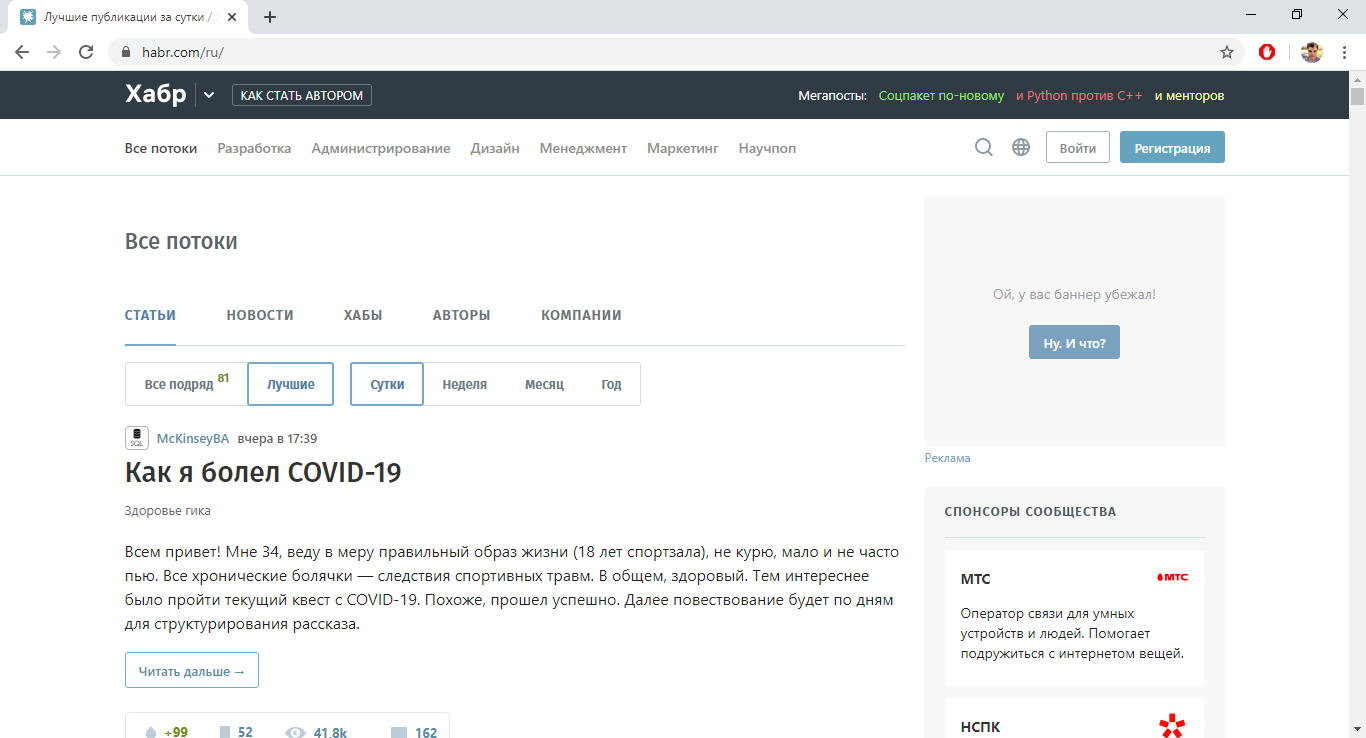


Рисунок 1.2 – «Хабр» - веб-приложение habr.com

Пользователи пишут сюда интересные материалы на разные темы и комментируют их. Тут можно задавать вопросы и получать ответы, обсуждать актуальные темы, искать работу или фриланс и напрямую общаться с крупными компаниями.

Одной из ключевых особенностей является открытость постов и комментариев к ним, быстрая регистрация и авторизация, user-friendly интерфейс.

Отличием нашего приложение от аналогов является сокрытие постов от незарегистрированных пользователей для большей приватности сообщества и тесного сотрудничества именно внутри системы.

Проектирование программного средства

ФИО

Подпись

Дата

Лист

1

КП 02.00.ПЗ

Разраб.

Чернявский А.Л.

Провер.

Бурмакова А.В.

Н. контр.

Бурмакова А.В.

Утверд.

Бурмакова А.В.

2 Разработка программного средства

Лит.

Листов

5

БГТУ 71171075, 2020

2.1 Проектирование серверной и клиентской частей

Серверная часть написана на Node.js с помощью Express, Mongoose, Socket.io. Node.js – это кроссплатформенная среда исполнения JS, которая выполняет JavaScript вне web-браузера. Node.js позволяет разработчикам использовать JavaScript для написания сценариев на стороне сервера, а также для создания динамического содержимого веб-страницы перед ее отправкой в web-браузер пользователя. Таким образом, мы объединяем разработку web-приложения вокруг одного языка программирования, а не разных языков для серверных и клиентских сценариев.

Express – это минимальная и гибкая среда web-приложений Node.js, которая предоставляет надежный набор функций для приложений.

Mongoose - ODM-библиотека (Object Data Modelling) для работы с MongoDB, которая позволяет сопоставлять объекты классов и документы коллекций из базы данных.

Socket.io – JS-библиотека для web-приложений и обмена данными в реальном времени поверх протокола WebSocket.

Приложение имеет клиент-серверную архитектуру (Рис. 2.1)

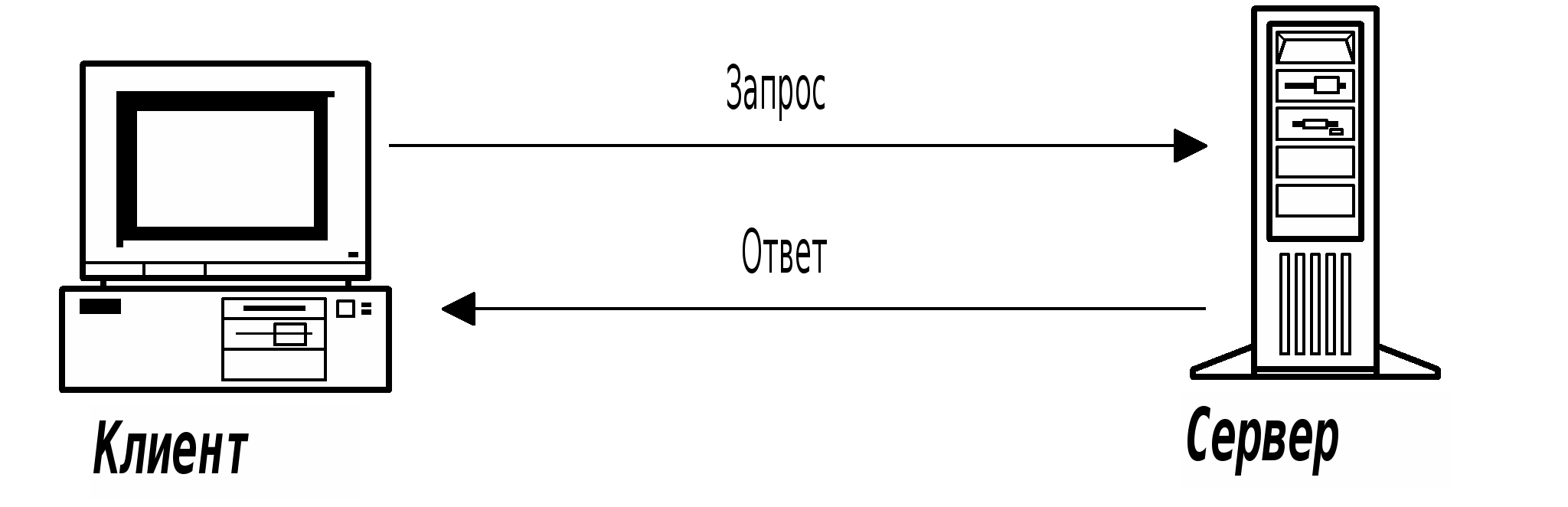


Рисунок 2.1 – Клиент-серверная архитектура

Клиент является браузерным приложением, которое посылает запросы (request) по сети с помощью протокола HTTP, а сервер должен принимать запросы от клиента и возвращать ответ (response). В нашем случае приложение разделено на две части. Это front-end (клиентская) и back-end (серверная).

Клиентская часть разработана с помощью библиотеки React. Следует отметить, что клиентская часть приложения имеет архитектуру Single Page Application (SPA-архитектуру). Отличия от классической Multi Page архитектуры можно рассмотреть на рисунке 2.2.

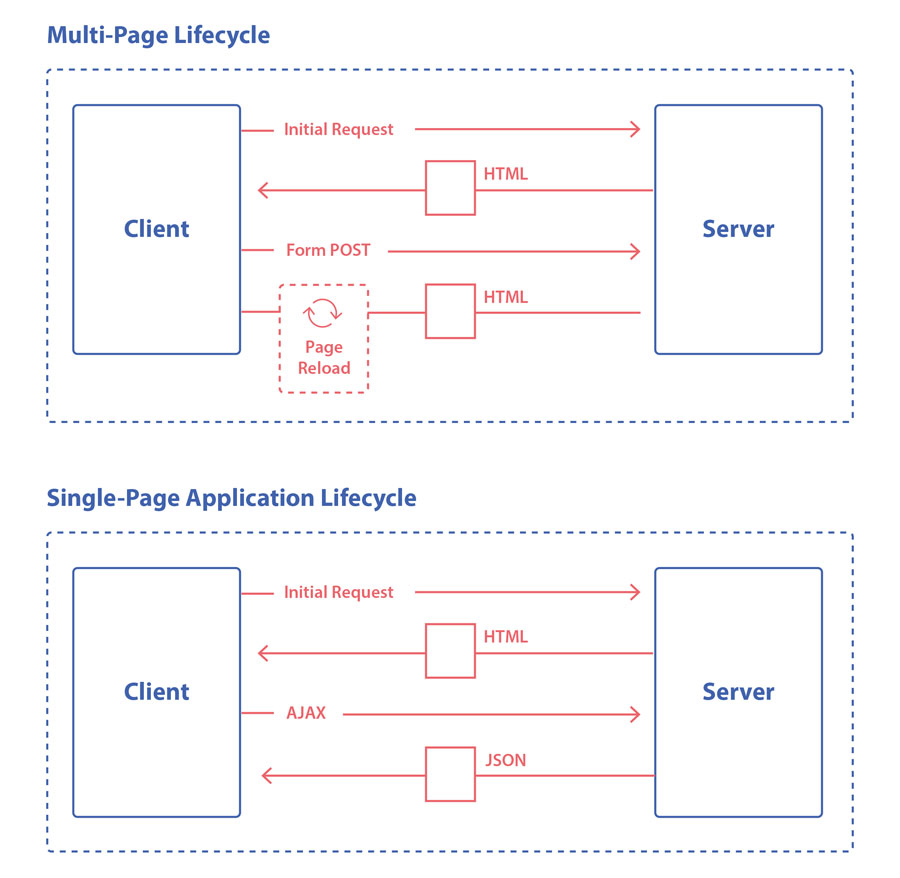


Рисунок 2.2 – Отличия SPA от Multi Page

Это значит, что в SPA архитектуре клиент запрашивает основную HTML-страницу только один раз, а затем общается с сервером с помощью AJAX запросов, при этом перезагрузки страницы не происходит, разметка меняется динамически.

В традиционных приложениях сервер при каждом запросе должен отсылать HTML-страницу.

Стоит отметить, что при разработке приложения учитывался MVC шаблон.

Модель предоставляет остальным компонентам приложения объектно-ориентированное отображение данных (например, данные о пользователе, список постов, список комментариев). Объекты модели осуществляют загрузку и сохранение данных в базе данных и реализуют бизнес-логику.

