

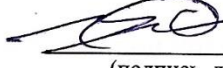
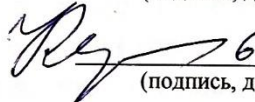
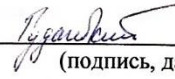
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	Кафедра
Вычислительной техники	Вычислительная техника
Направление подготовки	09.04.01 «Информатика и вычислительная техника»
Магистерская программа	Вычислительные машины, комплексы, системы и сети

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ
РАБОТА МАГИСТРА**

на тему

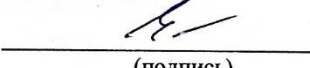
**СЕРВИС АВТОРИЗАЦИИ ПОЛЬЗОВАТЕЛЕЙ ПО
БИОМЕТРИЧЕСКИМ ДАННЫМ**

Студент	 (подпись, дата)	06.06.18 Миронов Андрей Андреевич (ФИО полностью)
Руководитель	 (подпись, дата)	6-06-18 С.А. Зинкин (фамилия, инициалы)
Нормоконтролёр	 (подпись, дата)	6.06.2018 А.В. Кучин (фамилия, инициалы)
Рецензент	 (подпись, дата)	08.06.2018 А.Р. Рудзинский (фамилия, инициалы)
	ин. сф. ООО «И» (должность, место работы)	1СБТС Development Рус

Работа допущена к защите (протокол зас. кафедры от 25.05.2018г. № 38-09-12)

Заведующий кафедрой 
(подпись) Д.В. Пашенко
(фамилия, инициалы)

Работа защищена с отметкой 
(подпись) (протокол заседания ГЭК от 14.06.18г. № 2)

Секретарь ГЭК 
(подпись) Е.И. Гурин
(фамилия, инициалы)

Пенза, 2018

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»

Заведующий кафедрой

 Д.В. Пашенко

«21» мая 2018г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
МАГИСТРА

1. Студент Миронов Андрей Андреевич гр. 16ВВм1 факультета вычислительной техники.

2. Тема работы «Сервис авторизации пользователей по биометрическим данным».

Тема утверждена приказом ПГУ № 979 а/о от " 3 " октября 2016

3. Руководитель работы д.т.н., профессор каф. ВТ ПГУ Зинкин С.А.

4. Задание на работу

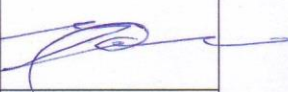
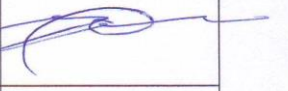
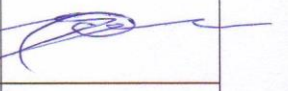
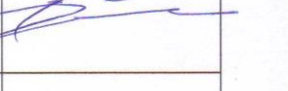
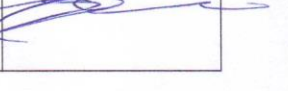
Назначение разработки: сервис авторизации пользователей по биометрическим данным.

Разработать систему в виде REST-ful веб-сервиса для авторизации пользователей на сторонних ресурсах сети Интернет на основе результатов сравнения их биометрических идентификаторов. Взаимодействие с системой должно быть доступно только пользователям, зарегистрированным в сервисе авторизации и предоставивших свои биометрические данные.

5. Перечень подлежащих разработке вопросов.

Требуется разработать базу данных, которая будет хранить данные для авторизации зарегистрированных пользователей, список устройств, с которых осуществлялись запросы на авторизацию и ссылку на контейнер с загруженными биометрическими данными. Со стороны клиента на сервис должны отправляться HTTP-запросы, содержащие биометрические данные пользователя, сервис должен принимать их, сопоставлять и возвращать отправителю ответ с результатами.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	21.05.18	
2.	Выбор программных средств для реализации	5	22.05.18	
3.	Проектирование базы данных	10	24.05.18	
4.	Разработка компонента авторизации в сервисе	10	26.05.18	
5.	Разработка REST-ful веб-сервиса	30	01.06.18	
6.	Отладка приложения на ЭВМ	10	02.06.18	
7.	Оформление выпускной работы	30	03.06.18	

Дата выдачи задания "21" Май 2018 г.

Руководитель магистерской работы  21.05.18 Зинкин С.А.
(подпись, дата)

Задание к исполнению принял студент  21.05.18 Миронов А.А..
(подпись, дата)

Работу к защите допустить
Декан факультета  09.06.2018 Фионова Л.Р.
(подпись, дата)

Заведующему кафедрой
"Вычислительная техника"
Д. В. Пащенко
студента 16ВВм1
Миронова Андрея Андреевича

ЗАЯВЛЕНИЕ

Прошу утвердить в качестве руководителя ВКР профессора кафедры
«Вычислительная техника» Зинкина Сергея Александровича.

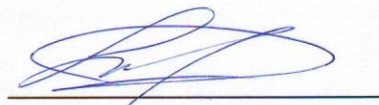
Руководитель – тема ВКР: «Сервис авторизации пользователей по
биометрическим данным».

Дата: 14_10_2016

Руководитель

A blue ink signature, likely of the supervisor, written over a horizontal line.

Студент

A blue ink signature, likely of the student, written over a horizontal line.

Ректору Пензенского
государственного университета
А. Д. Гулякову
студента 2 курса ФВТ
очной формы обучения
Миронова Андрея Андреевича

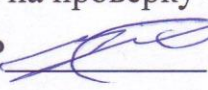
ЗАЯВЛЕНИЕ

Я, Миронов Андрей Андреевич, магистрант 2 курса очной формы обучения направления подготовки 09.04.01 Информатика и вычислительная техника, магистерская программа «Вычислительные машины, комплексы, системы и сети», прошу провести проверку с использованием системы «Антиплагиат» выпускной квалификационной работы магистра на тему «Сервис авторизации пользователей по биометрическим данным», выполненной мной самостоятельно, на содержание элементов плагиата.

Все прямые заимствования из печатных и электронных источников, а также из защищенных ранее выпускных квалификационных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

Я ознакомлен с действующим Стандартом университета «Выпускная квалификационная работа обучающихся по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры», согласно которому обнаружение плагиата является основанием для недопуска ВКР к защите и отчисления из университета.

ВКР представлена на проверку «4» 06 2018 г.

Руководитель ВКР  Зинкин С.А.

Протокол

проверки на оригинальность в системе «Антиплагиат.ВУЗ» выпускной квалификационной работы на тему «Сервис авторизации пользователей по биометрическим данным» магистранта 2 курса Миронова А.А. направления подготовки 09.04.01 Информатика и вычислительная техника, магистерская программа «Вычислительные машины, комплексы, системы и сети».

Руководитель ВКР д.т.н., профессор каф. ВТ Зинкин С.А.



Отчет о проверке на заимствования №1

Автор: Миронов Андрей Андреевич pn294@hotmail.com / ID: 3296484

Проверяющий: Миронов Андрей Андреевич (pn294@hotmail.com) / ID: 3296484

Отчет предоставлен сервисом «Антиплагиат» <http://www.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 9
Начало загрузки: 04.06.2018 21:21:54
Длительность загрузки: 00:00:02
Имя исходного файла: ВКР_Миронов
Размер текста: 1509 КБ
Символов в тексте: 164201
Слов в тексте: 18506
Число предложений: 2976

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 04.06.2018 21:21:57
Длительность проверки: 00:00:04
Комментарий: не указано
Модуль поиска:

ЗАИМСТВОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
8,68% ■	0%	91,32%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использованием корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты, общепотребительные выражения, фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которому шла проверка, по отношению к общему объему документа.
Заимствования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определения корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Доля в тексте	Источник	Ссылка	Актуален на	Модуль поиска	Блоков в отчете	Блоков в тексте
[01]	2,44%	3,32%	Сбытовая деятельность на рынке автомобильных зап...	http://revolution.arbit.ru	25 Авг 2017	Модуль поиска Интернет	74	87
[02]	1,29%	1,76%	Spring Data JPA Example Examples Java Code Geeks - 2017	https://examples.java.codegeeks.com/	15 Июл 2017	Модуль поиска Интернет	30	42
[03]	0,66%	1,57%	Spring Security - Hibernate. Подкажите пожалуйста, в ч...	http://cyberforum.ru	22 Ноя 2016	Модуль поиска Интернет	27	70

Еще источников: 17
Еще заимствований: 4,06%

« 4 » 06 2018 г.

С.А. Зинкин

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет вычислительной техники

Кафедра «Вычислительная техника»

ОТЗЫВ

на выпускную квалификационную работу магистра

Миронова Андрея Андреевича

Тема работы: Сервис авторизации пользователей по биометрическим данным.

Цель работы: Разработка универсального сервиса, позволяющего пользователям авторизовывать свои действия на сторонних сервисах сети интернет.

Направление: 09.04.01 «Информатика и вычислительная техника»

Профиль: «Вычислительные машины, комплексы, системы и сети»

Выпускная квалификационная работа магистра, выполненная магистрантом Миронов А.А., относится к области разработки веб-сервисов.

Благодаря подробному анализу предметной области были выделены основные способы, методы и технологии для реализации поставленной цели. Были разработаны компонент авторизации пользователей в системе, компонент загрузки и изменения биометрических данных пользователей, компонент авторизации пользователей на сторонних сервисах сети Интернет, детально спроектирована и реализована база данных для них. Разработанные компоненты были объединены в единую систему.

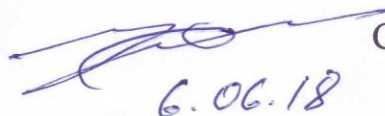
Поставленные в работе исследовательские и практические задачи магистрантом Мироновым А.А. выполнены в полном объеме. Пояснительная записка оформлена в соответствии с действующими стандартами.

С поставленными задачами Миронов А.А. справился успешно. Степень оригинальности работы 91,32%.

Считаю, что выпускная квалификационная работа магистра выполнена на профессиональном уровне и заслуживает оценки «отлично», а магистрант Миронов А.А. заслуживает присвоения ему квалификации магистра по направлению подготовки 09.04.01

Руководитель выпускной квалификационной работы магистра

Д.т.н. профессор каф. ВТ



С. А. Зинкин

6.06.18

РЕЦЕНЗИЯ

на выпускную квалификационную работу магистра студента группы 16ВВм1
Пензенского государственного университета

Миронова Андрея Андреевича

на тему: *«Сервис авторизации пользователей по биометрическим данным»*

Выпускная квалификационная работа магистра Миронова А.А. посвящена решению актуальной задачи по авторизации и контролю идентификации пользователей.

Разработанная система позволяет идентифицировать и авторизовывать действия пользователя сервиса на сторонних ресурсах сети Интернет. Система предоставляет ресурсам возможность идентифицировать пользователя по биометрическим данным при помощи двухэтапной авторизации.

Предложенная студентом двухэтапная авторизация по биометрическим данным позволяет увеличить точность идентификации пользователя, так как на каждом этапе используются разные биометрические идентификаторы.

Достоинствами разработки можно считать универсальность разработанной системы, возможность использования ее на различных мобильных и стационарных платформах, а также в различных сферах жизнедеятельности.

Незначительным недостатком работы является краткий анализ предметной области.

Результаты проекта могут быть с успехом использованы для дальнейшего развития идеи создания сервиса авторизации пользователей по биометрическим данным.

Считаю, что выпускная квалификационная работа магистра Миронова А.А. может быть допущена к защите, заслуживает оценки **«отлично»**, а ее автор присвоения ему степени магистра по направлению 09.04.01 – «Информатика и вычислительная техника».

Генеральный директор
ООО «ЮБПС Девелопмент Рус»



А.Р. Рудаковский
08.06.2018 г.

Реферат

Магистерская работа содержит 68 листов, 7 рисунков, 28 источников, 3 приложения.

ВЕБ-СЕРВИС, АВТОРИЗАЦИЯ, БАЗА ДАННЫХ, БИОМЕТРИЯ, SPRING FRAMEWORK, КЛИЕНТ-СЕРВЕР, HTTP, REST-FUL, JAVA

Цель работы – разработать сервис авторизации пользователей по биометрическим данным.

Объектом разработки является сервис авторизации пользователей на сторонних ресурсах сети Интернет. Сервис идентифицирует пользователей по биометрическим данным.

В результате проведенной работы создан веб-сервис, полностью удовлетворяющий поставленной задаче. Была спроектирована и смоделирована база данных, предназначенная для хранения информации для последующей авторизации пользователей. Был разработан алгоритм двухэтапной авторизации пользователя по биометрическим данным. Был создан веб-сервис, выполняющий авторизацию пользователей по биометрическим данным на сторонних ресурсах сети Интернет.

Данный проект является универсальным и подразумевает одновременное использование на различных ресурсах сети Интернет.

СОДЕРЖАНИЕ

Введение.....	6
1 Постановка задачи.....	9
2 Обзор технологий.....	10
2.1 Язык программирования.....	10
2.2 Среда разработки программного обеспечения.....	11
2.3 Spring Framework.....	13
2.3.1 Модуль Spring MVC.....	14
2.3.2 Spring Data.....	15
2.4 База данных.....	17
2.5 Пул соединений с базой данных.....	17
2.6 Контейнер сервлетов.....	18
2.7 Система контроля версий.....	19
2.8 Средство автоматизированной сборки проекта.....	21
3 Анализ предметной области.....	24
4 Разработка базы данных.....	27
4.1 Проектирование базы данных.....	27
4.2 Проектирование таблиц базы данных.....	28
4.2.1 Проектирование таблицы пользователей.....	29
4.2.2 Проектирование таблицы с данными авторизации в сервисе.....	29
4.2.3 Проектирование таблиц с ссылками на биометрические контейнеры пользователей.....	30
4.2.4 Проектирование таблицы с авторизованными устройствами пользователей.....	30
4.2.5 Проектирование таблицы токенов.....	30
4.3 Проектирование базы данных.....	31
4.3.1 Абстрактный класс AuthUser.....	32
4.3.2 Класс PersonInfo.....	34
4.3.3 Класс SimpleAuthUser.....	36
4.3.4 Класс Enroll.....	37
4.3.5 Класс EnrollFace.....	39

4.3.6 Класс FaceFeature	40
4.3.7 Класс EnrollVoice	41
4.3.8 Класс AuthorizedDevice.....	42
4.3.9 Класс Client	43
4.3.10 Класс UserHash	44
5 Разработка приложения	46
5.1 Конфигурации приложения	47
5.2 Конфигурация приложения для работы с сервисом Google reCAPTCHA	49
5.3 Создание классов сущностей.....	50
5.4 Создание репозиториев	52
5.5 Создание контроллеров.....	55
5.6 Создание компонента авторизации пользователя на сторонних ресурсах сети Интернет.....	58
6 Руководство	60
6.1 Руководство по сборке проекта.....	60
6.1.1 Сборка проекта при помощи интегрированной среды разработки..	60
6.1.2 Сборка проекта при помощи Maven.....	60
6.2 Руководство по размещению проекта на сервере	61
6.3 Руководство по регистрации пользователя в сервисе авторизации	61
6.4 Руководство по работе с сервисом авторизации пользователей по биометрическим данным через сторонние сервисы	63
Заключение	67
Список используемых источников.....	68
Приложение А Листинги (обязательное).....	70
Приложение А.1 Листинги классов сущностей.....	70
Приложение А.1.1 Файл «AuthUser.java»	70
Приложение А.1.2 Файл «PersonInfo.java»	73
Приложение А.1.3 Файл «SimpleAuthUser.java».....	75
Приложение А.1.4 Файл «Enroll.java».....	77
Приложение А.1.5 Файл «EnrollFace.java»	80
Приложение А.1.6 Файл «EnrollFinger.java»	80
Приложение А.1.7 Файл «EnrollVoice.java».....	81

Приложение А.1.8 Файл «FaceFeature.java»	81
Приложение А.1.9 Файл «AuthorizedDevice.java»	83
Приложение А.1.10 Файл «Client.java»	84
Приложение А.1.11 Файл «UserHash.java»	86
Приложение А.2 Листинги контроллеров	89
Приложение А.2.1 Файл «AuthController.java»	89
Приложение А.2.2 Файл «GeneralizeController.java»	98
Приложение А.2.3 Файл «HomeController.java»	104
Приложение А.2.4 Файл «LoginController.java»	104
Приложение А.2.5 Файл «MyCabinetController.java»	105
Приложение А.2.6 Файл «PersonificationController.java»	106
Приложение А.2.7 Файл «RegistrationController.java»	109
Приложение А.2.8 Файл «ShopController.java»	118
Приложение А.2.9 Файл «WebAuthController.java»	120
Приложение А.3 Листинги репозиториев	125
Приложение А.3.1 Файл «AuthorizedDeviceRepository.java»	125
Приложение А.3.2 Файл «AuthUserRepository.java»	126
Приложение А.3.3 Файл «ClientRepository.java»	126
Приложение А.3.4 Файл «EnrollFaceRepository.java»	127
Приложение А.3.5 Файл «EnrollFingerRepository.java»	128
Приложение А.3.6 Файл «EnrollRepository.java»	128
Приложение А.3.7 Файл «EnrollVoiceRepository.java»	129
Приложение А.3.8 Файл «SimpleAuthUserRepository.java»	130
Приложение А.3.9 Файл «UserHashRepository.java»	130
Приложение Б Диаграммы UML (рекомендуемое)	132
Приложение Б.1 Диаграмма развертывания	132
Приложение Б.2 Диаграмма сущность-связь	133
Приложение В Презентация (рекомендуемое)	134

ВВЕДЕНИЕ

В настоящее время информационные технологии развиваются достаточно стремительно. Те вещи, которые ранее казались чем-то из разряда фантастики, сегодня кажутся вполне обыденными. Одними из предназначений информационных технологий являются упрощение, ускорение выполнения, а также минимизация расходов ресурсов на решение как организационных, так и повседневных задач.

Говоря о таких субъектах как крупная организация или отдельно взятый человек, необходимо обратить внимание на то, что все они используют информационные технологии для хранения, обработки и распространения различных по объему и содержанию данных. На примере обычного человека это могут быть поиск и сохранение различной информации, находящейся в открытом доступе, а также публикация информации в Интернет.

Однако в случае хранения, обработки и распространения конфиденциальной информации, необходимы различные системы для контроля доступа пользователей. Это необходимо для того, чтобы никто иной, кроме определенного круга лиц, не смог получить доступ к конфиденциальной информации и использовать ее во вред источнику информации.

На данный момент, самым, пожалуй, популярным способом авторизации пользователя, для предоставления ему прав пользования системой, является авторизация по логину и паролю. Идея такой авторизации сводится к тому, что никто иной, кроме владельца аккаунта системы не сможет получить доступ к системе и действовать от лица этого пользователя, так как логин и пароль от такого аккаунта знает только его владелец или, за редким исключением, администратор системы, зарегистрировавший данный аккаунт. Однако данные входа могут быть забыты пользователем, в этом случае помогает простое восстановление пароля. Но если же данные для входа были

скомпрометированы и постороннее лицо получило доступ к конфиденциальной информации, то, помимо смены пароля от скомпрометированного аккаунта, необходимо провести ряд мероприятий по оповещению всех участников обмена информации, а также по уменьшению возможного ущерба, который может быть вызван в результате получения доступа к конфиденциальной информации посторонним лицом.

Большинство алгоритмов шифрования паролей пользователей не могут дать абсолютной гарантии защиты от простого подбора паролей, а также от возможного возникновения коллизий.

Помимо достаточно распространенного метода авторизации по логину и паролю, существует еще также авторизации пользователя по его уникальным физиологическим или поведенческим чертам, или же просто аутентификация по биометрии пользователя.

Как известно, каждый человек обладает практически уникальным набором физиологических и поведенческих черт, что позволяет системам биометрической аутентификации достаточно точно проверять соответствие субъекта и того, за кого он себя выдает.

Так же существуют комбинированные системы биометрической аутентификации, которые проверяют субъект по двум и более физиологическим и/или поведенческим параметрам, что дает более точную оценку и повышает безопасность системы в целом. Подобные виды аутентификации пользователя позволяют людям быть спокойным за конфиденциальность своих данных, а также избавляют от необходимости запоминания паролей от различных систем.

Как было сказано выше, информационные технологии стремительно развиваются. Прогресс в развитии биометрических систем аутентификации тоже не стоит на месте: благодаря развитию техники, оптимизации и разработке алгоритмов идентификации, системы стали работать гораздо точнее и быстрее, по сравнению с тем, как работали несколько лет назад.

Подобные технологии, ранее недоступные обычным пользователям, со временем становятся более распространенными. В настоящее время современные мобильные устройства обладают такими функциями как идентификация пользователя по лицу, сетчатке глаза, отпечаткам пальцев и даже по образцу голоса.

Целью данной работы является разработка системы авторизации пользователей по биометрическим данным. Разрабатываемая система позволит идентифицировать пользователя по некоторым физиологическим и поведенческим параметрам, таким как:

- геометрия лица;
- снимок отпечатка пальца;
- образец голоса пользователя.

Так же эта система будет комбинирована. Это означает, что для полной идентификации пользователя необходимо предоставить как минимум два вышеописанных параметра. В сравнении с системами авторизации пользователей, где методом авторизации выступает логин и пароль, для разрабатываемой системы можно сказать, что в качестве логина пользователя будет выступать геометрия его лица, а в качестве пароля может выступать образец голоса пользователя или снимок отпечатка пальца.

Плюсом разрабатываемой системы может служить ее универсальность. Данную систему можно использовать практически для любой задачи, где требуется аутентификации пользователя, этими задачами могут быть:

- финансовые операции;
- получения доступа к ресурсам сети интернет;
- разблокирование устройства, например, персонального компьютера или смартфона и т.д.

