МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных

систем и технологий

**КУРСОВАЯ РАБОТА**

по дисциплине: ”Операционные системы и системное программирование”

на тему: **«**Платежная система “VaraPay”»

Выполнил**:** ст.гр.

10701118 **Воробей И.А.**

Принял**:** к.т.н. доц. **Бородуля А.В.**

Минск 2020

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра программного обеспечения информационных

систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по дисциплине: ” Операционные системы и системное программирование ”

на тему: «Платежная система “VaraPay”»

Исполнитель**:** Воробей И.А.

Руководитель**:** к.т.н. доц. Бородуля А.В.

Минск 2020

**Оглавление**

[ВВЕДЕНИЕ 5](#_Toc58781535)

[1. Формулировка задачи 6](#_Toc58781536)

[1.1 Требования 6](#_Toc58781537)

[1.2 Предметная область 7](#_Toc58781538)

[2. Описание программы 8](#_Toc58781539)

[2.1 Физическая структура программы 8](#_Toc58781540)

[2.2 Схема базы данных 12](#_Toc58781541)

[2.3 Описание beck end части программы 13](#_Toc58781542)

[База данных 13](#_Toc58781543)

[Структура Java кода 14](#_Toc58781544)

[Интеграция с web сервисом криптобиржи “Livecoin” 15](#_Toc58781545)

[Интеграция с email 15](#_Toc58781546)

[Безопасность 15](#_Toc58781547)

[Валидация 15](#_Toc58781548)

[2.4 Описание front end части программы 16](#_Toc58781549)

[Адаптивность 16](#_Toc58781550)

[Локализация 16](#_Toc58781551)

[Валидация 16](#_Toc58781552)

[3. Руководство пользователя 17](#_Toc58781553)

[3.1 Client 17](#_Toc58781554)

[3.2 Admin 25](#_Toc58781555)

[4. Модульное тестирование 28](#_Toc58781556)

[1.1 Тестирование с иcпользованием Mockito 28](#_Toc58781557)

[ЗАКЛЮЧЕНИЕ 31](#_Toc58781558)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 32](#_Toc58781559)

[Приложение А. 33](#_Toc58781560)

[Листинг программы 33](#_Toc58781561)

# ВВЕДЕНИЕ

Преимущества кроссплатформенной разработки очевидны. Если программа написана под несколько платформ – у неё будет больше пользователей. Затраты на разработку будут гораздо меньше, чем если бы под каждую платформу программа писалась бы с нуля. Разработчик не привязан к единственной платформе, а поэтому приобретает иммунитет к vendor lock-in. Разработка кроссплатформенного приложения заставляет задуматься о хорошем дизайне программы, избегать привязки к “нестандартностям” и недокументированным функциям платформы, что приводит к меньшему числу багов. Есть и рекурсивный аргумент: если программа написана “в кроссплатформенной манере” – добавление поддержки ещё одной платформы происходит гораздо легче.

# 1. Формулировка задачи

## 1.1 Требования

Необходимо написать (построить) веб-систему согласно требованиям, приведенным ниже, и реализовать определенную функциональность.

Общие требования к проекту:

Приложение реализовать применяя технологии Servlet и JSP. Архитектура приложения должна соответствовать шаблонам Layered architecture и Model-View-Controller. Информация о предметной области должна хранится в БД:

* данные в базе хранятся на кириллице, рекомендуется применять кодировку utf-8
* технология доступа к БД – JDBC (только JDBC)
* для работы с БД в приложении должен быть реализован пул соединений
* при проектировании БД рекомендуется не использовать более 6-8 таблиц
* доступ к данным в приложении осуществлять с использованием шаблона DAO.

Интерфейс приложения должен быть интернационализирован; выбор языков: английский, русский.

Приложение должно корректно обрабатывать возникающие исключительные ситуации, в том числе вести их журналирование. В качестве логгера использовать Log4j.

Классы и другие сущности приложения должны быть грамотно структурированы по пакетам и иметь отражающую их функциональность название.

При реализации бизнес-логики приложения следует при необходимости использовать шаблоны проектирования (например, шаблоны GoF: Factory Method, Command, Builder, Strategy, State, Observer etc), а также необходимо избегать процедурного стиля программирования.

Для хранения пользовательской информации между запросами использовать сессию.

Для обработки объектов запроса(request) и ответа(response) применить фильтры (например, для установки параметра кодировки запроса/ответа).

При реализации страниц JSP следует использовать теги библиотеки JSTL, использовать скриплеты запрещено. Обязательным требованием является реализация и использование пользовательского тега. Просмотр “длинных списков” желательно организовывать в постраничном режиме.

Документацию к проекту необходимо оформить согласно требованиям javadoc.

Оформление кода должно соответствовать Java Code Convention.

Общие требования к функциональности проекта:

* Вход(sign in) и выход(sign out) в/из системы.
* Регистрация.
* Просмотр информации (например: просмотр всех курсов, имеющихся кредитных карт, счетов и т.д.)
* Удаление информации (например: отмена заказа, медицинского назначения, отказ от курса обучения и т.д.)
* Добавление и модификация информации (например: создать и отредактировать курс, создать и отредактировать заказ и т.д.)

## 1.2 Предметная область

Система Платежи. Клиент имеет одну или несколько Кредитных Карт, каждая из которых соответствует некоторому Счету в системе платежей. Клиент может при помощи Счета сделать Платеж, заблокировать Счет и пополнить Счет. Администратор снимает блокировку.

# 2. Описание программы

## 2.1 Физическая структура программы

Софт реализован на Java 14 в среде разработки Intelij Idea.

Все части программы реализованы в различных классах. Были задействованы такие технологии как Tomcat, Servelt API, Postgresql, Gradle, JSP, JSTL, TestNg, Mockito, JavaDoc, Log4J2, JDBC, HTML, CSS, JS, и т.д.

Результатом разделения и проектирования стала следующая структура классов и файлов. Пакет com.epam.varapay представлен на рисунках 2.1 – 2.6.

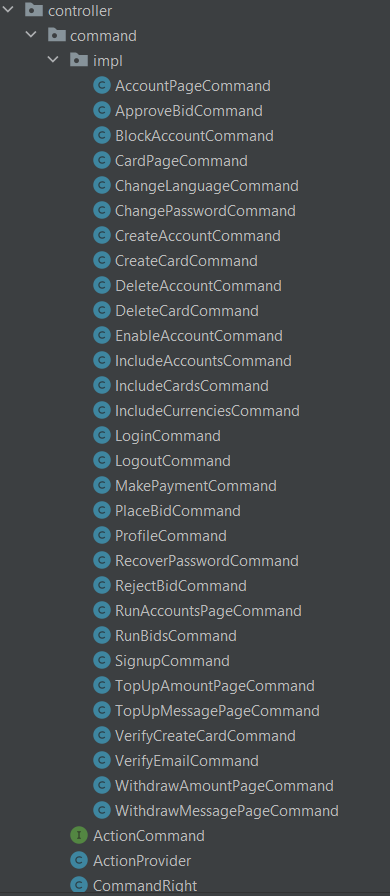


Рисунок 2.1 – Структура контроллера часть 1

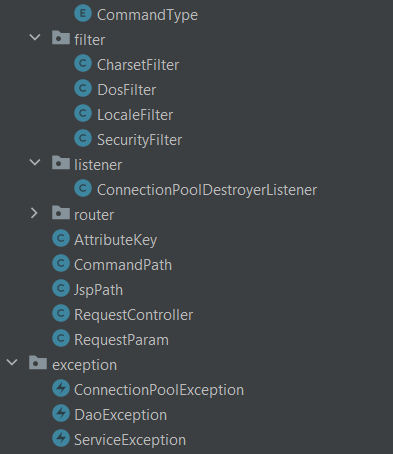


Рисунок 2.2 – Структура контроллера часть 2

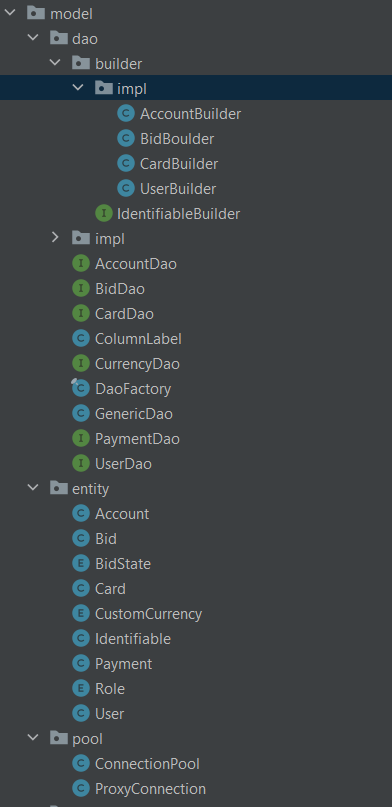


Рисунок 2.3 – Структура модели часть 1

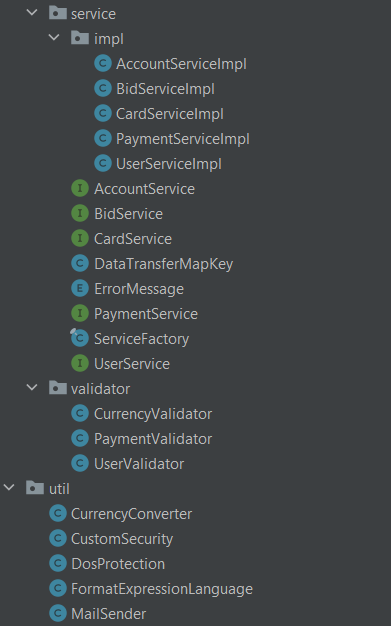


Рисунок 2.4 – Структура модели часть 2

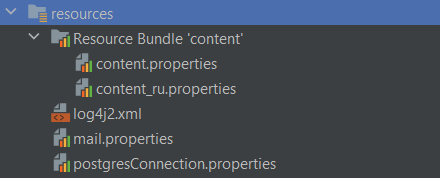


Рисунок 2.5 – Структура дирриктории с ресурсами

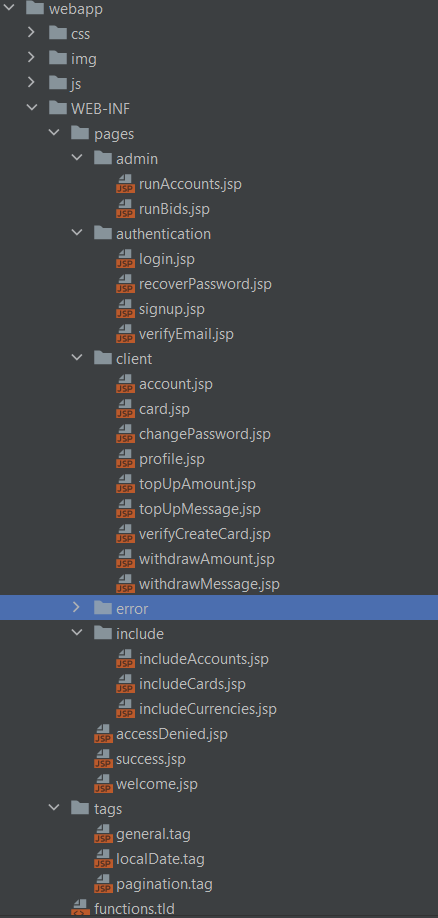


Рисунок 2.6 – Структура дирриктории webapp

## 2.2 Схема базы данных

Схема базы данных показана на рисунке 2.7.

