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, которая позволяет сопоставлять объекты классов и документы коллекций из базы данных;

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.

Аналитический обзор литературы

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 Аналитический обзор литературы

Лит.

Листов

2

БГТУ 71171075, 2020

Раньше, для администрирования заказов в магазинах, вели бухгалтерский учет, что осложняло весь процесс. Сейчас же используются базы данных с разработанным для них программным обеспечением, позволяя уменьшить объем бумажных работ и оптимизировать работу всей системы магазинов, попутно увеличивая прибыль компании, использующей современные технологии.

Постепенно с развитием программного обеспечения ЭВМ появились идеи создания управляющих систем, которые позволяли бы накапливать, хранить и обновлять взаимосвязанные данные по целому комплексу решаемых задач, например, при автоматизации бухгалтерского учета на предприятии. Эти идеи нашли свое воплощение в системах управления базами данных (СУБД). С появлением персональных компьютеров СУБД становятся наиболее популярным средством обработки табличной информации. Они являются инструментальным средством проектирования банков данных при обработке больших объемов информации.

СУБД, сама по себе, не достаточно удобна для обычного пользователя. Поэтому для облегчения взаимодействия пользователя с данными разрабатываются сайты, дающие кроссплатформенность и не привязанность пользователя к операционной системе.

Базы данных, написанные для различных интернет-магазинов, используются для структуризации информации в заказах. В открытом доступе рассмотреть напрямую системы управления магазинами невозможно, так они пишутся на заказ. Но можно рассмотреть структуру организации товаров среди онлайн-магазинов. Одним из таких магазинов является белорусский «e-dostavka.by», принадлежащий ООО «Евроторг».

Рассмотрим в качестве примера заказ с данного сайта на рисунке 1.1.

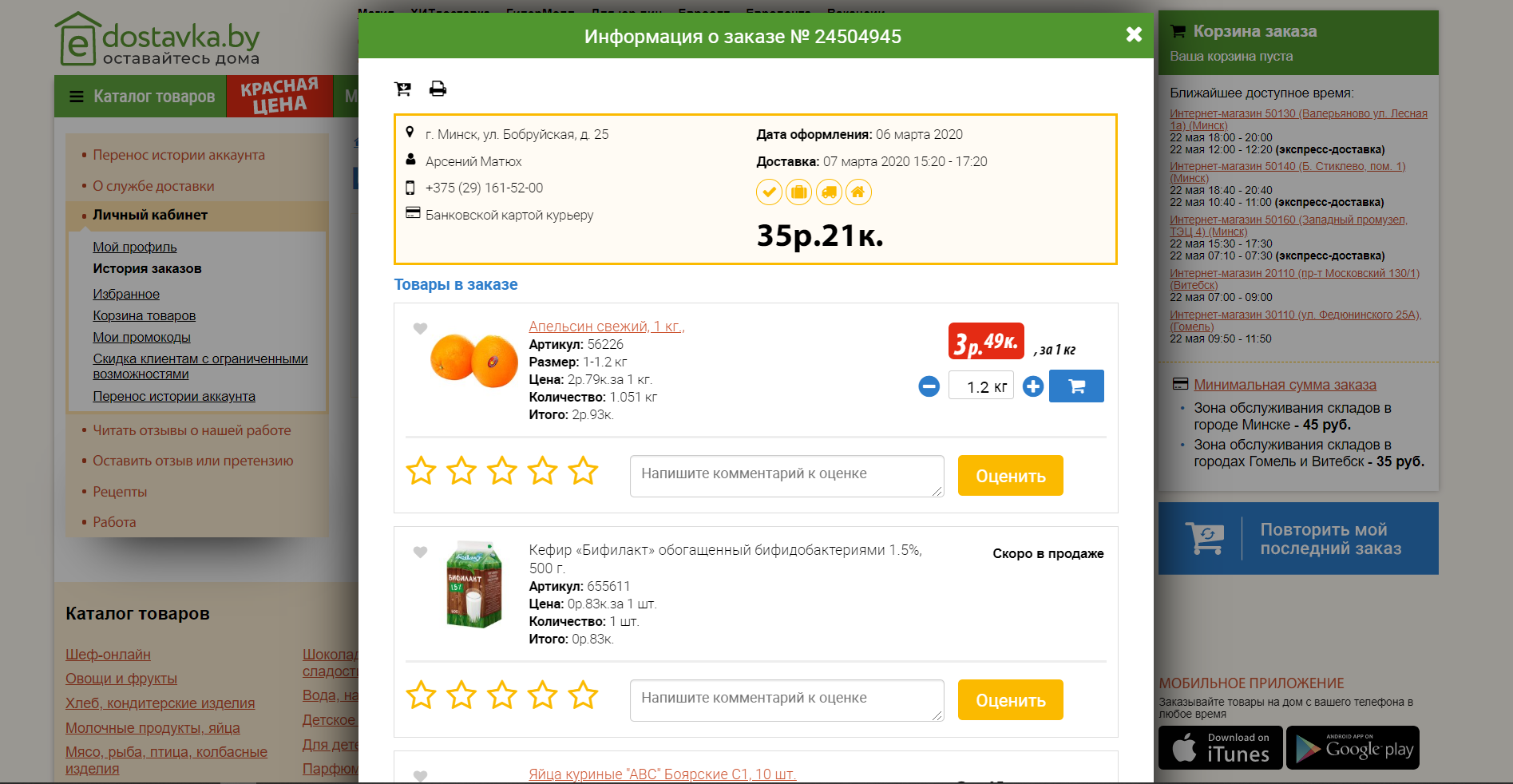


Рисунок 1.1 – Пример заказа в «e-dostavka.by»

Тут структуру заказа можно разделить на несколько отдельных информационных категорий:

* заказ;
* клиент;
* адрес доставки;
* артикул;
* список заказанных товаров;
* товар.

У товара есть следующие основные особенности:

* артикул (идентификатор);
* цена;
* название;
* количество;
* итоговая стоимость.

Стоит отметить одну важную особенность, заказанные товары формируют собой единый заказ, которые собирают в единое целое, для единовременной доставки.

Так же рассмотрим интернет-магазин «aliexpress.com», страница из списка заказов представлена на рисунке 1.2.

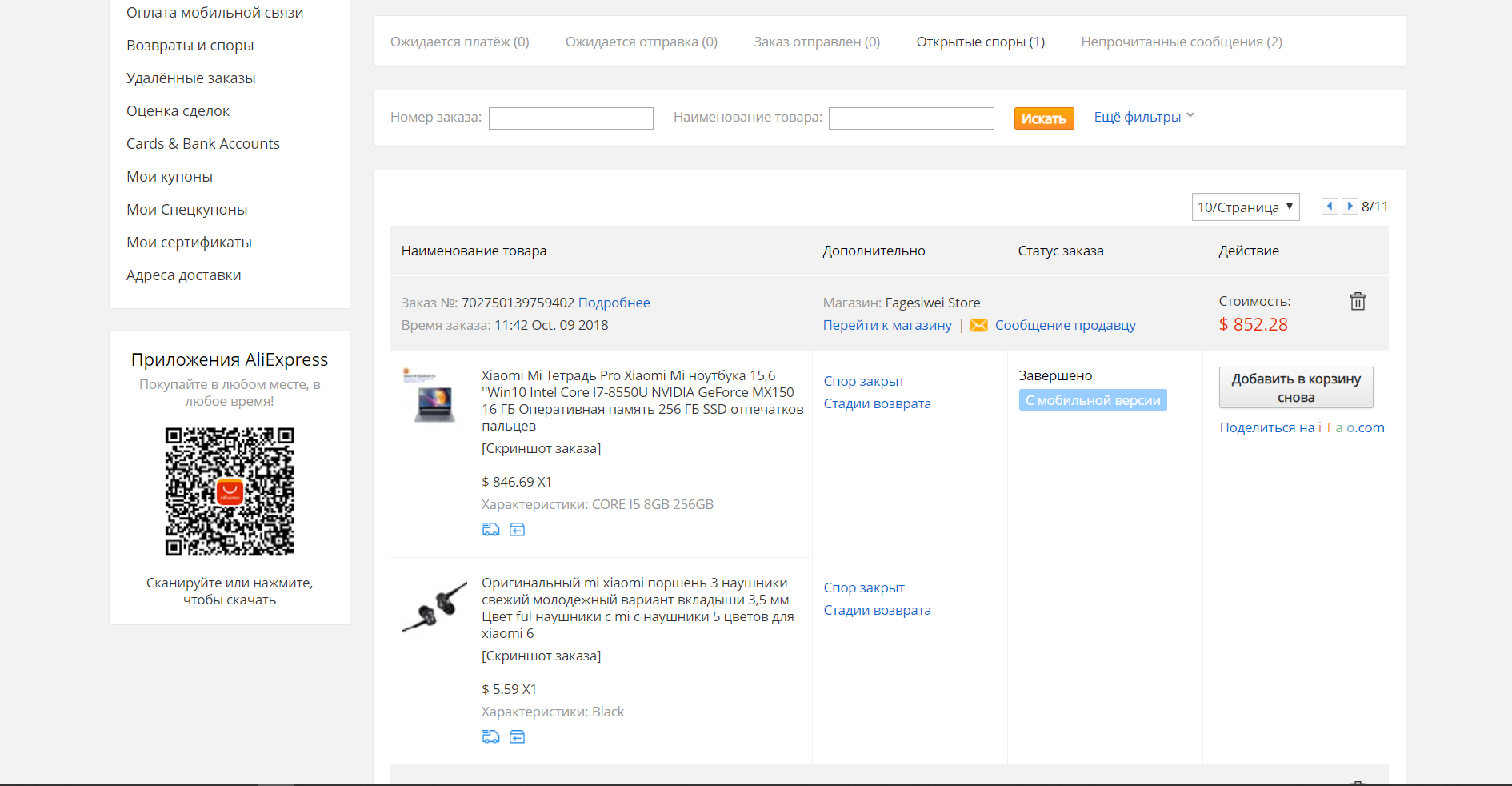


Рисунок 1.2 – Пример сайта «aliexpress.com»

На данном сайте заказы формируются в единый из нескольких товаров, исключением является заказ у одного продавца. Это обоснованно тем, что «aliexpress.com» представляет собой не просто магазин, а интернет-площадку для сторонних продавцов, которые сами формируют и отправляют заказ.

Тут структуру заказа состоит из следующих информационных категорий:

* заказ;
* магазин;
* адрес доставки;
* номер заказа;
* список заказанных товаров;
* товар.