1 ПОСТАНОВКА ЗАДАЧИ

Целью работы является разработка сервиса авторизации пользователей по биометрическим данным. Для ее достижения необходимо выполнить следующий ряд подзадач:

- выбрать наиболее подходящие средства и методы для решения поставленной задачи;
- спроектировать базу данных для хранения информации, необходимой для авторизации пользователей;
- реализовать структуру разработанной базы данных программным способом;
- разработать модули регистрации и авторизации в сервисе;
- разработать модули загрузки, сохранения и изменения биометрических данных пользователей;
- разработать модуль авторизации действий пользователей на сторонних ресурсах сети Интернет;
- обеспечить корректность выполняемых запросов;
- протестировать систему.

Выполнение перечисленных подзадач поэтапно и в указанной последовательности позволит максимально точно достигнуть поставленной цели в кратчайшие сроки.

2 ОБЗОР ТЕХНОЛОГИЙ

2.1 Язык программирования

В качестве языка программирования для данной работы был выбран язык Java версии 8. Впервые он был представлен миру в 1995г. компанией Sun Microsystems [1]. Благодаря своей кроссплатформенности, объектной ориентированностью и простотой изучения, данный язык стал одним из самых популярных языков программирования среди разработчиков программного обеспечения. В настоящее время он используется в огромном количестве различных устройств, таких как:

- персональные компьютеры;
- смартфоны
- автомобили
- часы;
- телевизоры;
- и прочая бытовая техника.

Так же следует сказать о том, в каких приложениях применим данный язык. В первую очередь Java используется для серверных приложений в крупных корпорациях таких как: банки, страховые компании, розничные сети и т.д. Помимо этого этот язык нашел широкое применение в веб-приложениях электронной коммерции. Менее популярен данный язык при разработке приложений для рабочего стола, программных средств и средств разработки, однако, зачастую такие приложения распространяются бесплатно, а разработчики выкладывают исходный код приложения в открытый доступ.

После загрузки Java пользователи получают Java Runtime Environment, которая состоит из виртуальной машины Java, известной как JVM, базовые классы платформы и дополнительные библиотеки. Однако, для разработки программного обеспечения на языке Java необходим так называемый Java

Development Kit. JDK – это полнофункциональный комплект для разработки программного обеспечения, который состоит из следующих частей:

- Java Runtime Environment;
- компиляторы;
- различные инструменты для создания, отладки и компиляции программ (например, JavaDoc и Java Debugger). [2]

Язык постоянно развивается, увеличивая быстродействие и добавляя различные нововведения, такие как: аннотации, лямбда-выражения, параллельные сортировки и т.д.

Огромное сообщество разработчиков часто создает, развивает и бесплатно распространяет различные библиотеки, которые позволяют сильно облегчить разработку приложений. Так же стоит заметить, что классы и методы в Java грамотно документированы, что позволяет на первых этапах относительно быстро разобраться как программисту с большим стажем разработки программного обеспечения на других языках программирования, так и начинающему разработчику.

2.2 Среда разработки программного обеспечения

Интегрированная среда разработки – это комплекс программных средств, который используется программистами для разработки программного обеспечения. Интегрированные среды разработки могут иметь различный внешний вид и иметь самый разный набор средств облегчающий разработку, отладку, тестирование и сборку программного обеспечения, однако обязательно включают в себя следующие компоненты:

- текстовый редактор;
- компилятор и/или интерпретатор;
- средства автоматизации сборки;
- отладчик. [3]

Пожалуй, самыми известными и распространенными средами интегрированной разработки для языка Java являются IntelliJ IDEA, Net Beans и Eclipse. Для данной работы была выбрана среда разработки IntelliJ IDEA от компании JetBrains. Она является коммерческой и распространяется в следующих изданиях:

- бесплатная IntelliJ IDEA Community Edition, которая помимо стандартных для интегрированных сред разработки средств, включает в себя поддержку большого числа языков, некоторых фреймворков, систем контроля версий, инструментов сборки и разворачивания и т.д.
- коммерческая IntelliJ IDEA Ultimate Edition, которая расширяет бесплатное издание, добавляя ко всему выше перечисленному огромное количество различных средств, ускоряющих и упрощающих процесс разработки программного обеспечения. [4]

Для данной работы была выбрана коммерческая версия IntelliJ IDEA, которая содержит весь необходимый инструментарий, требуемый для выполнения поставленной задачи:

- поддержка языка Java;
- поддержка фреймворка Spring;
- поддержка фреймворка Thymeleaf;
- поддержка фреймворка Freemarker;
- инструменты для работы с базами данных;
- поддержка системы контроля версий SVN;
- встроенный контейнер сервлетов Tomcat;
- система автоматизированной сборки Maven.

Наличие данных инструментов подразумевает отсутствие необходимости переключения между различными программами для разработки, отладки и сборки приложения, что, несомненно удобно и

позволяет решать задачи непосредственно по мере необходимости без затрат времени на поиск, запуск и настройку необходимых программ.

2.3 Spring Framework

Spring представляет собой универсальный фреймворк с открытым исходным кодом. Самая первая версия была написана и опубликована в 2002 году Родом Джонсоном. В 2003 году данный фреймворк был выпущен под лицензией Apache 2.0. С тех пор Spring постоянно развивается и к настоящему времени завоевал огромную популярность среди разработчиков на языке Java.

Spring состоит из большого числа модулей, которые можно рассматривать как отдельные фреймворки – они могут использоваться отдельно друг от друга, но наиболее большая функциональность достигается при их совместном использовании [5]. Сам Spring Framework, известный также как Spring Core является ядром платформы и предоставляет базовые средства для создания приложений [6]. Для корректной работы данного фреймворка, его необходимо сконфигурировать. В настоящее время это можно сделать несколькими способами:

- Конфигурация модулей при помощи XML файлов. При использовании данного метода получается достаточно большой объем громоздких, нечитаемых XML файлов, разобраться в которых порой бывает очень тяжело.
- Конфигурация при помощи аннотаций. Данный метод позволяет настроить фреймворк непосредственно в классах приложения при помощи аннотаций. Данный способ относительно быстр и удобен. Однако, он был доступен не всегда: аннотации в языке Java появились с версии 5.0, но разработчики быстро адаптировали фреймворк и в свет вышла версия Spring Framework 2.5 [7].

- Конфигурация при помощи классов конфигурации. Программное создание бинов. Реализует принцип соглашения по конфигурации [8].
- Конфигурация при помощи внешних файлов с настройками. По сути является аналогом конфигурации при помощи XML файлов.

2.3.1 Модуль Spring MVC

Концепция Model View Controller была описана Трюгве Реенскаутом в 1978г. и представляет собой схему разделения данных приложения, пользовательского интерфейса и управляющей логики на отдельные компоненты:

- Модель. Предоставляет данные и реагирует на команды контроллера, изменяя свое состояние.
- Представление. Отображает данные модели пользователю, реагируя на изменение модели.
- Контроллер. Интерпретирует действия пользователя, оповещая модель о необходимости изменений [9].

В листинге 1 представлен пример создания контроллера.

Листинг 1. Пример создания контроллера.

```
@Controller
public class MyCabinetController
{
    ...
    @RequestMapping(value = Path.MY_CABINET, method =
RequestMethod.GET)
    public String myCabinetPage(Model model)
    {
        AuthUser user = currentUser.getUser();
        model.addAttribute("authUser",user);

        model.addAttribute("androidAppLink",mobileProperty.getAndroidAppLink());
    }
}
```

```

        return Template.MY_CABINET;
    }
    ...
}

```

Как видно из листинга 1, класс, предназначенный для обработки запросов помечается аннотацией `@Controller`. Для того, чтобы связать URL запроса и метод контроллера, необходимо последний пометить аннотацией `@RequestMapping`, в параметрах которой указать строкой с URL значение параметра `value` (или `path`), а также указать тип запроса, на который должен реагировать этот метод. В некоторых случаях фреймворк сам понимает какие данные будет возвращать программист. На примере листинга выше, возвращаемым типом указан `String`, это значит, что после выполнения метода обратно будет передано название шаблона веб-страницы, который Spring подгрузит, обработает, сформирует конечную страницу и отдаст ее пользователю.

2.3.2 Spring Data

Модуль Spring Data работает с системами управления реляционными базами данных, использует JDBC и ORM средства [5]. Данное средство помогает осуществлять обмен данными между приложением и базой данных. Spring Data позволяет создавать интерфейсы репозитория. Для этого необходимо создать интерфейс, унаследованный от одного из следующих интерфейсов: `Repository`, `CrudRepository`, `PagingAndSortingRepository`, указав при наследовании два типа данных: первый тип указывает класс-сущность, с которым будет осуществляться взаимодействие, второй указывает на тип идентификатора сущности. Далее работа с созданным репозиторием будет происходить непосредственно через интерфейс, Spring сам создает его имплементацию, основываясь на названии методов, а также аннотаций

@Query и подобных, которые в качестве параметра принимают HQL запрос. В листинге 2 представлен пример репозитория с методами без аннотаций.

Листинг 2. Пример репозитория с методами без аннотаций.

```
public interface EnrollFingerRepository extends  
CrudRepository<EnrollFinger, Long> {  
    List<EnrollFinger> findAllByClient(Client client);  
}
```

Из приведенного выше примера видно, что был создан интерфейс, унаследованный от интерфейса CrudRepository. Данный интерфейс содержит только один метод, который называется findAllByClient. Фреймворк учитывает название метода, тип возвращаемых данных и передаваемые параметры и самостоятельно составляет запрос в базу данных. В листинге 3 представлен пример репозитория с методами, содержащими аннотацию @Query.

Листинг 3. Пример репозитория с методами, содержащими аннотацию @Query.

```
public interface ClientRepository extends  
CrudRepository<Client, Long>  
{  
    @Query("select c from Client c where c.email = :login")  
    Client findByLogin(@Param("login") String login);  
}
```

Из примера выше виден интерфейс ClientRepository, унаследованный от интерфейса CrudRepository. Данный интерфейс содержит только один метод и он аннотирован при помощи аннотации @Query. Внутри этой аннотации программист сам указывает какой запрос на языке HQL необходимо отправлять в базу данных при обращении к этому методу.

2.4 База данных

База данных – это организованная структура, которая предназначена для хранения, изменения и обработки взаимосвязанной информации [10].

В настоящее время существует достаточно большое количество различных баз данных. Все они имеют свой ряд преимуществ и недостатков. Однако среди всех было решено выделить PostgreSQL. Следует отметить, что указанная база данных является одной из наиболее развитых из открытых СУБД. У PostgreSQL множество различных возможностей, например:

- поддержка сложных структур;
- широкий спектр встроенных и определяемых пользователем, типов данных;
- и т.д.;

Говоря о данных, хотелось бы выделить следующие достоинства, связанные с размерами данных в PostgreSQL:

- максимальный размер базы данных неограничен;
- максимальный размер одной таблицы составляет 32 TB;
- максимальный размер одной строки составляет 1.6 GB;
- максимальный размер одного поля составляет 1 GB;
- максимальное количество строк в таблице неограниченно;
- максимальное количество столбцов в таблице варьируется от 250 до 1600 в зависимости от типов столбца;
- максимальное количество индексов неограниченно. [11]

PostgreSQL поддерживается на всех современных Unix системах, а также Windows и Mac OS X. [12]

2.5 Пул соединений с базой данных

В качестве пула соединений с базой данных был выбран Hikari Connection Pool.

Приложение может обмениваться с базой данных информацией, используя соединение. Если создавать его при каждом обращении к базе данных, то получится проигрыш по времени: на выполнение транзакции может уйти до нескольких миллисекунд, однако на создание соединения может уйти до нескольких секунд.

Пул соединений с базой данных это способ решения такой проблемы. Он подразумевает, что в распоряжении имеется некоторый набор (пул) соединений к базе данных и когда пользователь запрашивает доступ, ему, из этого пула, выдается уже открытое соединение. Если все открытые соединения заняты, то создается новое. После освобождения пользователем соединения, оно становится доступным для других пользователей. Те соединения, которые долго не используются, закрываются [13].

Для сервиса авторизации скорость играет ключевую роль, соответственно к выбору пула соединений с базой данных надо подойти со всей ответственностью.

Для данной работы был выбран пул соединений HikariCP. Hikari Connection Pool один из самых быстрых пулов соединений на сегодняшний день. Это яркий пример того, что всегда стоит брать под сомнение эффективность некоторых решений, даже если их используют миллионы людей. Этот пул соединений прекрасный пример того, как микро оптимизации, которые по отдельности не смогут дать прироста быстродействия, в совокупность позволяют создать очень эффективный и быстрый инструмент [14].

2.6 Контейнер сервлетов

Сервлет является интерфейсом Java, реализация которого позволяет расширить функциональные возможности сервера. Сервлет взаимодействует с клиентами по принципу запрос-ответ [15].

Контейнер сервлетов – программа, представляющая собой сервер, который занимается системной поддержкой сервлетов и обеспечивает их жизненный цикл в соответствии с правилами, определенными в спецификациях. Он может работать как самостоятельный веб-сервер, быть поставщиком веб-страниц для другого веб-сервера или быть интегрированным в сервер приложений [16].

Для данной выпускной квалификационной работы был выбран контейнер сервлетов Tomcat, разрабатываемый Apache Software Foundation. Он представляет из себя контейнер сервлетов с открытым исходным кодом, реализующий спецификацию сервлетов, JavaServer Pages, JavaServer Faces. Apache Tomcat состоит из следующих компонентов:

- Catalina – непосредственно сам контейнер сервлетов, реализующий спецификацию сервлетов Servlet API.
- Coyote – компонент сетка HTTP. Он прослушивает все входящие подключения на TCP порту, отправляет запросы в механизм Tomcat для обработки и отправляет ответ назад запрашивающему клиенту.
- Jasper – механизм JSP Tomcat’а. Он анализирует JSP файлы для их компиляции в Java код. Во время выполнения, Jasper, может автоматически отслеживать изменения в JSP-файле и перекомпилировать его [17].

Данный контейнер сервлетов является легковесным, простым для использования, а также кроссплатформенным.

2.7 Система контроля версий

Система контроля версий – это программное обеспечение для облегчения работы с изменяющейся информацией. Она позволяет хранить несколько версий одного и того же документа и, при необходимости,

возвращаться к более ранним версиям, а также определять, кто и когда делал вносил какие-либо изменения в документ.

Данные системы наиболее широко используются при разработке программного обеспечения и используются для хранения исходных кодов разрабатываемой программы.

Основными возможностями систем контроля версий являются:

- создание разных вариантов одного и того же документа, так называемые ветки, с общей историей изменения до и с разными после точки ветвления;
- возможность узнать кто и когда изменил какой-либо документ;
- ведение журнала изменений, в который пользователи системы могут записывать пояснения по каждому изменению документа;
- контроль прав доступа пользователей, который разрешает или запрещает чтение или изменение данных для каждого отдельного взятого пользователя или групп пользователей в целом [18].

Для данной работы была выбрана система контроля версий Subversion, также известная как SVN. Apache Subversion является свободной централизованной системой управления версиями. Под централизованной понимается то, что данные хранятся в едином хранилище, которое может располагаться как на локальном диске, так и на сетевом сервере.

При сохранении данных используется так называемая дельта-компрессия, т.е. при записи данных система находит отличия новой версии от предыдущей и записывает только их, что позволяет избежать дублирования данных.

Однако, следует не забывать про недостатки данной системы контроля версий:

- Проблемы при переименовании файлов. Subversion не всегда может корректно обработать операции по переименованию файлов, особенно если одновременно с переименованием

выполняется операция изменения содержимого файла. Также проблемы могут возникнуть, если файл, который был переименован в локальной копии, другой пользователь изменил в хранилище.

- Слабая поддержка слияния ветвей. До версии 1.5 все операции слияния ветвей пользователям приходилось отслеживать вручную. В настоящее время Subversion достаточно хорошо поддерживает типовые сценарии слияния, однако в более сложных случаях возможны проблемы. Слияние переименованных файлов и директорий не поддерживается.
- Невозможность удаления данных из хранилища. Информация, помещенная в хранилище Subversion, останется там навсегда. В Subversion не предусмотрено никакого штатного способа окончательного удаления файла из хранилища. Существуют сторонние программы, для автоматизации процесса создания дампа хранилища, его обработки штатной утилитой и последующем восстановлении хранилища из дампа, однако для этого требуется временное ограничение доступа к хранилищу, а также вмешательство администратора с достаточно высокими привилегиями для того, чтобы полностью стереть старое хранилище и заменить его новым [19].

2.8 Средство автоматизированной сборки проекта

Автоматизация сборки – это автоматизация широкого спектра различных задач, применяемых разработчиками в повседневной деятельности. Автоматизация включает в себя следующие действия:

- компиляция исходного кода в бинарный код;
- сборка бинарного кода;

- выполнение тестов;
- разворачивание программы на производственной платформе;
- написание сопроводительной документации или описание изменений новой версии.

Автоматизация сборки имеет следующие преимущества:

- улучшение качества продукта;
- ускорение процесса компиляции и линковки;
- избавление от лишних действий;
- минимизация некорректных сборок;
- избавление от привязки к конкретному человеку;
- ведение истории сборок и релизов для разбора выпусков;
- экономия времени и денег, благодаря причинам, указанным выше [20].

Для данной работы было выбрано средство автоматизированной сборки проектов Maven. Maven – фреймворк для автоматизации сборки проектов на основе их структуры в файлах на языке Project Object Model (POM), который является подмножеством языка XML. Maven используется для построения и управления проектами, написанными на Java, C#, Ruby, Scala и других языках [21].

Также, данное средство автоматизированной сборки проектов имеет ряд следующих преимуществ:

- Независимость от операционной системы. Сборка происходит в любой операционной системе, а файл проекта используется один и тот же.
- Управление зависимостями. Maven позволяет управлять сложными зависимостями. При подключении одних зависимостей, подключаются также и те, от которых зависят первые.

- Возможна сборка из командной строки. Такая возможность часто необходима, например, на серверах с установленной операционной системой без графического интерфейса пользователя.
- Хорошая интеграция со средами разработки. Основные среды разработки на java легко открывают проекты, которые собираются при помощи Maven. Также стоит отметить, что в выбранной среде разработки IntelliJ IDEA, данный инструмент установлен по умолчанию и его можно использовать из интерфейса интегрированной среды разработки сразу после ее установки.
- Декларативное описание проекта [22]. В файлах описания проекта содержится его спецификация, а не отдельные команды выполнения [21].

3 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Биометрия предполагает систему распознавания людей по одной или более физическим или поведенческим чертам. В области информационных технологий биометрические данные используются в качестве формы управления идентификаторами доступа и контроля доступа.

Биометрические данные можно разделить на два основных класса:

- Физиологические. Относятся к форме тела. В качестве примера можно привести: отпечатки пальцев, распознавание лица, дезоксирибонуклеиновая кислота, ладонь руки, сетчатка глаза, запах, голос.
- Поведенческие. Связаны с поведением человека. Например, походка и речь.

Биометрическая система может работать в двух режимах:

- Верификация. Сравнение один к одному с биометрическим шаблоном. Проверяет, что человек тот, за кого он себя выдает. Верификация может быть осуществлена по смарт-карте, имени пользователя или идентификационному номеру.
- Идентификация. Сравнение один ко многим: после «захвата» биометрических данных идет соединение с биометрической базой данных для определения личности. Идентификация личности проходит успешно, если биометрический образец уже есть в базе данных [23].

Системы аутентификации, использующие для удостоверения личности биометрические данные называются биометрическими системами аутентификации. Процесс доказательства и проверки подлинности заявленного пользователем имени, через предъявление пользователем своего биометрического образа и путем преобразования этого образа в соответствии

с заранее определенным протоколом аутентификации называется биометрической аутентификацией.

Методы аутентификации делятся на два вида:

- статические;
- динамические;

К статическим методам аутентификации можно отнести:

- аутентификацию по отпечатку пальца;
- аутентификацию по радужной оболочке глаза;
- аутентификацию по сетчатке глаза;
- аутентификацию по геометрии руки;
- аутентификацию по геометрии лица;
- аутентификацию по термограмме лица.

К динамическим методам аутентификации можно отнести:

- аутентификацию по голосу;
- аутентификацию по рукописному почерку.

В данной работе биометрическая система аутентификации является комбинированной или мультимодальной, т.е. такой биометрической системой аутентификации, которая применяет различные дополнения для использования нескольких типов биометрических характеристик, что позволяет соединить несколько типов биометрических технологий в системах аутентификации в одной. Это позволяет удовлетворить самые строгие требования к эффективности системы аутентификации. Например, аутентификация по отпечаткам пальцев может легко сочетаться со сканированием руки. Такая структура может использовать все виды биометрических данных человека и может применяться там, где приходится форсировать ограничения одной биометрической характеристики. Комбинированные системы являются более надежными с точки зрения возможности имитации биометрических данных человека, так как труднее

подделать целый ряд характеристик, чем фальсифицировать один биометрический признак [24].

У биометрических систем весьма перспективное будущее. Биометрические приложения аутентификации личности включают системы контроля доступа, системы учета рабочего времени, рабочие станции, сети и доступа к домену, приложения входа в систему, защиту данных, удаленный доступ к ресурсам, сопровождения сделок и веб-безопасности, и в последнее время широко применяются в области распознавания биометрической подписи. Данные технологии отнюдь не мир научной фантастики. На протяжении многих лет биометрия используется во многих малых и крупных компаниях, офисах и промышленных предприятиях. Инвестирование в покупку биометрических считывателей значительно сократилось и использование биометрических данных стало обычным явлением [25].