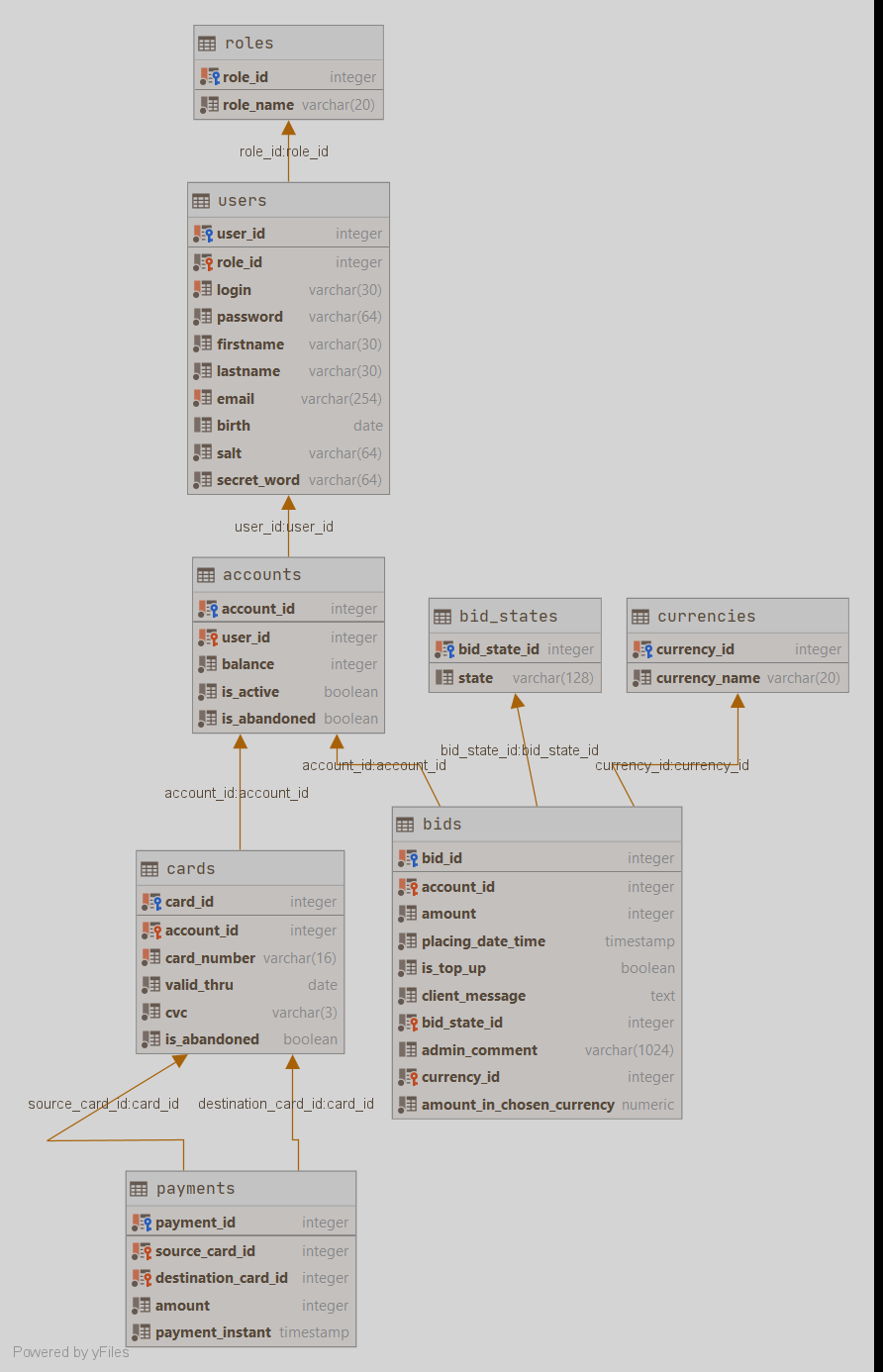


Рисунок 2.7 – Схема базы данных

## 2.3 Описание beck end части программы

### ****База данных****

В основе модели лежит база дынных. В данном проекте используется Postgresql. Таблицы базы данных соответствуют третьей НФ, то есть база является нормализованной. Данная БД является примером OLTP. Обработка транзакций в реальном времени. Требуется минимальное время отклика. Используются INSERT, UPDATE, DELETE. Объем такой базы не большой. Потеря данных приводит к денежной и юридитеческой ответсвенности. Избыточность недопустима, так как это усложняет операции изменения данных(изменение одного объекта затрагивает большое количество строк). Имеют более сложные связи.

В базе есть 8 таблиц. Все связи один ко многим. Для обращения к базе из приложения использую JDBC. Реализован пул соединений. Для создания объекта Connection необходимо:

1. import java.sql.\*;
2. Регистрация драйвера. Есть два способа:
3. Class.forName(“org.postgresql.Driver”); или DriverManager.registerDriver(new org.postgresql.Driver());
4. DriverManager.getConnection(String url, Properties props); есть три перегрузки

В приложении используется разные уровни изоляции транзакций:

0 — Неподтверждённое чтение (Read Uncommited, Dirty Read, грязное чтение) — чтение незафиксированных изменений своей транзакции и конкурирующих транзакций, возможны нечистые, неповторяемые чтения и фантомы

1 — Подтверждённое чтение (Read Commited) — чтение всех изменений своей транзакции и зафиксированных изменений конкурирующих транзакций, нечистые чтения невозможны, возможны неповторяемые чтения и фантомы

2 — Повторяемое чтение (Repeatable Read, Snapshot) — чтение всех изменений своей транзакции, любые изменения, внесённые конкурирующими транзакциями после начала своей, недоступны, нечистые и неповторяемые чтения невозможны, возможны фантомы

3 — Упорядоченный — (Serializable, сериализуемый) — [упорядоченные (сериализуемые)](http://ru.wikipedia.org/w/index.php?title=Сериализуемость&action=edit&redlink=1) транзакции. Идентичен ситуации, при которой транзакции выполняются строго последовательно, одна после другой. То есть транзакции, результат действия которых не зависит от порядка выполнения шагов транзакции (запрещено чтение всех данных, изменённых с начала транзакции, в том числе и своей транзакцией). Фантомы невозможны.

В postgresql уровни

READ\_UNCOMMITED = READ\_COMMITED

REPEATABLE\_READ = SERIALIZABLE

Отличие SERIALIZABLE от READ\_COMMITED в том что при READ\_COMMITED если

1. Одновременно начались две транзакции
2. Первая изменила данные
3. Вторая изменила те же данные(вторая автоматически блокируется и ждет)
4. Первая закомитилась. (Вторая разблокировалась и перопределила изменения первой)

То при SERILIZABLE когда первая комитится во второй будет ошибка(это вынудит сделать ролбек второй транзакции). Это происходит по тому что вторая транзакция началсь вместе с первой(пока не закомитилась первая). А если бы вторая началсь после того как закомитилась первая, то втроая бы выполнилась успешно.

Если две транзакции обновляют разные строки, то вторая транзакция не будет блокироваться, а сразу выполнится, никаких огранизений нет.

### Структура Java кода

Был использыван патерн проктирования MVC. Предполагает разделение данных приложени и управляющей логики на 3 отдельных компонента. Модификация каждого осуществляется независимо. Применение шаблона на практике приводит к тому, что базовая тречслойная архитектура распадается на многоуровневую. Причем один уровень знает о существовании только следующего за ним. Модель представляет данные и методы работы с ними.

Кроме того, были реализованы следующие паттерны: command, proxy, dto, flyweight, strategy, singleton, chain of responsibility, state.

Были соблюдены принципы SOLID, Yagni, Kiss, Dry. Пример Liskov substitution principle : подклассы не могут замещать поведение базовых классов, а могут лишь дополнять. Если s является подтипом t, тогда объекты типа t в программе могут быть заменены на объекты типа s. Если в функцию приходит родитель, то если передать наследника, все сработает. Подкласс не должен требовать от вызывающего кода больше чем базовый класс и не должен предоставлять вызывающему коду меньше чем базовый класс. Пример несоблюдения: parant.m() – ok; child.init(); cild.m(); - ok; But simply child.m() – error. Клиентский код не должен знать с каким типом он заботает(проверка instanceof – значит принцип не соблюдается).

Иногда незвозможно соблюсти LSP при наследовании, квадрат не может быть наследником прямоугольника, в противном случае LSP несоблюден.

Шаблонный метод увеличивает шансы того что LSP будет соблюден. Программист который наследует свой класс от класса с шаблонным методом скорее всего будет вынужден реализовать предполагаемую логику. Запустить двигатель(подключить стартер, подготовить топливо..)

Dependency inversion. Используй интерфейсы вместо конретных реализаций. Модули зависят от абстракций.

### Интеграция с web сервисом криптобиржи “Livecoin”

Приложени позволяет проводить операции с криптовалютой. Обращение к сервру биржи происходит каждую минуту для обновления цен на криптовалюты. Для каждой криптовалюты запущен отдельный поток, в котором и происходит обращение с таймером в бесконечном цикле.

### Интеграция с email

Регистрация пользователей осуществляется с подтверждением email. Это позволяет дополнительно верефицировать операции свзянные с переводом денег. Так же можно востановит забытый пароль.

### Безопасность

Пароли хешируются с помощью SHA-256 с солью. Таким образом, даже если злоумышленние получит доступ к БД, он не сможет идентифицировать одиноковые пароли.

Если злоумышленник получит логин и пароль, то он все равно не сможет вывести деньги без доступа к почте пользывателя.

Присутсвует защита от Dos.

### ****Валидация****

**Валидация на сервере полностью покрывает все случаи и является независимой от валидации на клиеите.**

## 2.4 Описание front end части программы

### Адаптивность

Благодаря адаптивной версте приложение можно запустить на устройстве с любой формой экрана. Для реализации break points были использованы CSS media queries. При сужении экрана header сворачивается в burger menu с аншимацией, также все элементы выстраиваются в ряд. Растояния между блоками уменьшаются, изменяется шрифт. Адаптивность дотупна на всех страницах.

### Локализация

Поддерживатся два языка RU, EN. Язык можно перключить на любой странице с сохранением состояния.

### Валидация

Кроме валидации на сервере присутсвует валидация на клиенте. Например проверка соответствия паролей при регистрации проверяется с помощью JS.

# 3. Руководство пользователя

## 3.1 Client

Каждая страница отображает Header, Footer, Sidebar одинаково кроме случаев с разными ролями пользователей. Тогда ссылки в хедере меняются. Главная страница продемонстрирована на рисунке 3.1. В sidebar отображаются актульные цены на криптовадюту. В header можно поменять язык.

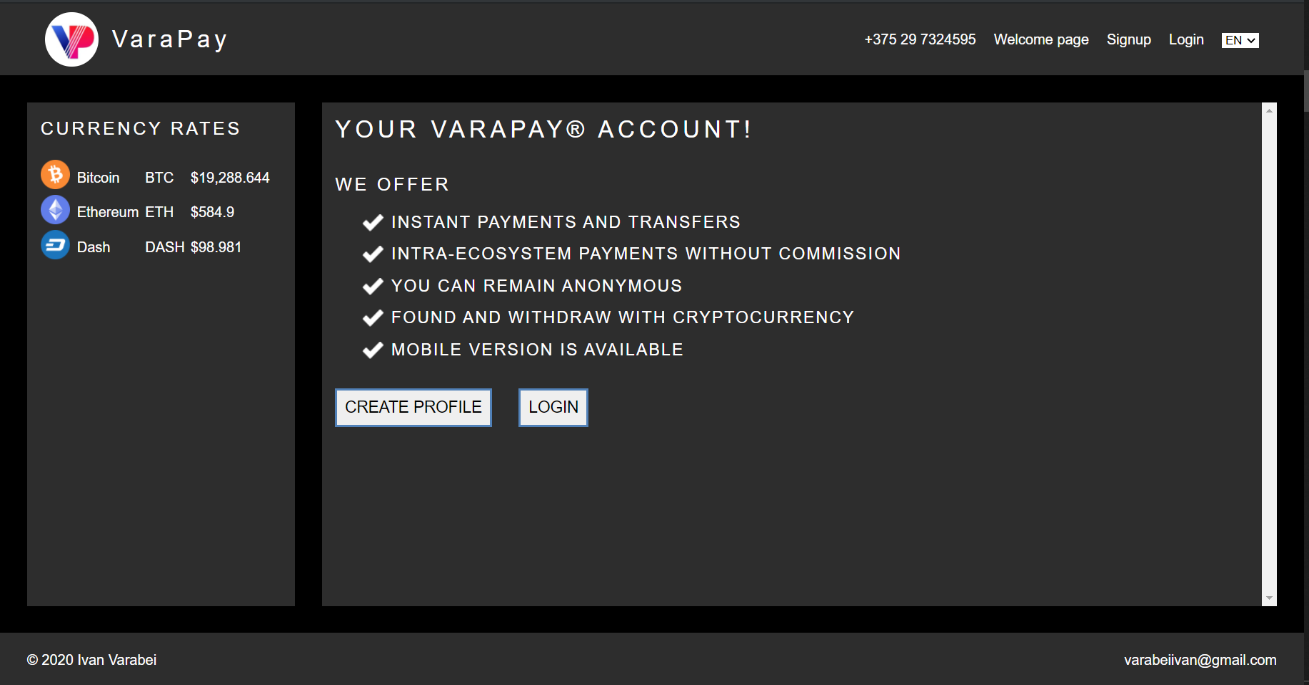


Рисунок 3.1 – Главная страница

Пример валидации на клиенте на странице регистрации показан на рисунке 3.2. При переключении на русский язык сообщения валидации так же сеняют язык. Пример валидации на сервере и на улиенте показан на рисунке 3.3.