У товара есть следующие основные особенности:

* цена;
* название;
* характеристики;
* количество.

Исходя из приведённых данных была разработана база данных, описанная в пункте 2.1.

Разработка 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

При разработке курсового проекта весь процесс был разбит на 3 следующие составляющие:

* база данных;
* сервер;
* клиент.

2.1 Проектирование структуры базы данных

Для реализации базы данных для перечней заказов было разработано 6 таблиц. Диаграмма базы со структурой связей представлена в приложении А.

Во всех таблицах есть столбцы createdAt и updatedAt, они обусловлены использованием на сервере Sequelize:

* createdAt – дата создания строки, тип datetimeoffset;
* updatedAt – дата изменения строки, тип datetimeoffset.

Таблицаordersпредставляет собой сам заказ, который состоит из столбцов:

* id – идентификатор заказа, тип int, первичный ключ;
* customer – имя заказчика, тип nvarchar;
* employeeId – идентификатор обработчика заказа, тип int;
* addresseeId – идентификатор адресата, тип int;
* status – статус заказа, тип nvarchar;
* shippedAt – дата доставки заказа, тип datetimeoffset;
* acceptedAt – дата принятия заказа, тип datetimeoffset.

Таблица processors представляет собой сотрудников, обрабатывающих заказ, состоит из столбцов:

* id – идентификатор сотрудника, тип int, первичный ключ;
* name – имя сотрудника, тип nvarchar;
* jobTitle – должность сотрудника, тип nvarchar;
* phone – номер телефона сотрудника, тип nvarchar.

Таблица addresseeInfos представляет собой адресатов. адресаты имеют адрес, на который должен быть доставлен заказ. Данная таблица состоит из следующих столбцов:

* id – идентификатор адресата, тип int, первичный ключ;
* firstName – имя адресата, тип nvarchar;
* lastName – фамилия адресата, тип nvarchar;
* email – адрес электронной почты адресата, тип nvarchar;
* shipId – идентификатор адреса, тип int.

Таблица ships представляет данные адресах доставки, состоит из столбцов:

* id – идентификатор адреса, тип int, первичный ключ;
* address – адрес доставки, тип nvarchar;
* zip – почтовый индекс, тип nvarchar;
* region – регион, тип nvarchar;
* country – страна, тип nvarchar.

Таблица products представляет собой список товаров в заказе, состоит из столбцов:

* id – идентификатор товара в заказе, тип int, первичный ключ;
* prod\_id – идентификатор товара, тип int;
* quantity – количество, тип int;
* order\_id – идентификатор заказа, тип int.

Таблица listproducts представляет собой товары, состоит из столбцов:

* id – идентификатор товара, тип int, первичный ключ;
* name – наименование товара, тип nvarchar;
* price – цена единицы товара, тип decimal;
* currency – валюта, тип nvarchar.

2.2 Проектирование модели базы данных

Для работы с базой данных на уровне сервера, было принято решение связать данные с концепциями объектно-ориентированных языков, то есть ORM.

ORM – технология программирования для связи базы данных с концепциями объектно-ориентированных языков программирования [2]. Данная технология позволяет проектировать работу с данными в терминах классов, а не таблиц данных. Она позволяет преобразовывать классы в данные, пригодные для хранения в базе данных, причем схему преобразования определяет сам разработчик. Благодаря технологии ORM нет необходимости писать SQL-код для взаимодействия с локальной базой данных.

В частности, был выбран пакет Sequelize [3], для реализации ORM в Node.js. Чтобы работать с данными необходимо было разработать модель данных [4]. Пример модуля с моделями данных представлен в виде листинга приложения Б.

Далее, после создания модели, необходимо образовать связи между таблицами модели [5]. Пример создания связей представлен ниже в листинге 2.1.

**Листинг 2.1 – Пример создания связей между таблицами модели**

|  |
| --- |
| customer.belongsTo(ship);              childOption.foreignKey = 'addresseeId';              order.belongsTo(customer, childOption);              childOption.foreignKey = 'employeeId';              order.belongsTo(processor, childOption);              childOption = {                  onDelete: 'CASCADE',                  hooks: true,                  foreignKey: 'prod\_id',              }; |

**Продолжение** **листинга 2.1**

|  |
| --- |
| listproduct.hasMany(product, childOption);              product.belongsTo(listproduct, childOption);                childOption.foreignKey = 'order\_id';              order.hasMany(product, childOption); |

Стоит обратить внимание на параметры onDelete и hooks, которые с данными значениями позволяют организовать каскадное удаление. То есть, если мы удалим заказ, то будут удалены и все связанные с ним значения, в данном случае будут удалены связанные товары из таблицы products.

2.3 Проектирование структуры сервера

При проектировании структуры сервера важно сделать так, чтобы она была легко читаема и легко расширяема. Для данных целей прекрасно подходит архитектурный паттерн MVC.

MVC подразумевает под собой разделение проекта на три компонента:

* Model – модель, организует бизнес-логику проекта, подчиняется командам контроллера и отправляет данные на представления;
* View – представление, отображает пользователю данные;
* Controller – контроллер, реагирует на команды пользователя, посылая дальнейшие команды пользователю, занимается маршрутизацией.

В данном проекте модель работает с базой данных. Модель состоит из модели базы данных, модуля для работы с базой данных и модуля API, для взаимодействия с моделью MVC. Пример архитектуры файлов модели в проекте на рисунке 2.1.

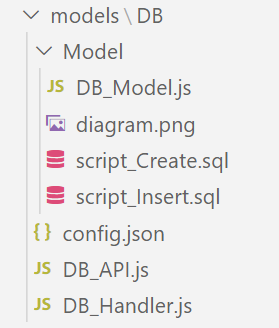


Рисунок 2.1 – Пример архитектуры файлов модели MVC

Контроллер, в данном проекте, выполняет функцию маршрутизатора запросов к серверу. Контроллер разделен на две части:

* маршрутизатор routing, который принимает запросы на представления;
* маршрутизатор api, который принимает запросы к модулю через его API.

Пример архитектуры файлов контроллера в проекте на рисунке 2.2.

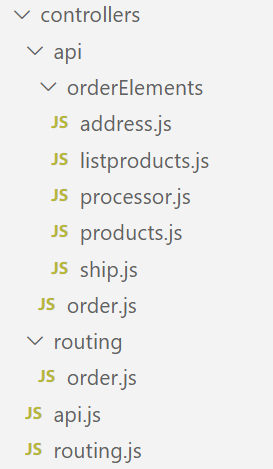


Рисунок 2.1 – Пример архитектуры файлов контроллера MVC

Проект является SPA приложением, поэтому контроллер routing отправляет клиенту html страницу, а контроллер api отправляет данные в формате json.

2.4 Проектирование приложения клиента

Так как проект является кроссплатформенным, клиентская часть была разработана для использования в браузере. Было принято реализовывать SPA приложение, поэтому оно должно уметь запрашивать у сервера необходимые данные и динамически изменять представление, в зависимости от полученных данных. Чтобы клиент не видел задержки, при запросе у сервера необходимых данных, клиент запрашивает данные асинхронно, с помощью promise-ориентированного fetch [6], позволяя отследить момент принятия данных с помощью then и ошибки с помощью catch. Пример реализации fetch-запроса представлен на листинге 2.2.

**Листинг 2.2 – Пример fetch-запроса**

|  |
| --- |
| let LINK = `/api/orders/${elem}`;      fetch(LINK, {method: 'GET'}).then(res => res.json()).then(res =>{          form = form.querySelector('[name=id]')          tempArray = [];          res.forEach(el => {              tempArray.push(el);              let option = document.createElement('option');              option.value = el.id;              option.innerHTML = el.id;              form.append(option);          });      })      .then(() => {          let i = 0;          sender = sender.querySelector('#id').innerHTML;          tempArray.forEach(el => {              ++i;              if(el.id == sender){                  form.selectedIndex = i;                  form.onchange();              }          });      })      .catch((err) => console.log(`Fetch ERROR by ${LINK}: ${err}`)); |

Обоснование технических приемов программирования

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 Обоснование технических приёмов программирования

Лит.

Листов

2

БГТУ 71171075, 2020

При проектировании программного средства в качестве программной платформы для сервера был выбран Node.js с пакетами Express.js, moment, body-parser, tedious и sequelize. Непосредственно для взаимодействия с Microsoft SQL Server нам необходимы только пакеты tedious и sequelize. Схема взаимодействия с Microsoft SQL Server представлена на рисунке 3.1.

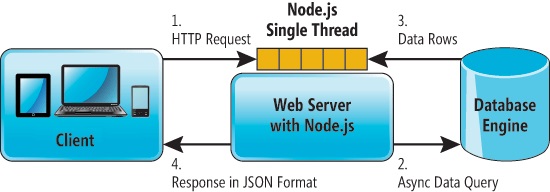


Рисунок 3.1 – Схема взаимодействия с Microsoft SQL Server

Для работы с базой данных используется пакет tedious, являющийся TDS драйвером [7]. Посредством этого пакета работает Sequelize, в который мы передаем необходимые конфигурации. Пример конфигурации, необходимой для подключения к Microsoft SQL Server, представлена ниже, в листинге 3.1.

**Листинг 3.1 – Пример конфигурации для подключения к СУБД**

|  |
| --- |
| {          "username": "Develer12",          "password": "admin",          "database": "TehOrder",          "host": "localhost",          "dialect": "mssql",          "dialectOptions": {              "multipleStatements": true          }      } |

Далее необходимо подключиться, пройти аутентификацию в СУБД и синхронизировать модель данных, задействовав ранее представленную конфигурацию. Пример подключения представлен ниже, в листинге 3.2.

**Листинг 3.2 – Подключение к Microsoft SQL Server**

|  |
| --- |
| sequelize = new Sequelize(config.server);          sequelize.authenticate().then(()=>{              sequelize.sync().catch(err=> console.log("SYNC ERROR: "+err));          })          .catch(err=>{console.log("Connection ERROR: "+err);}); |

При работе с датами, их формат у клиента и СУБД будет отличаться. Потому, чтобы привести их к единому виду, был использован пакет moment [8]. Пример приведения к формату дат у клиента приведен в листинге 3.3.

**Листинг 3.4 – Пример использования moment**

|  |
| --- |
| moment(results.acceptedAt).format('DD.MM.YYYY'); |

Пакет Express.js используется для упрощения работы с Node.js, позволяя облегчить синтаксис. Так же, за счет продвинутых средств маршрутизации, функции Router, можно разделить управление маршрутизацией между несколькими модулями. Пример передачи управления маршрутизацией другому модулю приведен в листинге 3.5.