4 РАЗРАБОТКА БАЗЫ ДАННЫХ

4.1 Проектирование базы данных

Для возможности авторизации пользователя на сторонних сервисах необходимо сначала зарегистрировать пользователя в сервисе авторизации пользователей по биометрическим данным. Для этого необходимо спроектировать модель пользователя, которая будет содержать персональные данные пользователя.

Для пользователя сервиса авторизации должны быть доступны следующие функции:

- регистрация в сервисе авторизации пользователей по биометрическим данным;
- предоставление сервису и изменение информации по биометрии для дальнейшей авторизации в сторонних сервисах;
- авторизация пользователя на сторонних сервисах при помощи разрабатываемого сервиса авторизации пользователей по биометрическим данным.

Для хранения персональных данных пользователя и данных для авторизации в сервисе предусмотрены разные таблицы.

Как было сказано выше, пользователям необходимо предоставлять свои биометрические данные для последующей авторизации на сторонних сервисах. Для этого необходимо создать несколько таблиц, разделенных по типу биометрии и ссылающихся на одну общую.

Не имеет смысла записывать непосредственно сами данные биометрии пользователя в таблицу базы данных, т.к. чрезмерное количество информации, записанной в базе данных, может привести к увеличению объема хранимой информации и замедлению скорости поиска пользователя, запросившего авторизацию.

Авторизация на сторонних сервисах происходит в два этапа, соответственно, пользователь авторизуется двумя разными запросами, из этого следует, что для хранения промежуточных результатов авторизации пользователя необходимо использовать отдельную таблицу базы данных.

Для ускорения авторизации необходимо создать таблицу, хранящую идентификаторы устройств, с которых были отправлены запросы на авторизацию.

4.2 Проектирование таблиц базы данных

Учитывая все выше сказанное, для достижения требуемого результата, а также максимальной функциональности разрабатываемого приложения, были созданы следующие таблицы базы данных:

- общая таблица с персональными данным пользователей (auth_user);
- таблица со специфичными для группы «КЛИЕНТ» данными пользователей (client);
- таблица устройств, которые запрашивали авторизацию пользователя (authorized_device);
- сводная таблица для связи пользователей группы «КЛИЕНТ» и авторизованных устройств, тип связи «многие-ко-многим» (client_authorized_device);
- таблица с общими данными по биометрии пользователей (enroll);
- таблица с ссылкой специфичными данными по биометрии лица с ссылкой на таблицу с данными лица (enroll_face);
- таблица со специфичными данными по биометрии голоса (enroll_voice);
- таблица со специфичными данными по биометрии отпечатков пальцев (enroll_finger);

- таблица с данными лица (face_features);
- таблица с данными авторизации пользователя в сервисе авторизации пользователей по биометрическим данным с ссылкой на пользователя системы (simple_auth_user);
- таблица с промежуточными данными авторизации пользователя в сторонних сервисах с ссылкой на пользователя системы (user_hash);

4.2.1 Проектирование таблицы пользователей

Различные группы пользователей системы хранят как общие данные, так и специфичные для определенной группы. В качестве общих данных используются такие данные как:

- идентификатор пользователя в базе данных;
- группа, к которой относится пользователь;
- статус пользователя;
- электронный почтовый адрес пользователя;
- телефон пользователя;
- ФИО пользователя;
- флаг блокировки пользователя.

Группа пользователей «КЛИЕНТ» хранит специфичные только для этой группы данные, а именно:

- список авторизованных устройств.

4.2.2 Проектирование таблицы с данными авторизации в сервисе

Для авторизации пользователя в разрабатываемом сервисе при помощи логина и пароля, создана отдельная таблица, в которой будут храниться ссылки

на записи в таблице пользователей, а также пароли пользователей для входа в систему в зашифрованном виде.

4.2.3 Проектирование таблиц с ссылками на биометрические контейнеры пользователей

Биометрические данные пользователей хранятся в отдельных файлах. Не имеет смысла записывать содержимое этих файлов напрямую в базу данных как из соображений безопасности, так и из-за излишней загрузки данными, которая приведет к снижению скорости выполнения запросов, что для сервиса авторизации является одним из ключевых факторов.

4.2.4 Проектирование таблицы с авторизованными устройствами пользователей

Пользователь может быть авторизован на различных устройствах, таких как терминалы оплаты, мобильные устройства, персональные компьютеры, различные сторонние сервисы и т.д. Предполагается, что пользователь будет использовать вышеперечисленные устройства более одного раза, соответственно, для увеличения скорости поиска пользователя, отправившего запрос на авторизацию, необходимо было создать таблицу, хранящую список устройств, с которых был отправлен запрос на авторизацию.

4.2.5 Проектирование таблицы токенов

Авторизация пользователей на сторонних ресурсах проходит в два этапа:

1. Пользователем предоставляется снимок лица с веб-камеры или любого другого подобного устройства.
2. Пользователем предоставляются иные биометрические данные, например, образец голоса или скан пальца.

Так как авторизация происходит более чем в один запрос к сервису, то на стороне сервиса авторизации необходимо хранить промежуточное состояние каждой конкретной авторизации. Данная таблица хранит следующие данные:

- идентификатор записи в таблице;
- ссылка на пользователя системы;
- случайно сгенерированная строка, предоставляемая пользователю после первого этапа авторизации;
- случайно сгенерированная секретная строка, которая используется для создания токена авторизации;
- штамп даты и времени последнего изменения записи в таблице;
- флаг истечения срока действия токена;
- идентификатор устройства, отправившего запрос;
- токен, выдаваемый пользователю после второго этапа авторизации.

4.3 Проектирование базы данных

При проектировании сервиса сначала создаются классы-сущности. При запуске приложения с определенными настройками, Spring Framework сам создаст таблицы и связи между ними, основываясь на ранее созданных этих классах.

Было принято решение разработать следующие классы:

- AuthUser.
- PersonInfo.

- SimpleAuthUser.
- Enroll.
- EnrollFace.
- EnrollFinger.
- EnrollVoice.
- FaceFeature.
- AuthorizedDevice.
- Client.
- UserHash.

Каждый класс представляет собой сущность базы данных.

4.3.1 Абстрактный класс AuthUser.

Данный класс является абстрактным и содержит модель основных данных пользователя. Он имеет следующие поля:

- Long id. Идентификатор пользователя в базе данных.
- Role role. Роль пользователя (CLIENT или ADMINISTRATOR).
- Status status. Статус пользователя (NEW_USER, REGISTERED_USER, CONFIRMED_USER, BANNED_USER).
- Boolean blocked. Флаг блокировки пользователя.
- String email. Электронный почтовый адрес пользователя.
- String phone. Контактный телефон пользователя.
- PersonInfo personInfo. Персональные данные пользователя.

Все последующие классы пользователей системы будут наследоваться от абстрактного класса AuthUser и иметь как свой набор полей и методов, так и общий. Данный класс необходим для создания различных типов пользователей системы, например:

- администраторов;

- модераторов;
- пользователей технической поддержки;
- обычных пользователей.

На данном этапе наличие в проекте таких типов пользователей как: администраторы, модераторы и пользователи технической поддержки не обязательно, соответственно создавать абстрактный класс от которого будет унаследован только один класс не имеет смысла. Однако, учитывая возможность расширения системы в дальнейшем было принято решение разделить общие данные пользователей и данные, которые принадлежат только обычным пользователям системы на два класса. В дальнейшем это может значительно ускорить доработку сервиса авторизации пользователей. В листинге 4 представлена структура абстрактного класса AuthUser.

Листинг 4. Структура абстрактного класса AuthUser.

```
@Entity(name = AuthUser.Database.ENTITY_NAME)
@Table(name = AuthUser.Database.TABLE_NAME)
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name =
AuthUser.Database.COLUMN_ROLE_NAME, discriminatorType =
DiscriminatorType.STRING)
public abstract class AuthUser
{
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Enumerated(value = EnumType.STRING)
    @Column(name = Database.COLUMN_ROLE_NAME, nullable =
false, insertable = false, updatable = false)
    private Role role = Role.CLIENT;

    @Enumerated(value = EnumType.STRING)
    @Column(name = Database.COLUMN_STATUS_NAME, nullable =
false, unique = false)
    private Status status = Status.NEW_USER;

    @Column(name = Database.COLUMN_BLOCKED_NAME, nullable =
false)
    private Boolean blocked = Boolean.TRUE;
```

```

        @Column(name = Database.COLUMN_EMAIL_NAME, nullable =
false, unique = true)
        private String email;

        @Column(name = Database.COLUMN_PHONE_NAME, nullable =
true, unique = true)
        private String phone;

        @Embedded
        private PersonInfo personInfo = new PersonInfo();
...
    public static class Database{
        public static final String ENTITY_NAME = "AuthUser";
        public static final String TABLE_NAME = "auth_user";

        public static final String COLUMN_ID_NAME = "id";
        public static final String COLUMN_ROLE_NAME =
"role";
        public static final String COLUMN_STATUS_NAME =
"status";
        public static final String COLUMN_BLOCKED_NAME =
"blocked";
        public static final String COLUMN_EMAIL_NAME =
"email";
        public static final String COLUMN_PHONE_NAME =
"phone";
        public static final String JOIN_COLUMN_AUTH_USER_ID
= "auth_user_id";
    }
}

```

4.3.2 Класс PersonInfo

Данный класс является контейнером для персональных данных пользователя системы и содержит в себе три поля:

- фамилия пользователя;
- имя пользователя;
- отчество пользователя.

PersonInfo помечен аннотацией `@Embeddable`, это означает, что объекты данного класса могут быть использованы в классах-сущностях, как поля, однако в базе данных под них не будет выделено отдельной таблицы – все поля

@Embeddable класса будут помещены в ту же таблицу, что и поля класса сущности. В листинге 5 представлена структура встраиваемого класса PersonInfo.

Листинг 5. Структура встраиваемого класса PersonInfo.

```
package ru.programsolutions.ubps.entity.auth;
import javax.annotation.Nullable;
import javax.persistence.Column;
import javax.persistence.Embeddable;
import javax.validation.constraints.NotNull;
import java.util.StringJoiner;

/**
 * Встраиваемый класс с личными данными пользователей
 * Содержит следующие поля:
 * <ul>
 * <li>{@link #firstName имя пользователя}</li>
 * <li>{@link #lastName фамилия пользователя}</li>
 * <li>{@link #patronymic отчество пользователя}</li>
 * </ul>
 */
@Embeddable
public class PersonInfo
{
    @Column(name = Database.COLUMN_FIRST_NAME_NAME, nullable
= false, unique = false, length = 32)
    private String firstName;

    @Column(name = Database.COLUMN_LAST_NAME_NAME, nullable
= false, unique = false, length = 32)
    private String lastName;

    @Column(name = Database.COLUMN_PATRONYMIC_NAME, nullable
= true, unique = false, length = 32)
    private String patronymic = "";
    ...
    public static class Database{
        public static final String COLUMN_FIRST_NAME_NAME =
"first_name";
        public static final String COLUMN_LAST_NAME_NAME =
"last_name";
        public static final String COLUMN_PATRONYMIC_NAME =
"patronymic";
    }
}
```

4.3.3 Класс SimpleAuthUser

Данный класс содержит информацию по авторизации пользователей на сайте и имеет следующие поля:

- идентификатор записи;
- ссылка на пользователя, которому принадлежат сведения по авторизации;
- пароль в зашифрованном виде;

В листинге 6 представлена структура класса SimpleAuthUser.

Листинг 6. Структура класса SimpleAuthUser.

```
package ru.programsolutions.ubps.entity.auth;

import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAut
hority;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

/**
 * Содержит данные авторизации пользователей
 */
@Entity(name = SimpleAuthUser.Database.ENTITY_NAME)
@Table(name = SimpleAuthUser.Database.TABLE_NAME)
public class SimpleAuthUser implements RegistrationAuth,
UserDetails {
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne
```

```

        @JoinColumn(name =
AuthUser.Database.JOIN_COLUMN_AUTH_USER_ID,
referencedColumnName = AuthUser.Database.COLUMN_ID_NAME)
        private AuthUser authUser;

        @Column(name = Database.COLUMN_PASSWORD_NAME, nullable =
false, unique = false, length = 128)
        private String password;
...
        @Override
        public Collection<? extends GrantedAuthority>
getAuthorities() {
            SimpleGrantedAuthority sia = new
SimpleGrantedAuthority(this.authUser.getRole().toString());
            List<SimpleGrantedAuthority> authList = new
ArrayList<>();
            authList.add(sia);
            return authList;
        }
...
        public static class Database {
            public static final String ENTITY_NAME =
"SimpleAuthUser";
            public static final String TABLE_NAME =
"simple_auth_user";

            public static final String COLUMN_ID_NAME = "id";
            public static final String COLUMN_PASSWORD_NAME =
"password";
        }
    }
}

```

4.3.4 Класс Enroll

Данный класс хранит ссылку на пользователя, предоставившего биометрические данные, тип биометрии и путь к файлу с биометрической информацией пользователя. В листинге 7 представлена структура класса Enroll.

Листинг 7. Структура класса Enroll.

```

@Entity(name = Enroll.Database.ENTITY_NAME)
@Table(name = Enroll.Database.TABLE_NAME)
@Inheritance(strategy = InheritanceType.JOINED)

```

```

@DiscriminatorColumn(name =
Enroll.Database.COLUMN_ENROLL_TYPE_NAME, discriminatorType =
DiscriminatorType.STRING)
public class Enroll {
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    protected Long id;

    @JsonManagedReference
    @ManyToOne
    @JoinColumn(name =
Client.Database.JOIN_COLUMN_CLIENT_ID_NAME,
referencedColumnName = AuthUser.Database.COLUMN_ID_NAME,
nullable = false, updatable = false)
    protected Client client;

    @Enumerated(value = EnumType.STRING)
    @Column(name = Database.COLUMN_ENROLL_TYPE_NAME,
insertable = false, updatable = false)
    protected EnrollType type;

    @Column(name = Database.COLUMN_PATH_NAME, nullable =
false, unique = true, length = 512)
    protected String path;

    @PostRemove
    private void removeFile() {
        File file = new File(this.path);
        boolean exists = false;
        try {
            exists = file.exists();
        } catch (SecurityException e) {
            System.err.println("SecurityException on file ["
+ file.getAbsolutePath() + "]; Read-access denied. Delete
file manually if exists");
        }
        if (exists) {
            try {
                file.delete();
            } catch (SecurityException e) {
                System.err.println("SecurityException on
file [" + file.getAbsolutePath() + "]; Delete-access denied.
Delete it manually");
            }
        }
    }
}

```

Как видно из примера выше, данный класс содержит в себе метод `removeFile()`, который помечен аннотацией `@PostRemove`. Данная аннотация

означает, что аннотированный ей метод будет выполнен автоматически после того, как из таблицы enroll базы данных будет удалена соответствующая запись. В этом методе описывается алгоритм удаления файла, хранящего биометрические данные пользователя, с жесткого диска устройства, на котором запущен сервис. Это позволяет сэкономить время и избежать таких ситуаций, связанных с удалением файла вручную, при которых данный метод случайно может быть вызван более одного раза или не вызван вовсе.

4.3.5 Класс EnrollFace

Данный класс является потомком класса Enroll и содержит ссылку на таблицу с дополнительной информацией по биометрии лица пользователя. В листинге 8 представлена структура класса EnrollFace.

Листинг 8. Структура класса EnrollFace.

```
package ru.programsolutions.ubps.entity.enroll;

import javax.persistence.*;

@Entity(name = EnrollFace.Database.ENTITY_NAME)
@Table(name = EnrollFace.Database.TABLE_NAME)
@DiscriminatorValue(EnrollFace.Database.DISCIMINATOR)
public class EnrollFace extends Enroll {
    @OneToOne(targetEntity = FaceFeature.class, cascade =
CascadeType.PERSIST)
    @JoinColumn(name =
FaceFeature.Database.JOIN_COLUMN_ID_NAME,
referencedColumnName = FaceFeature.Database.COLUMN_ID_NAME)
    private FaceFeature faceFeature;

    public EnrollFace() {
        this.type = EnrollType.FACE;
    }

    public FaceFeature getFaceFeature() {
        return faceFeature;
    }

    public void setFaceFeature(FaceFeature faceFeature) {
```

```

        this.faceFeature = faceFeature;
    }

    public static class Database {
        public static final String ENTITY_NAME =
"EnrollFace";
        public static final String TABLE_NAME =
"enroll_face";
        public static final String DISCRIMINATOR = "FACE";
    }
}

```

4.3.6 Класс FaceFeature

Данный класс содержит дополнительную информацию по биометрии лица пользователя, которая позволяет ускорить поиск пользователя при повторном запросе авторизации, путем исключения тех пользователей, чьи характеристики не подходят под заданные. В листинге 9 представлена структура класса FaceFeature.

Листинг 9. Структура класса FaceFeature.

```

package ru.programsolutions.ubps.entity.enroll;

import javax.persistence.*;

@Entity(name = FaceFeature.Database.ENTITY_NAME)
@Table(name = FaceFeature.Database.TABLE_NAME)
public class FaceFeature {
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = Database.COLUMN_AVERAGE_AGE_NAME)
    private Integer averageAge;

    @Enumerated(value = EnumType.STRING)
    @Column(name = Database.COLUMN_GENDER_NAME)
    private Gender gender;

    public FaceFeature() {
    }
}

```

```

        public FaceFeature(Integer averageAge, Gender gender) {
            this.averageAge = averageAge;
            this.gender = gender;
        }

        public static class Database {
            public static final String ENTITY_NAME =
"FaceFeature";
            public static final String TABLE_NAME =
"face_feature";

            public static final String COLUMN_ID_NAME = "id";
            public static final String COLUMN_AVERAGE_AGE_NAME =
"average_age";
            public static final String COLUMN_GENDER_NAME =
"gender";

            public static final String JOIN_COLUMN_ID_NAME =
TABLE_NAME + "_" + COLUMN_ID_NAME;
        }
    }
}

```

4.3.7 Класс EnrollVoice

Данный класс является потомком класса Enroll, однако на данный момент не содержит никаких дополнительных данных, описывающих предоставленный пользователем образец голоса. В листинге 10 представлена структура класса EnrollVoice.

Листинг 10. Структура класса EnrollVoice.

```

package ru.programsolutions.ubps.entity.enroll;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity(name = EnrollVoice.Database.ENTITY_NAME)
@Table(name = EnrollVoice.Database.TABLE_NAME)
@DiscriminatorValue(EnrollVoice.Database.DISCIMINATOR)
public class EnrollVoice extends Enroll{

    public EnrollVoice() {
    }
}

```

```

        public static class Database{
            public static final String ENTITY_NAME =
"EnrollVoice";
            public static final String TABLE_NAME =
"enroll_voice";
            public static final String DISCRIMINATOR = "VOICE";
        }
    }
}

```

4.3.8 Класс AuthorizedDevice

Данный класс хранит информацию по авторизованным устройствам пользователей. В листинге 11 структура класса AuthorizedDevice.

Листинг 11. Структура класса AuthorizedDevice.

```

/**
 * Информация об авторизованных устройствах пользователей
 */
@Entity(name = AuthorizedDevice.Database.ENTITY_NAME)
@Table(name = AuthorizedDevice.Database.TABLE_NAME)
public class AuthorizedDevice {
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = Database.COLUMN_DEVICE_IDENTITY_NAME,
unique = true, length = 255)
    private String deviceId;

    @ManyToMany(targetEntity = Client.class)
    @JoinTable(name =
Client.Database.JOIN_TABLE_CLIENT_AUTHORIZED_DEVICE_NAME,
        joinColumns = @JoinColumn(name =
Database.JOIN_COLUMN_AUTHORIZED_DEVICE_ID_NAME,
referencedColumnName = Database.COLUMN_ID_NAME),
        inverseJoinColumns = @JoinColumn(name =
Client.Database.JOIN_COLUMN_CLIENT_ID_NAME,
referencedColumnName = AuthUser.Database.COLUMN_ID_NAME))
    private List<Client> clients = new ArrayList<Client>();
    ...
    public static class Database {
        public static final String ENTITY_NAME =
"AuthorizedDevice";
        public static final String TABLE_NAME =
"authorized_device";
    }
}

```

```

        public static final String COLUMN_ID_NAME = "id";
        public static final String
COLUMN_DEVICE_IDENTITY_NAME = "device_identity";
        public static final String
JOIN_COLUMN_AUTHORIZED_DEVICE_ID_NAME = TABLE_NAME + "_" +
COLUMN_ID_NAME;
    }
}

```

4.3.9 Класс Client

Данный класс является наследником класса AuthUser и представляет из себя модель обычного пользователя системы с ролью «КЛИЕНТ». В данном случае, клиент хранит те же самые данные, что и AuthUser, а также список авторизованных устройств. В листинге 12 представлена структура класса Client.