На рисунке 2.3 можно увидеть взаимодействие между слоями MVC-приложения.

Так как в нашем случае в клиентской части используется SPA, то View будет являться JSON-ответ вместо отдельной HTML-страницы под конкретный action.

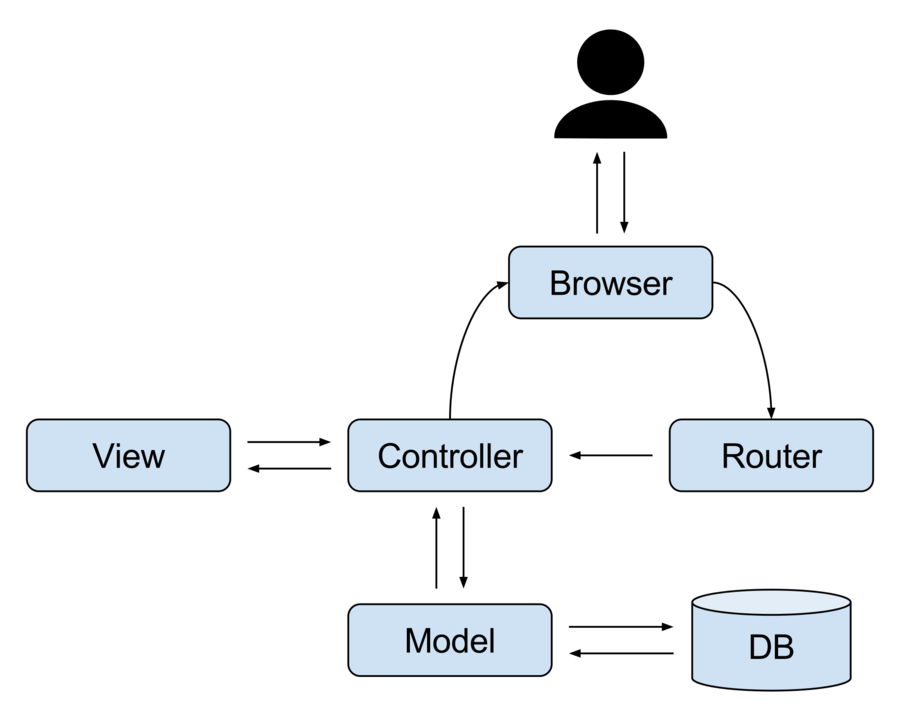


Рисунок 2.3 – Взаимодействие между слоями MVC

Структуру файлов и папок приложения можно увидеть на рисунке 2.4.

В папке «config» находятся файлы конфигурации, а также файл подключения к базе данных, «routes» – маршрутизация и контроллеры, «models» – модели данных, «middleware» – промежуточная обработка некоторых запросов, например тех, которые должны выполняться только для авторизованных пользователей. Файл «server.js» является точкой запуска приложения.

В папке «client» расположены файлы клиентской части. Приложение-клиент представляет собой web-приложение, написанное на языках JavaScript, HTML, CSS. Для рендеринга страниц используется React. React – JavaScript-библиотека для создания пользовательских интерфейсов. Создаются простые представления для каждого состояния вашего приложения, и React будет эффективно обновлять и отрисовывать только нужные компоненты при изменении данных. Компоненты React реализуют метод render(), который принимает входные данные и возвращает данные для отображения. Используется XML-подобный синтаксис JSX. В нашем приложении используется функциональные компоненты. Для отслеживания состояния используется технология Redux.



Рисунок 2.4 – Структура файлов и папок приложения

Для отправки запросов с SPA приложения разработаны «end-points». Их можно увидеть в приложении А

2.2 Структура моделей данных

MongoDB – это документарная база данных. Вместо хранения данных в таблицах, состоящих из отдельных строк, как в реляционных базах, MongoDB сохраняет данные в коллекциях, состоящих из документов. Документ – это большой JSON объект без заранее определенного формата и схемы.

Для приложения был создан кластер MongoDB в облачном сервисе MongoDB Atlas. Используются три коллекции (Рис. 2.5)

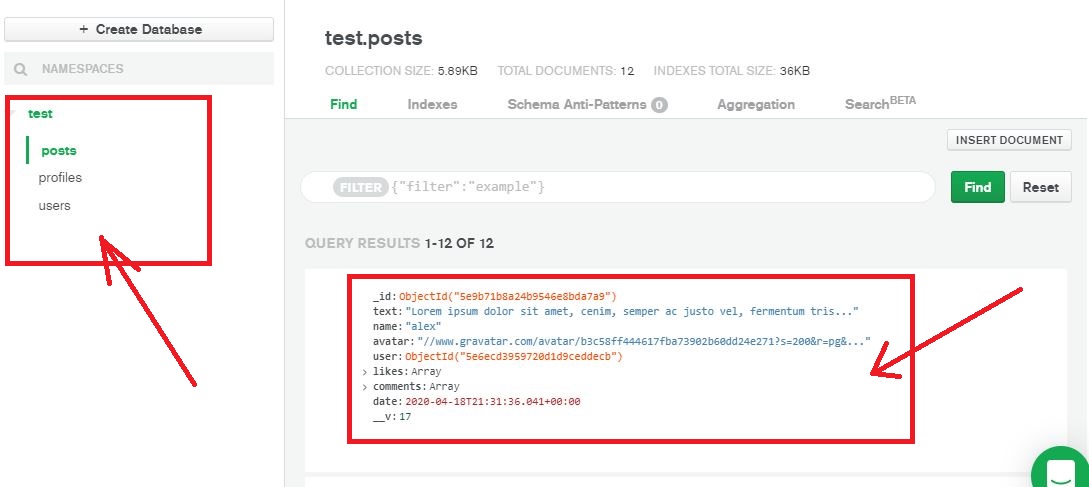


Рисунок 2.5 – Кластер в MongoDB Atlas

Схема в Mongoose определяет метаданные модели – ее свойства, типы данных и ряд другой информации. Схема данных представлена в Приложении Б. Она содержит три основных коллекции:

* posts – посты;
* profiles – профили пользователей (содержит данные о пользователях, информацию об образовании и прошлых проектах и т.д.);
* users – пользователи (данные для входа в систему – заполняются при регистрации и авторизации)

Разработка функциональной модели и модели данных ПС

ФИО

Подпись

Дата

Лист

1

КП 03.00.ПЗ

Разраб.

Чернявский А.Л.

Провер.

Бурмакова А.В.

Н. контр.

Бурмакова А.В.

Утверд.

Бурмакова А.В.

3 Разработка функциональной модели и модели данных ПС

Лит.

Листов

2

БГТУ 71171075, 2020

3.1 Выполняемые функции

Разрабатываемое приложение представляет собой социальную платформу для программистов, где главная цель – это обсуждение каких-либо тем. В результате были созданы необходимые модели для создания постов и комментариев к ним. У каждого пользователя есть возможность заполнить информацию о себе (о своих предыдущих и текущих проектах, об образовании).

В приложении есть система оценивания постов в виде «лайков».

Также добавлены уведомления при действиях со стороны других пользователей («лайки», комментарии).

Функционально программное средство должно выполнять следующие задачи:

* регистрация и авторизация пользователей;
* добавление и редактирование информации об образовании и проектах пользователя;
* создание, изменения, удаление постов и комментариев к ним;
* возможность оценивать комментарии «лаками»;
* уведомления пользователей о действиях со стороны других пользователей.

При авторизации используется JSON Web Token (JWT), при любом запросе клиент должен его посылать на сервер. JWT имеет некоторые преимущества – он самодостаточен, все необходимые данные для аутентификации можно хранить прямо в самом токене.

Мы можем рассмотреть устройство токена. JWT состоит из трёх основных частей: заголовка (header), нагрузки (payload) и подписи (signature). Заголовок и нагрузка формируются отдельно, а затем на их основе вычисляется подпись с помощью npm-пакета «jsonwebtoken». После первого входа в веб-приложение клиенту возвращается сгенерированный сервером токен. На стороне клиента этот токен кладется в LocalStorage браузера. При каждом последующем запросе токен извлекается из LocalStorage и посылается в HTTP-заголовке ‘x-auth-token’. На стороне сервера пришедший токен проверяется с помощью метода «verify» библиотеки. В приложении Д находится middleware по проверке авторизации пользователя.

3.2 Модель данных

В рамках курсового проекта было разработано три коллекции.

Коллекция users хранит в себе информацию о пользователях для входа в систему:

* name – ;
* email – ;
* password – ;
* avatar – ;
* date –

Тестирование, проверка работоспособности

ФИО

Подпись

Дата

Лист

1

КП 04.00.ПЗ

Разраб.

Чернявский А.Л.

Провер.

Бурмакова А.В.

Н. контр.

Бурмакова А.В.

Утверд.

Бурмакова А.В.

4 Тестирование, проверка работоспособности и анализ данных

Лит.

Листов

11

БГТУ 71171075, 2020

Перейдя на начальную страницу сайта, мы увидим две основные части: список заказов и информация о заказах. Так как заказ еще не выбран, то пользователь будет оповещен, что страница пока что пуста. Начальная страница продемонстрирована ниже, на рисунке 4.1.