**Листинг 3.5 – Пример передачи управления маршрутизацией**

|  |
| --- |
| const Route = express.Router();  const order = require(\_\_dirname + '/api/order');  Route.use('/orders', order); |

Для упрощения доступа к телу запроса был использован пакет body-parser, представляющий собой промежуточный обработчик для Express.js. Пример подключения промежуточного обработчика приведен в листинге 3.6.

**Листинг 3.6 – Пример подключения промежуточного обработчика body-parser**

|  |
| --- |
| const express = require('express');  const bodyParser = require('body-parser');  const app = express();  app.use(bodyParser.json()); |

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.

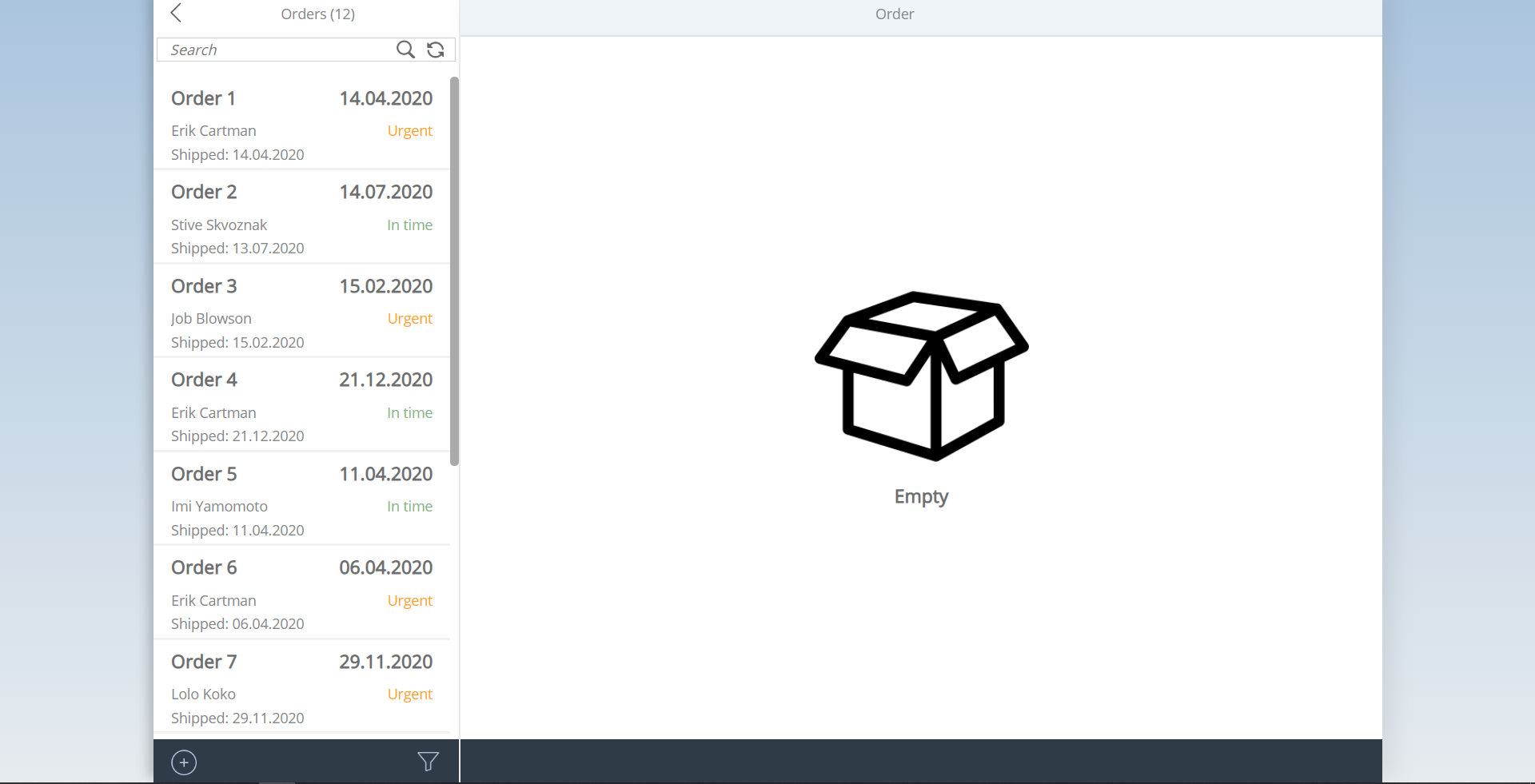


Рисунок 4.1 – Страница авторизации

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.

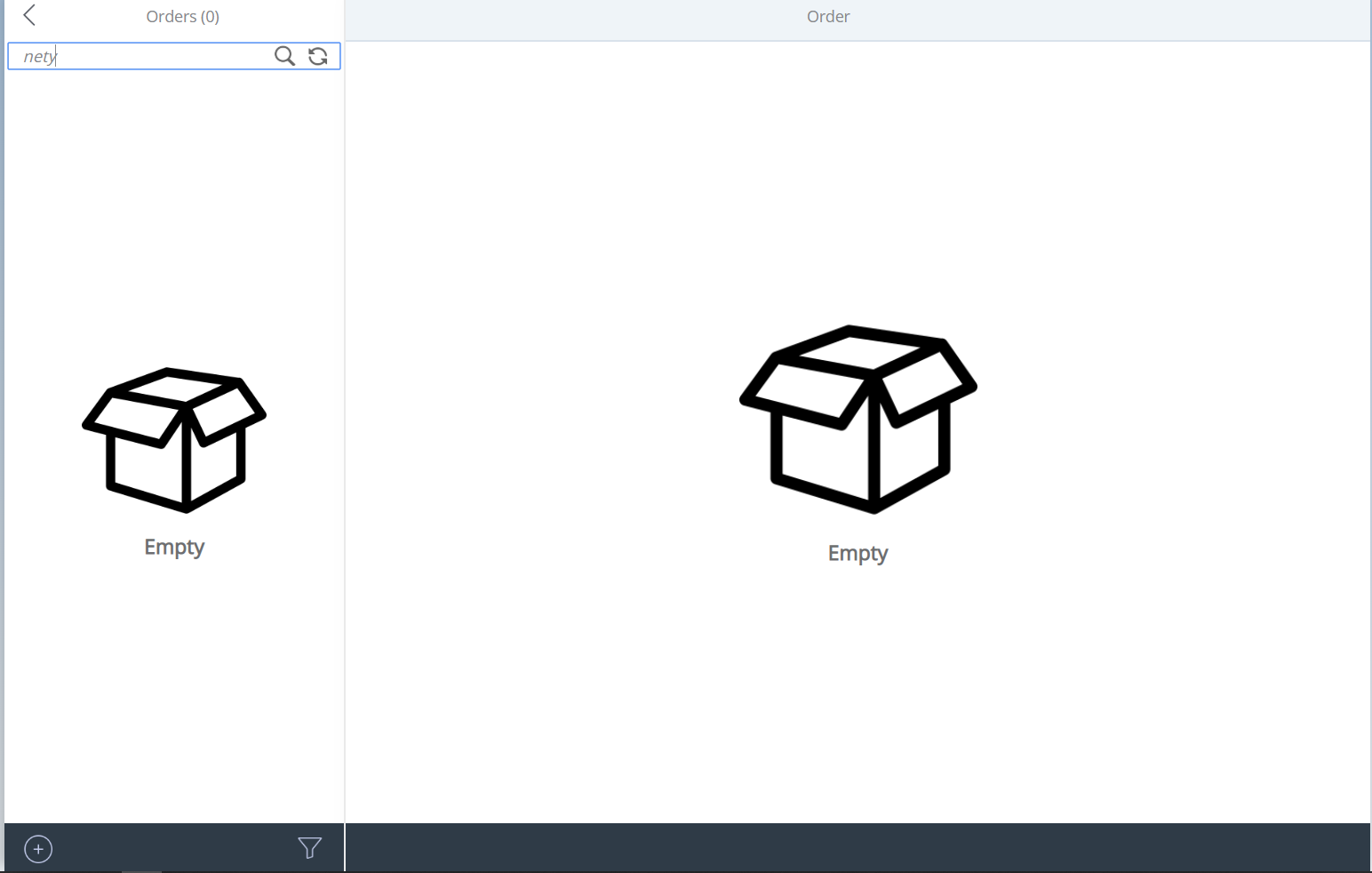


Рисунок 4.2 – Пример отсутствия искомого заказа

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.