Листинг 12. Структура класса Client.

```

package ru.programsolutions.ubps.entity;

import ru.programsolutions.ubps.entity.auth.AuthUser;
import ru.programsolutions.ubps.entity.auth.Role;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

/**
 * Класс с информацией по пользователю типа
<code>КЛИЕНТ</code>
 */
@Entity(name = Client.Database.ENTITY_NAME)
@Table(name = Client.Database.TABLE_NAME)
@DiscriminatorValue(value = Client.Database.DISCIMINATOR)
public class Client extends AuthUser {
    @ManyToMany(targetEntity = AuthorizedDevice.class,
cascade = CascadeType.PERSIST)
    @JoinTable(name =
Database.JOIN_TABLE_CLIENT_AUTHORIZED_DEVICE_NAME,
        joinColumns = @JoinColumn(name =
Database.JOIN_COLUMN_CLIENT_ID_NAME, referencedColumnName =
AuthUser.Database.COLUMN_ID_NAME),

```

```

        inverseJoinColumn = @JoinColumn(name =
AuthorizedDevice.Database.JOIN_COLUMN_AUTHORIZED_DEVICE_ID_N
AME, referencedColumnName =
AuthorizedDevice.Database.COLUMN_ID_NAME))
        private List<AuthorizedDevice> authorizedDevices = new
ArrayList<>();
...
    public static class Database {
        public static final String ENTITY_NAME = "Client";
        public static final String TABLE_NAME = "client";
        public static final String DISCRIMINATOR = "CLIENT";
        public static final String
JOIN_TABLE_CLIENT_AUTHORIZED_DEVICE_NAME =
"client_authorized_device";
        public static final String
JOIN_COLUMN_CLIENT_ID_NAME = "client_id";
    }
}

```

4.3.10 Класс UserHash

Данный класс хранит данные для аутентификации пользователей на сторонних ресурсах. В листинге 13 представлена структура класса UserHash.

Листинг 13. Структура класса UserHash.

```

/**
 * Хранит данные для аутентификации пользователя на
 * сторонних ресурсах
 */
@Entity(name = UserHash.Database.ENTITY_NAME)
@Table(name = UserHash.Database.TABLE_NAME)
public class UserHash {
    @JsonIgnore
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne
    @JoinColumn(name =
AuthUser.Database.JOIN_COLUMN_AUTH_USER_ID, referencedColumnName =
AuthUser.Database.COLUMN_ID_NAME)
    private AuthUser authUser;

    @Column(name = Database.COLUMN_HASH_NAME, nullable =
false)

```

```

        private String hash;

        @JsonIgnore
        @Temporal(TemporalType.TIMESTAMP)
        @Column(name = Database.COLUMN_LAST_MODIFIED_NAME,
nullable = false)
        private Date lastModified = new Date();

        @JsonIgnore
        @Column(name = Database.COLUMN_EXPIRED_NAME, nullable =
false)
        private Boolean expired = Boolean.FALSE;

        @JsonIgnore
        @Column(name = Database.COLUMN_DEVICE_ID_NAME, nullable
= false)
        private String deviceId;

        @JsonIgnore
        @Column(name = Database.COLUMN_AUTH_HASH_NAME)
        private String authHash;

        @JsonIgnore
        @Column(name = Database.COLUMN_AUTH_SECRET_NAME)
        private String authSecret;
...
        public boolean checkHash(String authHash) {
            BCryptPasswordEncoder bCryptPasswordEncoder = new
BCryptPasswordEncoder(12);
            boolean matches = bCryptPasswordEncoder.matches(hash
+ authSecret, authHash);
            return matches;
        }
    }
}

```

5 РАЗРАБОТКА ПРИЛОЖЕНИЯ

При разработке приложения были созданы следующие компоненты системы:

- основной файл конфигурации фреймворка Spring;
- файлы конфигурации BcryptPasswordEncoder, Google reCAPTCHA, настройки размещения мобильных приложений, настройки хранилища файлов, настройки валидации объектов, настройки минимального порога верификации биометрических данных;
- классы конфигурации фреймворка Spring;
- классы для работы с сервисом Google reCAPTCHA;
- классы модели данных, объекты которых хранятся в базе данных;
- интерфейсы репозиториев;
- классы контроллеров;
- классы для отправки сообщений пользователям на их электронную почту;
- классы свойств;
- сервисы нотификации пользователей на сайте и поиска пользователей по биометрическим данным;
- сервис размещения и хранения загруженных пользователями файлов;
- классы-валидаторы;
- утилитные классы.

5.1 Конфигурации приложения

Как было сказано ранее Spring Framework поддерживает конфигурацию через внешние файлы настроек, Java код, аннотации и XML файлы. В проекте использованы первые три способа.

Внешние файлы настроек представляют из себя файлы с расширением `properties`. В этих файлах настройка основных компонентов Spring происходит следующим образом:

1. Указывается название свойства, чье значение будет выставлено в этой строке.
2. Ставится знак равенства, выступающий разделителем между именем свойства и значением.
3. Указывается значение, которое будет присвоено свойству

Для конфигурации подключения к базе данных, сервиса отправления электронных почтовых сообщений, доступа к ресурсам сервиса, и т.п. использовался метода конфигурации при помощи Java кода и аннотаций. В листинге 14 представлен пример конфигурации подключения к базе данных при помощи Java кода и аннотаций.

Листинг 14. Пример конфигурации подключения к базе данных при помощи Java кода и аннотаций.

```
@Configuration
@EnableTransactionManagement
public class JpaConfig implements
TransactionManagementConfigurer
{
    @Value("${dataSource.driverClassName}")
    private String driver;
    @Value("${dataSource.url}")
    private String url;
    @Value("${dataSource.username}")
    private String username;
    @Value("${dataSource.password}")
    private String password;
    @Value("${hibernate.dialect}")
    private String dialect;
    ...
}
```

```

@Value("${entitymanager.packages.to.scan}")
private String[] emPackagesToScan;

@Bean(name = "dataSource")
public DataSource configureDataSource() {
    HikariConfig config = new HikariConfig();
    config.setDriverClassName(driver);
    config.setJdbcUrl(url);
    config.setUsername(username);
    config.setPassword(password);
    return new HikariDataSource(config);
}

@Bean
public LocalContainerEntityManagerFactoryBean
entityManagerFactory() {
    LocalContainerEntityManagerFactoryBean
entityManagerFactoryBean = new
LocalContainerEntityManagerFactoryBean();

entityManagerFactoryBean.setDataSource(configureDataSource()
);

...
}
...
}

```

Как видно из примера выше, был создан класс, который выполняет конфигурацию подключения к базе данных. Значение полей класса, помеченных аннотацией `@Value`, задаются фреймворком автоматически на стадии инициализации. Значения берутся из конфигурационных файлов.

В этой аннотации указывается у какого свойства в конфигурационном файле необходимо взять значение и присвоить аннотированному полю класса. Spring, взяв значение свойства попытается привести его к типу аннотированного поля.

Так как в проекте используется сторонняя реализация пула подключений к базе данных, то его необходимо настроить вручную и передать бину фабрики менеджеров сущностей.

5.2 Конфигурация приложения для работы с сервисом Google reCAPTCHA

ReCAPTCHA – система для защиты веб-сайтов от интернет-ботов и одновременной помощи в оцифровке текстов книг. Для использования данного сервиса необходимо зайти на страницу управления ключами ReCaptcha API и зарегистрировать новый ключ [26]. Рисунок 1 содержит пример заполнения формы регистрации сайта.

Регистрация сайта

Название

ubps

Выберите тип reCAPTCHA ?

☐ reCAPTCHA v2
Выполнять проверку пользователей с помощью флажка "Я не робот"

☒ Invisible reCAPTCHA
Выполнять проверку пользователей в фоновом режиме

☐ reCAPTCHA для Android
Проверка пользователей в приложении для Android.

Домены
(по одному в строке)

localhost
demo.ubps.co.uk
landf.ru
192.168.5.150
192.168.19.190

☒ Примите условия использования reCAPTCHA.

You agree to explicitly inform visitors to your site that you have implemented reCAPTCHA on your site and that their use of reCAPTCHA is subject to the Google [Privacy Policy](#) and [Terms of Use](#).

By accessing or using the reCAPTCHA APIs, you agree to the Google APIs [Terms of Use](#), Google [Terms of Use](#), and to the Additional Terms below. Please read and understand all applicable terms and policies before accessing the APIs.

► [Условия использования reCAPTCHA](#)

☒ Отправлять владельцам оповещения ?

Регистрация

Рисунок 1 - Регистрация ключа в сервисе Google reCAPTCHA

При регистрации ключа выдается пара ключей «ключ-секретный ключ». Первый ключ добавляется в HTML-код сайта. Секретный ключ нужен для связи сайта с Google.

Для проверки пользователя на стороне сервера были использованы классы интеграции с сервисом Google reCAPTCHA [27].

5.3 Создание классов сущностей

В данном проекте классы сущности играют ключевую роль. На основе них при первом запуске программы формируются и настраиваются таблицы базы данных и их поля. В листинге 15 представлен пример создания класса сущности UserHash.

Листинг 15. Пример создания класса сущности UserHash.

```
@Entity(name = UserHash.Database.ENTITY_NAME)
@Table(name = UserHash.Database.TABLE_NAME)
public class UserHash {
    @JsonIgnore
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne
    @JoinColumn(name =
AuthUser.Database.JOIN_COLUMN_AUTH_USER_ID, referencedColumnN
ame = AuthUser.Database.COLUMN_ID_NAME)
    private AuthUser authUser;

    @Column(name = Database.COLUMN_HASH_NAME, nullable =
false)
    private String hash;

    @JsonIgnore
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = Database.COLUMN_LAST_MODIFIED_NAME,
nullable = false)
    private Date lastModified = new Date();

    @JsonIgnore
```

```

        @Column(name = Database.COLUMN_EXPIRED_NAME, nullable =
false)
        private Boolean expired = Boolean.FALSE;

        @JsonIgnore
        @Column(name = Database.COLUMN_DEVICE_ID_NAME, nullable
= false)
        private String deviceId;

        @JsonIgnore
        @Column(name = Database.COLUMN_AUTH_HASH_NAME)
        private String authHash;

        @JsonIgnore
        @Column(name = Database.COLUMN_AUTH_SECRET_NAME)
        private String authSecret;
        ...
    }

```

Из приведенного выше примера видно, что класс аннотирован двумя аннотациями. Их предназначение в следующем:

- Аннотация `@Entity` указывает, что данный класс является сущностью, а его параметр `name`, указывает на то, по какому имени к этому классу необходимо обращаться в hql-запросах. Классы отмеченные аннотацией `@Entity` должны содержать пустой конструктор с любым модификатором доступа.
- Аннотация `@Table` указывает на то, что данный класс будет связан с таблицей в базе данных, а ее параметр `name`, указывает на наименование этой таблицы.

У полей данного класса можно заметить такие аннотации как: `@Id`, `@GeneratedValue`, `@Column`, `@OneToOne`, `@JoinColumn`, `@Temporal` и `@JsonIgnore`.

- Аннотация `@Id` указывает на первичный ключ данного класса.
- Аннотация `@GeneratedValue` указывается вместе с аннотацией `@Id` и указывает на стратегию генерации или сам генератор уникальных значений.

- Аннотация `@Column` определяет к какому столбцу в таблице относится данное поле. Данная аннотация имеет такие параметры как `name`, указывающий на наименование поля таблицы, `unique`, указывающий на то, должно ли быть значение этого поля уникальным, `nullable`, указывающий на то, может ли у данного поля отсутствовать значение и `length`, указывающий на длину столбца (например для значений типа `String`) [28].
- Аннотация `@OneToOne` указывает, что этот класс относится к аннотируемому классу как один-к-одному.
- Аннотация `@JoinColumn` указывает на то, с каким столбцом таблицы аннотируемого класса необходимо выставить связь. Указанный столбец аннотируемого класса будет добавлен к таблице этого класса. Для более точной настройки и во избежание возможных ошибок маппинга, в данной аннотации стоит проставить значение параметру `referencedColumnName`, который в качестве значения принимает строку с названием связываемого столбца.
- Аннотация `@Temporal` указывает на то, что в данном столбце будет храниться объект типа «ВРЕМЯ», причем можно указать в каком формате: штамп даты, штамп времени или штамп даты и времени.

5.4 Создание репозитория

Репозитории предоставляют программе доступ к данным, хранящимся в базе данных. В разработанном приложении под каждый класс-сущность были созданы свои интерфейсы репозитория. Spring Framework берет реализацию данных интерфейсов на себя, тем самым заметно упрощая и ускоряя разработку приложения.

По умолчанию, в унаследованных от интерфейса `CrudRepository<T, ID>` репозиториях (где `T` – тип объекта, а `ID` – тип идентификатора), есть следующие методы:

- `long count()` – возвращает число типа `long`, значение которого указывает на то, сколько объектов заданного типа сохранено в базе данных;
- `void delete(ID id)` – удаляет из базы данных объект типа `T` с указанным идентификатором `id`.
- `void delete(T entity)` – удаляет из базы данных объект типа `T`, соответствующий переданному `entity`;
- `void delete(Iterable<T> entities)` – удаляет из базы данных объекты типа `T`, соответствующие переданным в `Iterable<T>` коллекции объектов `entities`;
- `void deleteAll()` – удаляет все объекты типа `T` из базы данных;
- `boolean exists(ID id)` – возвращает `true`, если в базе данных существует запись (сохранен объект типа `T`) с идентификатором указанным идентификатором `id`;
- `Iterable<T> findAll()` – возвращает `Iterable<T>` коллекцию всех объектов, сохраненных в базе данных;
- `Iterable<T> findAll(Iterable<ID> ids)` – возвращает `Iterable<T>` коллекцию всех объектов с указанными идентификаторами;
- `T findOne(ID id)` – возвращает объект с идентификатором `id`, либо `null`, если такого объекта не существует в базе данных;
- `T save(T entity)` – сохраняет переданный объект типа `T` в базе данных и, при удачном сохранении, возвращает сохраненный в базе данных объект;
- `Iterable<T> save(Iterable<T> entities)` – сохраняет в базе данных все объекты из переданной `Iterable<T>` коллекции `entities` и

возвращает коллекцию `Iterable<T>` объектов, сохраненных в базе данных.

Однако, иногда может потребоваться написать свои методы для взаимодействия с базой данных, это можно сделать несколькими способами:

- при помощи аннотации `@Query` и подобных;
- при помощи объявления метода, удовлетворяющему специальному паттерну Spring Data Repository.

При использовании метода с аннотацией `@Query`, Spring Framework подставляет в реализацию метода строку с запросом к базе данных, указанную в качестве параметра упомянутой выше аннотации.

При использовании паттерна, Spring Framework, основываясь на названии метода, типе возвращаемого значения и переданных параметрах, генерирует запрос к базе данных автоматически. В листинге 16 представлен пример создания репозитория.

Листинг 16. Пример создания репозитория.

```
package ru.programsolutions.ubps.repository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import ru.programsolutions.ubps.entity.auth.AuthUser;
import ru.programsolutions.ubps.entity.auth.Role;

public interface AuthUserRepository extends
    CrudRepository<AuthUser, Long>
{
    @Query("select au.role from AuthUser au where au.email =
:login")
    Role getAuthRole(@Param("login") String login);

    AuthUser findByLogin(String login);
}
```

Как видно из примера выше, был создан интерфейс репозитория, унаследованный от `CrudRepository`, при этом указано, что в качестве класса

сущности используется `AuthUser`, а в качестве класса идентификатора используется `Long`. У данного интерфейса есть два метода:

- `Role getAuthRole(@Param("login") String login)` – помечен аннотацией `@Query`. Параметр этой аннотации – строка запроса на языке `hql`, в качестве именованного параметра `login`, в строку запроса будет передан параметр метода с аннотацией `@Param`, значение которого совпадает с названием именованного параметра. После выполнения запроса полученный результат будет приведен к указанному, в качестве возвращаемого, типу.
- `AuthUser findByLogin(String login)` – соответствует паттерну репозитория `Spring Data`. При реализации данного метода фреймворком будет сконструирована строка запроса на поиск и возврат первого объекта, чье значение свойства `login` совпадает с переданным параметром.

5.5 Создание контроллеров

В разработанном приложении огромную роль играют так называемые контроллеры. Они представляют из себя классы, внутри которых описаны методы, вызываемые фреймворком при получении HTTP запроса от пользователя. Контроллеры помечаются аннотацией `@Controller` и также могут быть помечены аннотацией `@RequestMapping`.

Аннотация `@RequestMapping` точно сконфигурировать какой метод класса будет вызван при определенных запросах пользователей. При конфигурации этой аннотации можно указать значения следующим ее параметрам:

- `value` – маппинг URL, поддерживает задание значения в стиле паттерна Ant, значение по умолчанию пустой массив строк;

- path – является синонимом параметра value, при присвоении значения к одному из этих параметров, указывать значение для параметра-синонима не нужно;
- name – наименование маппинга, значение по умолчанию пустая строка;
- method – указание с какими методами HTTP-запроса связывать метод контроллера, по умолчанию пустой массив объектов перечисления RequestMethod;
- params – указание входящих параметров и их значений, при совпадении которых необходимо вызвать метод контроллера, значение по умолчанию пустой массив строк;
- headers – аналогично параметру params, однако для заголовков запроса, значение по умолчанию пустой массив строк;
- consumes – указание при каких MIME-типах данных входящего запроса будет вызван метод контроллера, значение по умолчанию пустой массив строк;
- produces – указание при каких MIME-типах данных, которых ожидает отправитель запроса, будет вызван метод контроллера, значение по умолчанию пустой массив строк.

подавляющее большинство запросов содержат в себе параметры. Для того, чтобы иметь к ним доступ теле метода, необходимо для каждого получаемого параметра в списке входящих параметров метода необходимо указать аннотацию `@RequestParam` в настройках которой можно задать значения следующим параметрам:

- value – указывает, значение какого параметра запроса будет присвоено аннотированному параметру метода, по умолчанию пустая строка;

- name – является синонимом параметра value, при присвоении значения к одному из этих параметров, указывать значение для параметра-синонима не нужно;
- required – указывает на обязательность присутствия параметра в запросе, по умолчанию true;
- defaultValue – указывает на то, какое значение будет присвоено аннотированному параметру метода, если параметр запроса с указанным именем отсутствует, по умолчанию используется значение «\n\t\t\n\t\t\n\uE000\uE001\uE002\n\t\t\t\t\n», что является заменой слова «null».

В контроллере также могут использоваться различные сервисы, компоненты, репозитории и другие контроллеры. Так как в программе они автоматически инициализируются фреймворком, то для доступа к ним из класса контроллера, необходимо добавить поле с типом, соответствующему необходимому, указать имя поля, а также пометить его аннотацией `@Autowired`. При инициализации контроллера Spring Framework сам присвоит этому полю ссылку на уже инициализированный компонент. В листинге 17 представлен пример создания контроллера.

Листинг 17. Пример создания контроллера.

```
@Controller
public class MyCabinetController
{
    @Autowired
    private CurrentUser currentUser;

    @Autowired
    private MobileProperty mobileProperty;

    @Autowired
    private NotificationService notificationService;

    @RequestMapping(value = Path.MY_CABINET, method =
RequestMethod.GET)
    public String myCabinetPage(Model model)
    {
```

```

        AuthUser user = currentUser.getUser();
        model.addAttribute("authUser",user);

model.addAttribute("androidAppLink",mobileProperty.getAndroidAppLink());

        return Template.MY_CABINET;
    }

    public class Path{
        public static final String MY_CABINET = "/profile";
    }

    private class Template{
        public static final String MY_CABINET = "profile";
    }
}

```

Из представленного выше листинга видно, что класс MyCabinetController помечен аннотацией @Controller имеет метод myCabinetPage, помеченный аннотацией @RequestMapping, который возвращает пользователю страницу с данными этого пользователя.

5.6 Создание компонента авторизации пользователя на сторонних ресурсах сети Интернет

Для авторизации пользователей на сторонних ресурсах сети Интернет был разработан алгоритм, осуществляющий авторизацию пользователей в два этапа:

1. Предварительная авторизация по лицу пользователя. Сторонним ресурсом сети Интернет осуществляется HTTP-запрос в разработанный сервис, содержащий контейнер с биометрическими данными пользователя, примерный возраст, пол и идентификатор устройства. Сервис анализирует полученные данные, формирует список подходящих под заданные параметры пользователей и сверяет контейнер с данными по биометрии с

теми, что были загружены пользователями ранее. После этого сервис формирует публичный уникальный токен, который будет передан при успешной предварительной авторизации пользователю и записывает в базу данных, уникальный секретный токен и на основе них формирует при помощи хэширования третий токен, который вместе со вторым будет записан в базу данных.

2. Конечная авторизация по голосу пользователя или его отпечатку пальца. Сторонним ресурсом сети Интернет осуществляется HTTP-запрос в разработанный сервис, содержащий идентификатор пользователя системы, публичный токен, контейнер с биометрическими данными пользователя и идентификатор устройства. Сначала, по базе данных, осуществляется поиск объекта, содержащего информацию по токенам. Далее осуществляется операция конкатенации публичного и секретного токенов. Затем выполняется операция сравнения конкатенированных и хэшированного на этапе предварительной авторизации токенов. В случае, если операция прошла успешно, сравнивается принятый от пользователя и хранящийся в базе данных контейнеры с биометрическими данными. Результат выполнения отправляется клиенту, инициировавшему запрос.

6 РУКОВОДСТВО

6.1 Руководство по сборке проекта

Сборка проекта может быть выполнена через интегрированную среду разработки IntelliJ IDEA или вручную.

6.1.1 Сборка проекта при помощи интегрированной среды разработки.