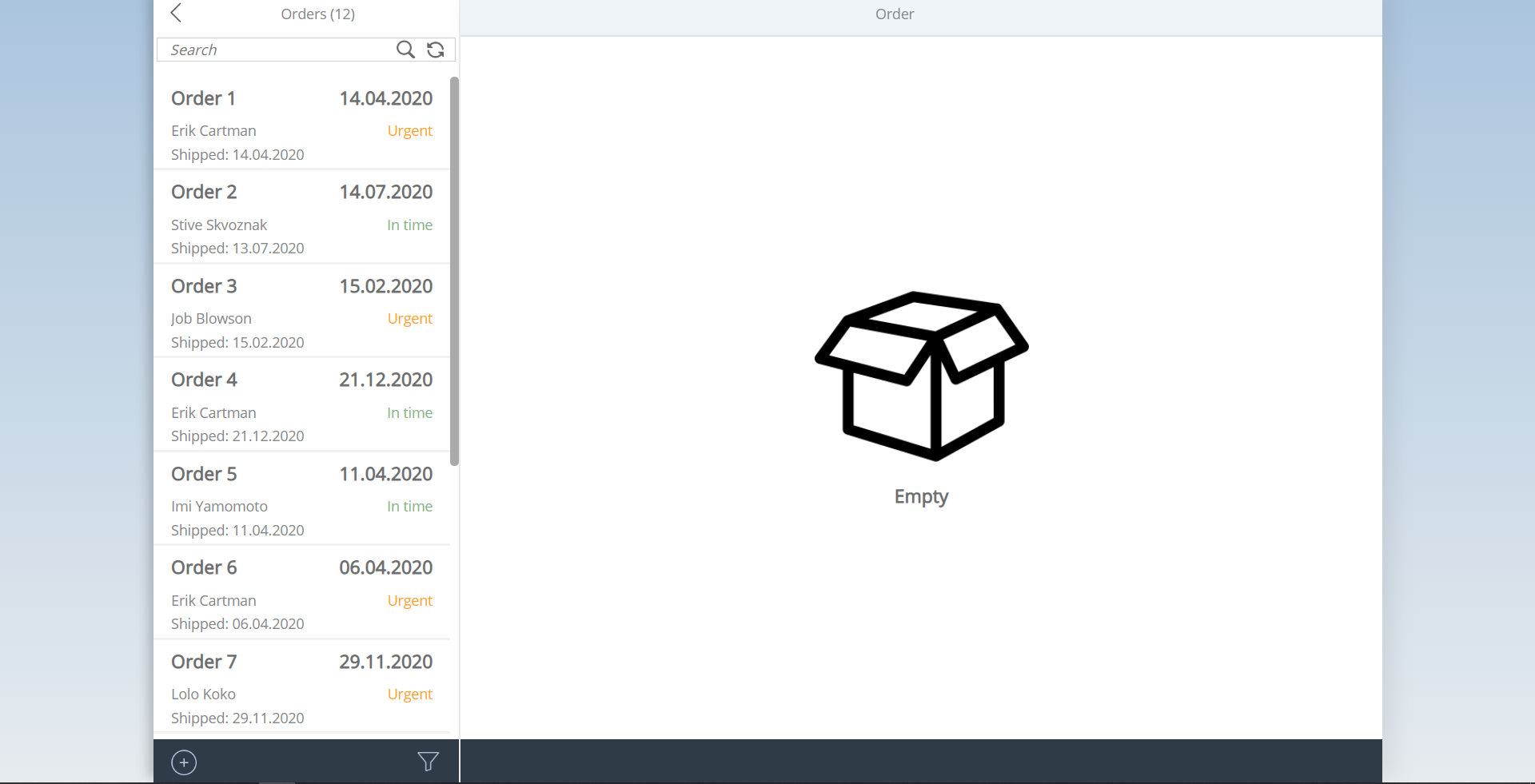


Рисунок 4.1 – Страница авторизации

Список заказов имеет функцию поиска, введя в поле искомую информацию и нажав на кнопку поиска, пользователь получит информацию по заданному критерию, если он имеется в базе данных. Если же искомый заказ отсутствует, то пользователь будет об этом оповещен. Пример отсутствия искомого заказа приведен на рисунке 4.2.

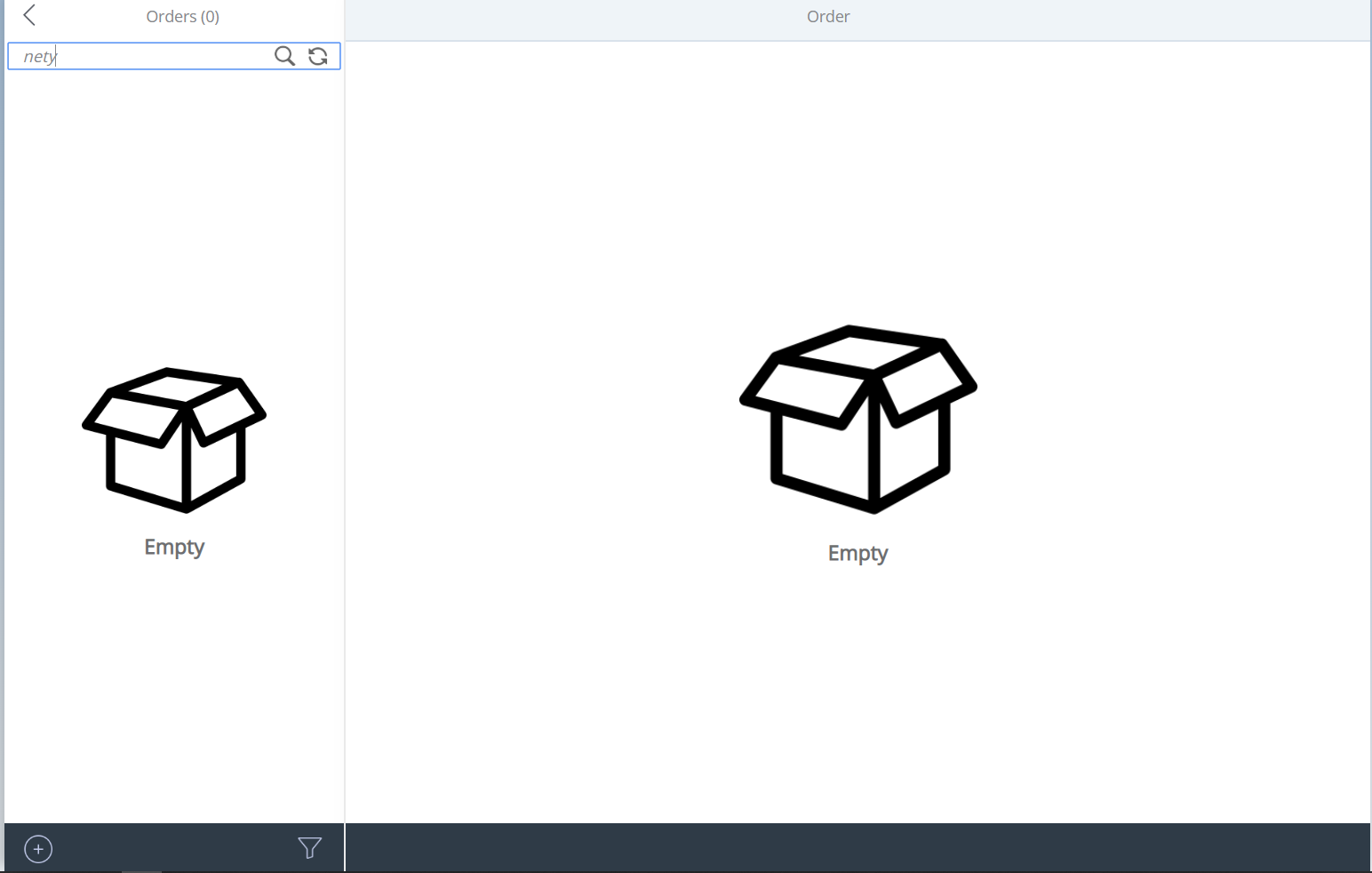


Рисунок 4.2 – Пример отсутствия искомого заказа

По дефолту поиск осуществляется по номеру заказа, имени заказчика, дате доставки и дате принятия заказа. Если же требуется произвести поиск по конкретному признаку, то по нажатию кнопки фильтра, в нижнем правом углу списка заказов, можно будет выбрать конкретный:

* All;
* Order №;
* Customer Name;
* Ship Date;
* Accepte Date.

Для выбора параметра будет открыто окошко на странице. Пример фильтрации поиска заказа приведен на рисунке 4.3.

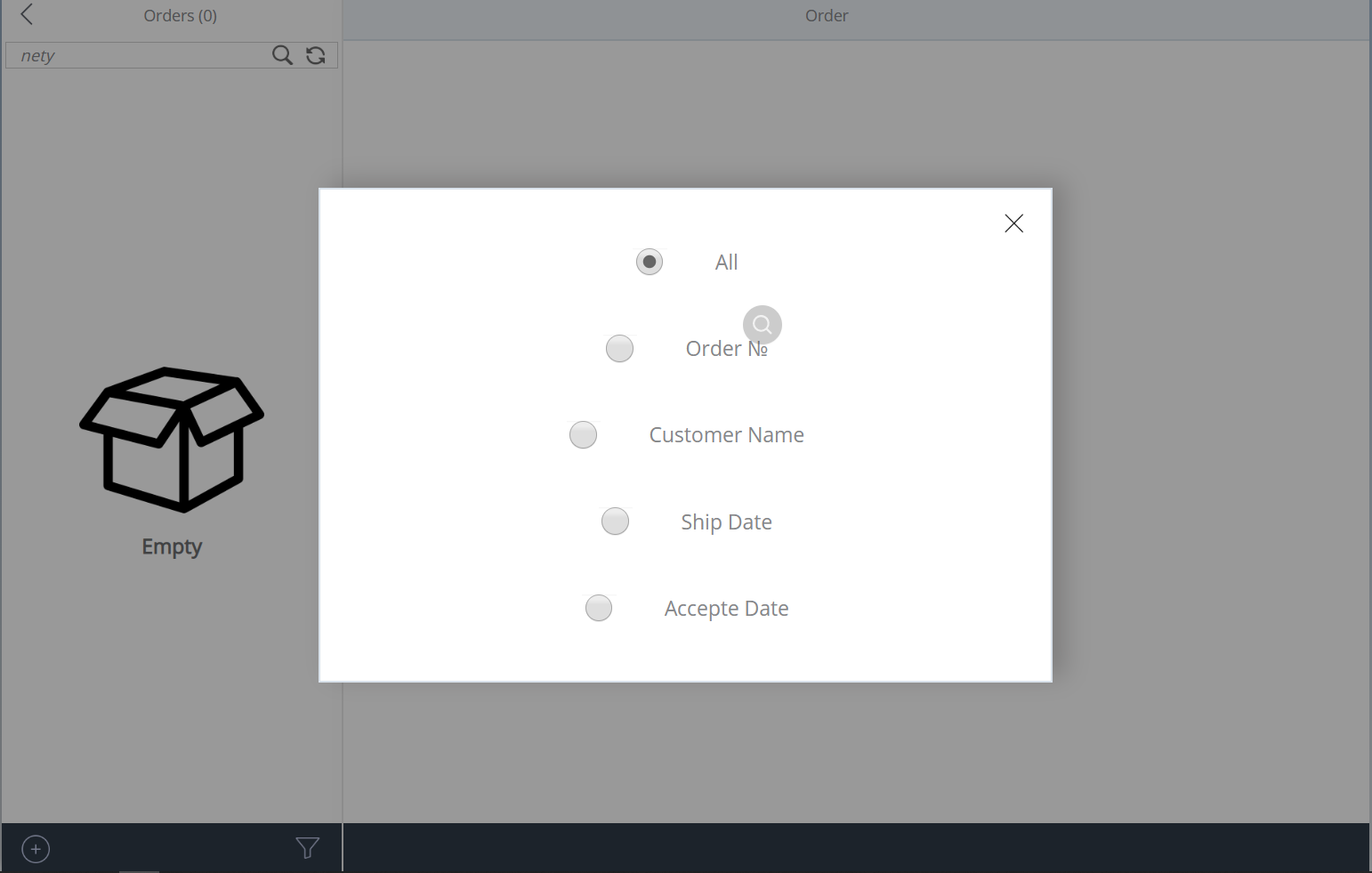


Рисунок 4.3 – Фильтрация поиска заказа

Далее нажмем на кнопку левее от кнопки фильтрации поиска, для создания нового заказа. Создание нового заказа представлено на рисунке 4.4.