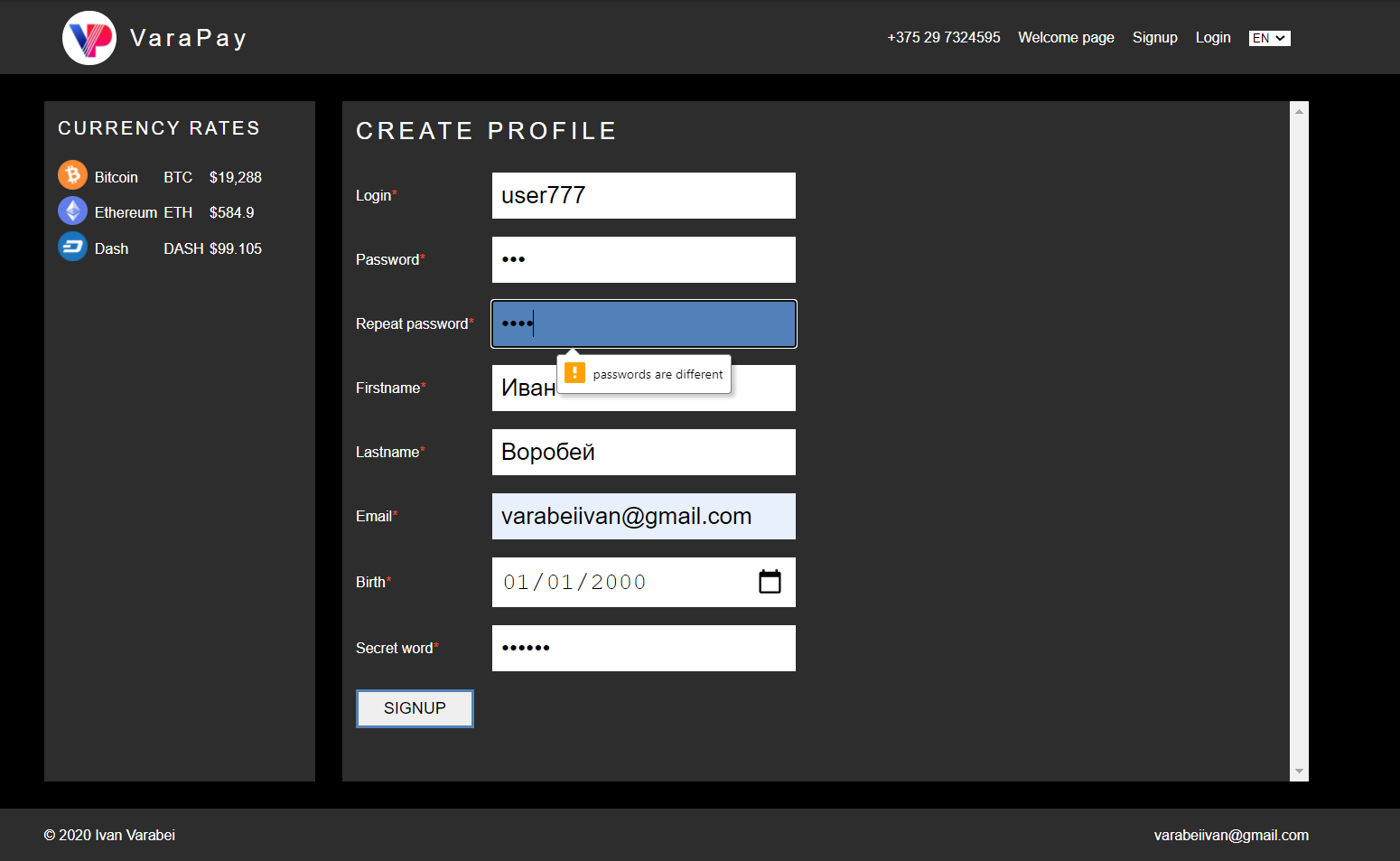


Рисунок 3.2 – Валидаци на клиенте

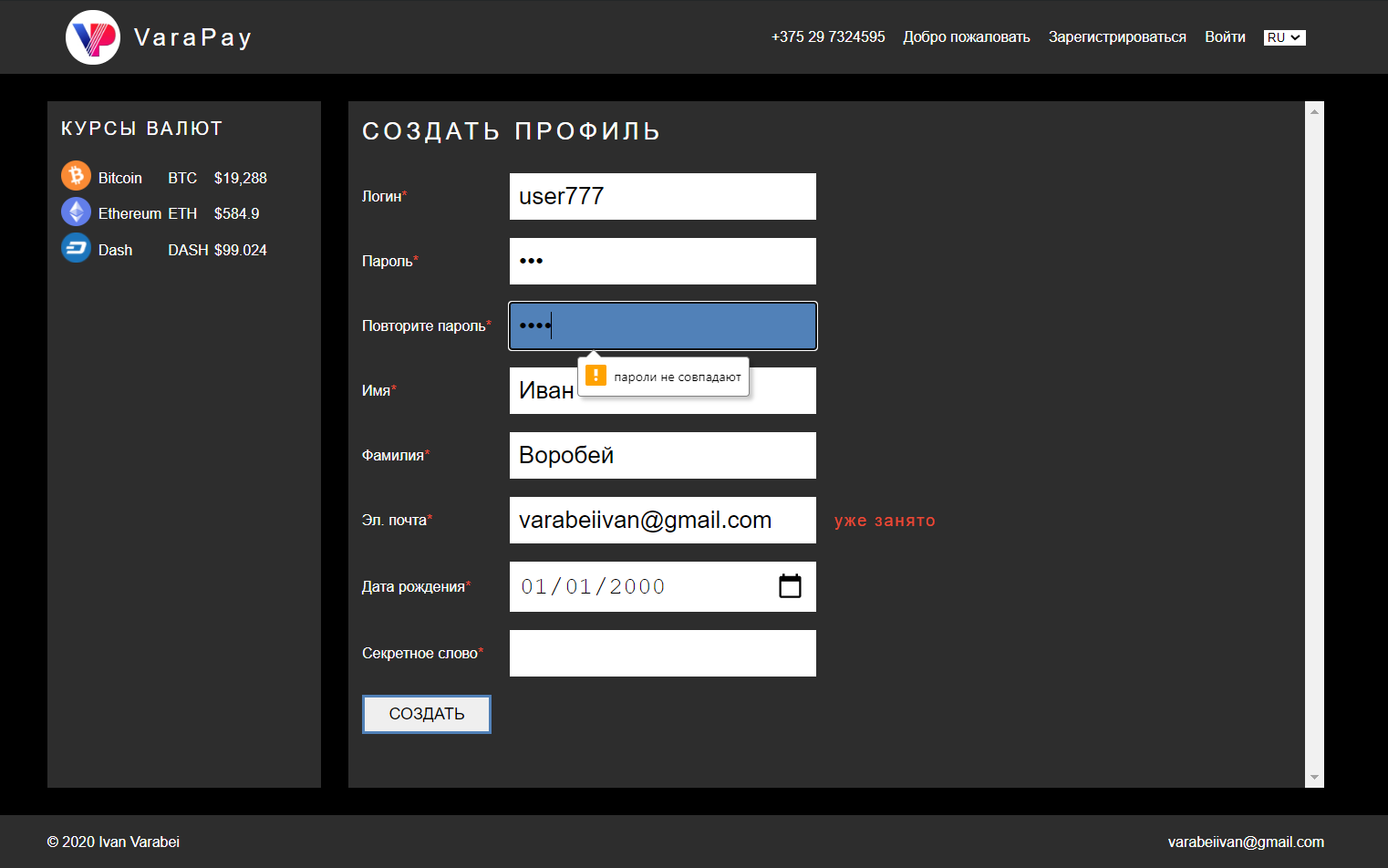


Рисунок 3.3 – Окно управления после запуска

Страница входа в систему показана на рисунке 3.4.

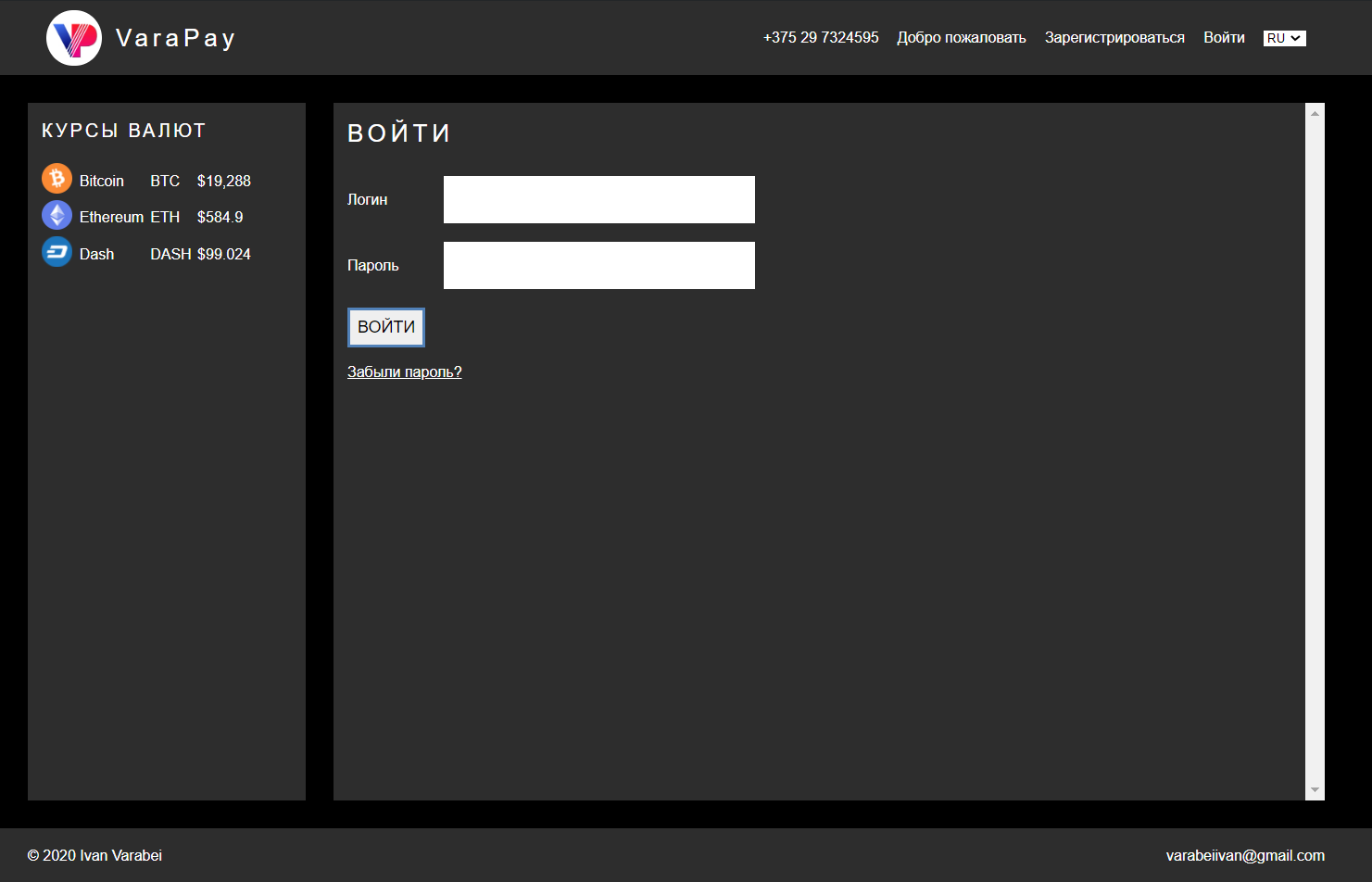


Рисунок 3.4 – Страница входа в систему

Страница восстановления пароля на рисунке 3.5.