Необходимо открыть проект интегрированную среду разработки IntelliJ IDEA. Если справа отсутствует вкладка «Maven Projects», то нужно перейти во вкладку «View» → «Tool Windows» и выбрать пункт «Maven Projects». Далее необходимо перейти во вкладку Maven Projects, раскрыть пункт с названием проекта, затем раскрыть пункт Lifecycle и выбрать пункт install. После этого, необходимо подождать пока Maven закончит сборку проекта. Собранный проект будет находится в директории target в виде *.war архива.

6.1.2 Сборка проекта при помощи Maven

Для начала необходимо скачать с официального сайта и разархивировать архив с приложением Maven в любую директорию. Затем необходимо открыть свойства компьютера и нажать на вкладку «Дополнительные параметры системы». После этого во вкладке «Дополнительно» нужно нажать на кнопку «Переменные среды». В любой из двух форм появившегося окна, выбрать переменную «Path» и добавить к ней путь до папки «bin» приложения Maven.

Для проверки корректности выполнения предыдущего шага, необходимо открыть командную строку и ввести команду «mvn -v». В результате выполнения команды на экран терминала должно вывестись

сообщение, содержащее версию продукта, путь до директории с Maven, версию Java, путь до директории с Java, локализацию по умолчанию и информацию об операционной системе.

После этого в окне командной строки нужно выполнить команду перехода в папку с проектом, а затем выполнить команду «mvn install». После этого необходимо дождаться, пока Maven закончит сборку проекта. Собранный проект будет находиться в директории «target» в виде *.war архива.

6.2 Руководство по размещению проекта на сервере

После сборки проекта необходимо при помощи приложения FileZilla установить связь с сервером, и скопировать собранный *.war архив в директорию «webapps», которая находится в директории, куда установлен Apache Tomcat. После выполнения операции копирования закрыть соединение.

Далее необходимо установить соединение с сервером при помощи приложения PuTTY. После того как соединение будет установлено, перейти в директорию, где находится Apache Tomcat, затем в директорию «bin». После этого в этой директории необходимо выполнить запуск контейнера сервлетов, путем ввода команды «./startup.sh».

6.3 Руководство по регистрации пользователя в сервисе авторизации

После запуска контейнера сервлетов Apache Tomcat, необходимо с помощью веб-браузера перейти на страницу веб-сервиса. Далее в правом верхнем углу страницы выбрать пункт «SIGN UP». На открывшейся странице заполнить форму регистрации и нажать на кнопку «CONTINUE». Рисунок 2 содержит пример заполнения формы регистрации пользователя.

SIGN UP

Миронов

Андрей

Patronymic (not required)

mironov.andrey.1994@gmail.com

.....

.....

[Sign in](#)
[Forgot password?](#)

CONTINUE

Рисунок 2 - Пример заполнения формы регистрации пользователя

После успешной регистрации, на указанный почтовый адрес придет письмо с ссылкой для активации аккаунта. Рисунок 3 содержит пример содержания письма с ссылкой для подтверждения аккаунта.

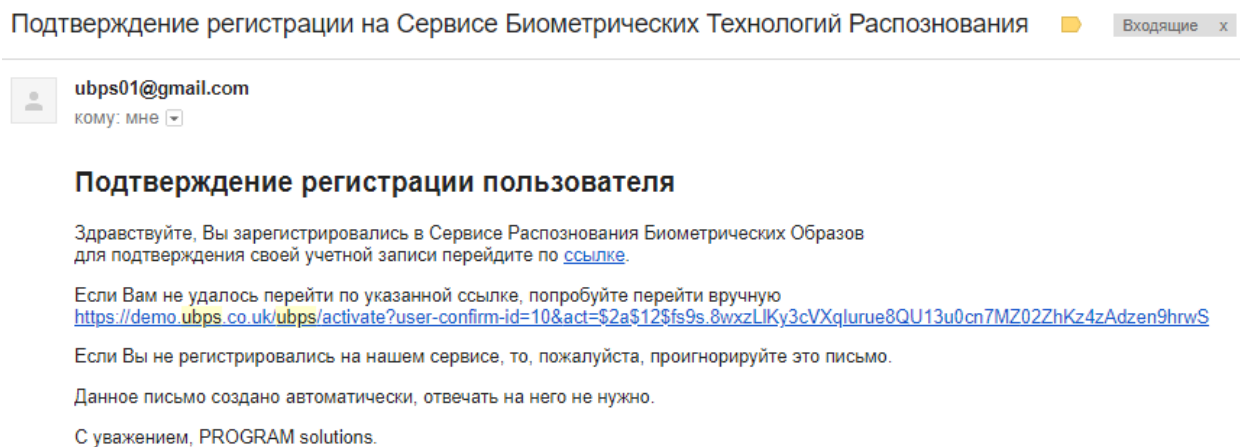


Рисунок 3 - Пример содержания письма

После этого пользователь может авторизоваться на сайте, перейти в личный кабинет и ознакомиться с подробным описанием дальнейших действий. Рисунок 4 содержит описание последовательности действий для использования приложения Demo Bank Application.

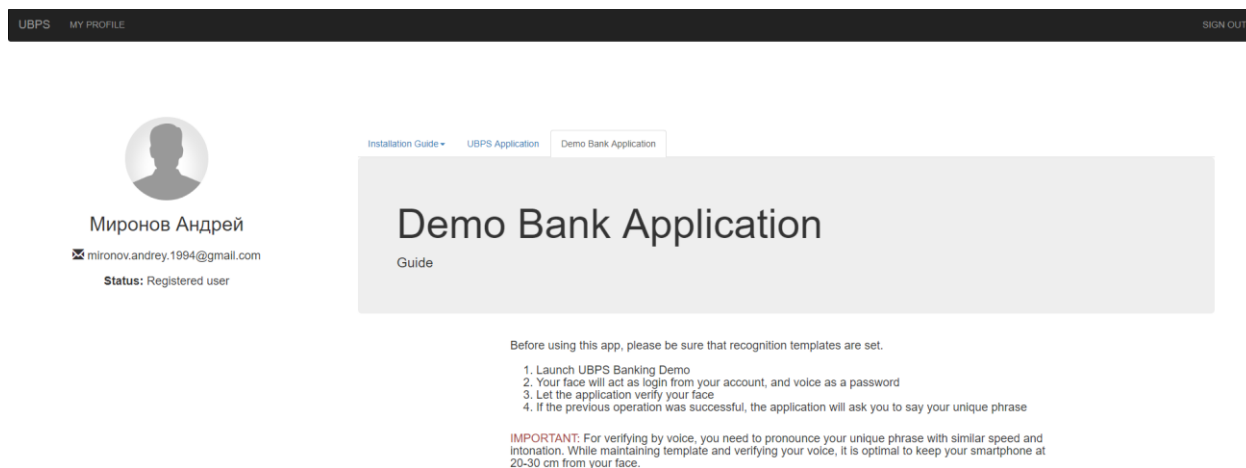


Рисунок 4 - Описания последовательности действий для использования приложения Demo Bank Application

6.4 Руководство по работе с сервисом авторизации пользователей по биометрическим данным через сторонние сервисы

Для примера работы с сервисом авторизации пользователей по биометрическим данным через сторонние сервисы был разработан демонстрационный сервис онлайн банка. Вход в личный кабинет пользователя данного сервиса осуществляется путем предоставления сервису лица пользователя через веб-камеру. После выполнения снимка веб-камерой, осуществляется процесс сбора биометрической информации по предоставленному снимку и отправляется в сервис авторизации пользователей по биометрическим данным. Рисунок 5 содержит пример авторизации пользователя по лицу.

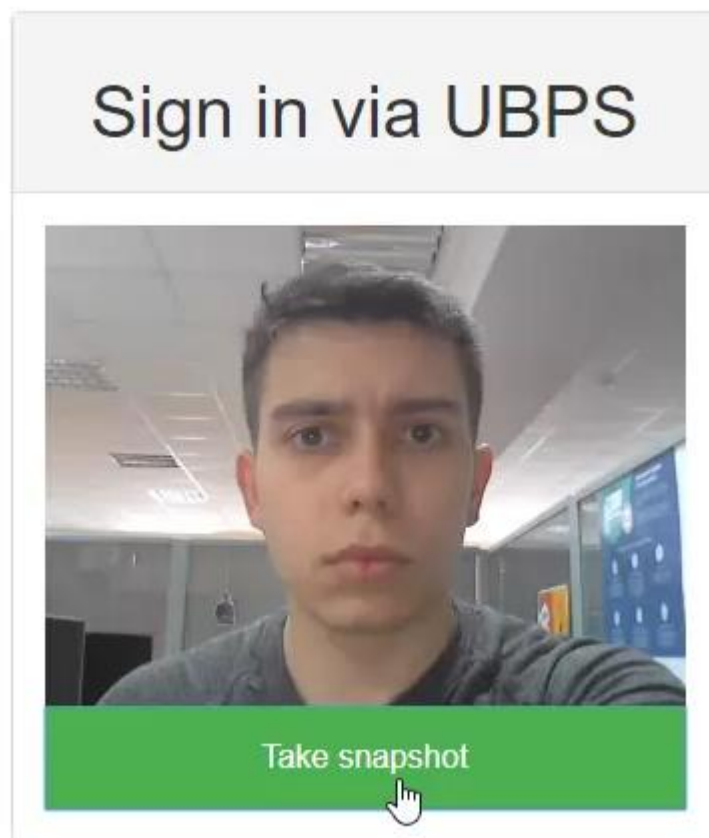


Рисунок 5 - Пример выполнения авторизации по лицу

Если пользователь с такими биометрическими данными будет найден, то обратно вернется ответ, содержащий токен для следующего запроса, веб-сайт уведомит пользователя об успешной авторизации по лицу, а форма с изображением с веб-камеры заменится на форму для предоставления голоса пользователя. Рисунок 6 содержит форму авторизации по голосу и уведомление об успешной авторизации по лицу.

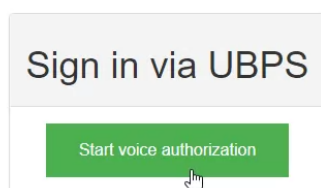


Рисунок 6 - Форма авторизации по голосу и уведомление об успешной авторизации по лицу

После полной авторизации пользователь получит доступ к личному кабинету пользователя, где сможет переводить другим пользователям системы виртуальные денежные средства. Рисунок 7 содержит снимок личного кабинета пользователя.

DEMO

LOGOUT

YOUR ACCOUNT NUMBER: 6

AMMOUNT: 8876.0

Transfer money:

Recipient account:

Amount:

TRANSFER

Deposit to account:

Amount:

DEPOSIT

Transaction ID	Sender	Recipient	Amount	Date	Status Code	Status Message
60	6	6	-17.0	2017-10-17 11:41:25.744	FAIL	Transaction aborted. Wrong amount value
59	6	6	-4.0	2017-10-17 11:41:21.648	FAIL	Transaction aborted. Wrong amount value
58	6	6	-5.0	2017-10-17 11:41:16.024	FAIL	Transaction aborted. Wrong amount value
57	6	6	-24.0	2017-10-17 11:36:33.811	OK	Transaction completed. Money added
56	6	1	100.0	2017-10-17 11:23:17.247	OK	Transaction completed.
52	6	1	-5000.0	2017-10-16 15:27:36.519	FAIL	Transaction aborted. Wrong amount value
51	6	1	-1.0	2017-10-16 15:24:15.243	FAIL	Transaction aborted. Wrong amount value
50	6	1	0.0	2017-10-16 15:17:58.405	OK	Transaction completed.
48	6	---	1.0	2017-10-16 15:14:58.468	FAIL	Transaction aborted. No recipient with specified account number
41	6	---	10.0	2017-10-16 15:08:38.442	FAIL	Transaction aborted. No recipient with specified account number
40	6	1	1000.0	2017-10-16 15:07:52.463	OK	Transaction completed.

Рисунок 7 - Личный кабинет пользователя

Для перевода виртуальных денежных средств другим пользователям системы необходимо выполнить процедуру, аналогичную авторизации на сайте.

ЗАКЛЮЧЕНИЕ

В процессе работы были определены и сформулированы требования к решению поставленной задачи, методика решения и инструменты реализации.

На языке Java с применением фреймворка Spring Framework был разработан сервис авторизации пользователей по биометрическим данным.

Были изучены особенности создания сервисов с применением фреймворка Spring Framework, особенности создания структуры модели базы данных при помощи классов сущностей и методы аутентификации пользователей.

На основе спроектированной базы данных, ее структура была реализованная программным способом. Были разработаны модули регистрации и авторизации пользователей, а также модуль загрузки, сохранения и изменения биометрических данных пользователей

Разработанная система является универсальной и может использоваться в любой сфере жизнедеятельности человека, где необходима аутентификация пользователей.

Выполненная работа полностью соответствует полученному заданию и может быть внедрена в уже работающие системы.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Что такое технология Java и каково ее применение?
URL: https://www.java.com/ru/download/faq/whatis_java.xml
2. В чем разница между JRE и JDK?
URL: <http://qaru.site/questions/11544/what-is-the-difference-between-jdk-and-jre>
3. Интегрированная среда разработки. URL:
https://ru.wikipedia.org/wiki/Интегрированная_среда_разработки
4. Edition comparasion matrix.
URL: https://www.jetbrains.com/idea/features/editions_comparison_matrix.html
5. Spring Framework.
URL: https://ru.wikipedia.org/wiki/Spring_Framework
6. Как писать на Spring в 2017. URL: <https://habr.com/post/333756/>
7. Spring Framework Annotations.
URL: <https://springframework.guru/spring-framework-annotations/>
8. Инициализация Spring: XML, аннотации, Java конфигурация и ... Groovy. URL: <https://easyjava.ru/spring/spring-framework/inicializaciya-spring-java-konfiguraciya-annotacii-xml-i-groovy/>
9. Model-View-Controller. URL: <https://ru.wikipedia.org/wiki/Model-View-Controller>
10. База данных. URL: <https://hostiq.ua/wiki/database/>
11. Чем PostgreSQL лучше других SQL баз данных с открытым исходным кодом. URL: <https://habr.com/post/282764/>
12. Что такое PostgreSQL.
URL: http://www.sai.msu.su/~megera/postgres/talks/what_is_postgresql.html

13. Database Connection Pool. URL: <https://habr.com/post/101342/>
14. HikariCP – самый быстрый пул соединений на java.
URL: <https://habr.com/post/269023/>
15. Сервлет (Java). URL: [https://ru.wikipedia.org/wiki/Сервлет_\(Java\)](https://ru.wikipedia.org/wiki/Сервлет_(Java))
16. Контейнер сервлетов.
URL: https://ru.wikipedia.org/wiki/Контейнер_сервлетов
17. Apache Tomcat. URL: https://ru.wikipedia.org/wiki/Apache_Tomcat
18. Система управления версиями.
URL: https://ru.wikipedia.org/wiki/Система_управления_версиями
19. Subversion. URL: <https://ru.wikipedia.org/wiki/Subversion>
20. Автоматизация сборки.
URL: https://ru.wikipedia.org/wiki/Автоматизация_сборки
21. Apache Maven. URL: https://ru.wikipedia.org/wiki/Apache_Maven
22. Основные преимущества Maven. URL: <http://www.apache-maven.ru/>
23. Биометрия. URL: <https://ru.wikipedia.org/wiki/Биометрия>
24. Биометрические системы аутентификации.
URL: http://ru.wikipedia.org/wiki/Биометрические_системы_аутентификации
25. Что такое биометрия. URL: <http://www.biometria.sk/ru/what-is-biometrics.html>
26. ReCAPTCHA. URL: <https://ru.wikipedia.org/wiki/ReCAPTCHA>
27. Registration with Spring – Integrate reCAPTCHA.
URL: <http://www.baeldung.com/spring-security-registration-captcha>
28. Руководство по Hibernate. Аннотации.
URL: <https://proselyte.net/tutorials/hibernate-tutorial/annotations/>

ПРИЛОЖЕНИЕ А

ЛИСТИНГИ

(обязательное)

Приложение А.1

Листинги классов сущностей

Приложение А.1.1

Файл «AuthUser.java»

```
package ru.programsolutions.ubps.entity.auth;
import javax.persistence.*;

/**
 * Класс информации пользователя. Содержит такие данные как:
 * <ul>фцуа
 * <li>{@link #id идентификатор пользователя}</li>
 * <li>{@link #role роль пользователя}</li>
 * <li>{@link #status статус пользователя}</li>
 * <li>{@link #blocked флаг блокировки
пользователя}</li>
 * <li>{@link #email электронный почтовый адрес
пользователя}</li>
 * <li>{@link #phone контактный телефон
пользователя}</li>
 * </ul>
 */
@Entity(name = AuthUser.Database.ENTITY_NAME)
@Table(name = AuthUser.Database.TABLE_NAME)
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name =
AuthUser.Database.COLUMN_ROLE_NAME, discriminatorType =
DiscriminatorType.STRING)
public abstract class AuthUser
{
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Enumerated(value = EnumType.STRING)
```

```

        @Column(name = Database.COLUMN_ROLE_NAME, nullable =
false, insertable = false, updatable = false)
        private Role role = Role.CLIENT;

        @Enumerated(value = EnumType.STRING)
        @Column(name = Database.COLUMN_STATUS_NAME, nullable =
false, unique = false)
        private Status status = Status.NEW_USER;

        @Column(name = Database.COLUMN_BLOCKED_NAME, nullable =
false)
        private Boolean blocked = Boolean.TRUE;

        @Column(name = Database.COLUMN_EMAIL_NAME, nullable =
false, unique = true)
        private String email;

        @Column(name = Database.COLUMN_PHONE_NAME, nullable =
true, unique = true)
        private String phone;

        @Embedded
        private PersonInfo personInfo = new PersonInfo();

        public AuthUser()
        {
        }

        public Long getId()
        {
            return id;
        }

        public void setId(Long id)
        {
            this.id = id;
        }

        public Role getRole()
        {
            return role;
        }

        public void setRole(Role role)
        {
            this.role = role;
        }

        public Status getStatus()
        {
            return status;
        }

        public void setStatus(Status status)
        {
            this.status = status;
        }
    }

```

```

    public Boolean getBlocked()
    {
        return blocked;
    }
    public void setBlocked(Boolean blocked)
    {
        this.blocked = blocked;
    }
    public String getEmail()
    {
        return email;
    }
    public void setEmail(String email)
    {
        this.email = email;
    }
    public String getPhone()
    {
        return phone;
    }
    public void setPhone(String phone)
    {
        this.phone = phone;
    }
    public PersonInfo getPersonInfo()
    {
        return personInfo;
    }
    public void setPersonInfo(PersonInfo personInfo)
    {
        this.personInfo = personInfo;
    }

    public static class Database{
        public static final String ENTITY_NAME = "AuthUser";
        public static final String TABLE_NAME = "auth_user";

        public static final String COLUMN_ID_NAME = "id";
        public static final String COLUMN_ROLE_NAME =
"role";
        public static final String COLUMN_STATUS_NAME =
"status";
        public static final String COLUMN_BLOCKED_NAME =
"blocked";
        public static final String COLUMN_EMAIL_NAME =
"email";
        public static final String COLUMN_PHONE_NAME =
"phone";

        public static final String JOIN_COLUMN_AUTH_USER_ID
= "auth_user_id";
    }

```

```
}
```

Приложение А.1.2

Файл «PersonInfo.java»

```
package ru.programsolutions.ubps.entity.auth;
import javax.annotation.Nullable;
import javax.persistence.Column;
import javax.persistence.Embeddable;
import javax.validation.constraints.NotNull;
import java.util.StringJoiner;

/**
 * Встраиваемый класс с личными данными пользователей
 * Содержит следующие поля:
 * <ul>
 * <li>{@link #firstName имя пользователя}</li>
 * <li>{@link #lastName фамилия пользователя}</li>
 * <li>{@link #patronymic отчество пользователя}</li>
 * </ul>
 */
@Embeddable
public class PersonInfo
{
    @Column(name = Database.COLUMN_FIRST_NAME_NAME, nullable
= false, unique = false, length = 32)
    private String firstName;

    @Column(name = Database.COLUMN_LAST_NAME_NAME, nullable
= false, unique = false, length = 32)
    private String lastName;

    @Column(name = Database.COLUMN_PATRONYMIC_NAME, nullable
= true, unique = false, length = 32)
    private String patronymic = "";

    public PersonInfo()
    {
    }

    public PersonInfo(@NotNull String firstName, @NotNull
String lastName)
    {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

```

    public PersonInfo(@NotNull String firstName, @NotNull
String lastName, @Nullable String patronymic)
    {
        this(firstName, lastName);
        if (patronymic != null)
        {
            this.patronymic = patronymic;
        }
    }

    public String getFirstName()
    {
        return firstName;
    }
    public void setFirstName(String firstName)
    {
        this.firstName = firstName;
    }
    public String getLastName()
    {
        return lastName;
    }
    public void setLastName(String lastName)
    {
        this.lastName = lastName;
    }
    public String getPatronymic()
    {
        return patronymic;
    }
    public void setPatronymic(String patronymic)
    {
        if (patronymic == null)
        {
            this.patronymic = "";
        }
        else
        {
            this.patronymic = patronymic;
        }
    }

    public String getFullName()
    {
        StringJoiner joiner = new StringJoiner(" ");
        joiner.add(this.lastName).add(this.firstName);
        if (!this.patronymic.isEmpty())
        {
            joiner.add(this.patronymic);
        }
        return joiner.toString();
    }

```

```

        public static class Database{
            public static final String COLUMN_FIRST_NAME_NAME =
"first_name";
            public static final String COLUMN_LAST_NAME_NAME =
"last_name";
            public static final String COLUMN_PATRONYMIC_NAME =
"patronymic";
        }
    }
}

```

Приложение А.1.3

Файл «SimpleAuthUser.java»