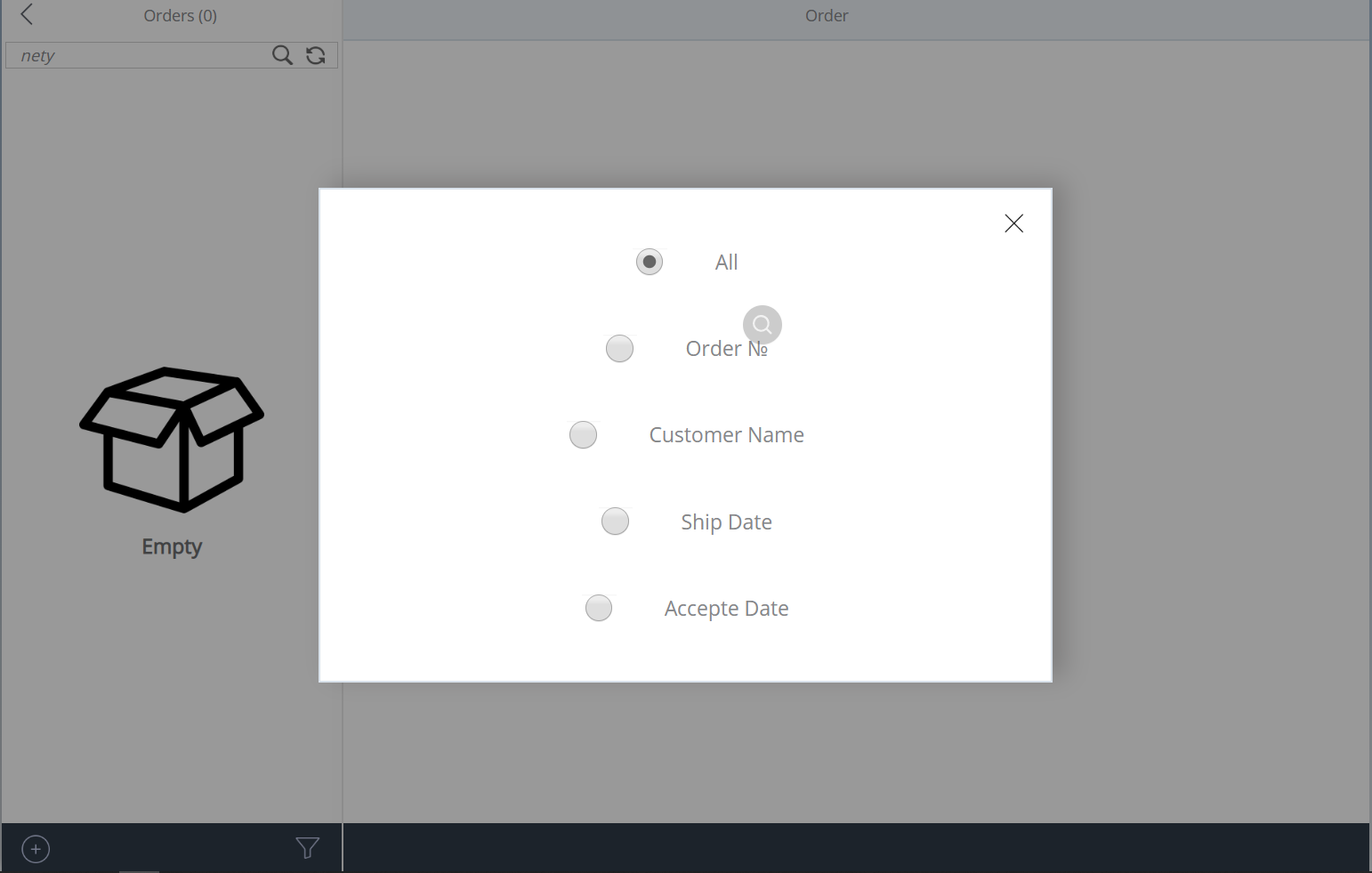


Рисунок 4.3 – Фильтрация поиска заказа

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

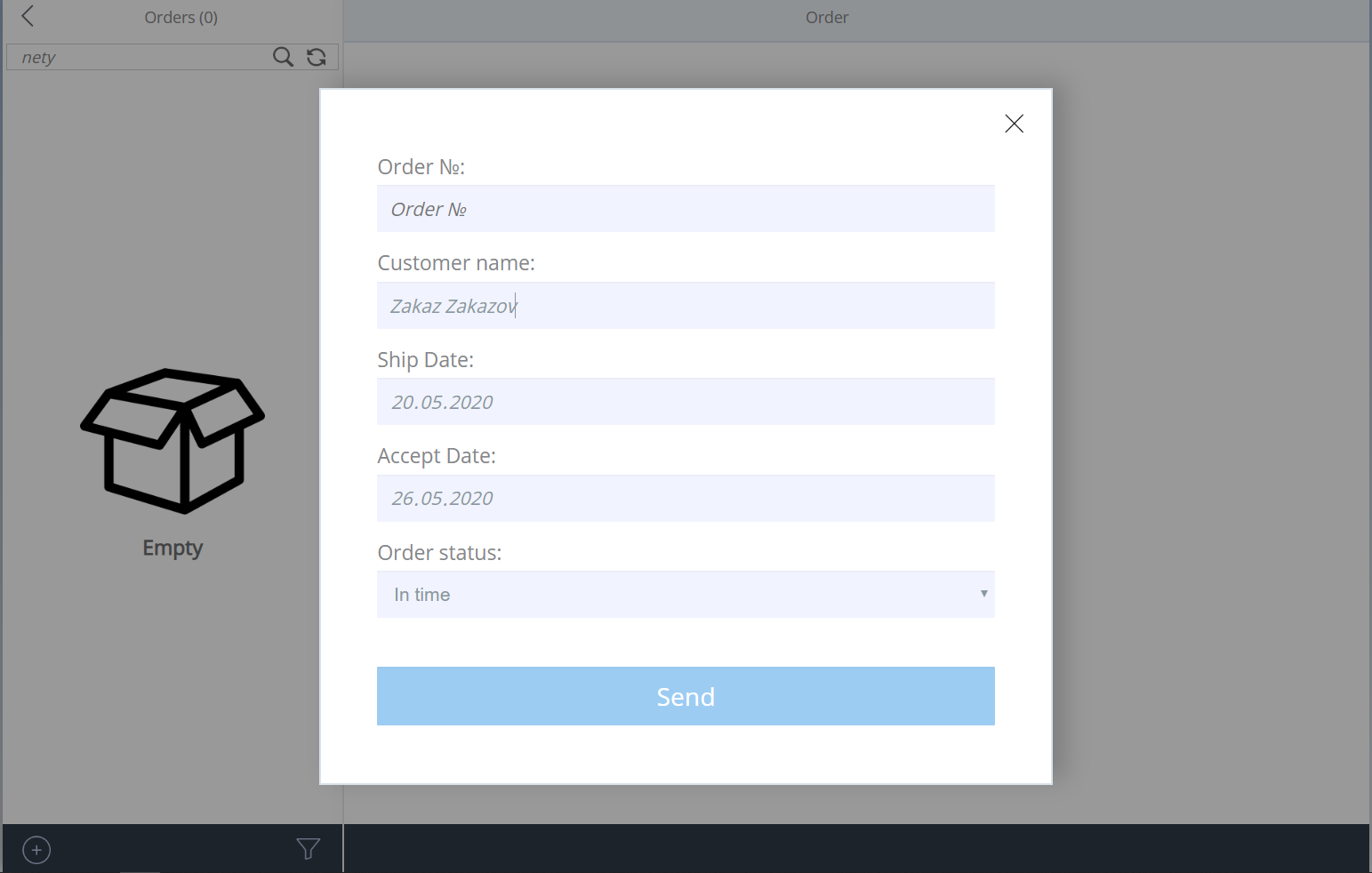


Рисунок 4.4 – Создание нового заказа

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.

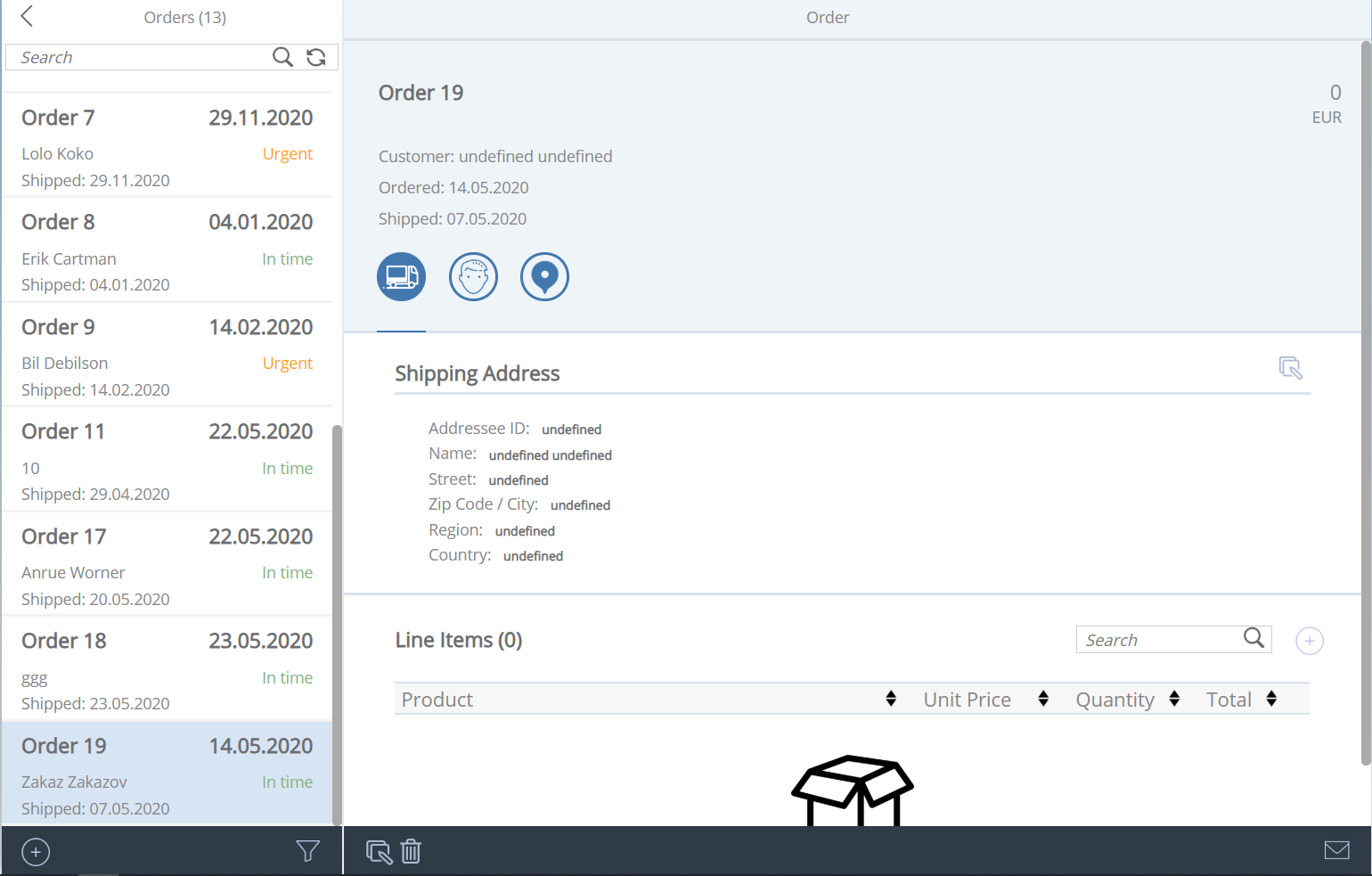


Рисунок 4.5 – Окно заказа

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.