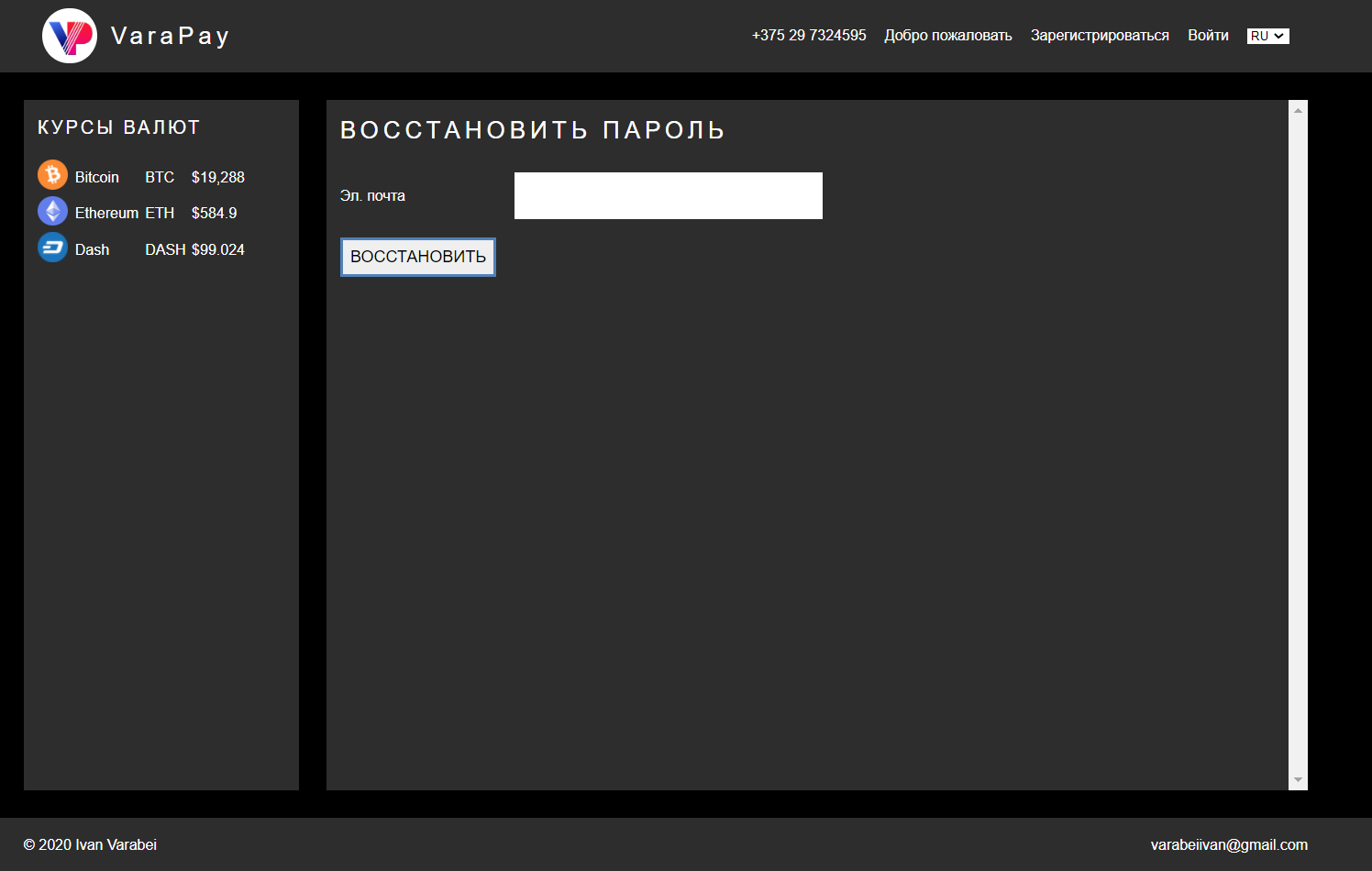


Рисунок 3.5 – Страница восстановления пароля

Страница профиля показана на рисунке 3.6. Тут пользователь видит свои счета и карты. Здесь можно создавать и удалять счета и карты.

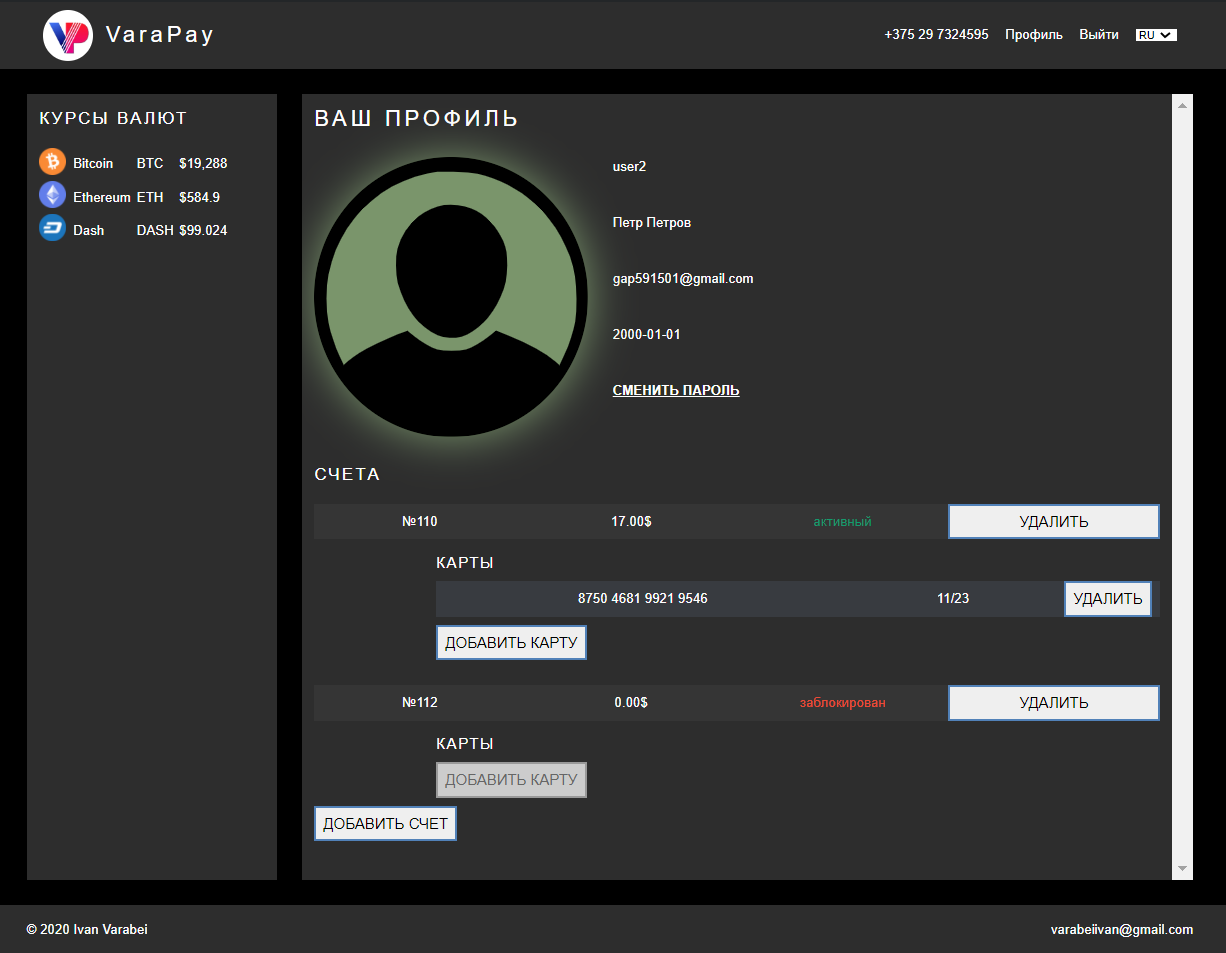


Рисунок 3.6 – Профиль

Страница изменения пароля на рисунке 3.7.

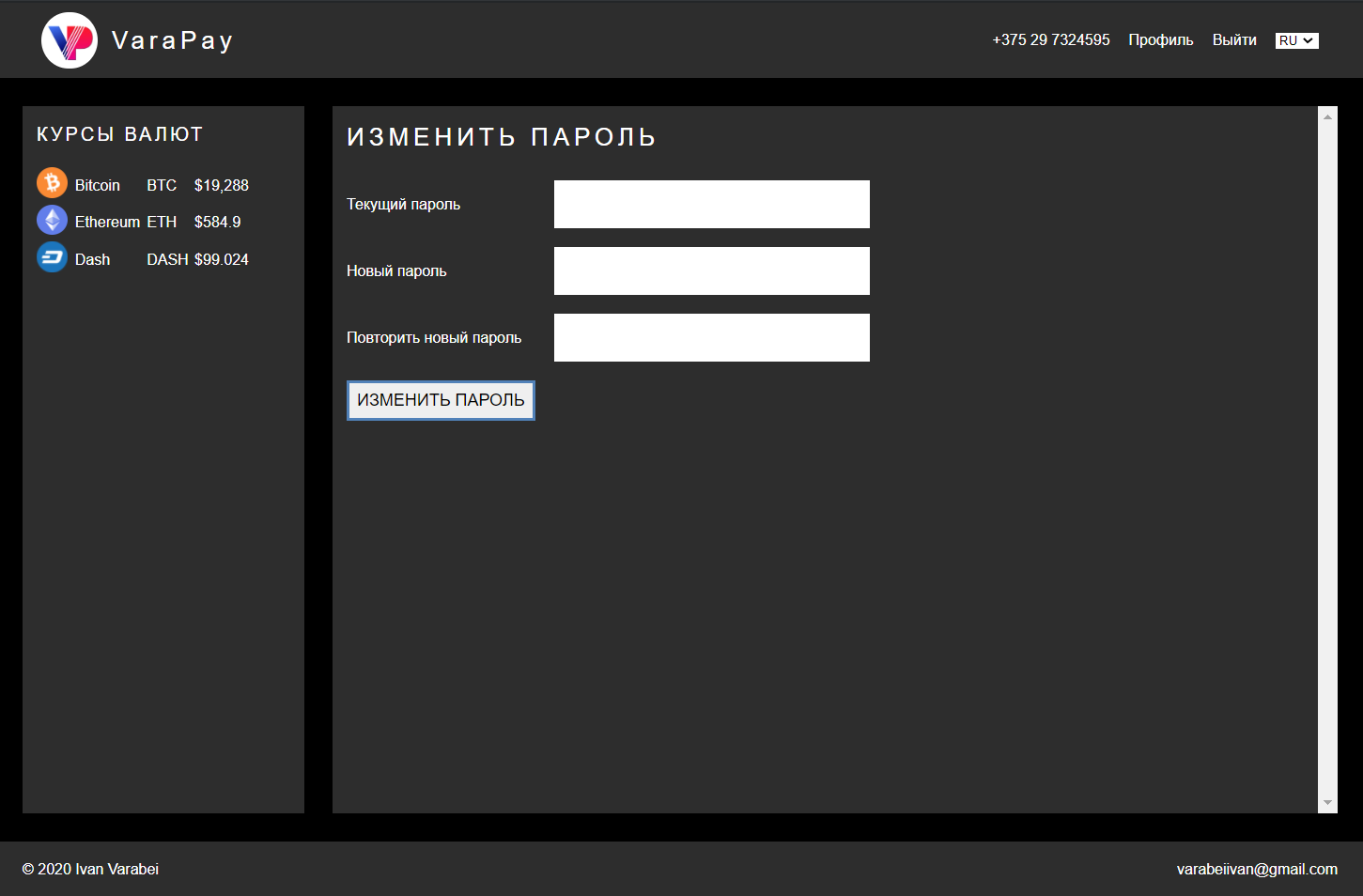


Рисунок 3.7 – Страница изменения пароля

Страница счета позволяет блокировать, пополнять и выводить деньги. Кроме того можно просматривать историю пополнений и выводов. Демонстрирует рисунок 3.8.