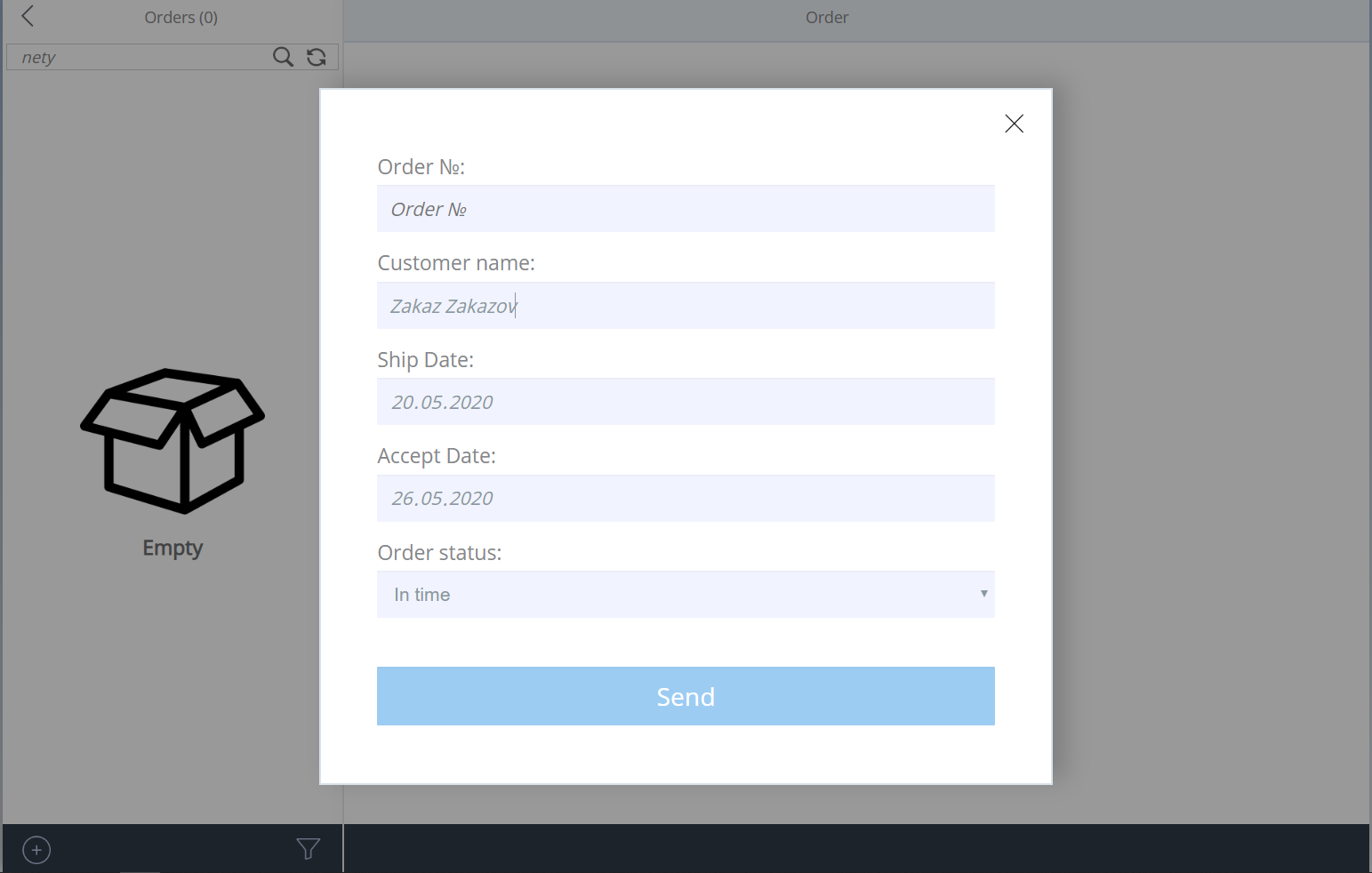


Рисунок 4.4 – Создание нового заказа

Все поля, кроме номера заказа, обязательны к заполнению. Номер заказа должен иметь уникальное значение, которое должен ввести пользователь, если же он оставит его пустым, то в базе данных номер будет назначен автоматически.

После, нажав на кнопку отправки, откроем добавленный нами заказ. Первоначально, в нём нет никакой информации, её необходимо заполнить самим. Пример окна заказа на рисунке 4.5.

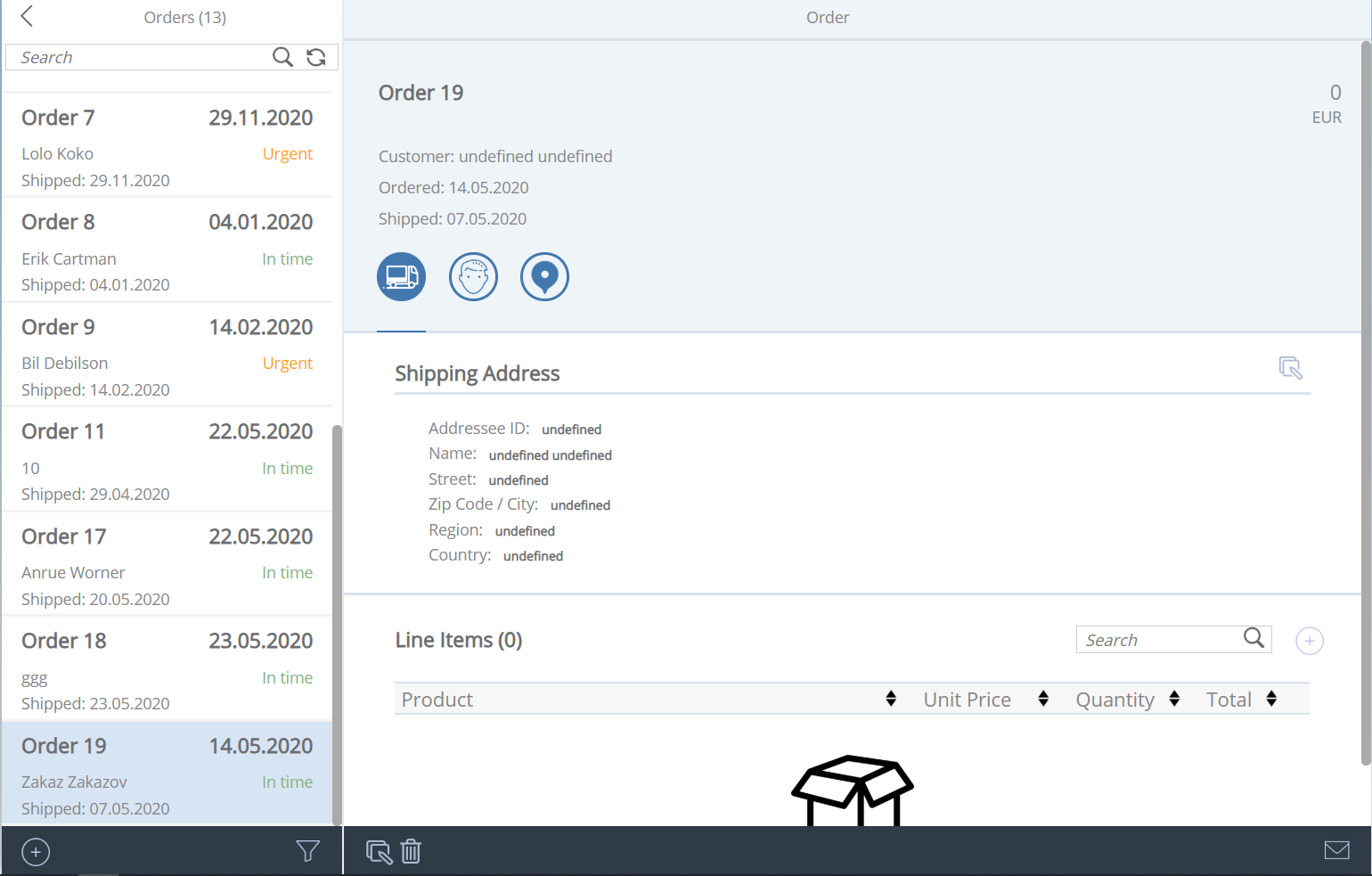


Рисунок 4.5 – Окно заказа

В окне заказа есть 3 кнопки, по нажатию на которые, будет отображена информация, связанная с заказом, а именно:

* информация об адресате и адресе доставки;
* информация об обработчике заказа;
* адрес доставки, отображаемый на карте.

Сначала добавим информацию об адресате и адресе доставки, выбрав первый пункт и нажав на кнопку редактирования справа. Появится окно, где необходимо выбрать по идентификатору адресата и отправить данные. Окно выбора адресата представлено на рисунке 4.6.

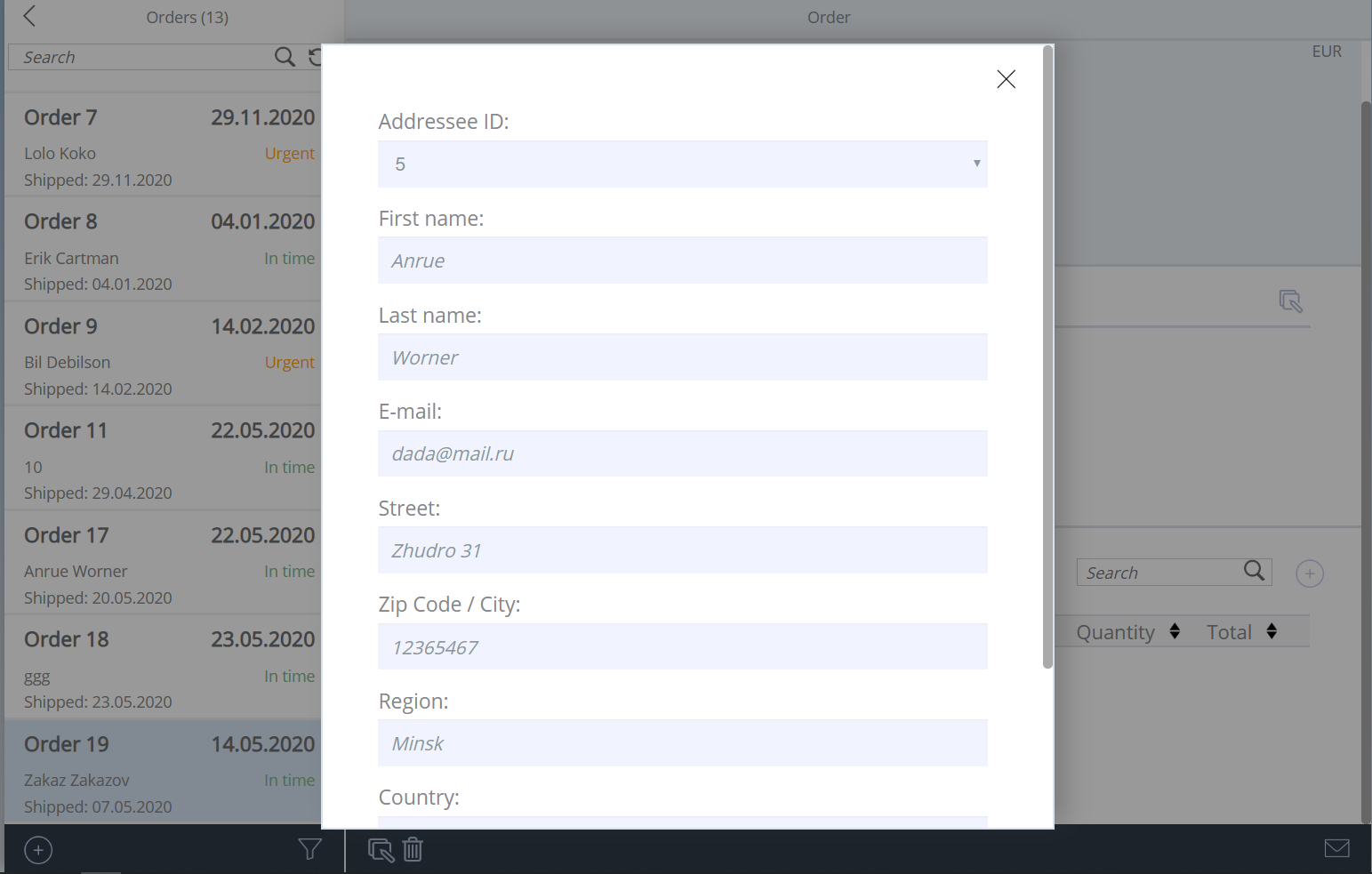


Рисунок 4.6 – Окно выбора адресата

После добавления информации об адресате, таким же образом можно эту же информацию изменить

Далее добавим информацию об обработчике заказа, выбрав второй пункт и нажав на кнопку редактирования справа. Появится окно, где необходимо выбрать по идентификатору обработчика и отправить данные. Окно выбора обработчика представлено на рисунке 4.7.