```

package ru.programsolutions.ubps.entity.auth;

import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAut
hority;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

/**
 * Содержит данные авторизации пользователей
 */
@Entity(name = SimpleAuthUser.Database.ENTITY_NAME)
@Table(name = SimpleAuthUser.Database.TABLE_NAME)
public class SimpleAuthUser implements RegistrationAuth,
UserDetails {
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne
    @JoinColumn(name =
AuthUser.Database.JOIN_COLUMN_AUTH_USER_ID,
referencedColumnName = AuthUser.Database.COLUMN_ID_NAME)
    private AuthUser authUser;
}

```

```

        @Column(name = Database.COLUMN_PASSWORD_NAME, nullable =
false, unique = false, length = 128)
        private String password;

        public SimpleAuthUser() {
        }

        public AuthUser getAuthUser() {
            return authUser;
        }

        public Long getId() {
            return id;
        }

        public void setId(Long id) {
            this.id = id;
        }

        public void setAuthUser(AuthUser authUser) {
            this.authUser = authUser;
        }

        public void setPassword(String password) {
            this.password = password;
        }

        @Override
        public Collection<? extends GrantedAuthority>
getAuthorities() {
            SimpleGrantedAuthority sia = new
SimpleGrantedAuthority(this.authUser.getRole().toString());
            List<SimpleGrantedAuthority> authList = new
ArrayList<>();
            authList.add(sia);
            return authList;
        }

        @Override
        public String getPassword() {
            return this.password;
        }

        @Override
        public String getUsername() {
            return this.authUser.getEmail();
        }

        @Override
        public boolean isAccountNonExpired() {
            return true;
        }

```

```

    }

    @Override
    public boolean isAccountNonLocked() {
        return !this.authUser.getBlocked();
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return !this.authUser.getBlocked();
    }

    @Override
    public AuthUser getRegistrationAuth() {
        return this.authUser;
    }

    public static class Database {
        public static final String ENTITY_NAME =
"SimpleAuthUser";
        public static final String TABLE_NAME =
"simple_auth_user";

        public static final String COLUMN_ID_NAME = "id";
        public static final String COLUMN_PASSWORD_NAME =
"password";
    }
}

```

Приложение A.1.4

Файл «Enroll.java»

```

package ru.programsolutions.ubps.entity.enroll;

import
com.fasterxml.jackson.annotation.JsonManagedReference;
import ru.programsolutions.ubps.entity.Client;
import ru.programsolutions.ubps.entity.auth.AuthUser;

import javax.persistence.*;
import java.io.File;

@Entity(name = Enroll.Database.ENTITY_NAME)

```

```

@Table(name = Enroll.Database.TABLE_NAME)
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name =
Enroll.Database.COLUMN_ENROLL_TYPE_NAME, discriminatorType =
DiscriminatorType.STRING)
public class Enroll {
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    protected Long id;

    @JsonManagedReference
    @ManyToOne
    @JoinColumn(name =
Client.Database.JOIN_COLUMN_CLIENT_ID_NAME,
referencedColumnName = AuthUser.Database.COLUMN_ID_NAME,
nullable = false, updatable = false)
    protected Client client;

    @Enumerated(value = EnumType.STRING)
    @Column(name = Database.COLUMN_ENROLL_TYPE_NAME,
insertable = false, updatable = false)
    protected EnrollType type;

    @Column(name = Database.COLUMN_PATH_NAME, nullable =
false, unique = true, length = 512)
    protected String path;

    public Enroll() {
    }

    public Enroll(Client client, EnrollType type, String
path) {
        this.client = client;
        this.type = type;
        this.path = path;
    }

    @PostRemove
    private void removeFile() {
        File file = new File(this.path);
        boolean exists = false;
        try {
            exists = file.exists();
        } catch (SecurityException e) {
            System.err.println("SecurityException on file ["
+ file.getAbsolutePath() + "]; Read-access denied. Delete
file manually if exists");
        }
        if (exists) {
            try {
                file.delete();
            }

```

```

        } catch (SecurityException e) {
            System.err.println("SecurityException on
file [" + file.getAbsolutePath() + "]; Delete-access denied.
Delete it manually");
        }
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Client getClient() {
        return client;
    }

    public void setClient(Client client) {
        this.client = client;
    }

    public EnrollType getType() {
        return type;
    }

    public void setType(EnrollType type) {
        this.type = type;
    }

    public String getPath() {
        return path;
    }

    public void setPath(String path) {
        this.path = path;
    }

    public static class Database{
        public static final String ENTITY_NAME = "Enroll";
        public static final String TABLE_NAME = "enroll";

        public static final String COLUMN_ID_NAME = "id";
        public static final String COLUMN_ENROLL_TYPE_NAME =
"type";
        public static final String COLUMN_PATH_NAME =
"path";
    }
}

```

Приложение А.1.5

Файл «EnrollFace.java»

```
package ru.programsolutions.ubps.entity.enroll;

import javax.persistence.*;

@Entity(name = EnrollFace.Database.ENTITY_NAME)
@Table(name = EnrollFace.Database.TABLE_NAME)
@DiscriminatorValue(EnrollFace.Database.DISCIMINATOR)
public class EnrollFace extends Enroll {
    @OneToOne(targetEntity = FaceFeature.class, cascade =
CascadeType.PERSIST)
    @JoinColumn(name =
FaceFeature.Database.JOIN_COLUMN_ID_NAME,
referencedColumnName = FaceFeature.Database.COLUMN_ID_NAME)
    private FaceFeature faceFeature;

    public EnrollFace() {
        this.type = EnrollType.FACE;
    }

    public FaceFeature getFaceFeature() {
        return faceFeature;
    }

    public void setFaceFeature(FaceFeature faceFeature) {
        this.faceFeature = faceFeature;
    }

    public static class Database {
        public static final String ENTITY_NAME =
"EnrollFace";
        public static final String TABLE_NAME =
"enroll_face";
        public static final String DISCRIMINATOR = "FACE";
    }
}
```

Приложение А.1.6

Файл «EnrollFinger.java»

```
package ru.programsolutions.ubps.entity.enroll;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity(name = EnrollFinger.Database.ENTITY_NAME)
```

```

@Table(name = EnrollFinger.Database.TABLE_NAME)
@DiscriminatorValue(value =
EnrollFinger.Database.DISCIMINATOR)
public class EnrollFinger extends Enroll {
    public EnrollFinger() {
    }

    public static class Database{
        public static final String ENTITY_NAME =
"EnrollFinger";
        public static final String TABLE_NAME =
"enroll_finger";
        public static final String DISCRIMINATOR = "FINGER";
    }
}

```

Приложение A.1.7

Файл «EnrollVoice.java»

```

package ru.programsolutions.ubps.entity.enroll;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity(name = EnrollVoice.Database.ENTITY_NAME)
@Table(name = EnrollVoice.Database.TABLE_NAME)
@DiscriminatorValue(EnrollVoice.Database.DISCIMINATOR)
public class EnrollVoice extends Enroll{

    public EnrollVoice() {
    }

    public static class Database{
        public static final String ENTITY_NAME =
"EnrollVoice";
        public static final String TABLE_NAME =
"enroll_voice";
        public static final String DISCRIMINATOR = "VOICE";
    }
}

```

Приложение A.1.8

Файл «FaceFeature.java»

```

package ru.programsolutions.ubps.entity.enroll;

```

```

import javax.persistence.*;

@Entity(name = FaceFeature.Database.ENTITY_NAME)
@Table(name = FaceFeature.Database.TABLE_NAME)
public class FaceFeature {
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = Database.COLUMN_AVERAGE_AGE_NAME)
    private Integer averageAge;

    @Enumerated(value = EnumType.STRING)
    @Column(name = Database.COLUMN_GENDER_NAME)
    private Gender gender;

    public FaceFeature() {
    }

    public FaceFeature(Integer averageAge, Gender gender) {
        this.averageAge = averageAge;
        this.gender = gender;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Integer getAverageAge() {
        return averageAge;
    }

    public void setAverageAge(Integer averageAge) {
        this.averageAge = averageAge;
    }

    public Gender getGender() {
        return gender;
    }

    public void setGender(Gender gender) {
        this.gender = gender;
    }

    public static class Database {
        public static final String ENTITY_NAME =
"FaceFeature";

```

```

        public static final String TABLE_NAME =
"face_feature";

        public static final String COLUMN_ID_NAME = "id";
        public static final String COLUMN_AVERAGE_AGE_NAME =
"average_age";
        public static final String COLUMN_GENDER_NAME =
"gender";

        public static final String JOIN_COLUMN_ID_NAME =
TABLE_NAME + "_" + COLUMN_ID_NAME;
    }
}

```

Приложение A.1.9

Файл «AuthorizedDevice.java»

```

package ru.programsolutions.ubps.entity;

import ru.programsolutions.ubps.entity.auth.AuthUser;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

/**
 * Информация об авторизованных устройствах пользователей
 */
@Entity(name = AuthorizedDevice.Database.ENTITY_NAME)
@Table(name = AuthorizedDevice.Database.TABLE_NAME)
public class AuthorizedDevice {
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = Database.COLUMN_DEVICE_IDENTITY_NAME,
unique = true, length = 255)
    private String deviceId;

    @ManyToMany(targetEntity = Client.class)
    @JoinTable(name =
Client.Database.JOIN_TABLE_CLIENT_AUTHORIZED_DEVICE_NAME,
        joinColumns = @JoinColumn(name =
Database.JOIN_COLUMN_AUTHORIZED_DEVICE_ID_NAME,
referencedColumnName = Database.COLUMN_ID_NAME),
        inverseJoinColumns = @JoinColumn(name =
Client.Database.JOIN_COLUMN_CLIENT_ID_NAME,
referencedColumnName = AuthUser.Database.COLUMN_ID_NAME))

```

```

private List<Client> clients = new ArrayList<Client>();

public AuthorizedDevice() {
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getDeviceId() {
    return deviceId;
}

public void setDeviceId(String deviceId) {
    this.deviceId = deviceId;
}

public List<Client> getClients() {
    return clients;
}

public void setClients(List<Client> clients) {
    this.clients = clients;
}

public static class Database {
    public static final String ENTITY_NAME =
"AuthorizedDevice";
    public static final String TABLE_NAME =
"authorized_device";
    public static final String COLUMN_ID_NAME = "id";
    public static final String
COLUMN_DEVICE_IDENTITY_NAME = "device_identity";
    public static final String
JOIN_COLUMN_AUTHORIZED_DEVICE_ID_NAME = TABLE_NAME + "_" +
COLUMN_ID_NAME;
}
}

```

Приложение А.1.10

Файл «Client.java»

```

package ru.programsolutions.ubps.entity;

import ru.programsolutions.ubps.entity.auth.AuthUser;

```

```

import ru.programsolutions.ubps.entity.auth.Role;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

/**
 * Класс с информацией по пользователю типа
<code>КЛИЕНТ</code>
 */
@Entity(name = Client.Database.ENTITY_NAME)
@Table(name = Client.Database.TABLE_NAME)
@DiscriminatorValue(value = Client.Database.DISCriminator)
public class Client extends AuthUser {
    @ManyToMany(targetEntity = AuthorizedDevice.class,
        cascade = CascadeType.PERSIST)
    @JoinTable(name =
        Database.JOIN_TABLE_CLIENT_AUTHORIZED_DEVICE_NAME,
        joinColumns = @JoinColumn(name =
            Database.JOIN_COLUMN_CLIENT_ID_NAME, referencedColumnName =
                AuthUser.Database.COLUMN_ID_NAME),
        inverseJoinColumns = @JoinColumn(name =
            AuthorizedDevice.Database.JOIN_COLUMN_AUTHORIZED_DEVICE_ID_N
                AME, referencedColumnName =
                    AuthorizedDevice.Database.COLUMN_ID_NAME))
    private List<AuthorizedDevice> authorizedDevices = new
        ArrayList<>();

    public Client() {
    }

    public List<AuthorizedDevice> getAuthorizedDevices() {
        return authorizedDevices;
    }

    public void setAuthorizedDevices(List<AuthorizedDevice>
        authorizedDevices) {
        this.authorizedDevices = authorizedDevices;
    }

    public static class Database {
        public static final String ENTITY_NAME = "Client";
        public static final String TABLE_NAME = "client";

        public static final String DISCRIMINATOR = "CLIENT";
        public static final String
            JOIN_TABLE_CLIENT_AUTHORIZED_DEVICE_NAME =
                "client_authorized_device";
        public static final String
            JOIN_COLUMN_CLIENT_ID_NAME = "client_id";
    }
}

```

Приложение А.1.11

Файл «UserHash.java»

```
package ru.programsolutions.ubps.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;
import org.apache.tomcat.util.codec.binary.Base64;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import ru.programsolutions.ubps.entity.auth.AuthUser;

import javax.persistence.*;
import java.util.Date;
import java.util.UUID;

/**
 * Хранит данные для аутентификации пользователя на
 * сторонних ресурсах
 */
@Entity(name = UserHash.Database.ENTITY_NAME)
@Table(name = UserHash.Database.TABLE_NAME)
public class UserHash {
    @JsonIgnore
    @Id
    @Column(name = Database.COLUMN_ID_NAME)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne
    @JoinColumn(name =
AuthUser.Database.JOIN_COLUMN_AUTH_USER_ID, referencedColumnName =
AuthUser.Database.COLUMN_ID_NAME)
    private AuthUser authUser;

    @Column(name = Database.COLUMN_HASH_NAME, nullable =
false)
    private String hash;

    @JsonIgnore
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = Database.COLUMN_LAST_MODIFIED_NAME,
nullable = false)
    private Date lastModified = new Date();

    @JsonIgnore
    @Column(name = Database.COLUMN_EXPIRED_NAME, nullable =
false)
    private Boolean expired = Boolean.FALSE;
```

```

        @JsonIgnore
        @Column(name = Database.COLUMN_DEVICE_ID_NAME, nullable
= false)
        private String deviceId;

        @JsonIgnore
        @Column(name = Database.COLUMN_AUTH_HASH_NAME)
        private String authHash;

        @JsonIgnore
        @Column(name = Database.COLUMN_AUTH_SECRET_NAME)
        private String authSecret;

        public UserHash() {
        }

        public String createAndStoreOneUseAuthHash() {
            authSecret = UUID.randomUUID().toString();
            BCryptPasswordEncoder bCryptPasswordEncoder = new
BCryptPasswordEncoder(12);
            String encode = bCryptPasswordEncoder.encode(hash +
authSecret);
            authHash = encode;
            return Base64.encodeBase64String(encode.getBytes());
        }

        public UserHash(AuthUser authUser, String deviceId) {
            this.authUser = authUser;
            this.hash = UUID.randomUUID().toString();
            this.deviceId = deviceId;
        }

        public UserHash(AuthUser authUser, String deviceId,
String hash) {
            this.authUser = authUser;
            this.hash = hash;
            this.deviceId = deviceId;
        }

        public Long getId() {
            return id;
        }

        public void setId(Long id) {
            this.id = id;
        }

        public AuthUser getAuthUser() {
            return authUser;
        }
    }

```

```

public void setAuthUser(AuthUser authUser) {
    this.authUser = authUser;
}

public String getHash() {
    return hash;
}

public void setHash(String hash) {
    this.hash = hash;
}

public Date getLastModified() {
    return lastModified;
}

public void setLastModified(Date lastModified) {
    this.lastModified = lastModified;
}

public Boolean isExpired() {
    return expired;
}

public void setExpired(Boolean expired) {
    this.expired = expired;
}

public String getDeviceId() {
    return deviceId;
}

public void setDeviceId(String deviceId) {
    this.deviceId = deviceId;
}

@PrePersist
@PreUpdate
private void refreshLastModified() {
    this.lastModified = new Date();
}

public void refreshHash() {
    this.hash = UUID.randomUUID().toString();
    this.expired = false;
    this.authHash = "";
    authSecret = "";
}

public void dropSession() {
    this.hash = "";
    this.expired = true;
}

```

```

        this.authHash = "";
        authSecret = "";
    }

    public boolean checkHash(String authHash) {
        BCryptPasswordEncoder bCryptPasswordEncoder = new
        BCryptPasswordEncoder(12);
        boolean matches = bCryptPasswordEncoder.matches(hash
+ authSecret, authHash);
        return matches;
    }

    public static class Database{
        public static final String ENTITY_NAME = "UserHash";
        public static final String TABLE_NAME = "user_hash";

        public static final String COLUMN_ID_NAME = "id";
        public static final String COLUMN_HASH_NAME =
"hash";
        public static final String COLUMN_LAST_MODIFIED_NAME
= "last_modified";
        public static final String COLUMN_EXPIRED_NAME =
"expired";
        public static final String COLUMN_DEVICE_ID_NAME =
"device_id";
        public static final String COLUMN_AUTH_HASH_NAME =
"auth_hash";
        public static final String COLUMN_AUTH_SECRET_NAME =
"auth_secret";
    }
}

```

Приложение A.2

Листинги контроллеров

Приложение A.2.1

Файл «AuthController.java»

```

package ru.programsolutions.ubps.controller;

import com.oep.faceroll.lib.ApiCalls;
import org.apache.commons.codec.binary.Base64;
import
org.springframework.beans.factory.annotation.Autowired;

```

```

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;
import ru.programsolutions.ubps.config.SecurityConfig;
import ru.programsolutions.ubps.entity.AuthorizedDevice;
import ru.programsolutions.ubps.entity.Client;
import ru.programsolutions.ubps.entity.UserHash;
import ru.programsolutions.ubps.entity.auth.AuthUser;
import ru.programsolutions.ubps.entity.auth.Role;
import ru.programsolutions.ubps.entity.auth.SimpleAuthUser;
import ru.programsolutions.ubps.entity.auth.Status;
import ru.programsolutions.ubps.entity.enroll.Enroll;
import ru.programsolutions.ubps.entity.enroll.EnrollType;
import ru.programsolutions.ubps.entity.enroll.FaceFeature;
import ru.programsolutions.ubps.entity.enroll.Gender;
import ru.programsolutions.ubps.property.VerifyProperty;
import ru.programsolutions.ubps.repository.*;
import
ru.programsolutions.ubps.service.AuthUserDetailsService;
import
ru.programsolutions.ubps.service.BcryptEncoderService;
import
ru.programsolutions.ubps.service.currentuser.CurrentUser;
import
ru.programsolutions.ubps.service.search.FaceAuthService;
import
ru.programsolutions.ubps.service.search.FaceSearchResult;
import ru.programsolutions.ubps.storage.FileStorageService;
import ru.programsolutions.ubps.util.DeviceResponse;
import ru.programsolutions.ubps.util.MultipartFileImpl;

import javax.servlet.ServletException;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.InputStream;
import java.util.List;

/**
 * Created by Tensky on 25.07.2017.
 */
@Controller

```

```

public class AuthController {
    @Autowired
    UserHashRepository hashRepository;

    @Autowired
    AuthUserDetailsService authUserDetailsService;

    @Autowired
    private BCryptPasswordEncoder encoderService;

    @Autowired
    SecurityConfig.MD5Encoder md5Encoder;

    @Autowired
    private AuthUserRepository authUserRepository;

    @Autowired
    private SimpleAuthUserRepository
simpleAuthUserRepository;

    @Autowired
    private FileStorageService storageService;

    @Autowired
    private EnrollRepository enrollRepository;

    @Autowired
    private VerifyProperty verifyProperty;

    @Autowired
    private CurrentUser currentUser;

    @Autowired
    private ClientRepository clientRepository;

    @Autowired
    private AuthorizedDeviceRepository
authorizedDeviceRepository;

    @Autowired
    private EnrollFaceRepository enrollFaceRepository;

    @Autowired
    private FaceAuthService faceAuthService;

    @Autowired
    private DeviceResponse deviceResponse;

    @RequestMapping(value = Path.GET_IP, method =
RequestMethod.GET)
    @ResponseBody
    public String getIp(ServletRequest servletRequest) {

```

```

        return servletRequest.getRemoteAddr();
    }

    @RequestMapping(value = AuthController.Path.GET_HASH,
        method = {RequestMethod.GET, RequestMethod.POST})
    @ResponseBody
    public ResponseEntity getHash(@RequestParam(name =
        Param.LOGIN, required = true) String login,
        @RequestParam(name =
        Param.PASSWORD, required = true) String password,
        @RequestParam(name =
        Param.DEVICE_ID, required = true) String deviceId) {
        SimpleAuthUser simpleAuthUser = null;
        try {
            //Получаем SimpleAuthUser для доступа к AuthUser
            //и хешированному паролю пользователя
            simpleAuthUser = (SimpleAuthUser)
            authUserDetailsService.loadUserByUsername(login);
        } catch (UsernameNotFoundException e) {
            //Пользователь не найден
            return
            ResponseEntity.status(HttpStatus.FORBIDDEN).body(null);
        }

        if
        (!encoderService.getPasswordEncoder().matches(password,
            simpleAuthUser.getPassword())) {
            //Пароль не верный
            return
            ResponseEntity.status(HttpStatus.FORBIDDEN).body(null);
        }

        AuthUser authUser =
        simpleAuthUser.getRegistrationAuth();

        //Если пользователь не подтвердил почту
        if (authUser.getStatus().equals(Status.NEW_USER) ||
            authUser.getStatus().equals(Status.BANNED_USER)) {
            return
            ResponseEntity.status(HttpStatus.FORBIDDEN).body(null);
        }