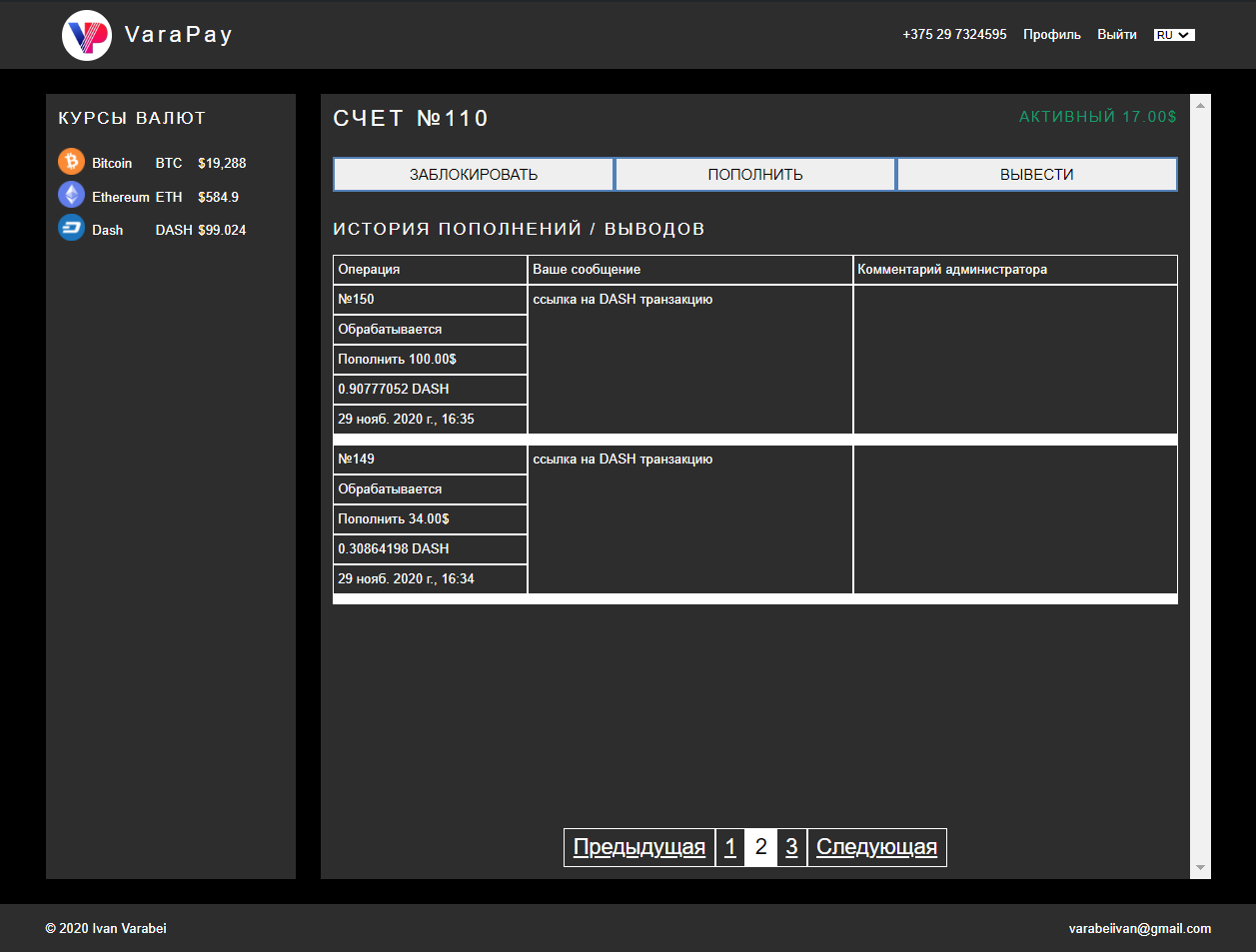


Рисунок 3.8 – Счёт

Для пополнения счета необходимо ввести желаемое количестов в долларх и выбрать криптовалюту в которой будет осуществляться пополнение рисунок 3.9. Далее после конвертации будет будет показана необходимая сумма в криптовалюте рисуное 3.10.