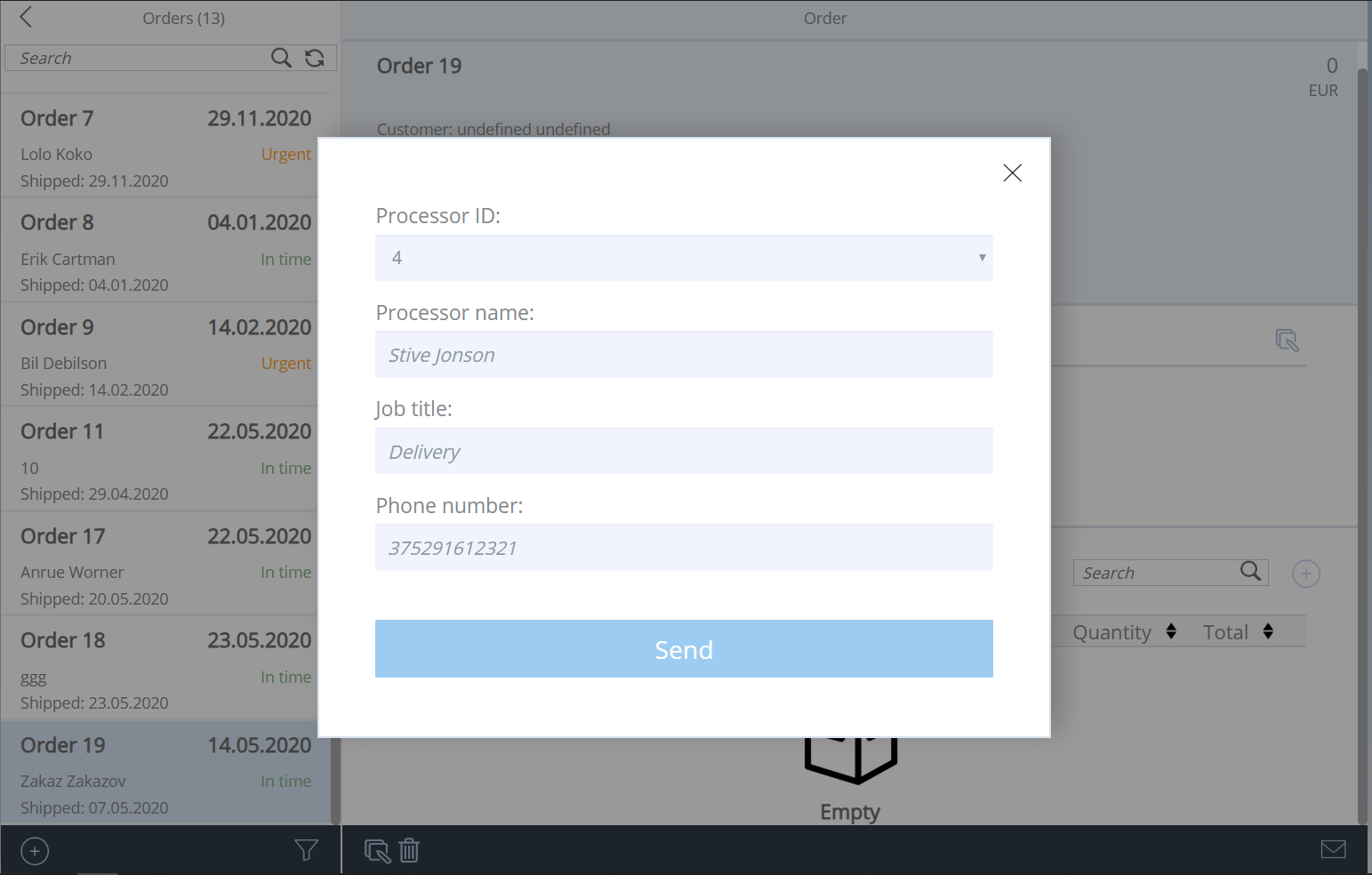


Рисунок 4.7 – Окно выбора обработчика

После добавления информации об обработчике, таким же образом можно эту же информацию изменить.

После добавления информации о заказе, можно рассмотреть, как выглядят эти данные. К примеру, в разделе обработчика находится его номер, по нажатию на который, будет предложено совершить звонок. Пример набора номера приведен на рисунке 4.8.

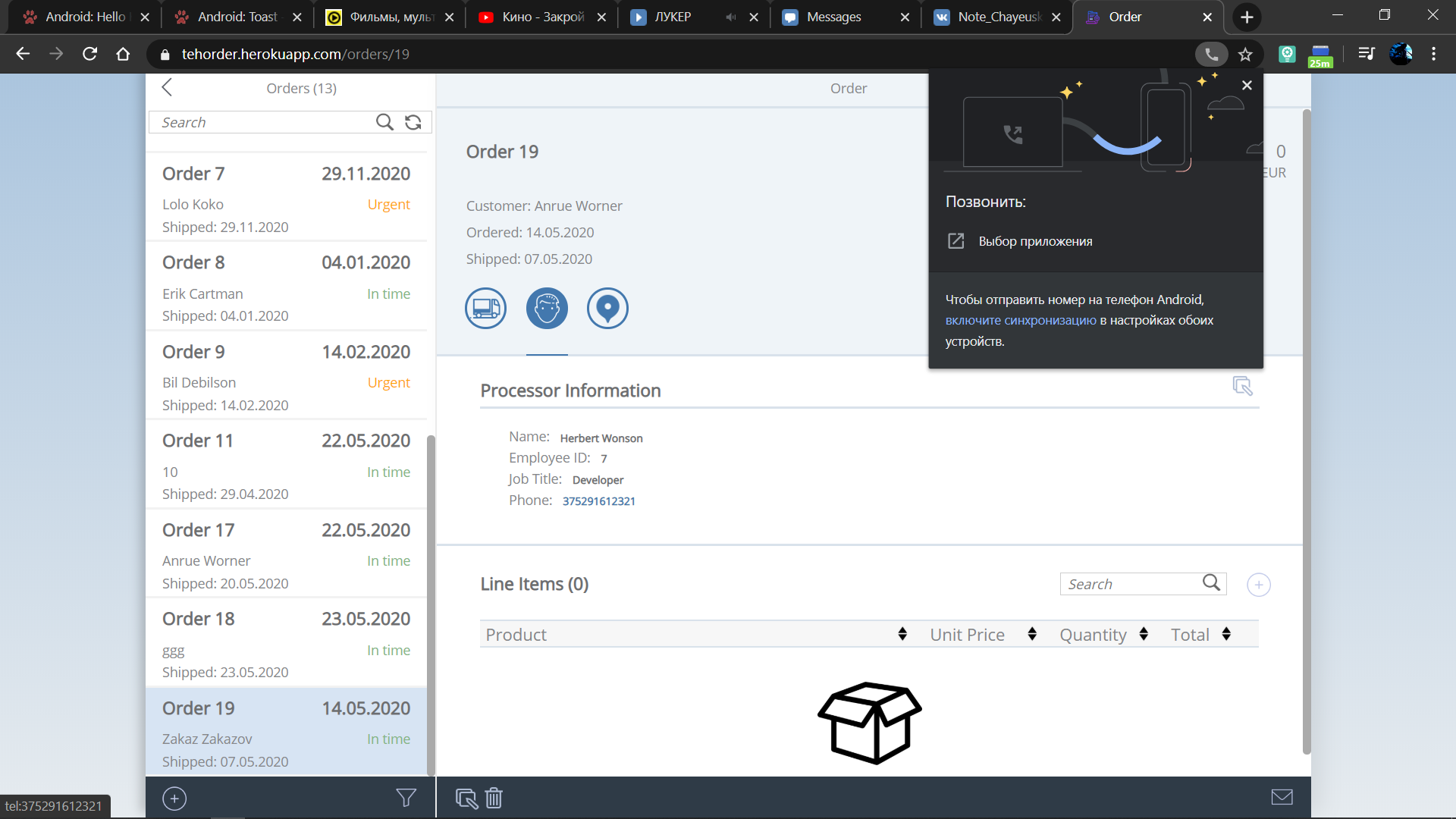


Рисунок 4.8 – Пример набора номера

В нижнем правом углу есть кнопка почты, по нажатию на которую, будет предложено отправить e-mail сообщение адресату. Пример отправки e-mail приведен на рисунке 4.9.

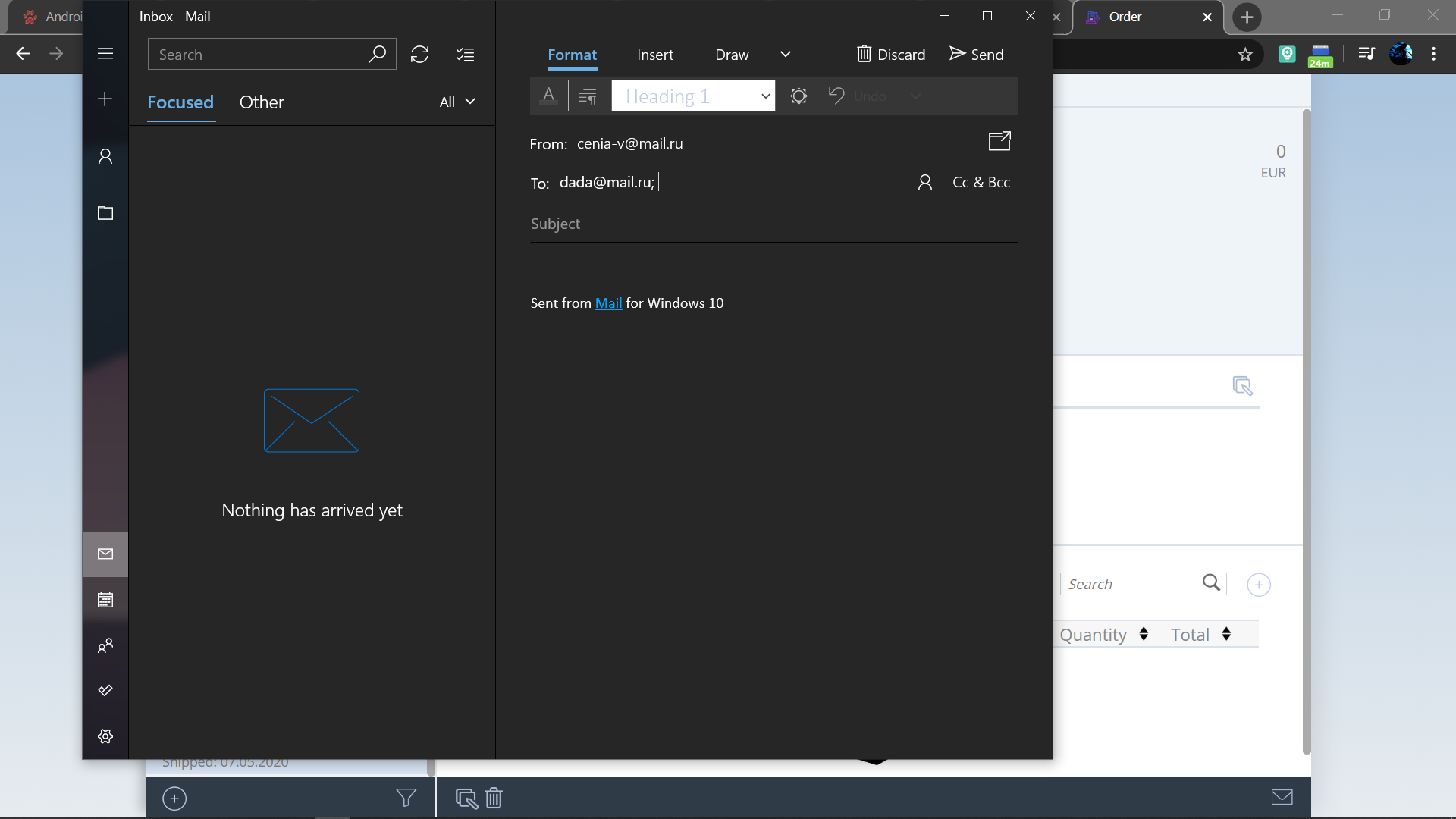


Рисунок 4.9 – Пример отправки e-mail

Выбрав третью кнопку в информации о заказе, будет открыта карта, отображающая адрес доставки. Пример отображения адреса на карте приведен на рисунке 4.10.