        //Получение "сессии"
        UserHash userHash =
        hashRepository.findByUserAndDeviceId(authUser, deviceId);

        //Если ее нет, то создаем
        if (userHash == null) {
            userHash = new UserHash(authUser, deviceId);
        }
    }

```

```

        userHash.refreshHash();

        //Сохраняем изменения в бд
        userHash = hashRepository.save(userHash);
        if (authUser.getRole().equals(Role.CLIENT)) {
            Client client =
clientRepository.findOne(authUser.getId());
            AuthorizedDevice authDevice =
authorizedDeviceRepository.findByDeviceIdentity(deviceId);
            if (authDevice == null) {
                authDevice = new AuthorizedDevice();
                authDevice.getClients().add(client);
                authDevice.setDeviceId(deviceId);
                authDevice =
authorizedDeviceRepository.save(authDevice);
            }
            List<Client> clients = authDevice.getClients();
            if (!clients.contains(client)) {
                clients.add(client);
                authorizedDeviceRepository.save(authDevice);
            }
        }
        //Отправляем пользователю
        DeviceResponse authenticated =
deviceResponse.createAuthenticated(userHash.getHash(),
userHash.getAuthUser(), userHash.getDeviceId());
        return ResponseEntity.ok(authenticated);
    }

    public DeviceResponse isHashValid(Long authId, String
hash, String deviceId) {
        AuthUser authUser =
authUserRepository.findOne(authId);

        Status status = authUser.getStatus();
        if (status.equals(Status.NEW_USER) ||
status.equals(Status.BANNED_USER))
            return deviceResponse.createNotAuthenticated();
        UserHash userHash = hashRepository.find(authId,
hash, deviceId);
        if (userHash == null) {
            return deviceResponse.createNotAuthenticated();
        }
        if (userHash.isExpired()) {
            return deviceResponse.createNotAuthenticated();
        }
        return
deviceResponse.createAuthenticated(userHash.getHash(),
userHash.getAuthUser(), userHash.getDeviceId());
    }

```

```

        @RequestMapping(value = Path.VERIFY_BY_APP, method =
RequestMethod.POST)
        @ResponseBody
        public ResponseEntity verify(@RequestParam(name =
Param.AUTH_ID) Long authId,
                                     @RequestParam(name =
Param.DEVICE_ID) String deviceId,
                                     @RequestParam(name =
Param.LOGIN_TOKEN) String token,
                                     @RequestParam(name =
Param.ENROLL, required = true) MultipartFile multipartFile,
                                     @RequestParam(name =
Param.ENROLL_TYPE, required = true) String enrollType) {
            DeviceResponse deviceResponse = isHashValid(authId,
token, deviceId);
            if (!deviceResponse.isAuthenticated())
                return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Not
authorized");

            try {
                ByteArrayOutputStream baos = new
ByteArrayOutputStream();
                int read;
                InputStream is = multipartFile.getInputStream();
                System.out.println("Voice mff: length =
"+multipartFile.getBytes().length + "; is.length =
"+is.available());
                while ((read= is.read())!=-1){
                    baos.write(read);
                }
                byte[] bytes = baos.toByteArray();
                bytes = Base64.decodeBase64(bytes);
                multipartFile = new
MultipartFileImpl(multipartFile.getName(),multipartFile.getO
riginalFilename(),bytes);
            } catch (Exception e) {
            }
            java.nio.file.Path submitted =
storageService.storeMultipartFile(multipartFile);

            EnrollType type = EnrollType.valueOf(enrollType);
            AuthUser authUser =
authUserRepository.findOne(authId);
            UserHash user =
hashRepository.findByUserAndDeviceId(authUser, deviceId);

```

```

        Enroll enrollDB = null;
        String generalized = null;
        try {
            enrollDB =
enrollRepository.findByAuthUser(authId, type);
            generalized = enrollDB.getPath();
        } catch (NullPointerException e) {
            return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Enroll not
found");
        }

        try {
            // FIXME: 02.02.2018 RETURN AUTH
//            int verify = 31011994;
            int verify;
            switch (type) {
                case FACE:
                    verify =
ApiCalls.verify(submitted.toString(), generalized);
                    break;
                case VOICE:
                    System.out.println("Generalized: " +
generalized);
                    System.out.println("Submitted: " +
submitted);
                    verify =
ApiCalls.verify_voice(submitted.toString(), generalized);
                    System.out.println("Score: " + verify);
                    break;
                case FINGER:
                    verify =
ApiCalls.verify_finger(submitted.toString(), generalized);
                    break;
                default:
                    verify = -1;
            }

            if (verify > 0) {
                String andStoreOneUseAuthHash =
user.createAndStoreOneUseAuthHash();
                hashRepository.save(user);
                return
ResponseEntity.ok(andStoreOneUseAuthHash);
            } else return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Not
verified");
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.FORBIDDEN).body(e.getMessag
e());
        } finally {

```

```

        File file = new
File(submitted.toAbsolutePath().toString());
        try {
            if (file.exists()) {
                file.delete();
            }
        } catch (SecurityException ignored) {
            System.err.println("Cannot delete file: " +
file.getAbsolutePath());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

@RequestMapping(value = Path.GET_HASH_BY_FACE, method =
RequestMethod.POST)
@ResponseBody
public ResponseEntity getHashByFace(@RequestParam(value
= Param.ENROLL) MultipartFile multipartFile,
                                     @RequestParam(value
= Param.DEVICE_ID) String deviceId,
                                     @RequestParam(value
= Param.AGE) Integer avgAge,
                                     @RequestParam(value
= Param.MALE) Boolean isMale,
                                     @RequestParam(name =
Param.ENCODED, defaultValue = "false", required = false)
Boolean encoded) {
    FaceFeature faceFeature = new FaceFeature(avgAge,
isMale ? Gender.MALE : Gender.FEMALE);
    java.nio.file.Path path;
    path =
storageService.storeMultipartFile(multipartFile);

    FaceSearchResult searchResult;
    try {
        searchResult = faceAuthService.search(deviceId,
faceFeature, path.toString());
    } catch (Exception e) {
        if
(e.getCause().getClass().getSimpleName().contains("NIOExcept
ion")) {
            return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Invalid
template format");
        }
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body
(e.getMessage());
    }
}

```

```

        if (!searchResult.isFound())
            return
        ResponseEntity.status(HttpStatus.FORBIDDEN).body("Not
        Authorized");

        Client client =
        clientRepository.findOne(searchResult.getClient().getId());

        if (!searchResult.isAuthorizedByDeviceId()) {
            AuthorizedDevice device =
            authorizedDeviceRepository.findByDeviceIdentity(deviceId);
            if (device == null) {
                device = new AuthorizedDevice();
                device.setDeviceId(deviceId);

                device.getClients().add(client);
                device =
                authorizedDeviceRepository.save(device);

                if
                (!client.getAuthorizedDevices().contains(device)) {

                    client.getAuthorizedDevices().add(device);
                    clientRepository.save(client);
                } else {
                    device.getClients().add(client);
                    device =
                    authorizedDeviceRepository.save(device);
                    if
                    (!client.getAuthorizedDevices().contains(device)) {

                        client.getAuthorizedDevices().add(device);
                        clientRepository.save(client);
                    }
                }
            }
        }

        try {
            storageService.removeDirectory(new
            File(path.toString()).getParentFile());
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Cannot remove parent
            directory of: " + path.toString());
        }

        UserHash userHash =
        hashRepository.findByUserAndDeviceId(client, deviceId);
        if (userHash == null) {
            userHash = new UserHash(client, deviceId);

```

```

        userHash.createAndStoreOneUseAuthHash();
        userHash = hashRepository.save(userHash);
    }

    userHash.refreshHash();
    userHash = hashRepository.save(userHash);

    DeviceResponse authenticated =
deviceResponse.createAuthenticated(userHash.getHash(),
userHash.getAuthUser(), userHash.getDeviceId());
    return ResponseEntity.ok(authenticated);
}

public class Param {
    public static final String LOGIN = "login";
    public static final String PASSWORD = "password";
    public static final String AUTH_ID = "auth_id";
    public static final String DEVICE_ID = "device_id";
    public static final String LOGIN_TOKEN =
"login_token";
    public static final String ENROLL_TYPE =
"enroll_type";
    public static final String DEBUG = "debug";
    public static final String ENROLL = "enroll";
    public static final String AGE = "age";
    public static final String MALE = "male";
    public static final String ENCODED = "encoded";
}

public class Path {
    public static final String CHECK_HASH = "/check-
hash";
    public static final String GET_HASH = "/get-hash";
    public static final String VERIFY_BY_APP =
"/verify";
    public static final String GET_HASH_BY_FACE = "/get-
hash-face";
    public static final String GET_IP = "/get-ip";
}
}

```

Приложение A.2.2

Файл «GeneralizeController.java»

```

package ru.programsolutions.ubps.controller;

import com.google.common.io.Files;
import com.oep.faceroll.lib.ApiCalls;
import com.oep.faceroll.lib.FaceFeatures;

```

```

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;
import ru.programsolutions.ubps.entity.Client;
import ru.programsolutions.ubps.entity.auth.Status;
import ru.programsolutions.ubps.entity.enroll.*;
import ru.programsolutions.ubps.repository.*;
import ru.programsolutions.ubps.storage.FileStorageService;
import ru.programsolutions.ubps.util.DeviceResponse;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.*;

@Controller
public class GeneralizeController {
    @Autowired
    private AuthController authController;

    @Autowired
    private FileStorageService storageService;

    @Autowired
    private EnrollRepository enrollRepository;

    @Autowired
    private ClientRepository clientRepository;

    @Autowired
    private EnrollFaceRepository enrollFaceRepository;

    @Autowired
    private EnrollVoiceRepository enrollVoiceRepository;

    @Autowired
    private EnrollFingerRepository enrollFingerRepository;

    @RequestMapping(value = Path.GENERALIZE, method =
RequestMethod.POST)
    @ResponseBody

```

```

        public ResponseEntity generalize(@RequestParam(name =
AuthController.Param.AUTH_ID) Long authId,
                                         @RequestParam(name =
AuthController.Param.LOGIN_TOKEN) String token,
                                         @RequestParam(name =
AuthController.Param.DEVICE_ID) String deviceId,
                                         @RequestParam(name =
AuthController.Param.ENROLL_TYPE) String et,
                                         @RequestParam(name =
AuthController.Param.ENROLL) List<MultipartFile>
multipartFiles,
                                         @RequestParam
Map<String, Object> requestMap) {
            EnrollType enrollType = EnrollType.valueOf(et);
            DeviceResponse response =
authController.isHashValid(authId, token, deviceId);
            if (!response.isAuthenticated())
                return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Not
authorized");

            Client client = clientRepository.findOne(authId);
            if (client == null) {
                return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Unknown
client");
            }

            List<java.nio.file.Path> filePaths =
storageService.storeMultipartFiles(multipartFiles);

            ArrayList<String> paths = new ArrayList<>();
            for (java.nio.file.Path filePath : filePaths) {
                paths.add(filePath.toString());
            }

            java.nio.file.Path parent =
storageService.dirByDate(Calendar.getInstance()).resolve(UUI
D.randomUUID().toString());
            new File(parent.toString()).mkdirs();

            File generalized = new
File(parent.resolve(enrollType.name() +
"_GENERAL").toString());
            try {
                generalized.createNewFile();
            } catch (IOException e) {
                for (java.nio.file.Path filePath : filePaths) {
                    storageService.removeDirectory(new
File(filePath.getParent().toString()));
                }
            }
        }
    }
}

```

```

        return
        ResponseEntity.status(HttpStatus.FORBIDDEN).body("Error
while saving user's authorization template");
    }

    FileOutputStream fos = null;
    Enroll enroll;
    try {
        fos = new FileOutputStream(generalized);
        switch (enrollType) {
            case FACE: {
                System.out.println("images count: "+
paths.size());
                System.out.println("multipartfiles
count: "+multipartFiles.size());
                for (String path : paths) {
                    System.out.println("image path:
"+path);
                }
                FaceFeatures faceFeatures =
ApiCalls.generalizeFace(paths, fos);
                if(faceFeatures == null){
                    return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Not
generalized");
                }
                int age = faceFeatures.getAge();
                Gender gender = faceFeatures.isMale() ?
Gender.MALE : Gender.FEMALE;
                FaceFeature faceFeature = new
FaceFeature(age, gender);
                enroll = saveGeneralFace(client,
generalized.getPath(), faceFeature);
                List<EnrollFace> enrolls =
enrollFaceRepository.findAllByClient(client);
                enrolls.remove(enroll);

                enrollRepository.delete(enrolls);

                if(client.getStatus().equals(Status.REGISTERED_USER))

                client.setStatus(Status.CONFIRMED_USER);

                client = clientRepository.save(client);
                break;
            }
            case VOICE: {
                Files.copy(new File(paths.get(0)), fos);
                enroll = saveGeneralVoice(client,
generalized.getPath());
                List<EnrollVoice> enrolls =
enrollVoiceRepository.findAllByClient(client);

```

```

        enrolls.remove(enroll);
        enrollRepository.delete(enrolls);

if(client.getStatus().equals(Status.REGISTERED_USER))

client.setStatus(Status.CONFIRMED_USER);

        client = clientRepository.save(client);
        break;
    }
    case FINGER:
    {
        Files.copy(new File(paths.get(0)), fos);
        enroll =
saveEnrollFinger(client,generalized.getPath());
        List<EnrollFinger> enrolls =
enrollFingerRepository.findAllByClient(client);
        enrolls.remove(enroll);
        enrollRepository.delete(enrolls);

if(client.getStatus().equals(Status.REGISTERED_USER))

client.setStatus(Status.CONFIRMED_USER);

        client = clientRepository.save(client);
        break;
    }
    default:
        throw new
IllegalArgumentException("Unhandled type: " + enrollType);
    }
    } catch (IOException e) {
        e.printStackTrace();
        for (java.nio.file.Path filePath : filePaths) {
            storageService.removeDirectory(new
File(filePath.getParent().toString()));
        }
        return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Error
while saving");
    } finally {
        if (fos != null) {
            try {
                fos.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

        try {
            for (java.nio.file.Path filePath : filePaths) {
                storageService.removeDirectory(new
File(filePath.getParent().toString()));
            }
        } catch (Exception ignored) {
        }

        if (enroll == null) {
            return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Database
saving error");
        } else return ResponseEntity.ok("Enroll saved");
    }

    private EnrollVoice saveGeneralVoice(Client client,
String generalizedPath) {
        EnrollVoice enrollVoice = new EnrollVoice();
        enrollVoice.setClient(client);
        enrollVoice.setPath(generalizedPath);
        enrollVoice.setType(EnrollType.VOICE);
        return enrollVoiceRepository.save(enrollVoice);
    }

    private EnrollFace saveGeneralFace(Client client, String
generalizedPath, FaceFeature faceFeature) {
        EnrollFace enrollFace = new EnrollFace();
        enrollFace.setClient(client);
        enrollFace.setPath(generalizedPath);
        enrollFace.setType(EnrollType.FACE);
        enrollFace.setFaceFeature(faceFeature);
        return enrollFaceRepository.save(enrollFace);
    }

    private EnrollFinger saveEnrollFinger(Client client,
String generalizedPath)
    {
        EnrollFinger enrollFinger = new EnrollFinger();
        enrollFinger.setClient(client);
        enrollFinger.setPath(generalizedPath);
        enrollFinger.setType(EnrollType.FINGER);
        return enrollFingerRepository.save(enrollFinger);
    }

    public class Path {
        public static final String GENERALIZE =
"/generalize";
    }
}

```

Приложение А.2.3

Файл «HomeController.java»

```
package ru.programsolutions.ubps.controller;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HomeController
{
    @RequestMapping(value = Path.HOME,method =
RequestMethod.GET)
    public String getHomePage(Model model)
    {
        return Template.HOME;
    }

    public class Path{
        public static final String HOME = "/";
    }

    private class Template{
        public static final String HOME = "home";
    }
}
```

Приложение А.2.4

Файл «LoginController.java»

```
package ru.programsolutions.ubps.controller;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class LoginController
{
```

```

        @RequestMapping(value = Path.LOGIN, method =
RequestMethod.GET)
        public String getLoginPage(Model model)
        {
            return Template.LOGIN;
        }

        public class Path
        {
            public static final String LOGIN = "/login";
        }

        private class Template
        {
            public static final String LOGIN = "login";
        }
    }

```

Приложение A.2.5

Файл «MyCabinetController.java»

```

package ru.programsolutions.ubps.controller;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;
import ru.programsolutions.ubps.entity.auth.AuthUser;
import ru.programsolutions.ubps.property.MobileProperty;
import
ru.programsolutions.ubps.service.currentuser.CurrentUser;
import
ru.programsolutions.ubps.service.notification.NotificationSe
rvice;

public class MyCabinetController {
    @Autowired
    private CurrentUser currentUser;

    @Autowired
    private MobileProperty mobileProperty;

    @Autowired
    private NotificationService notificationService;

```

```

    @RequestMapping(value = Path.MY_CABINET, method =
RequestMethod.GET)
    public String myCabinetPage(Model model) {
        AuthUser user = currentUser.getUser();
        model.addAttribute(Attribute.AUTH_USER, user);

model.addAttribute(Attribute.ANDROID_APPLICATION_LINK,
mobileProperty.getAndroidAppLink());

        return Template.MY_CABINET;
    }

    public static class Attribute {
        public static final String AUTH_USER = "authUser";
        public static final String ANDROID_APPLICATION_LINK
= "androidAppLink";
    }

    public class Path {
        public static final String MY_CABINET = "/profile";
    }

    private class Template {
        public static final String MY_CABINET = "profile";
    }
}

```

Приложение А.2.6

Файл «PersonificationController.java»

```

package ru.programsolutions.ubps.controller;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;
import ru.programsolutions.ubps.entity.Client;

```

```

import ru.programsolutions.ubps.entity.auth.AuthUser;
import ru.programsolutions.ubps.entity.auth.Status;
import ru.programsolutions.ubps.entity.enroll.Enroll;
import ru.programsolutions.ubps.entity.enroll.EnrollType;
import
ru.programsolutions.ubps.repository.AuthUserRepository;
import ru.programsolutions.ubps.repository.ClientRepository;
import ru.programsolutions.ubps.repository.EnrollRepository;
import ru.programsolutions.ubps.storage.FileStorageService;
import ru.programsolutions.ubps.util.DeviceResponse;

import java.util.ArrayList;
import java.util.List;

@Controller
public class PersonificationController
{
    @Autowired
    private FileStorageService storageService;

    @Autowired
    private UserDetailsService detailsService;

    @Autowired
    private EnrollRepository enrollRepository;

    @Autowired
    private AuthUserRepository authUserRepository;

    @Autowired
    private ClientRepository clientRepository;

    @Autowired
    private AuthController authController;

    @RequestMapping(value = Path.SAVE_ENROLL_FACE, method =
RequestMethod.POST)
    @ResponseBody
    public ResponseEntity saveEnrollFace(@RequestParam(value
= Param.AUTH_ID, required = true) Long userId,
                                         @RequestParam(value
= Param.HASH, required = true) String hash,
                                         @RequestParam(value
= Param.DEVICE_ID, required = true) String deviceId,
                                         @RequestParam(value
= Param.ENROLL) List<MultipartFile> multipartFiles,
                                         @RequestParam(value
= Param.ENROLL_TYPE) EnrollType enrollType)
    {

```

```

        DeviceResponse deviceResponse =
authController.isHashValid(userId, hash, deviceId);
        if (!deviceResponse.isAuthenticated())
            return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Not
Authenticated");

        AuthUser authUser =
authUserRepository.findOne(userId);

        List<java.nio.file.Path> paths =
storageService.storeMultipartFiles(multipartFiles);
        List<Enroll> enrolls = new
ArrayList<>(paths.size());

        Client client =
clientRepository.findOne(authUser.getId());
        for (java.nio.file.Path path : paths)
        {
            enrolls.add(new
Enroll(client, enrollType, path.toString()));
        }

        if (enrolls.size() > 0)
        {
            enrolls = ((List<Enroll>)
enrollRepository.save(enrolls));

            if
(client.getStatus().equals(Status.REGISTERED_USER))
            {
                client.setStatus(Status.CONFIRMED_USER);
                clientRepository.save(client);
            }
            return
ResponseEntity.status(HttpStatus.OK).body(true);
        }
        return
ResponseEntity.status(HttpStatus.OK).body(false);
    }

    @RequestMapping(value = Path.SAVE_ENROLL_FACE, method =
RequestMethod.GET)
    public String testUploadPage()
    {
        return Template.TEST_UPLOAD_PAGE;
    }

    public static class Param{
        public static final String AUTH_ID = "auth_id";

```

```

        public static final String HASH = "hash";
        public static final String DEVICE_ID = "deviceId";
        public static final String ENROLL = "enroll";
        public static final String ENROLL_TYPE =
"enroll_type";
    }

    public class Path
    {
        public static final String SAVE_ENROLL_FACE =
"/save/enroll/face";

    }

    private class Template
    {
        public static final String TEST_UPLOAD_PAGE = "test-
upload-page";
    }
}

```

Приложение A.2.7

Файл «RegistrationController.java»