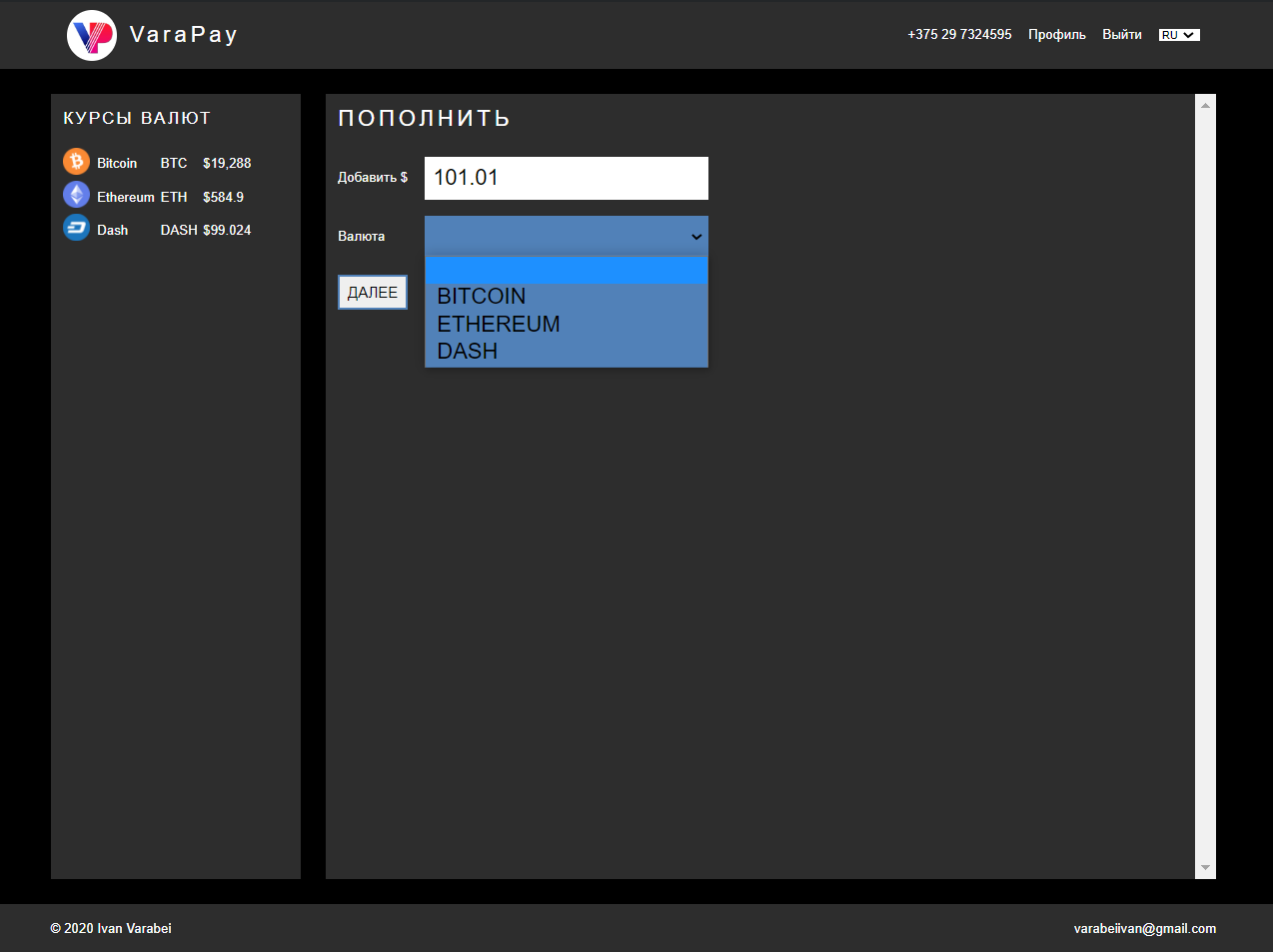


Рисунок 3.9 – Ввод количества и выбор валюты

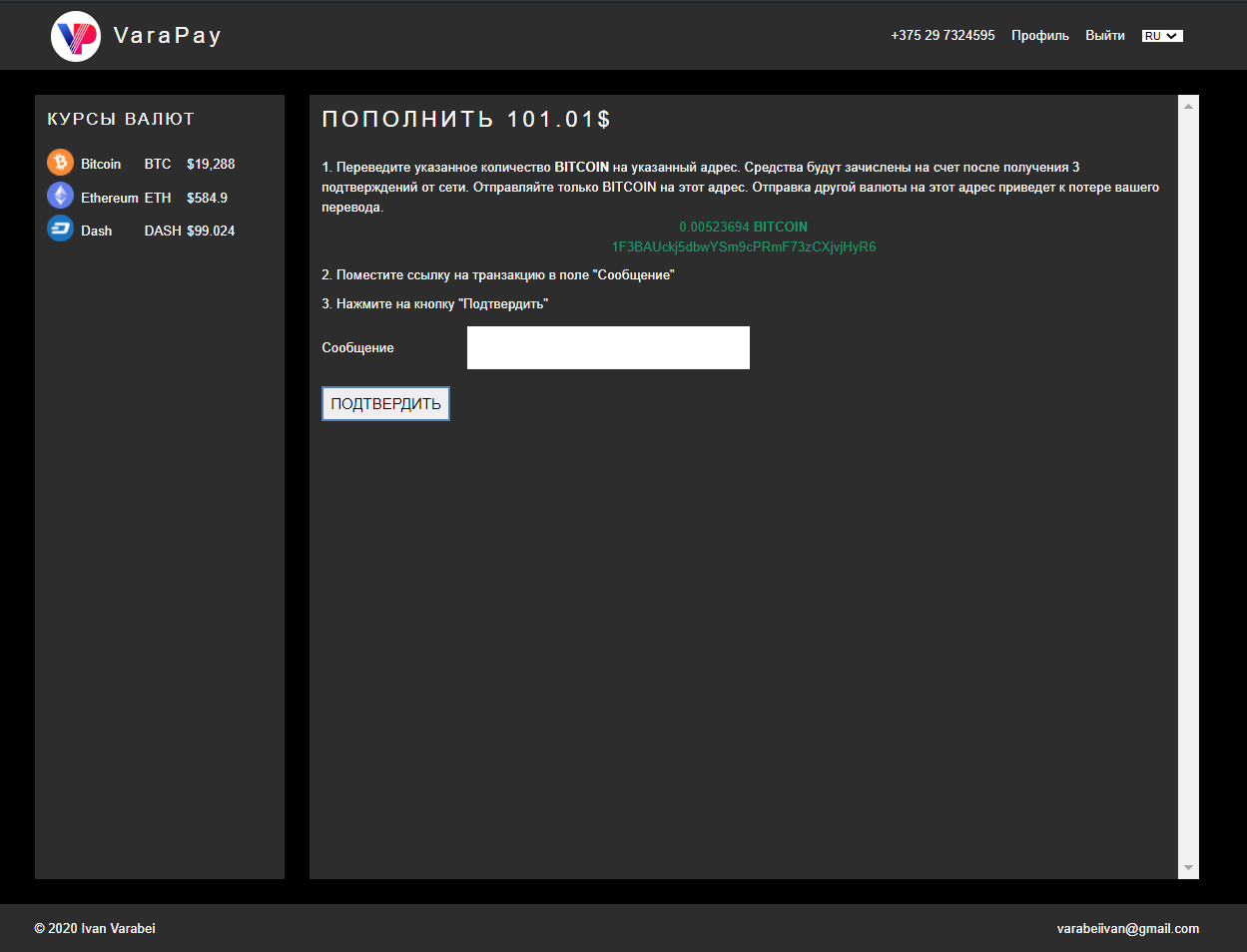


Рисунок 3.10 – Демонстрация суммы в криптовалюте

Приложение позволяет посылать денежные транзакции другим пользователям. Это можно сделать на стрнице карты. На этой стрнице так же отображается история транзакций с пагинацией. Валидация на клинте не позволяет ввести большее кличество денег, чем есть на счете. Продемонстрированно на рисунке 3.11. Та же страница после сжатия экрана продемонстрирована на рисунках 3.12, 3.13. После нажатия на значек бургера выпадает header, продемонстрировано на рисунке 3.14.

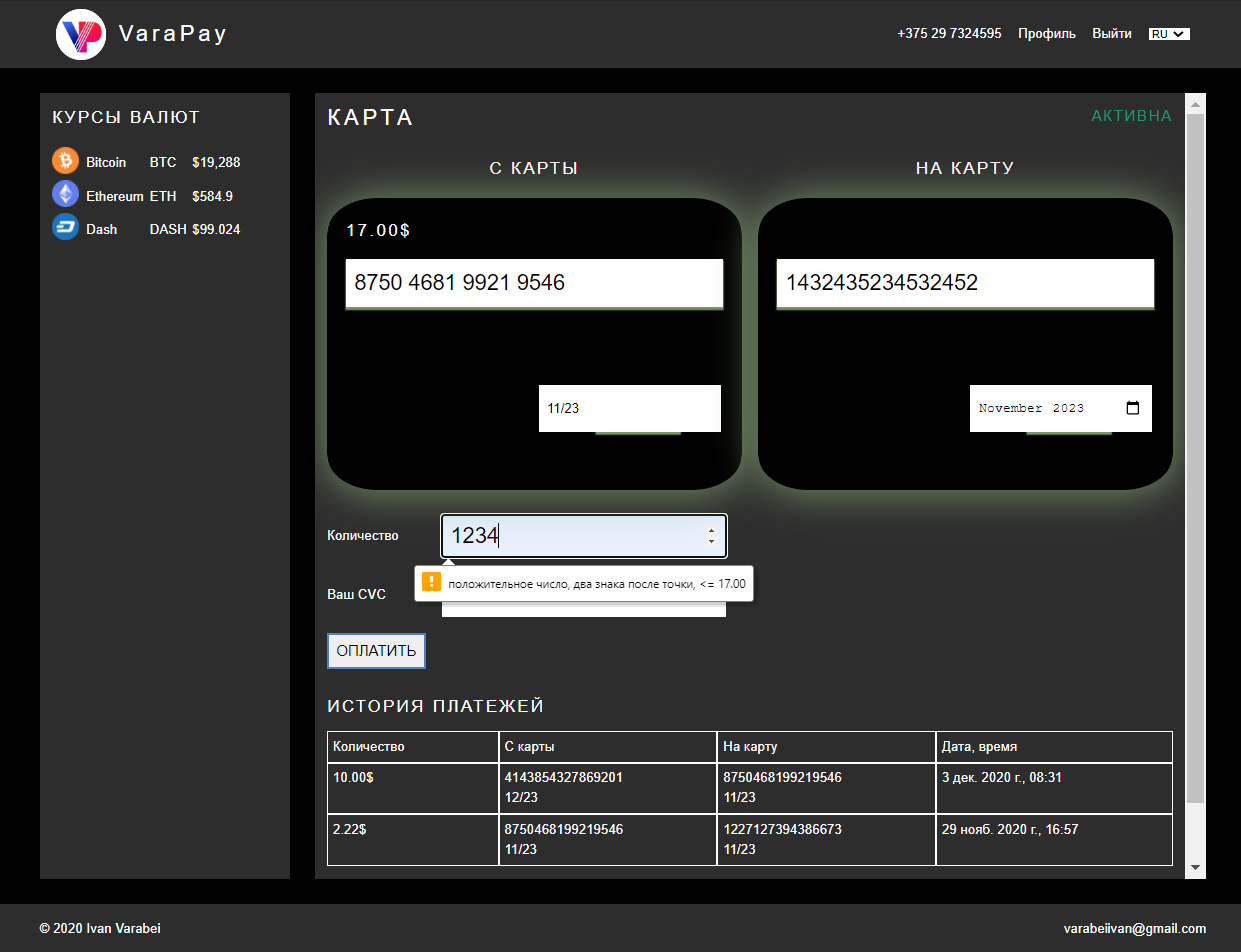


Рисунок 3.11 – Страница создания платежа

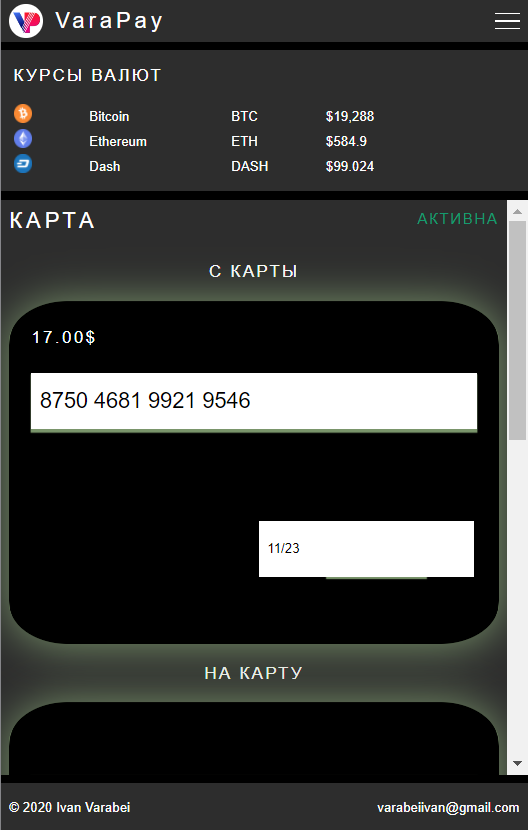


Рисунок 3.12 – Адаптация под мобильное устройство первая часть страницы

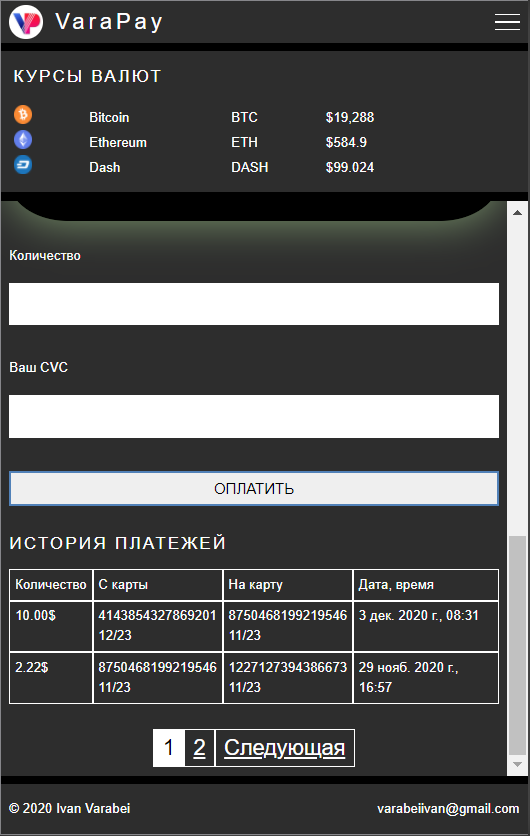


Рисунок 3.13 – Адаптация под мобильное устройство вторая часть страницы

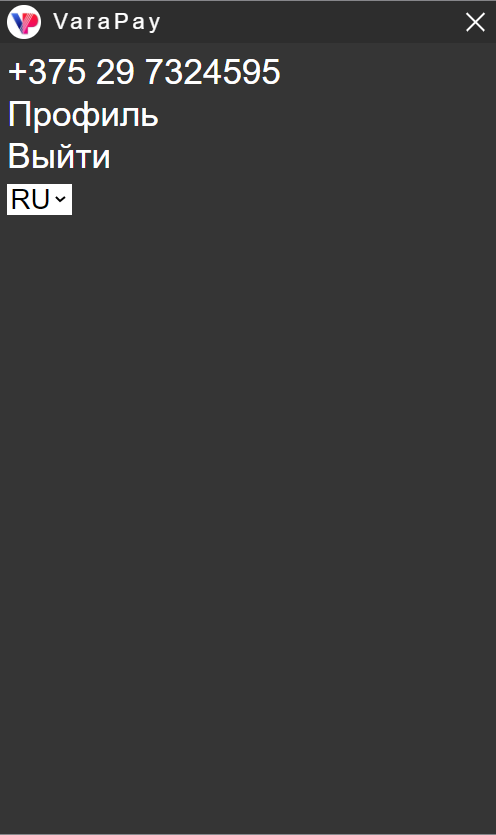


Рисунок 3.14 – Header в мобильном режиме

## 3.2 Admin

Администратор не может сам иметь счета. Кнопка «добавить счет» перебросит на страницу с сообщением запрещающим действие. Профиль администратора показан на рисунке 3.15.

У администратора есть три основные задачи:

* Проверять входящие платежи когда клиент оставляет заявку на поплнеиние. Администратор либо принимает заявку либо отклоняет.
* Отправлять криптовалюту на кошелки клиентов, когда они оставляют заявку на вывод. Администратор либо принимает заявку либо отклоняет. Продемонстрированно на рисунке 3.16.
* Разблокировать заблокированные счета по звонку клиента. Клиент звнонит администратору по телефону указанному в header, называет секретное слово, которое клиент ввёл при регистрации. Администратор на ходит заблокированные счета этого клиента и вводит секретное слово, если совпадает, то счет разблокирутся. Стрница поиска счетов на рисунке 3.17.