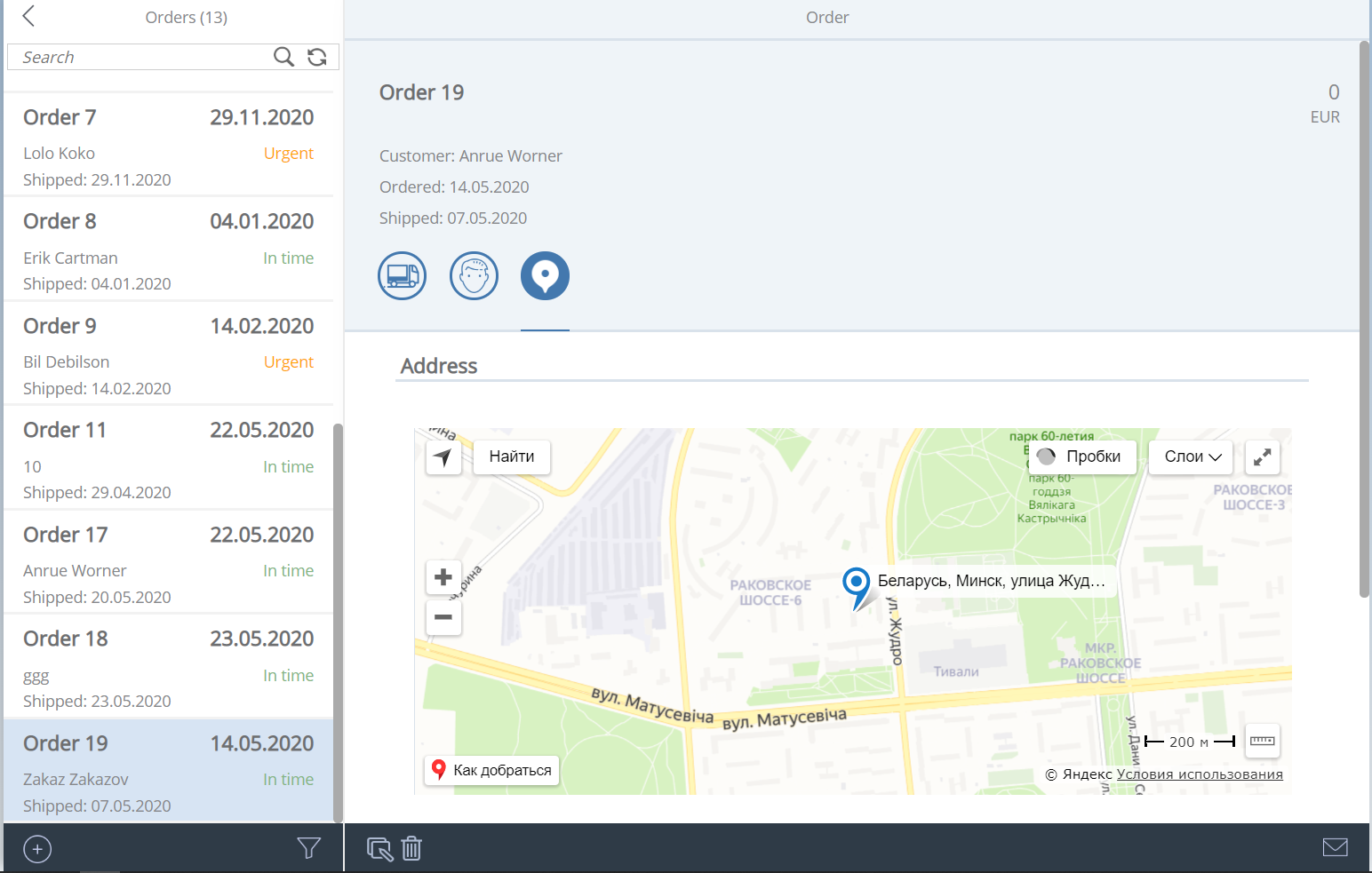


Рисунок 4.10 – Отображение адреса на карте

Далее рассмотрим список товаров в заказе. Пример списка товаров в заказе приведен на рисунке 4.11.

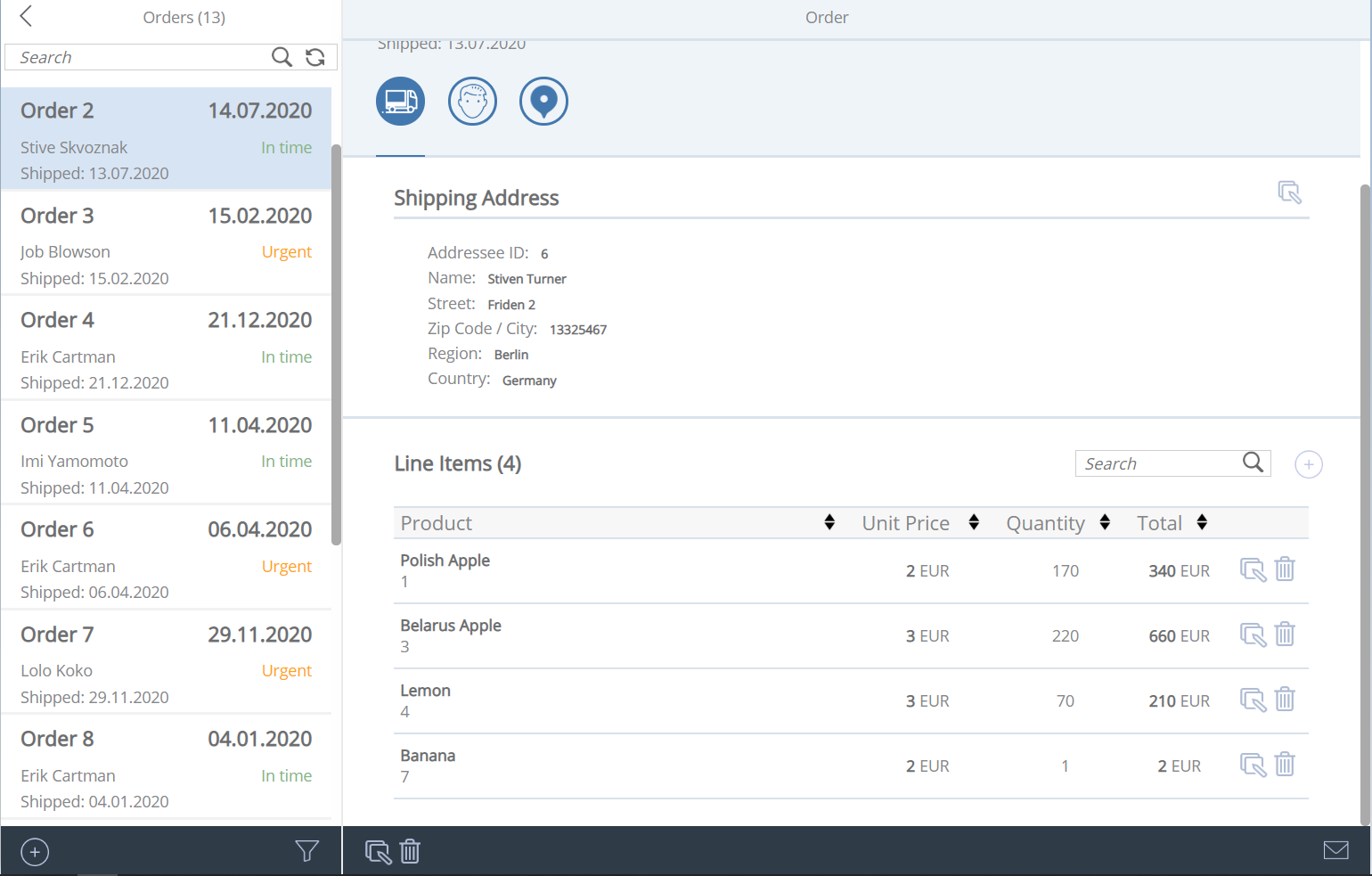


Рисунок 4.11 – Пример списка товаров в заказе

Список товаров имеет функцию сортировки по параметрам в трех позициях: стандартная, по возрастанию и по убыванию. Параметры, по которым возможно осуществить сортировку:

* название товара;
* цена единицы товара;
* количество товара;
* общая стоимость товара.

Поиска товара приведен на рисунке 4.12.

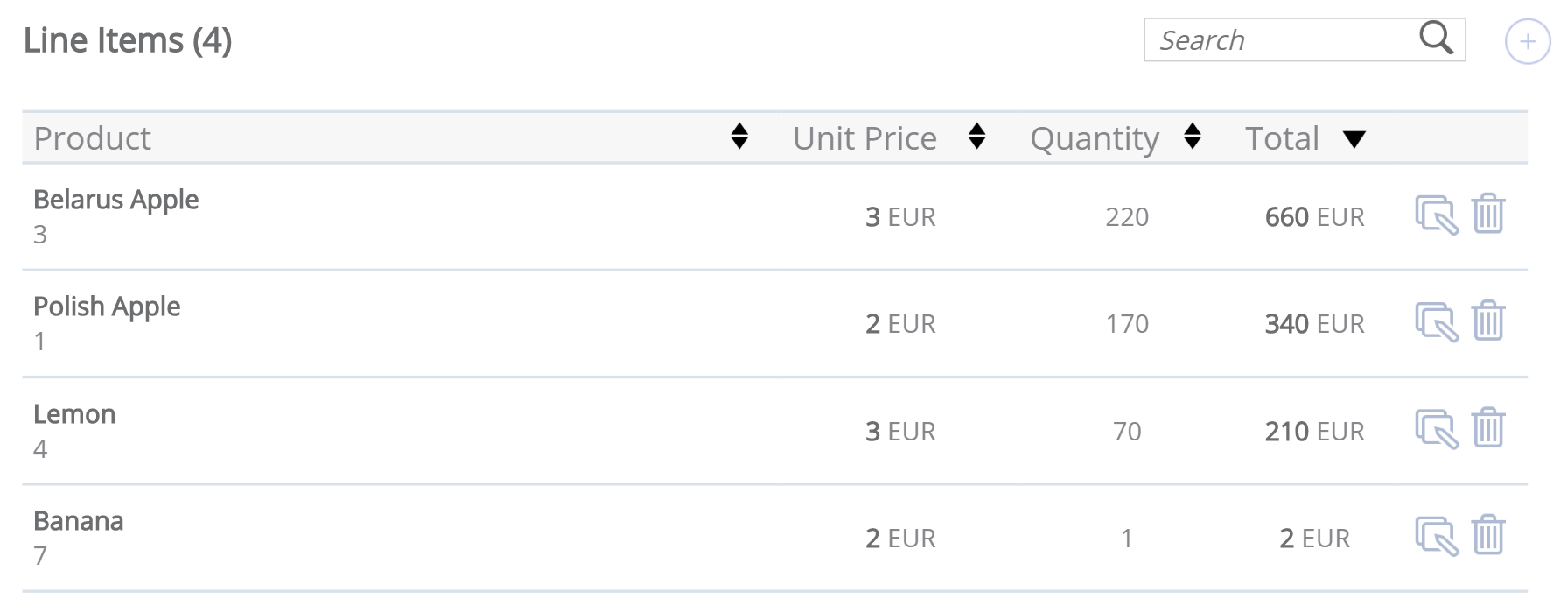


Рисунок 4.12 – Сортировка товара

Также возможен поиск товара по подстроке. Поиска товара приведен на рисунке 4.13.