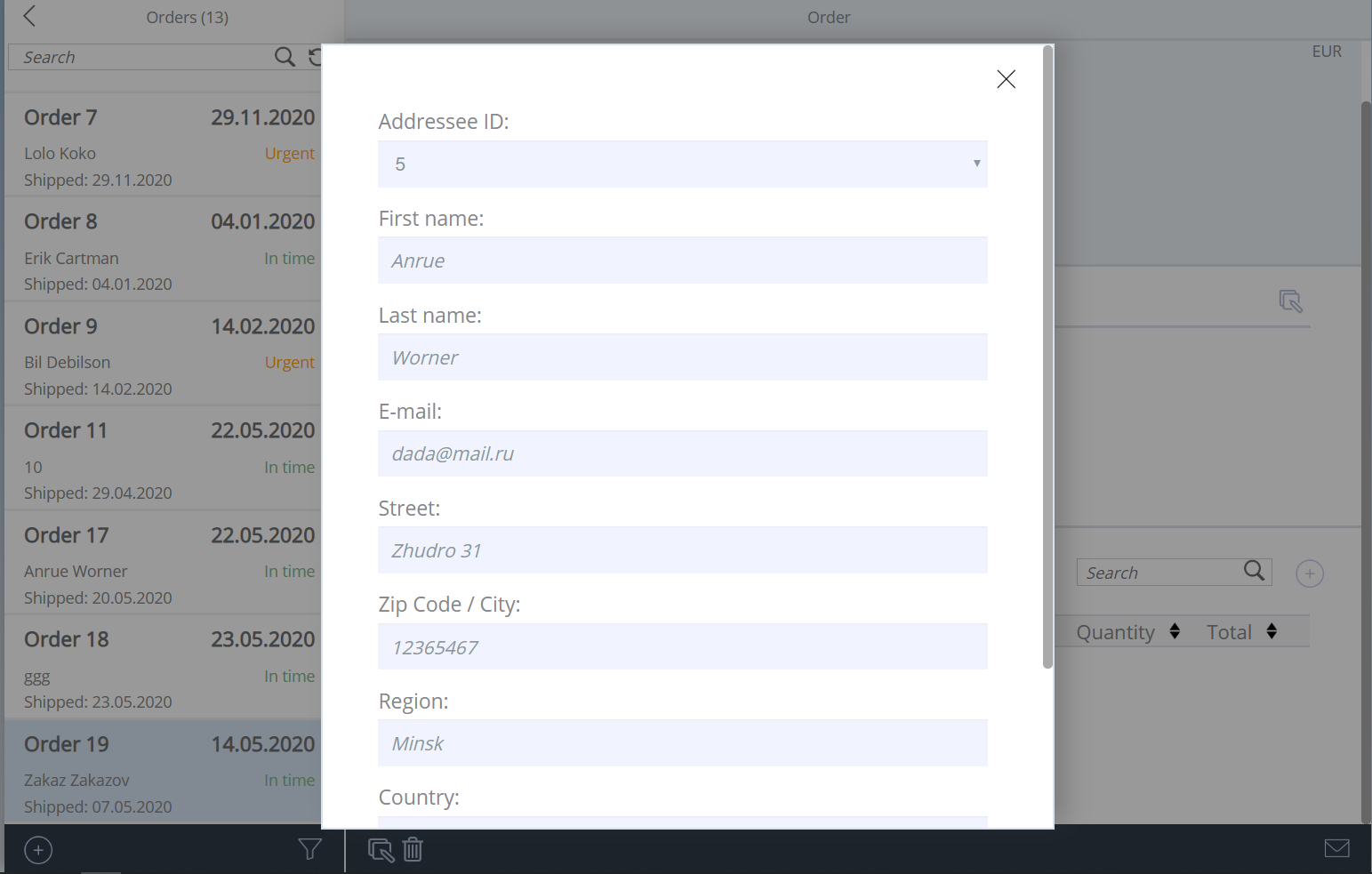


Рисунок 4.6 – Окно выбора адресата

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.

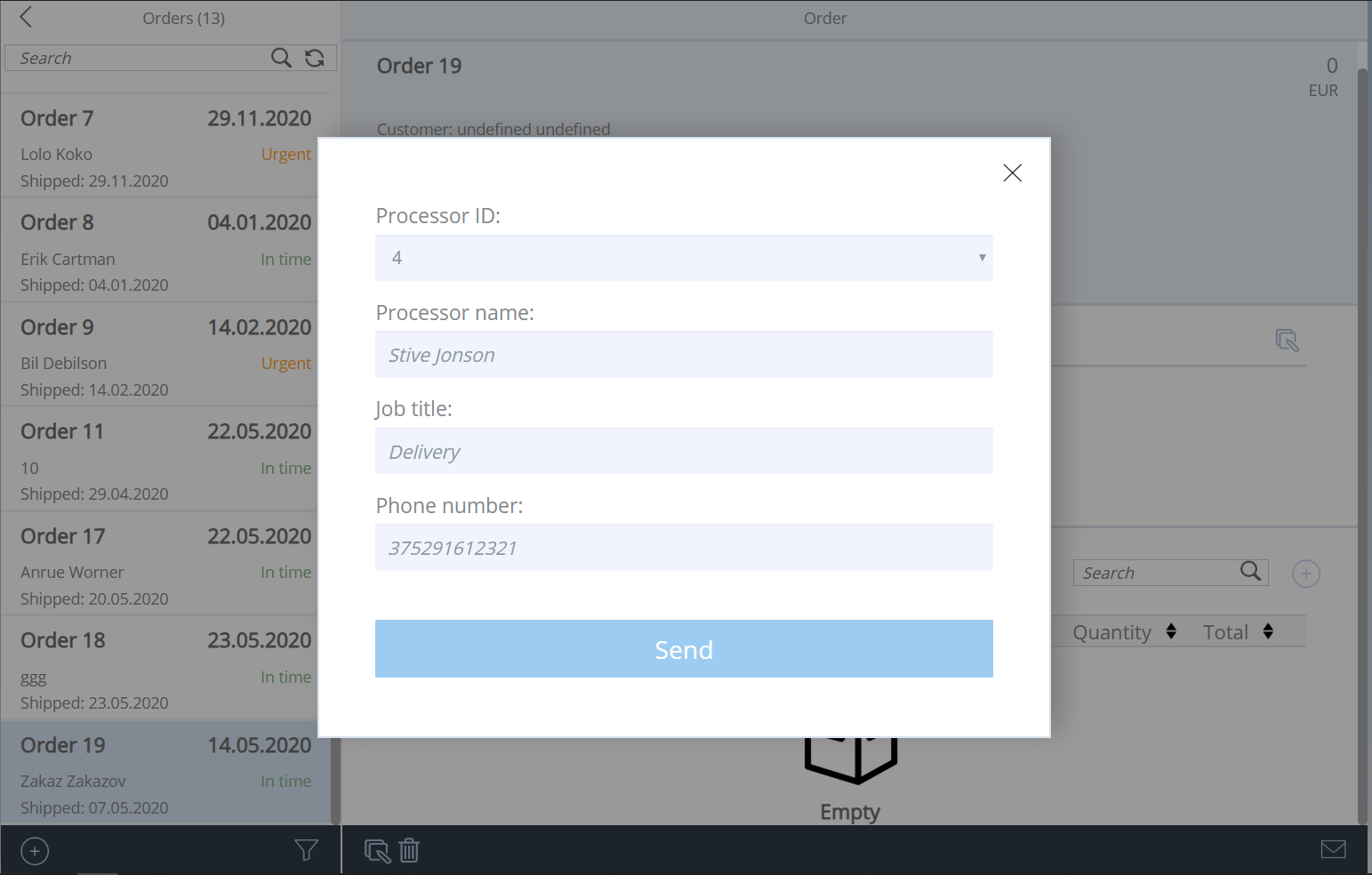


Рисунок 4.7 – Окно выбора обработчика

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.

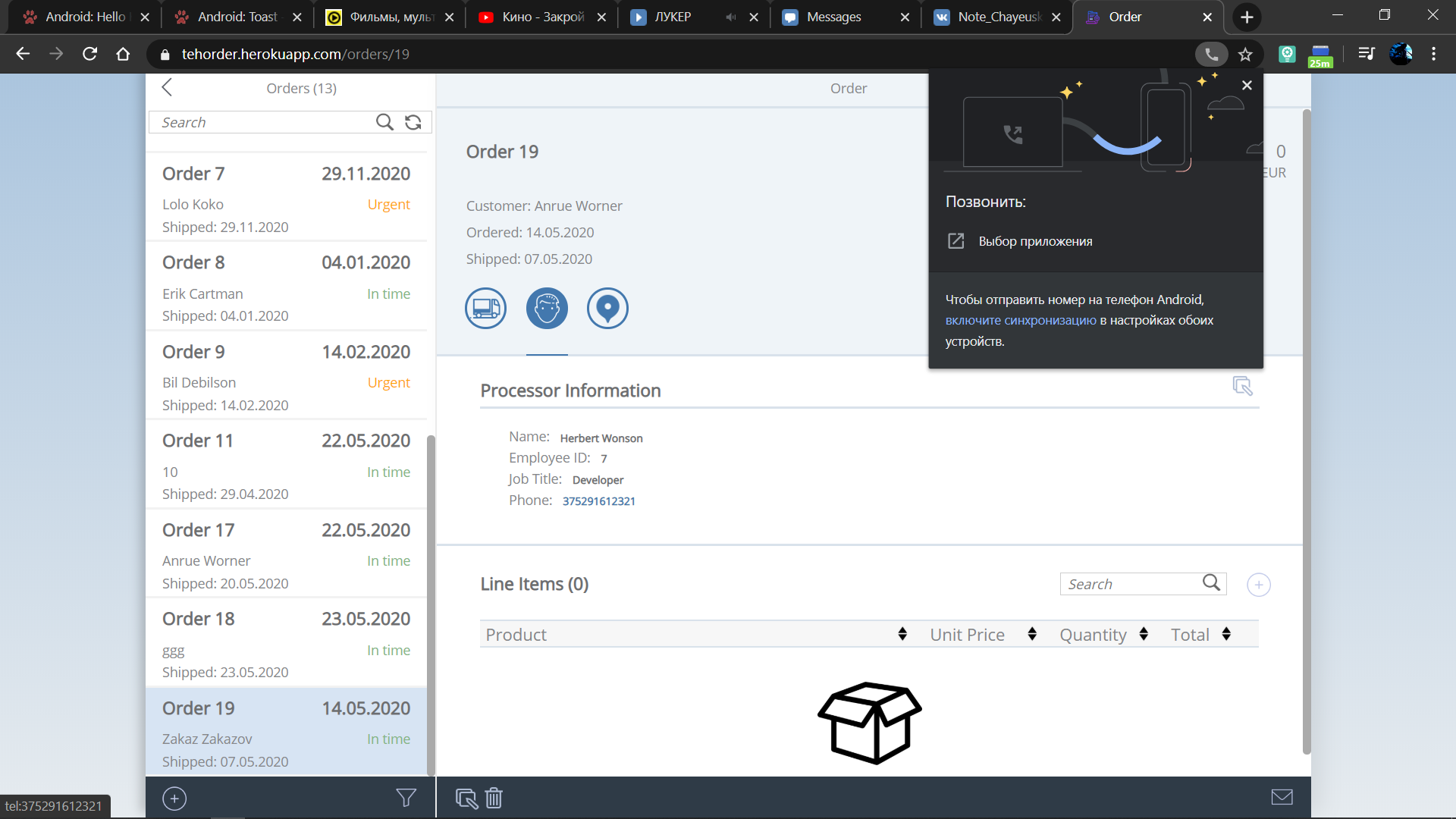


Рисунок 4.8 – Пример набора номера

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.