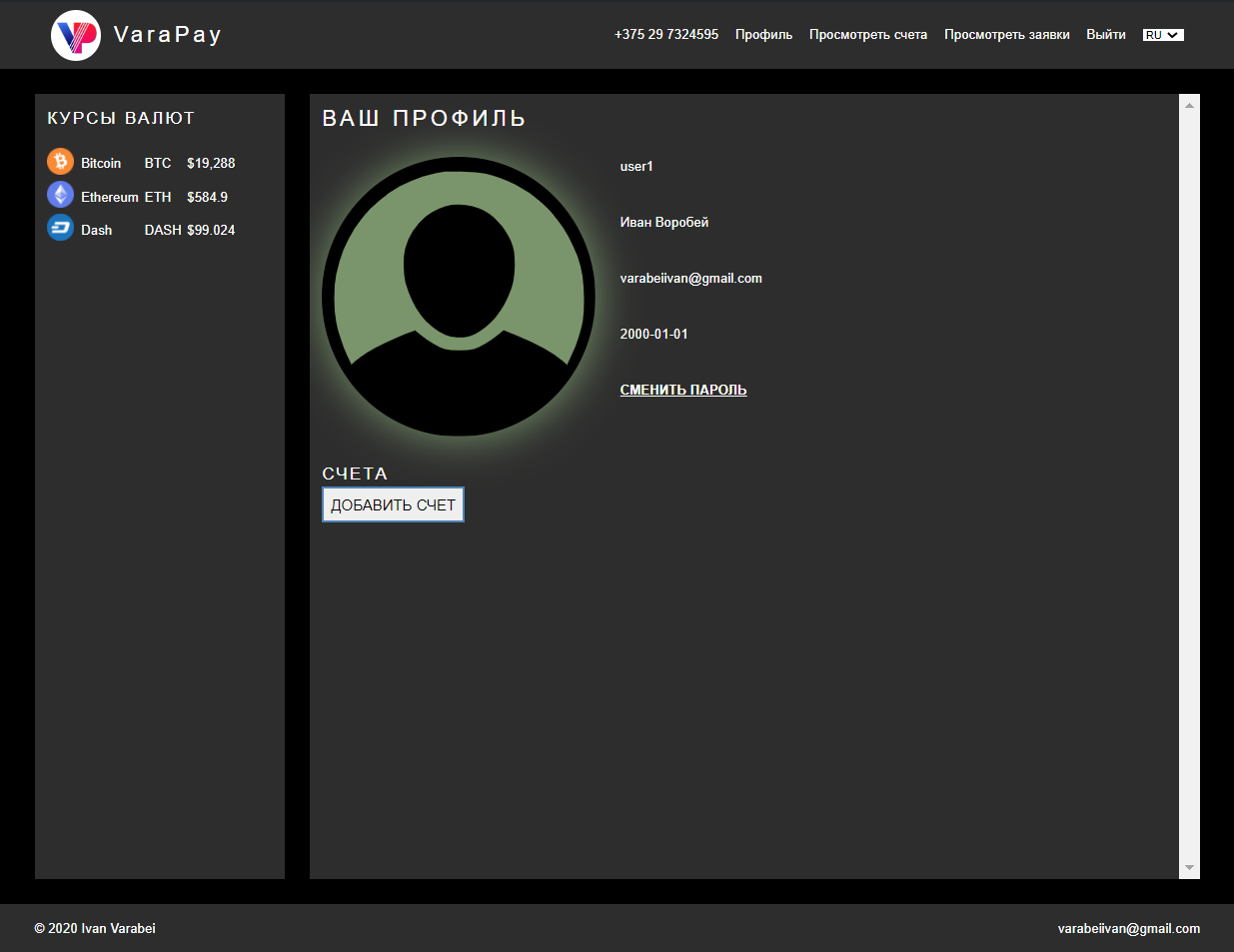


Рисунок 3.15 – Профиль администратора

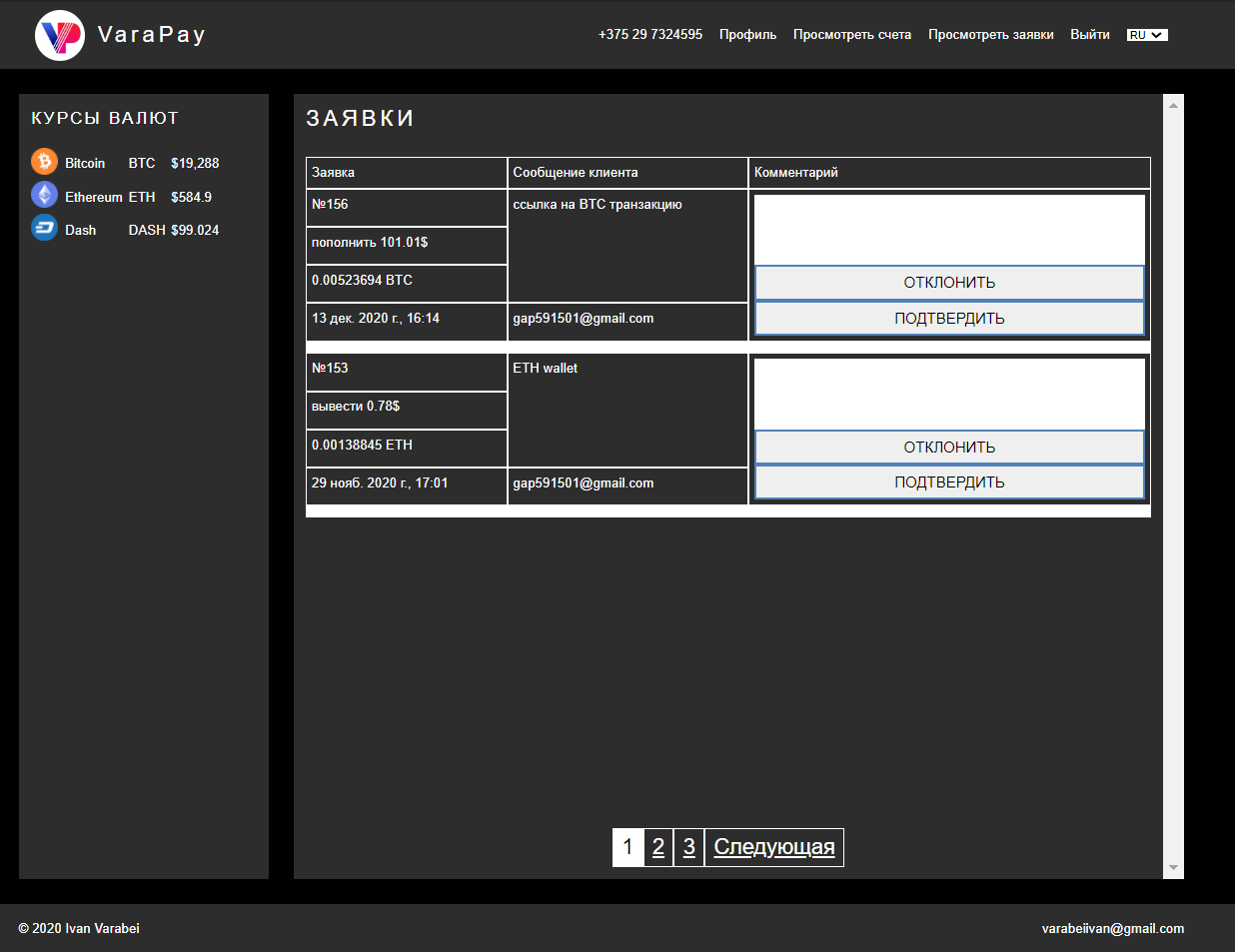


Рисунок 3.16 – Страница заявок

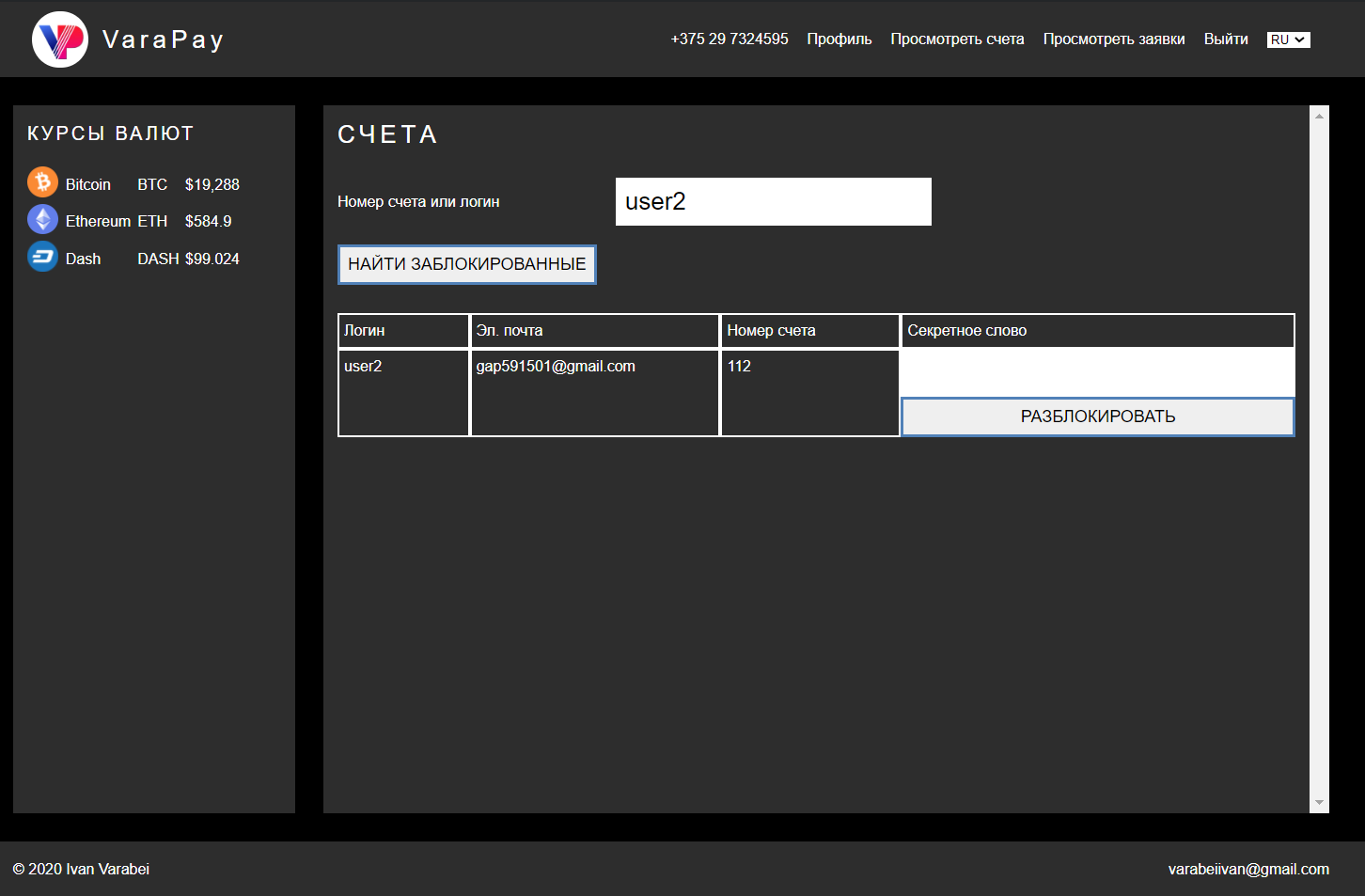


Рисунок 3.17 – Страница поиска счетов

# 4. Модульное тестирование

## 1.1 Тестирование с иcпользованием Mockito

Были проведены модульные тесты. Были использованы Mockito, TestNg. В качетсве заглушек используются и Mock и Spy. Для тестирования валидаторов использутся тесты с параметрами. Структура пактов с тестовыми классами приведена на рисунке 4.1. Результаты запуска тестов на рисунке . Всего запущено 79 тестовых методов. Результаты запуска тестов на рисун ках 4.2 – 4.4.