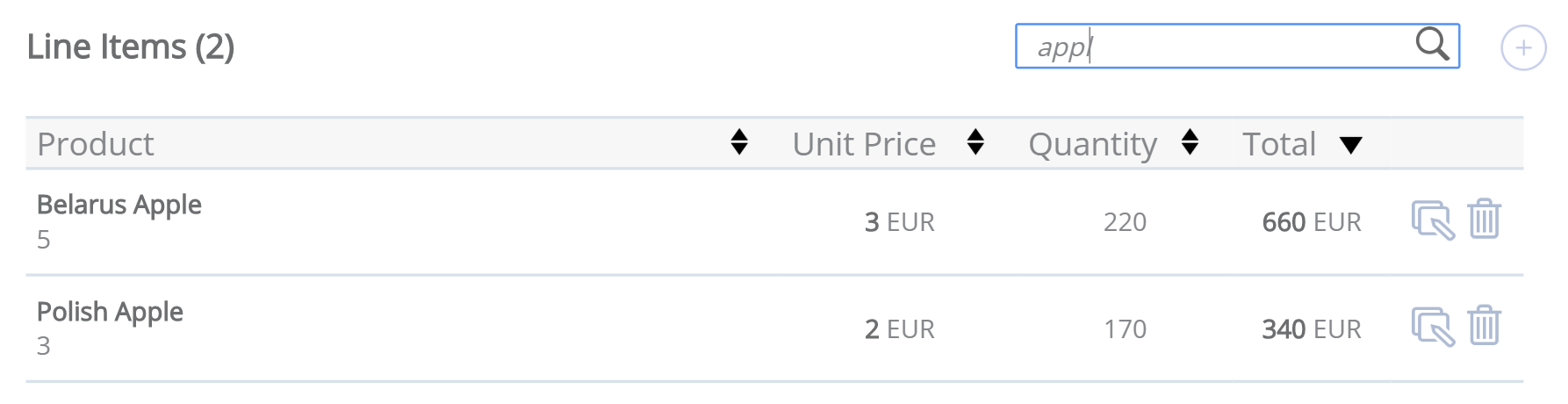


Рисунок 4.13 – Пример поиска товара

А теперь добавим новый товар, нажав на кнопку добавления в правом верхнем углу списка товаров. Для добавления необходимо выбрать идентификатор продукта и его количество. Добавление товара приведено на рисунке 4.14.

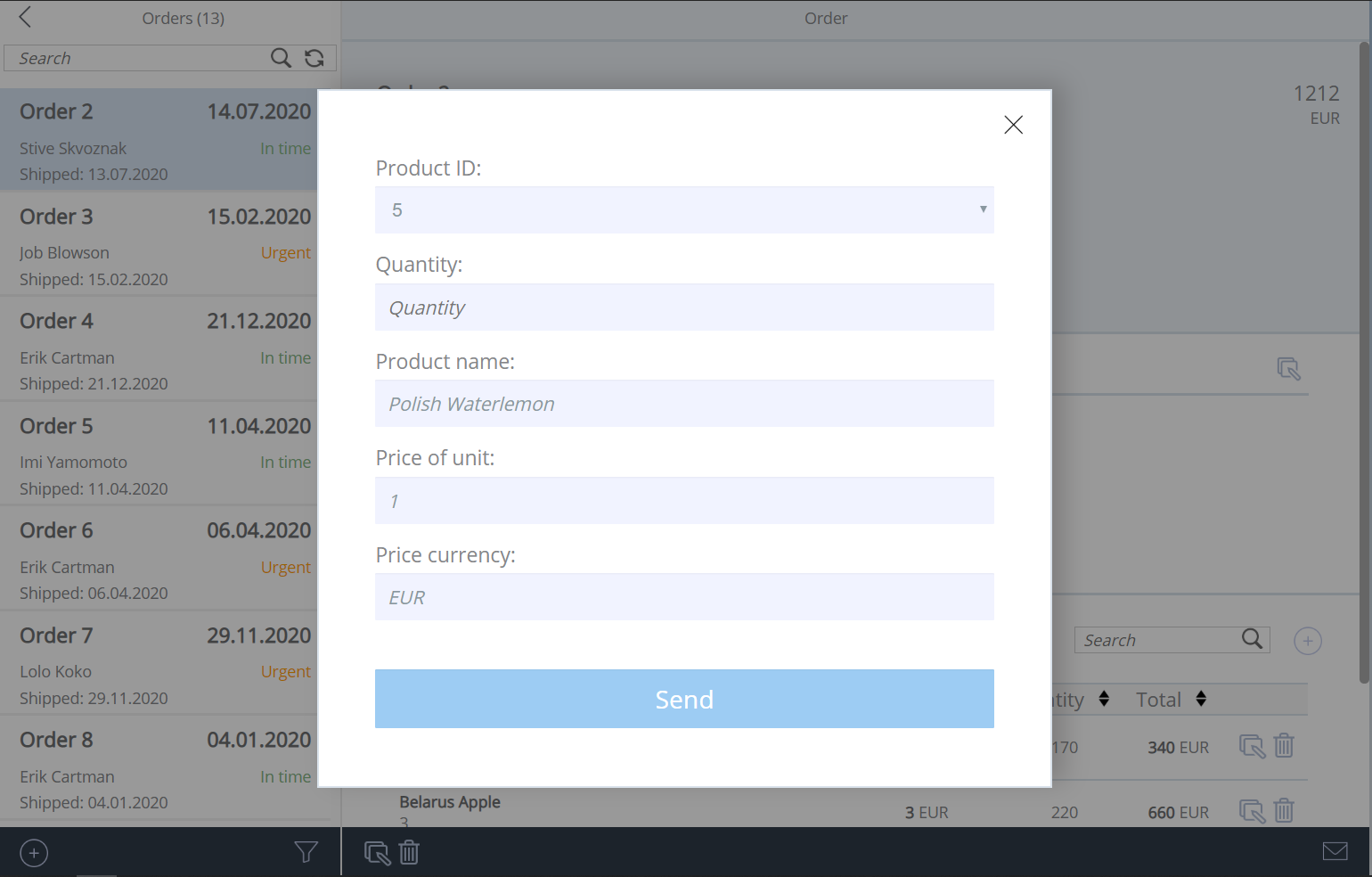


Рисунок 4.14 – Добавление товара

Добавленный товар можно отредактировать или же вовсе удалить. Для этого, напротив требуемого товара, есть кнопки управления. Кнопки изменения товара продемонстрирован на рисунке 4.15.

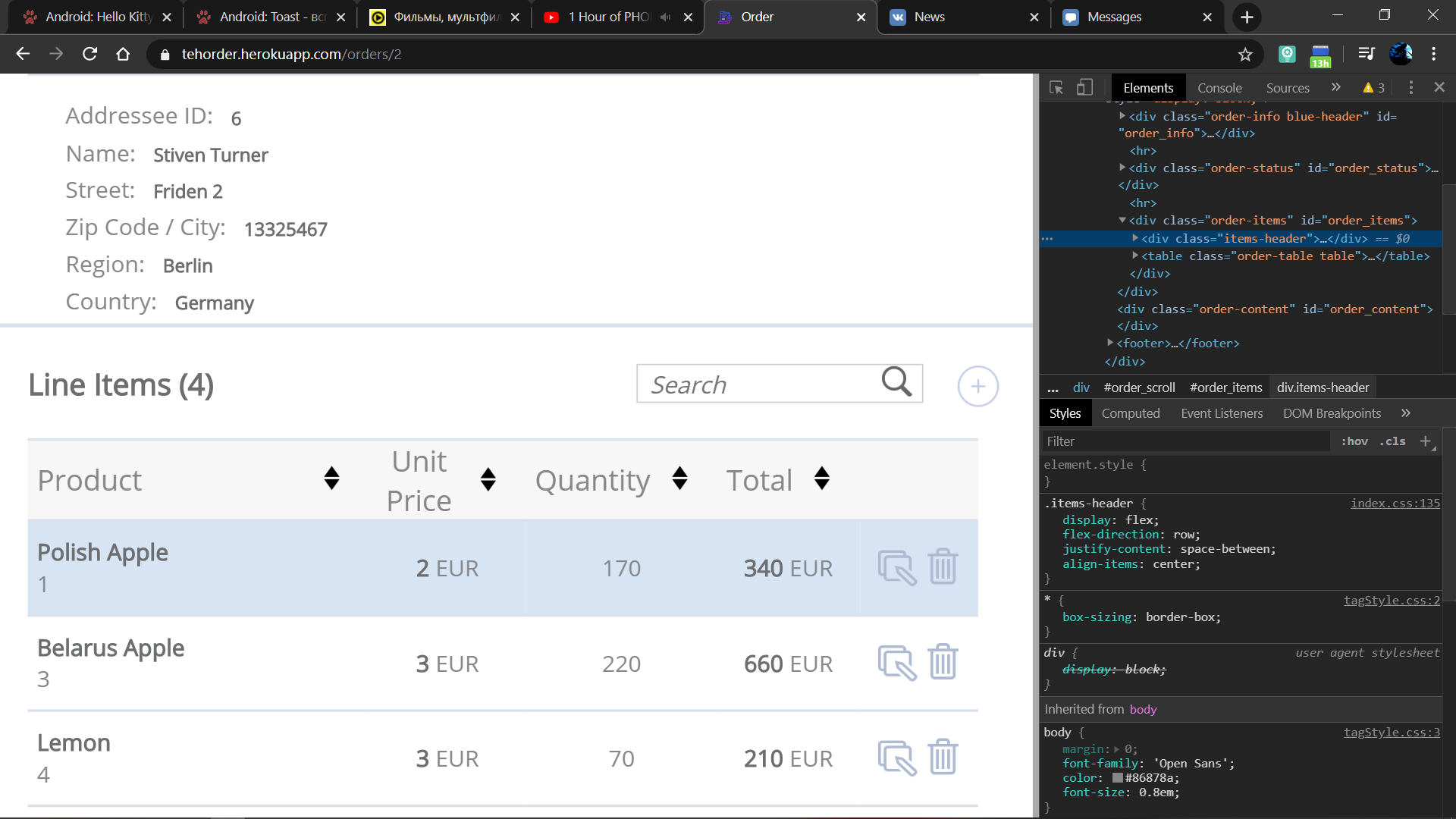


Рисунок 4.15 – Кнопки изменения товара

Созданный заказ можно отредактировать или же вовсе удалить. Для этого в окне открытого заказа, слева внизу, есть кнопки управления. Кнопки изменения заказа продемонстрированы на рисунке 4.16.