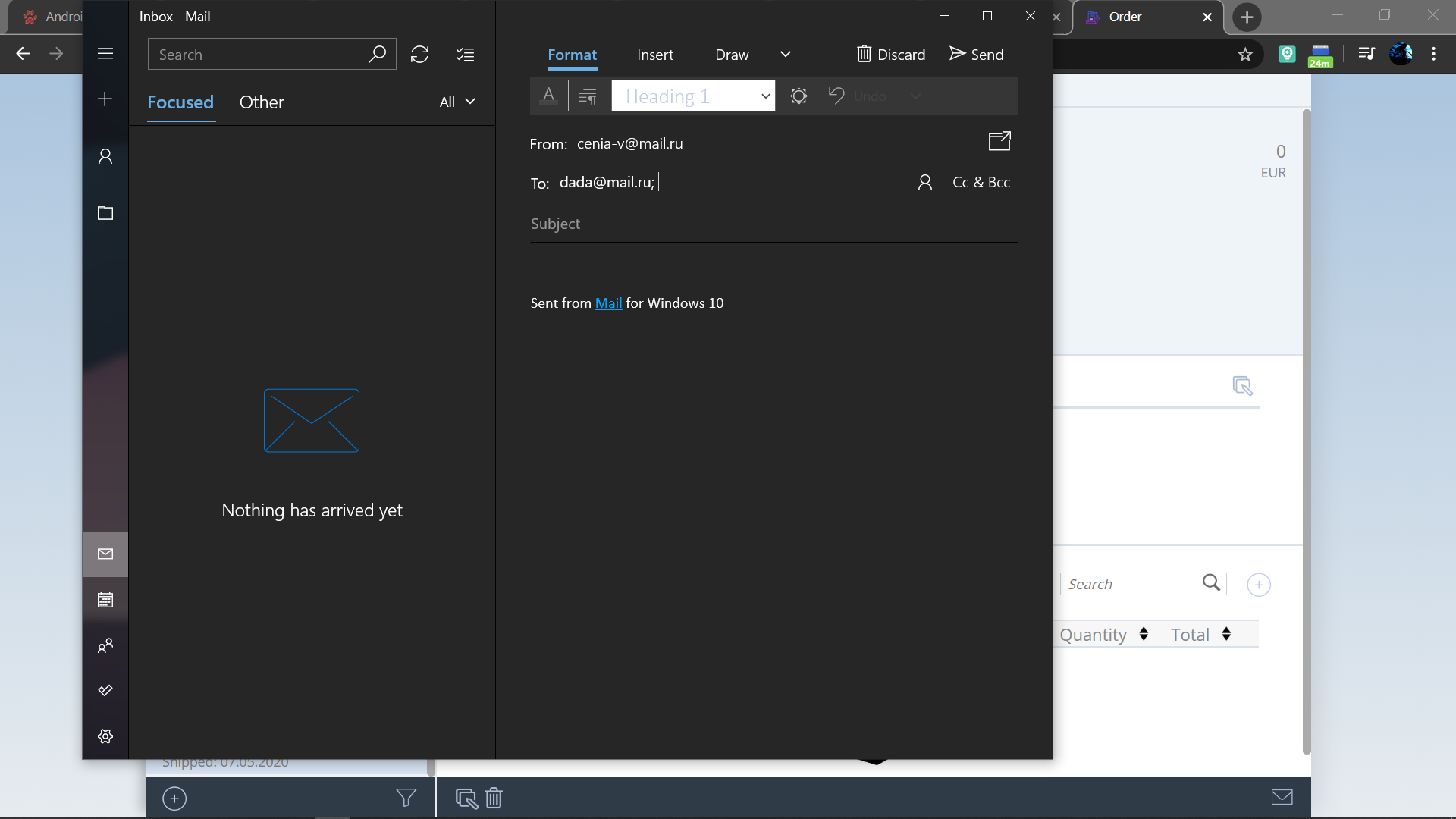


Рисунок 4.9 – Пример отправки e-mail

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

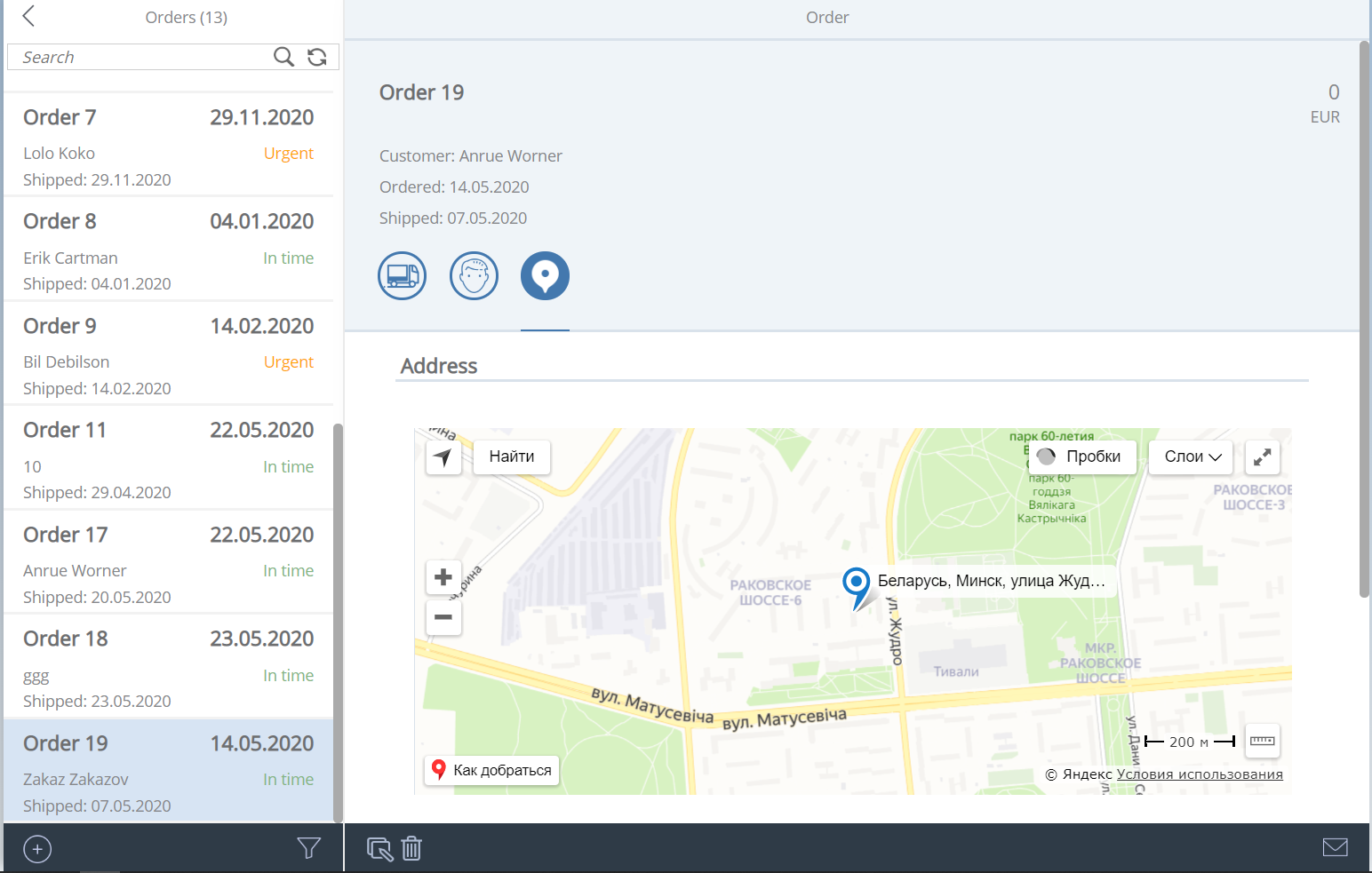


Рисунок 4.10 – Отображение адреса на карте

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

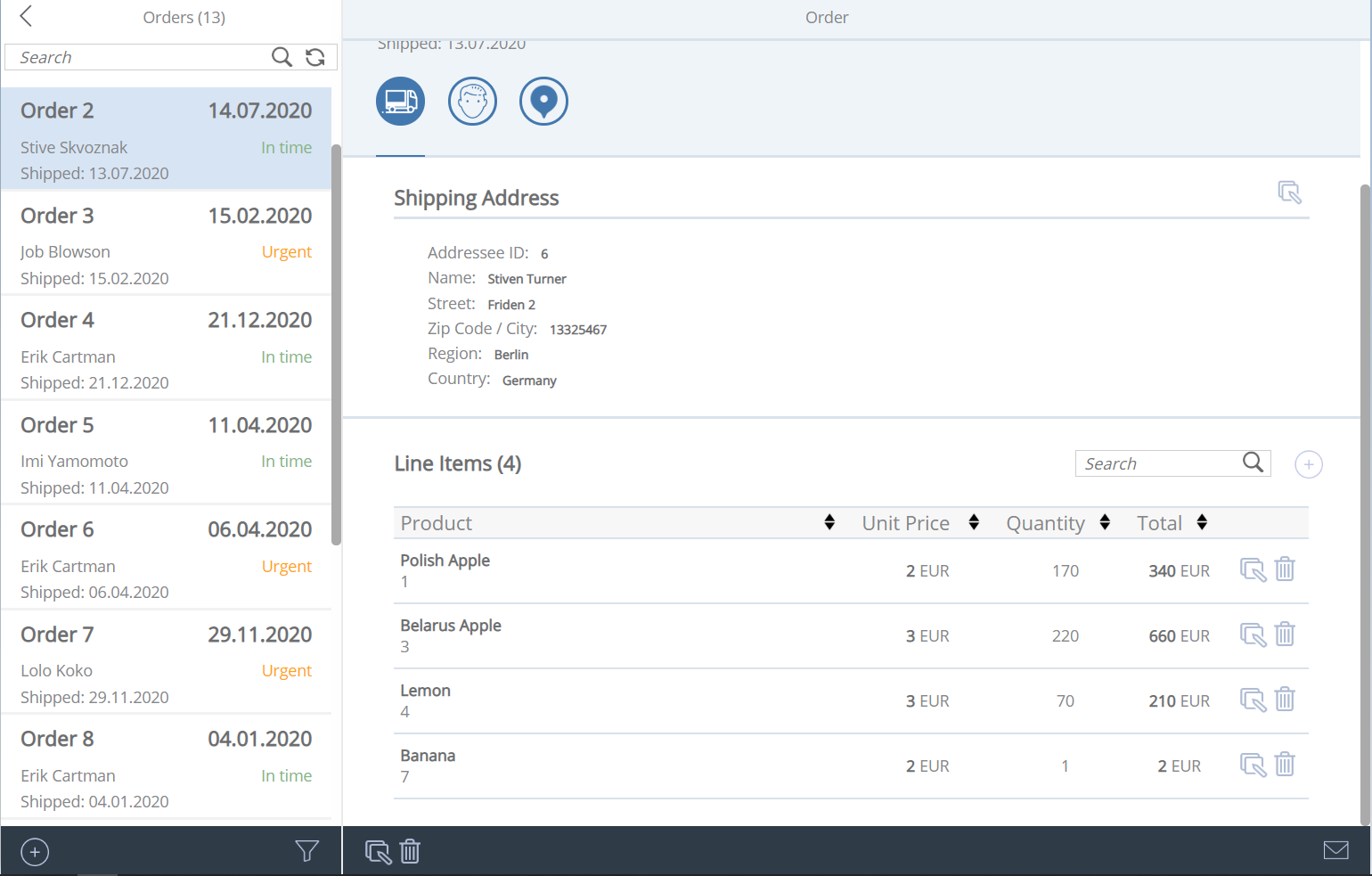


Рисунок 4.11 – Пример списка товаров в заказе