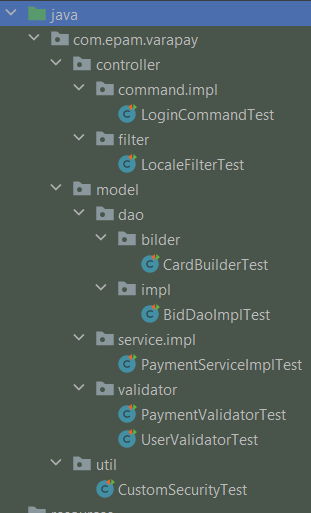


Рисунок 4.1 – Структара пакетов с тестовыми классами



Рисунок 4.2 – Результат запуска тестов часть 1

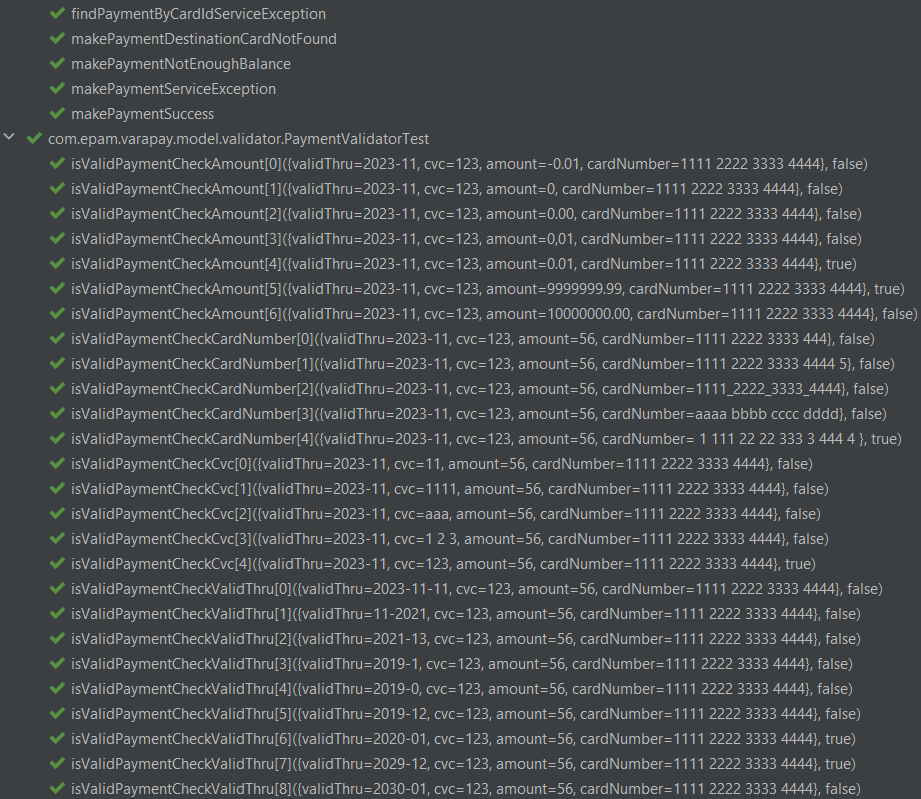


Рисунок 4.3 – Результат запуска тестов часть 2

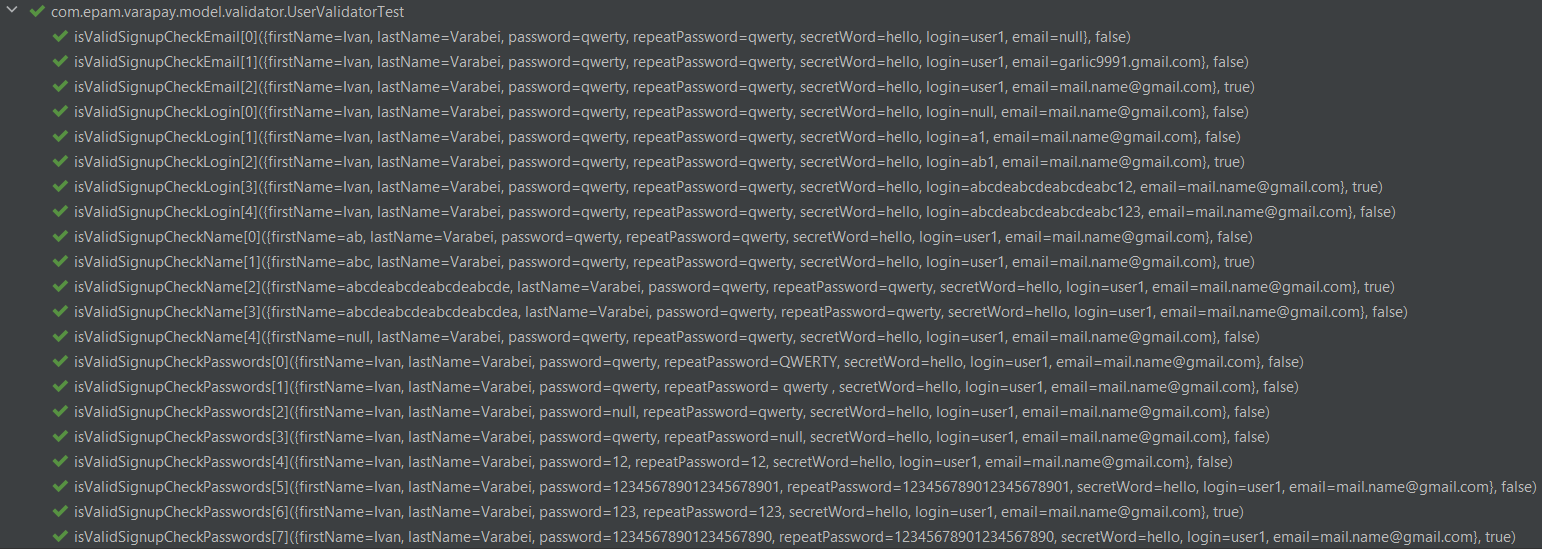


Рисунок 4.4 – Результат запуска тестов часть 3

# ЗАКЛЮЧЕНИЕ

Разработанный софт позволяет пользователям совершать анонимные транзакции без комиссии внутри эко-системы. Программа автоматизирует работу банка, тем сымым позволяет бизнесу сократить издержи. Кроме того пользователям предоставляется возможность обменивать криптовалюту.

Реализован весь функционал в соответсвии с техническим заданием к проекту. Высокий уровень покрытия тестами свидетельстует о качественном программном обеспечении. Была использывана методология разработки TDD. Модульное тестирование проводилось с использыванием заглушек. Кроме того были проведены интеграционные тесты.

Приложение демонстрирует высокий уровень безопасности. Приложение реализует защиту от DOS, XSS атак. Данные пользвователей надежно храняться в захешированном с солью виде.

Приятный дизайн и адаптивность front end части позволит привлеч еще больше клиентов и использывать приложение на разных устройствах, тем самым увеличивив оборот денежной массы проходящей через приложение. Для реализации burger menu и валидации на клиенте использыван JavaScript.

Успешные тесты и стабильная работа, продемонстрированная на испытаниях, говорит о готовности программы к релизу и запуску ее первой пробной версии.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Java. Сетевое программирование / В. Кумар [и др.]. – М.: Лори, 2007.- 400 с.
2. Фурунжиев Р.И. – Устройство Tomcat. Монография. - Мн.: Выш.шк, 2013. - 452 с.
3. Орлов, С.А. Организация ЭВМ и систем: Учебник для вузов / С.А. Орлов, Б.Я. Цилькер. - 2-е изд. - СПб: Питер, 2011. - 688с.
4. Фурунжиев Р.И., Гурский Н.Н. - Применение математических методов и ЭВМ. Программное моделирование систем. - Мн.: Выш. шк. 1991. – 250 с.
5. C.Бобровский - Java8. Учебный курс.

# Приложение А.

## Листинг программы

**DDL & DML script for whole data base**

create function generate\_random(length integer) returns text

language sql

as $$

WITH chars AS (

SELECT unnest(string\_to\_array('0 1 2 3 4 5 6 7 8 9', ' ')) AS \_char

),

charlist AS

(

SELECT \_char FROM chars ORDER BY random() LIMIT $1

)

SELECT string\_agg(\_char, '')

FROM charlist

;

$$;

alter function generate\_random(integer) owner to postgres;

CREATE OR REPLACE function generate\_random\_unique\_card\_number(length integer) returns text

language plpgsql

as $$

declare

i integer := 0;

begin

raise exception 'not implemented';

end;

$$;

CREATE OR REPLACE function generate\_random\_unique\_card\_number(length integer) returns text

language plpgsql

as $$

declare

chars text[] := '{0,1,2,3,4,5,6,7,8,9}';

result text := '';

i integer := 0;

begin

if length < 0 then

raise exception 'Given length cannot be less than 0';

end if;

for i in 1..length loop

result := result || chars[1+random()\*(array\_length(chars, 1)-1)];

end loop;

if not exists(select 1 from cards where card\_number = result) then

return result;

end if;

if exists(select 1 from cards where card\_number = result) then

return generate\_random\_unique\_card\_number(length);

end if;

end;

$$;

alter function generate\_random\_unique\_card\_number(integer) owner to postgres;

create table roles

(

role\_id serial not null

constraint roles\_pkey

primary key,

role\_name varchar(20) not null

);

alter table roles owner to postgres;

create table users

(

user\_id serial not null

constraint users\_pkey

primary key,

role\_id integer default 2 not null

constraint fk\_user\_role1

references roles,

login varchar(30) not null

constraint users\_login\_key

unique,

password varchar(64) not null,

firstname varchar(30) not null,

lastname varchar(30) not null,

email varchar(254) not null

constraint users\_email\_key

unique,

birth date,

salt varchar(64) not null,

secret\_word varchar(64) not null

);

create index users\_login\_email\_idx

on users (login, email);

alter table users owner to postgres;

create table accounts

(

account\_id serial not null

constraint accounts\_pkey

primary key,

user\_id integer not null

constraint fk\_accounts\_users1

references users

on update cascade on delete cascade,

balance integer default 0 not null,

is\_active boolean default true not null,

is\_abandoned boolean default false not null

);

create index accounts\_user\_id\_idx

on accounts (user\_id);

alter table accounts owner to postgres;

create table cards

(

card\_id serial not null

constraint cards\_pkey

primary key,

account\_id integer not null

constraint fk\_card\_account1

references accounts,

card\_number varchar(16) default generate\_random\_unique\_card\_number(16) not null

constraint cards\_cardnumber\_key

unique,

valid\_thru date default (now() + '3 years'::interval) not null,

cvc varchar(3) default generate\_random(3) not null,

is\_abandoned boolean default false not null

);

create index cards\_card\_number\_idx

on cards (card\_number);

create index cards\_account\_id\_idx

on cards (account\_id);

alter table cards owner to postgres;

create table payments

(

payment\_id serial not null

constraint payments\_pkey

primary key,

source\_card\_id integer not null

constraint fk\_payments\_cards1

references cards,

destination\_card\_id integer not null

constraint fk\_payments\_cards2

references cards,

amount integer not null,

payment\_instant timestamp default now() not null

);

create index payments\_source\_card\_id\_idx

on payments (source\_card\_id);

create index payments\_destination\_card\_id\_idx

on payments (destination\_card\_id);

alter table payments owner to postgres;

create table bid\_states

(

bid\_state\_id serial not null

constraint bid\_states\_pkey

primary key,

state varchar(128)

);

alter table bid\_states owner to postgres;

create table currencies

(

currency\_id serial not null

constraint currencies\_pkey

primary key,

currency\_name varchar(20) not null

);

alter table currencies owner to postgres;

create table bids

(

bid\_id serial not null

constraint topupbids\_pkey

primary key,

account\_id integer not null

constraint fk\_topupbids\_accounts1

references accounts,

amount integer not null,

placing\_date\_time timestamp default now() not null,

is\_top\_up boolean not null,

client\_message text not null,

bid\_state\_id integer default 1 not null

constraint bid\_state\_id\_1

references bid\_states,

admin\_comment varchar(1024) default NULL::character varying,

currency\_id integer not null

constraint fk\_currencies1

references currencies,

amount\_in\_chosen\_currency numeric not null

);

create index bids\_account\_id\_idx

on bids (account\_id);

alter table bids owner to postgres;

insert into currencies (currency\_id, currency\_name) values

(0, 'BITCOIN'),

(1, 'ETHEREUM'),

(2, 'DASH');

insert into roles (role\_name) values

('admin'),

('client');

insert into bid\_states (state) values

('in\_progress'),

('approved'),

('rejected');