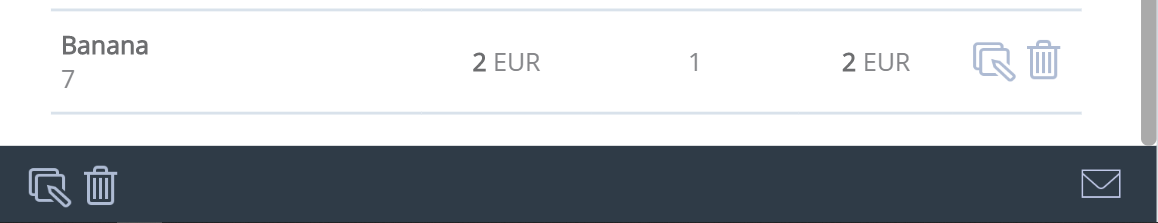


Рисунок 4.16 – Кнопки изменения заказа

Для полноценной поддержки кроссплатформенности, приложение должно иметь адаптивный интерфейс, чтобы пользователю было комфортно пользоваться им не только на полноформатных устройствах, но и на смартфонах. Пример мобильной версии приложения продемонстрирован на рисунке 4.17.

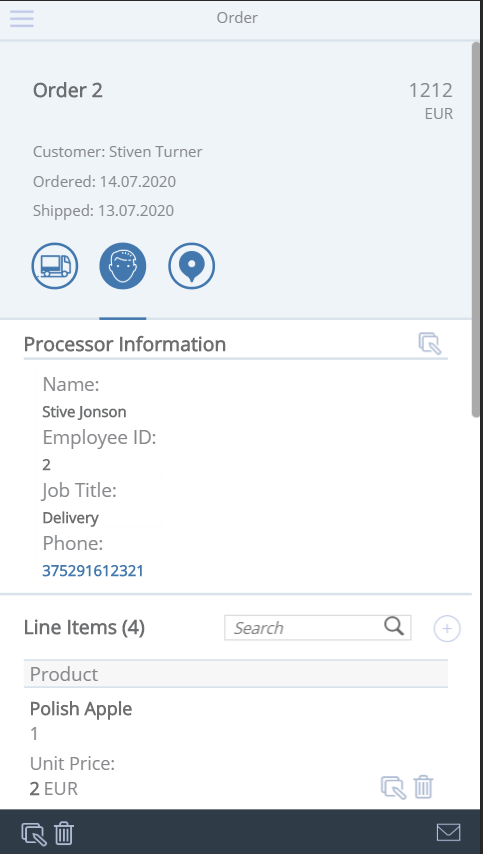


Рисунок 4.17 – Пример мобильной версии приложения

В данном варианте интерфейса, весьма проблематично разместить информацию и список заказов одновременно, поэтому список был помещен в боковую шторку. Для ее открытия в верхнем левом углу расположена кнопка. Если же требуется закрытие, то можно нажать на кнопку, либо же на темную область. Пример боковой шторки продемонстрирован на рисунке 4.18.

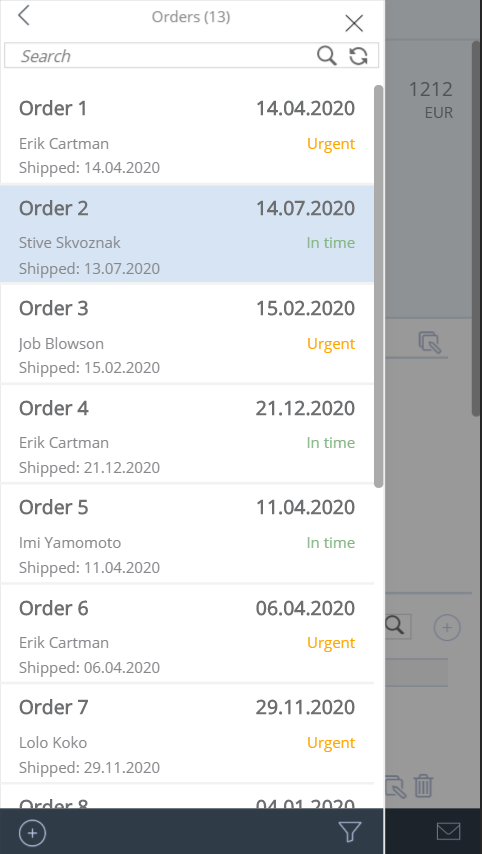


Рисунок 4.18 – Боковая шторка в мобильной версии приложения

Заключение

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Чернявский А.Л.

Провер.

Бурмакова А.В.

Н. контр.

Бурмакова А.В.

Утверд.

Бурмакова А.В.

Заключение

Лит.

Листов

1

БГТУ 71171075, 2020

В процессе решения поставленной задачи была достигнута поставленная цель по созданию программного средства «Перечень заказов». Основой целью курсового проекта стало проектирование кроссплатформенного приложение, которое помогло облегчить взаимодействие с пользователем. Это было достигнуто за счёт гибкой программной платформы сервера на Node.js и адаптивности сайта, в качестве клиентской части.

При разработке выполнены следующие пункты:

* просмотр перечня заказов;
* добавление заказов;
* корректирование заказов;
* удаление заказов;
* осуществление поиска заказов по различным признакам.

В курсовом проекте были реализованы следующие задачи:

* Создание базы данных;
* Создание сервера;
* Создание клиента с пользовательским интерфейсом;
* Реализация функций работы приложения;
* Тестирование программного продукта.

Данный проект является показательным примером программного продукта, реализующий систему работы со списком заказов, позволяя охватить необходимые требования для их администрирования, позволяя работать как самими заказами, так и с сущностями, связанными с ними.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объёме.

Список литературы

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Чернявский А.Л.

Провер.

Бурмакова А.В.

Н. контр.

Бурмакова А.В.

Утверд.

Бурмакова А.В.

Список литературы

Лит.

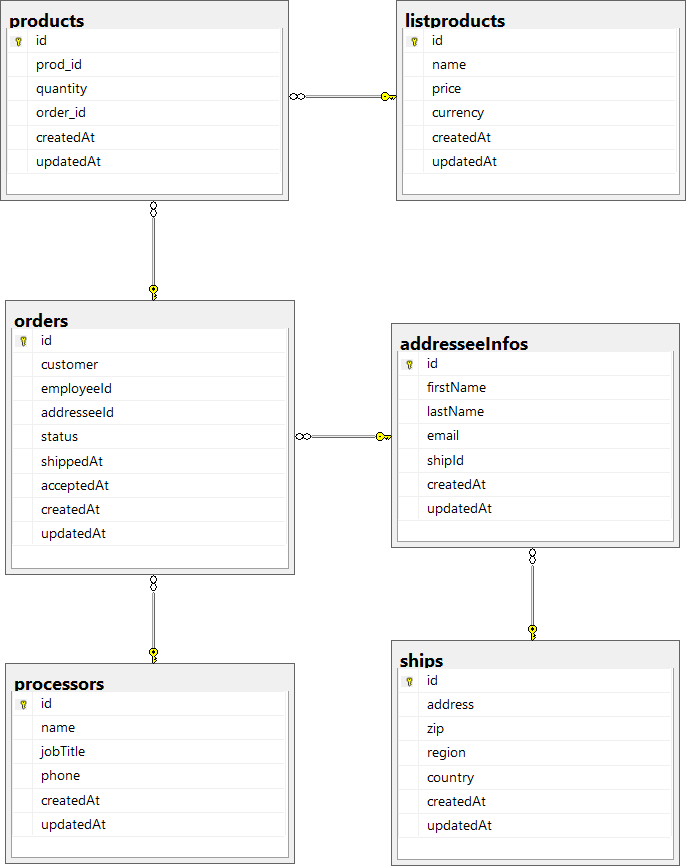
Листов

1

БГТУ 71171075, 2020

1. About Node.js [Электронный ресурс] / OpenJS Foundation. – Режим доступа: https://nodejs.org/en/about/. – Дата доступа: 20.04.2020.
2. What is Object/Relational Mapping? [Электронный ресурс] / Red Hat. – Режим доступа: https://hibernate.org/orm/what-is-an-orm/. – Дата доступа: 20.04.2020.
3. Sequelize [Электронный ресурс] / OpenJS Foundation. – Режим доступа: https://www.npmjs.com/package/sequelize. – Дата доступа: 20.04.2020.
4. Model Instances [Электронный ресурс] / Sequelize. – Режим доступа: https://sequelize.org/master/manual/model-instances.html. – Дата доступа: 22.04.2020.
5. Associations [Электронный ресурс] / Sequelize. – Режим доступа: https://sequelize.org/master/manual/assocs.html. – Дата доступа: 22.04.2020.
6. Fetch API [Электронный ресурс] / Mozilla. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/Fetch\_API. – Дата доступа: 28.04.2020.
7. Overview [Электронный ресурс] / Mike D Pilsbury. – Режим доступа: http://tediousjs.github.io/tedious/index.html. – Дата доступа: 20.04.2020.
8. Moment.js [Электронный ресурс] / MIT. – Режим доступа: https://momentjs.com/. – Дата доступа: 28.04.2020.

Приложение А



Приложение Б

|  |
| --- |
| const ship = (Sequelize, sequelize) =>{      return sequelize.define('ship', {          id: {              type: Sequelize.INTEGER,              autoIncrement: true,              primaryKey: true          },          address: {              type: Sequelize.STRING          },          zip: {              type: Sequelize.STRING          },          region: {              type: Sequelize.STRING          },          country: {              type: Sequelize.STRING          }      });  };  const processor = (Sequelize, sequelize) =>{      return sequelize.define('processor', {          id: {              type: Sequelize.INTEGER,              autoIncrement: true,              primaryKey: true          },          name: {              type: Sequelize.STRING          },          jobTitle: {              type: Sequelize.STRING          },          phone: {              type: Sequelize.STRING          }      });  };  const customer = (Sequelize, sequelize) =>{      return sequelize.define('addresseeInfo', {          id: {              type: Sequelize.INTEGER,              autoIncrement: true,              primaryKey: true          },          firstName: {              type: Sequelize.STRING          },          lastName: {              type: Sequelize.STRING          },          email: {              type: Sequelize.STRING          },          shipId: {              type: Sequelize.INTEGER,              references: {                  model: 'ships',                  key: 'id'              }          }      });  };  const order = (Sequelize, sequelize) =>{      return sequelize.define('order', {          id: {              type: Sequelize.INTEGER,              autoIncrement: true,              primaryKey: true          },          customer: {              type: Sequelize.STRING          },          employeeId: {              type: Sequelize.INTEGER,              references: {                  model: 'processors',                  key: 'id'              }          },          addresseeId: {              type: Sequelize.INTEGER,              references: {                  model: 'addresseeInfos',                  key: 'id'              }          },          status: {              type: Sequelize.STRING          },          acceptedAt: {              type: Sequelize.DATEONLY          },          shippedAt: {              type: Sequelize.DATEONLY          }      });  };  const listproduct = (Sequelize, sequelize) =>{      return sequelize.define('listproduct',      {          id: {              type: Sequelize.INTEGER,              autoIncrement: true,              primaryKey: true          },          name: {              type: Sequelize.STRING          },          price: {              type: Sequelize.DECIMAL(12,2)          },          currency: {              type: Sequelize.STRING          }      });  };  const product = (Sequelize, sequelize) =>{      return sequelize.define('product',      {          id: {              type: Sequelize.INTEGER,              autoIncrement: true,              primaryKey: true          },          prod\_id: {              type: Sequelize.INTEGER,              references: {                  model: 'listproducts',                  key: 'id'              }          },          quantity: {              type: Sequelize.INTEGER          },          order\_id: {              type: Sequelize.INTEGER,              references: {                  model: 'orders',                  key: 'id'              }          }      });  }; |