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.

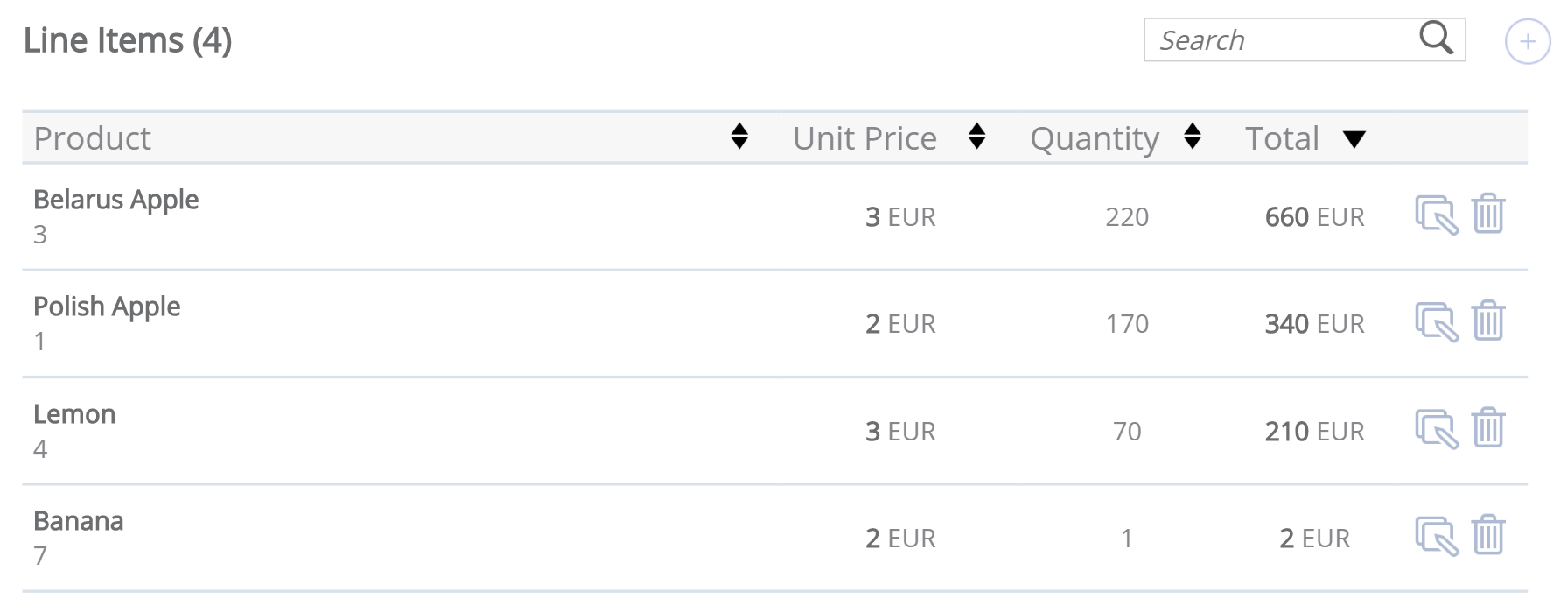


Рисунок 4.12 – Сортировка товара

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

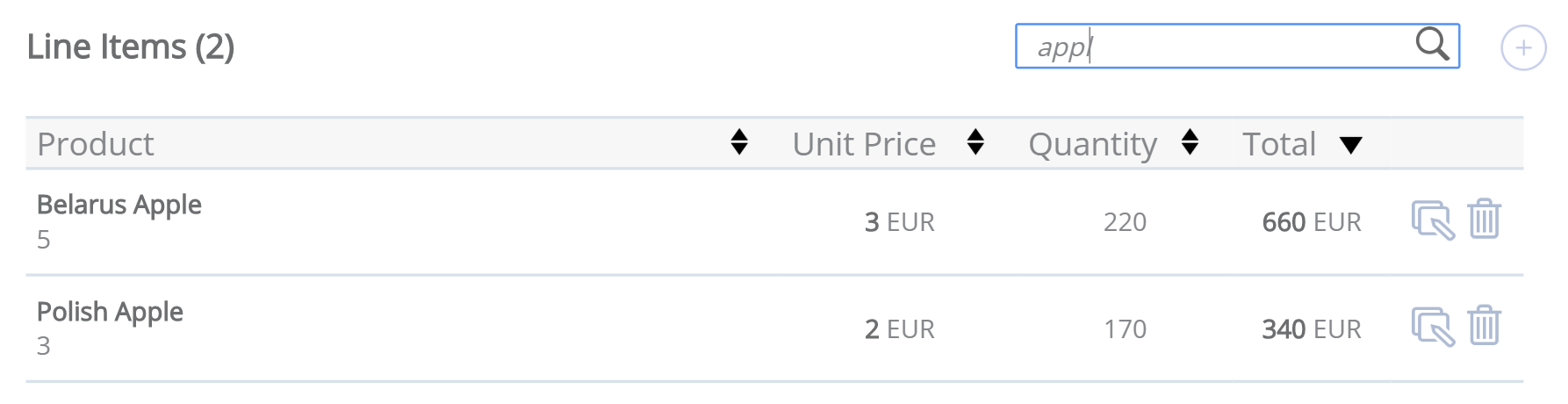


Рисунок 4.13 – Пример поиска товара

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.

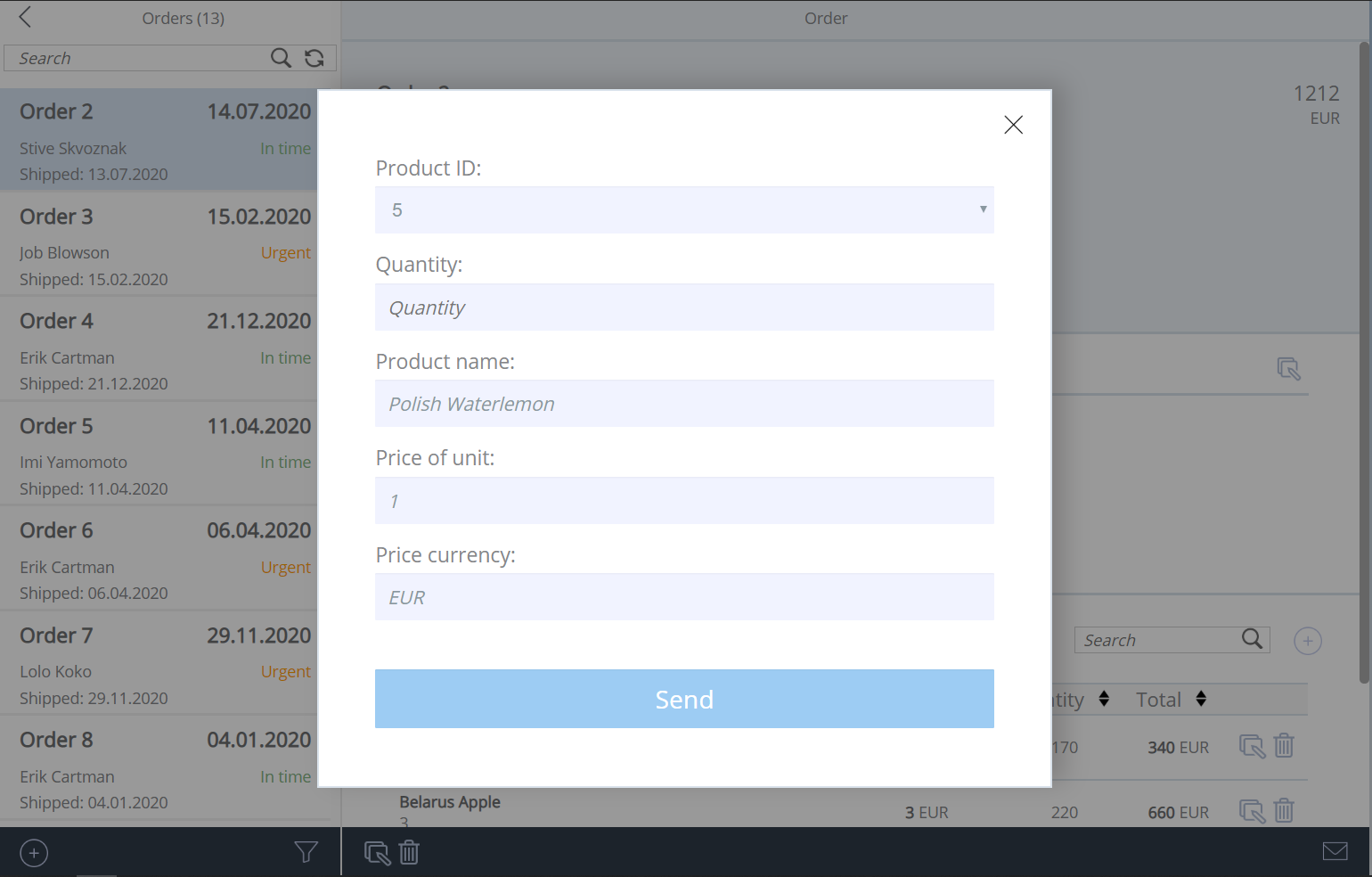


Рисунок 4.14 – Добавление товара

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

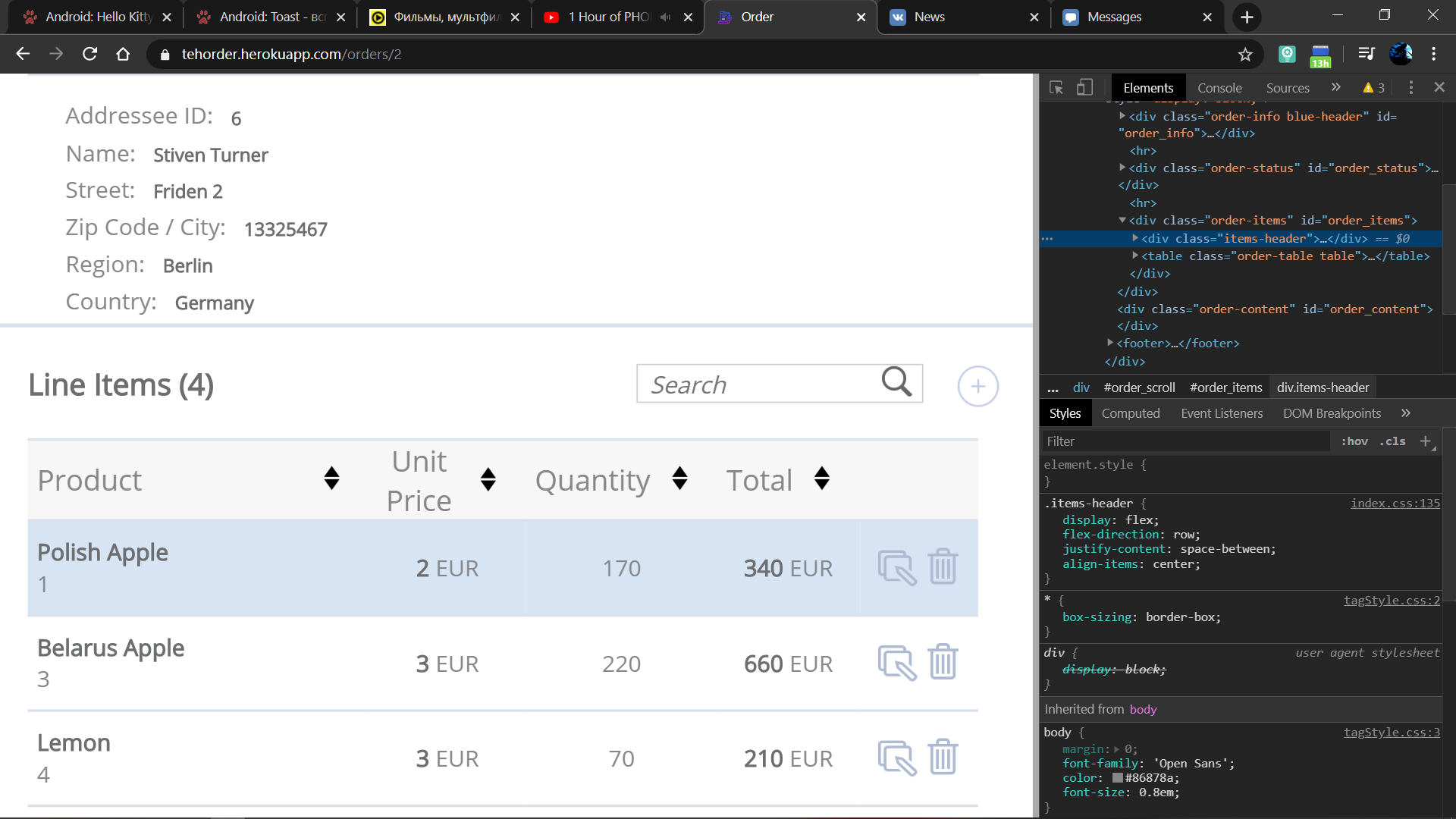


Рисунок 4.15 – Кнопки изменения товара

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.

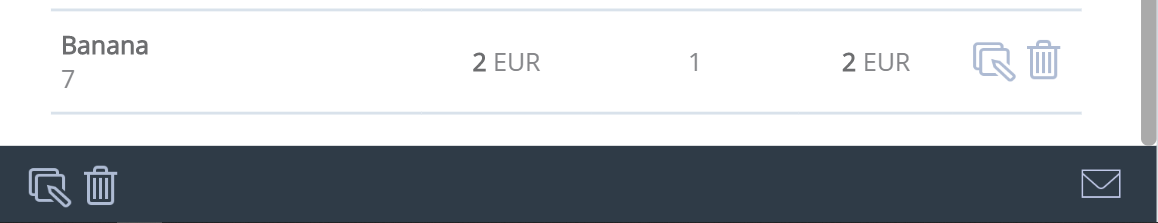


Рисунок 4.16 – Кнопки изменения заказа

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.

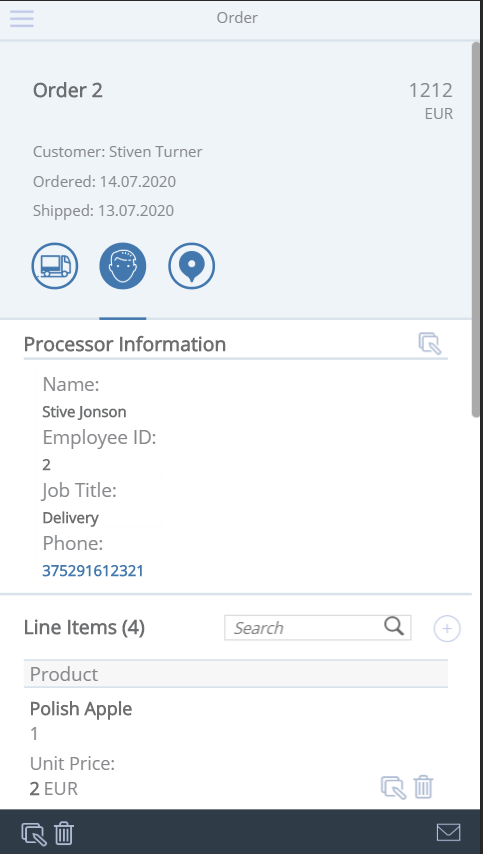


Рисунок 4.17 – Пример мобильной версии приложения

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.

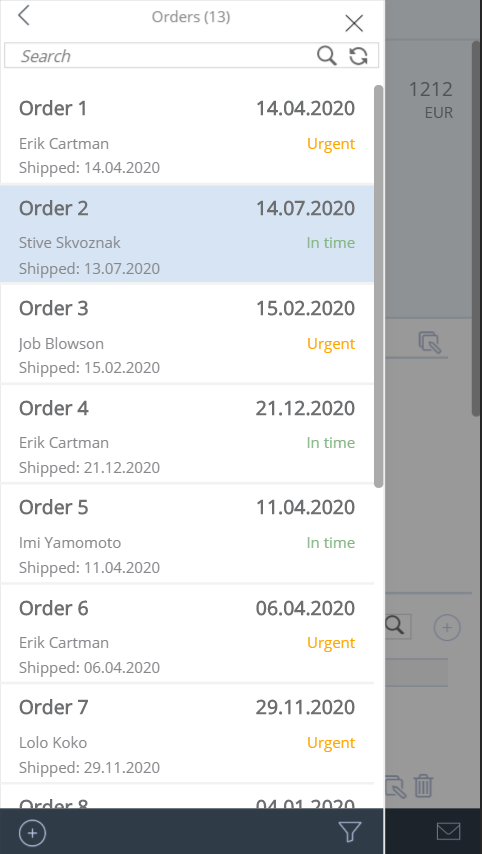


Рисунок 4.18 – Боковая шторка в мобильной версии приложения

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.

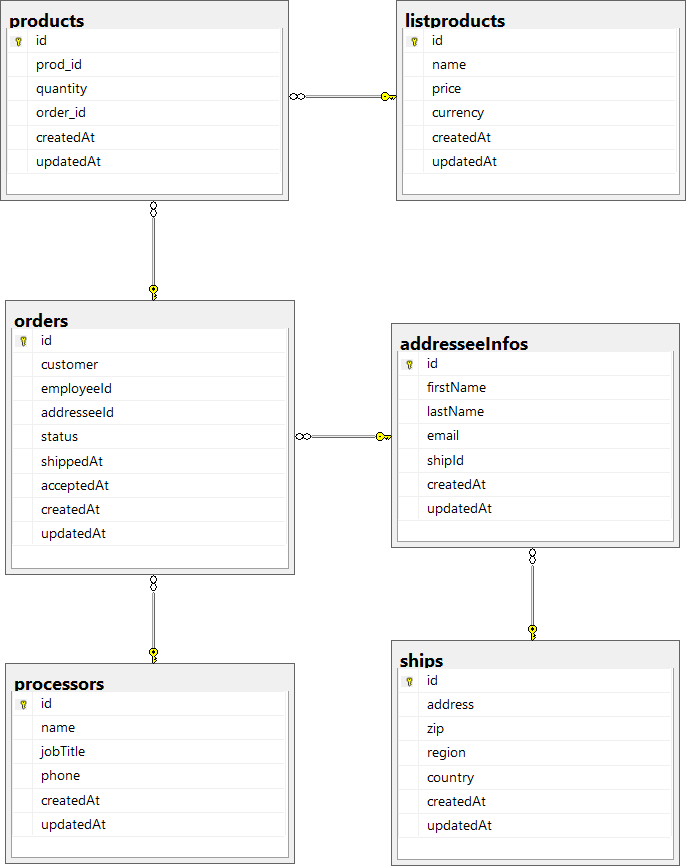
Листов

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'              }          }      });  }; |