```

package ru.programsolutions.ubps.controller;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.MailException;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import
org.springframework.validation.BeanPropertyBindingResult;
import org.springframework.validation.BindingResult;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.annotation.*;
import ru.programsolutions.ubps.captcha.CaptchaService;
import
ru.programsolutions.ubps.captcha.ReCaptchaInvalidException;
import
ru.programsolutions.ubps.captcha.ReCaptchaUnavailableExcepti
on;
import ru.programsolutions.ubps.config.SecurityConfig;
import ru.programsolutions.ubps.entity.Client;
import ru.programsolutions.ubps.entity.auth.*;
import
ru.programsolutions.ubps.mail.registration.RegistrationConfir
mMail;

```

```

import
ru.programsolutions.ubps.mail.registration.RegistrationConfir
mParam;
import ru.programsolutions.ubps.property.CaptchaProperty;
import ru.programsolutions.ubps.property.SiteProperty;
import
ru.programsolutions.ubps.repository.AuthUserRepository;
import ru.programsolutions.ubps.repository.ClientRepository;
import
ru.programsolutions.ubps.repository.SimpleAuthUserRepository
;
import
ru.programsolutions.ubps.service.AuthUserDetailsService;
import
ru.programsolutions.ubps.service.BcryptEncoderService;
import ru.programsolutions.ubps.service.MailService;
import
ru.programsolutions.ubps.service.notification.NotificationSe
rvice;
import ru.programsolutions.ubps.util.Redirect;
import ru.programsolutions.ubps.util.SimpleResponseMessage;
import
ru.programsolutions.ubps.util.validator.AuthUserValidator;
import
ru.programsolutions.ubps.util.validator.SimpleAuthUserValida
tor;

import javax.servlet.ServletContext;
import java.util.List;

@Controller
public class RegistrationController {
    @Autowired
    AuthUserDetailsService userDetailsService;

    @Autowired
    private CaptchaService captchaService;

    @Autowired
    private SecurityConfig securityConfig;

    @Autowired
    private NotificationService notificationService;

    @Autowired
    private BcryptEncoderService encoderService;

    @Autowired
    private AuthUserValidator authUserValidator;

    @Autowired
    private AuthUserRepository authUserRepository;

```

```

        @Autowired
        private SimpleAuthUserRepository
simpleAuthUserRepository;

        @Autowired
        private SimpleAuthUserValidator simpleAuthUserValidator;

        @Autowired
        private ClientRepository clientRepository;

        @Autowired
        private MailService mailService;

        @Autowired
        private SiteProperty siteProperty;

        @SuppressWarnings("SpringJavaAutowiredInspection")
        @Autowired
        private ServletContext servletContext;

        @RequestMapping(value = Path.REGISTRATION, method =
RequestMethod.GET)
        public String getRegistrationPage(Model model) {
            Client client = new Client();
            /*client.setAuthType(AuthType.SIMPLE);*/
            model.addAttribute("AuthUser", client);
            return Template.REGISTRATION;
        }

        @RequestMapping(value = Path.REGISTRATION, method =
RequestMethod.POST, params = Param.REGISTRATION_MOBILE)
        @ResponseBody
        public SimpleResponseMessage
registerUserMobile(@RequestParam(name = Param.EMAIL,
required = true) String email,

        @RequestParam(name = Param.LASTNAME, required = true) String
lastName,

        @RequestParam(name = Param.FIRSTNAME, required = true)
String firstName,

        @RequestParam(name = Param.PATRONYMIC, required = false)
String patronymic,

        @RequestParam(name = Param.PASSWORD, required = true) String
password,

        @RequestParam(name = Param.RE_PASSWORD, required = true)
String rePassword) {
            if (authUserRepository.findByLogin(email) != null) {

```

```

        return SimpleResponseMessage.forbidden("User
with specified email already registered");
    }

    if (!password.equals(rePassword)) {
        return
SimpleResponseMessage.forbidden("Passwords are not equals");
    }

    Client client = new Client();
    client.setEmail(email);
    client.setPersonInfo(new PersonInfo(firstName,
lastName, patronymic));
    client.setRole(Role.CLIENT);

    BeanPropertyBindingResult bindingResult = new
BeanPropertyBindingResult(client, "client");
    authUserValidator.validate(client, bindingResult);
    if (bindingResult.hasErrors()) {
        return
SimpleResponseMessage.forbidden(bindingResult.getAllErrors()
.get(0).getDefaultMessage());
    }

    client = clientRepository.save(client);
    if (client == null) {
        return
SimpleResponseMessage.internalServerError("Cannot save new
client");
    }

    SimpleAuthUser sau = new SimpleAuthUser();
    sau.setAuthUser(client);
    sau.setPassword(password);

    bindingResult = new BeanPropertyBindingResult(sau,
"sau");
    simpleAuthUserValidator.validate(sau,
bindingResult);

    if (bindingResult.hasErrors()) {
        clientRepository.delete(client);
        return
SimpleResponseMessage.forbidden(bindingResult.getAllErrors()
.get(0).getDefaultMessage());
    }

    sau.setPassword(encoderService.getPasswordEncoder().encode(s
au.getPassword()));
    sau = simpleAuthUserRepository.save(sau);
    if(sau == null){

```

```

        clientRepository.delete(client);
        return
SimpleResponseMessage.internalServerError("Cannot save new
client");
    }

    RegistrationConfirmParam confirmParam = new
RegistrationConfirmParam(client,encoderService.encodeActivat
ionLink(encoderService.getRawLink(client)),servletContext,si
teProperty);
    RegistrationConfirmMail mail = new
RegistrationConfirmMail(confirmParam);
    try {
        mailService.sendEmail(mail);
    } catch (MailException e) {
        e.printStackTrace();
        simpleAuthUserRepository.delete(sau);
        clientRepository.delete(client);
        return
SimpleResponseMessage.internalServerError("Cannot send
confirm message to email");
    }
    return SimpleResponseMessage.ok("You have been
registered, please check your e-mail to confirm
registration");
}

@SuppressWarnings("Duplicates")
@RequestMapping(value = Path.REGISTRATION, method =
RequestMethod.POST)
public String registerUser(@ModelAttribute(name =
Attribute.AUTH_USER) Client client,
                           BindingResult bindingResult,
                           @RequestParam(value =
Param.PASSWORD, required = true) String password,
                           @RequestParam(value =
Param.RE_PASSWORD, required = true) String rePassword,
                           @RequestParam(value =
CaptchaProperty.G_RECAPTCHA_RESPONSE, required = true)
String reCaptchaResponse
    ) {

    AuthUser userInDatabase =
authUserRepository.findByLogin(client.getEmail());
    if (userInDatabase != null) {

notificationService.addErrorMessage("Пользователь с таким
адресом уже зарегистрирован");
        return Redirect.to(Path.REGISTRATION);
    }

    try {

```

```

captchaService.processResponse(reCaptchaResponse);
    } catch (ReCaptchaInvalidException e) {
        notificationService.addErrorMessage("Неверная
капча");
        return Redirect.to(Path.REGISTRATION);
    } catch (ReCaptchaUnavailableException e) {
        System.err.println("Капча недоступна");
        // TODO: 28.07.2017 Если капча недоступна, пока
разрешает регистрацию
    }

    if (!password.equals(rePassword)) {
        notificationService.addErrorMessage("Пароли не
совпадают");
        return Redirect.to(Path.REGISTRATION);
    }

    if (client.getPersonInfo().getPatronymic() == null)
{
        client.getPersonInfo().setPatronymic("");
    }
    /*if (client.getAuthType() == null)
        client.setAuthType(AuthType.SIMPLE);*/
    client.setBlocked(true);
    client.setStatus(Status.NEW_USER);
    client.setRole(Role.CLIENT);

    authUserValidator.validate(client, bindingResult);

    if (bindingResult.hasErrors()) {
        List<ObjectError> errors =
bindingResult.getAllErrors();
        for (ObjectError error : errors) {

notificationService.addErrorMessage(error.getDefaultMessage(
));
        }
        return Redirect.to(Path.REGISTRATION);
    }

    try {
        client = clientRepository.save(client);
    } catch (Exception e) {
        client = null;
        e.printStackTrace();
    }

    if (client == null) {
        notificationService.addErrorMessage("Не удалось
сохранить пользователя");
    }

```

```

        return Redirect.to(Path.REGISTRATION);
    }

    SimpleAuthUser simpleAuthUser = new
SimpleAuthUser();

simpleAuthUser.setAuthUser(authUserRepository.findOne(client
.getId()));
    simpleAuthUser.setPassword(password);

    BeanPropertyBindingResult simpleBindingResult = new
BeanPropertyBindingResult(simpleAuthUser, "simpleAuthUser");

    simpleAuthUserValidator.validate(simpleAuthUser,
simpleBindingResult);
    if (simpleBindingResult.hasErrors()) {
        List<ObjectError> errors =
simpleBindingResult.getAllErrors();
        for (ObjectError error : errors) {

notificationService.addErrorMessage(error.getDefaultMessage(
));
        }
        clientRepository.delete(client);
        return Redirect.to(Path.REGISTRATION);
    }

simpleAuthUser.setPassword(encoderService.getPasswordEncoder
().encode(simpleAuthUser.getPassword()));

    try {
        simpleAuthUser =
simpleAuthUserRepository.save(simpleAuthUser);
    } catch (Exception e) {
        e.printStackTrace();
        simpleAuthUser = null;
        clientRepository.delete(client);
    }

    if (simpleAuthUser == null) {
        notificationService.addErrorMessage("Не удалось
сохранить пользователя");
        clientRepository.delete(client);
        return Redirect.to(Path.REGISTRATION);
    }

    RegistrationConfirmParam confirmParam = new
RegistrationConfirmParam(client,

encoderService.encodeActivationLink(encoderService.getRawLin
k(client)), servletContext, siteProperty);

```

```

        RegistrationConfirmMail mail = new
RegistrationConfirmMail(confirmParam);

        try {
            mailService.sendEmail(mail);
        } catch (Exception e) {
            e.printStackTrace();
            if (e instanceof MailException)
                notificationService.addErrorMessage("Ошибка
отправки письма");
            else {

notificationService.addErrorMessage("Неизвестная ошибка");
            }

            simpleAuthUserRepository.delete(simpleAuthUser);
            clientRepository.delete(client);
            return Redirect.to(Path.REGISTRATION);
        }
        notificationService.addInfoMessage("На указанный
электронный почтовый адрес было выслано письмо для
подтверждения регистрации");
        return Redirect.to(LoginController.Path.LOGIN);
    }

    @RequestMapping(value = Path.ACTIVATE_USER, method =
RequestMethod.GET)
    public String activateUser(@RequestParam(value =
RegistrationConfirmParam.LinkParam.USER_CONFIRM_ID, required
= true) Long userId,
                                @RequestParam(value =
RegistrationConfirmParam.LinkParam.ACTIVATION_SECRET,
required = true) String secret) {
        Object one = clientRepository.findOne(userId);
        AuthUser authUser = (AuthUser) one;
        if (authUser == null) {
            notificationService.addErrorMessage("Неизвестный
пользователь");
            return Redirect.to(Path.REGISTRATION);
        }

        if (authUser.getStatus().equals(Status.BANNED_USER)
&& authUser.getBlocked()) {
            notificationService.addErrorMessage("Вы были
заблокированы в системе");
            return Redirect.to(Path.REGISTRATION);
        }

        String rawLink =
encoderService.getRawLink(authUser);

```

```

        if
(encoderService.getPasswordEncoder().matches(rawLink,
secret)) {
            switch (authUser.getStatus()) {
                case BANNED_USER:
                    notificationService.addErrorMessage("Вы
были заблокированы в системе");
                    return Redirect.to(Path.REGISTRATION);
                case CONFIRMED_USER:
                case REGISTERED_USER: {

notificationService.addWarningMessage("Вы уже подтвердили
регистрацию");
                    return
Redirect.to(LoginController.Path.LOGIN);
                }
                case NEW_USER: {

notificationService.addSuccessMessage("Регистрация прошла
успешно");
                    authUser.setBlocked(false);

authUser.setStatus(Status.REGISTERED_USER);
                    authUserRepository.save(authUser);
                    return
Redirect.to(LoginController.Path.LOGIN);
                }
                default:
                    return
Redirect.to(LoginController.Path.LOGIN);
            }
        } else {
            notificationService.addErrorMessage("Неверный
ключ");
            return Redirect.to(Path.REGISTRATION);
        }
    }

    public class Param {
        public static final String REGISTRATION_MOBILE =
"registration=mobile";
        public static final String EMAIL = "e-mail";
        public static final String FIRSTNAME = "first_name";
        public static final String LASTNAME = "last_name";
        public static final String PATRONYMIC =
"patronymic";
        public static final String PASSWORD = "password";
        public static final String RE_PASSWORD =
"re_password";
    }

```

```

        public static class Attribute{
            public static final String AUTH_USER = "AuthUser";
        }

        public class Path {
            public static final String REGISTRATION =
"/registration";
            public static final String ACTIVATE_USER =
"/activate";
        }

        private class Template {
            public static final String REGISTRATION =
"registration";
            public static final String REGISTRATION_CONFIRM_FRL
= "registration-confirm.ftl";
        }
    }
}

```

Приложение А.2.8

Файл «ShopController.java»

```

package ru.programsolutions.ubps.controller;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.util.Base64Utils;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import ru.programsolutions.ubps.entity.UserHash;
import
ru.programsolutions.ubps.repository.AuthUserRepository;
import
ru.programsolutions.ubps.repository.UserHashRepository;
import ru.programsolutions.ubps.util.DeviceResponse;

@Controller
public class ShopController {
    @Autowired
    UserHashRepository userHashRepository;
}

```

```

@Autowired
AuthUserRepository authUserRepository;

@Autowired
AuthController authController;

@Autowired
DeviceResponse deviceResponse;

@RequestMapping(value = Path.SHOP_CHECK, method =
RequestMethod.GET)
@ResponseBody
public ResponseEntity checkAuth(@RequestParam(name =
AuthController.Param.AUTH_ID) Long authId,
                                @RequestParam(name =
Param.AUTH_HASH) String auth_hash,
                                @RequestParam(name =
AuthController.Param.LOGIN_TOKEN) String login_token,
                                @RequestParam(name =
AuthController.Param.DEVICE_ID) String deviceId) {

    //Ищем хеш пользователя
    System.out.println(String.format("SHOP_CHECK:
%s:%s", Param.AUTH_HASH, auth_hash));
    try {
        auth_hash = new
String(Base64Utils.decodeFromString(auth_hash));
    } catch (Exception e) {
        return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Cannot
decode auth_hash");
    }
    System.out.println(String.format("SHOP_CHECK:
decoded %s:%s", Param.AUTH_HASH, auth_hash));

    UserHash userHash = userHashRepository.find(authId,
login_token, deviceId);

    if (userHash == null) {
        System.out.println("::User not found");
        return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("No user
hash found");
    }

    //не уверен, что это здесь нужно.
    boolean authenticated =
authController.isHashValid(authId, login_token,
deviceId).isAuthenticated();

```

```

        if (!authenticated) {
            System.out.println("::Not authenticated");
            return
        }
        ResponseEntity.status(HttpStatus.FORBIDDEN).body("Not
authenticated");
    }

    if (userHash.checkHash(auth_hash)) {
        userHash.dropSession();
        userHashRepository.save(userHash);
        return
        ResponseEntity.ok(deviceResponse.createAuthenticated(login_t
oken, authId, deviceId)/*true*/);
    } else {
        System.out.println("::No biometrics auth");
        return
        ResponseEntity.status(HttpStatus.FORBIDDEN).body("No
Biometrics auth");
    }
}

public class Param {
    public static final String AUTH_HASH = "auth_hash";
}

public class Path {
    public static final String SHOP_CHECK =
"/checkshop";
}

private class Template {
    public static final String SHOP_CHECK = "checkshop";
}
}

```

Приложение А.2.9

Файл «WebAuthController.java»

```

package ru.programsolutions.ubps.controller;

import com.oep.faceroll.lib.ApiCalls;
import com.oep.faceroll.lib.FaceFeatures;
import org.apache.commons.codec.binary.Base64;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;

```

```

import org.springframework.ui.Model;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;
import ru.programsolutions.ubps.entity.enroll.EnrollType;
import ru.programsolutions.ubps.repository.EnrollRepository;
import
ru.programsolutions.ubps.service.currentuser.CurrentUser;
import
ru.programsolutions.ubps.service.search.FaceAuthService;
import ru.programsolutions.ubps.storage.FileStorageService;
import ru.programsolutions.ubps.util.MultipartFileImpl;

import java.io.*;
import java.util.UUID;

@Controller
public class WebAuthController {
    @Autowired
    private CurrentUser currentUser;

    @Autowired
    private EnrollRepository enrollRepository;

    @Autowired
    private FileStorageService storageService;

    @Autowired
    private FaceAuthService faceAuthService;

    @Autowired
    private AuthController authController;

    @RequestMapping(value = Path.WEB_AUTH, method =
RequestMethod.GET)
    public String webAuthPage(Model model) {
        model.addAttribute(Attribute.VERIFY_PATH,
Path.WEB_AUTH_VERIFY);
        return Template.WEB_AUTH;
    }

    @RequestMapping(value = Path.WEB_AUTH_VERIFY_VOICE,
method = RequestMethod.GET)
    public String verifyVoicePage() {
        return Template.WEB_AUTH_VOICE;
    }
}

```

```

    @RequestMapping(value = Path.WEB_AUTH_VOICE, method =
RequestMethod.POST)
    @ResponseBody
    public ResponseEntity webAuthVoice(@RequestParam(name =
Param.VOICE) MultipartFile voiceRaw,
                                     @RequestParam(name =
AuthController.Param.AUTH_ID) Long authId,
                                     @RequestParam(name =
AuthController.Param.DEVICE_ID) String deviceId,
                                     @RequestParam(name =
AuthController.Param.LOGIN_TOKEN) String token) {
        File f = null;
        File enroll = null;
        try {
            InputStream is = voiceRaw.getInputStream();
            String uuid = UUID.randomUUID().toString();
            String rawPostfix = "_VOICE_RAW";
            String enrollPostfix = "_VOICE_ENROLL";

            while ((f = new File(uuid +
rawPostfix)).exists()) {
                uuid = UUID.randomUUID().toString();
            }
            f.createNewFile();
            FileOutputStream fos = new FileOutputStream(f);
            {
                int read;
                while ((read = is.read()) != -1) {
                    fos.write(read);
                    fos.flush();
                }
            }
            fos.close();
            String enrollPath = f.getAbsolutePath() +
enrollPostfix;
            String enrollName = uuid + enrollPostfix;
            enroll = new File(enrollPath);
            ApiCalls.enrollVoice(f.getAbsolutePath(),
enrollPath);

            FileInputStream fis = new
FileInputStream(enroll);
            ByteArrayOutputStream baos = new
ByteArrayOutputStream();
            {
                int read;
                while ((read = fis.read()) != -1) {
                    baos.write(read);
                }
            }
            byte[] enrollBytes = baos.toByteArray();

```

```

        MultipartFileImpl multipartFile = new
MultipartFileImpl(enrollName, enrollName, enrollBytes);
        ResponseEntity response =
authController.verify(authId, deviceId, token,
multipartFile, EnrollType.VOICE.name());
        return response;
    } catch (IOException e) {
        return
createResponseEntity(HttpStatus.INTERNAL_SERVER_ERROR,
"Cannot get data from recorded voice");
    } catch (SecurityException e) {
        return
createResponseEntity(HttpStatus.INTERNAL_SERVER_ERROR,
"Cannot create temporary file.");
    } catch (Exception e) {
        e.printStackTrace();
        return
createResponseEntity(HttpStatus.INTERNAL_SERVER_ERROR,
"Cannot recognize voice");
    } finally {
        deleteFile(f);
        deleteFile(enroll);
    }
}

@RequestMapping(value = Path.WEB_AUTH_FACE, method =
RequestMethod.POST)
@ResponseBody
public ResponseEntity webAuthFace(@RequestParam(name =
Param.FACE) String source,
                                   @RequestParam(name =
AuthController.Param.DEVICE_ID) String deviceId) {
    File f = null;
    String enrollPath = null;
    try {
        byte[] face =
Base64.decodeBase64(source.getBytes());
        String dir =
storageService.getRoot().toString();
        String uuid = UUID.randomUUID().toString();
        f = new File(dir, uuid);
        while (f.exists()) {
            uuid = UUID.randomUUID().toString();
            f = new File(dir, uuid);
        }
        f.createNewFile();
        FileOutputStream fos;
        fos = new FileOutputStream(f);
        fos.write(face);
        fos.flush();
        fos.close();
        //TEMPORARY FILE CREATED, CREATE ENROLL

```

```

        enrollPath = f.getAbsolutePath() + "_FACE";
        FaceFeatures ffs =
ApiCalls.enrollFace(f.getAbsolutePath(), enrollPath);
        FileInputStream fis = new
FileInputStream(enrollPath);
        ByteArrayOutputStream baos = new
ByteArrayOutputStream();
        int read;
        while ((read = fis.read()) != -1) {
            baos.write(read);
        }

        MultipartFileImpl multipartFile = new
MultipartFileImpl(uuid, uuid, baos.toByteArray());
        ResponseEntity response =
authController.getHashByFace(multipartFile, deviceId,
ffs.getAge(), ffs.isMale(), false);
        return response;
    } catch (Exception e) {
        e.printStackTrace();
        return
createResponseEntity(HttpStatus.INTERNAL_SERVER_ERROR,
"Error occurred");
    } finally {
        deleteFile(f);
        if (enrollPath!=null) {
            deleteFile(new File(enrollPath));
        }
    }
}

private void deleteFile(File file) {
    if (file != null && file.exists()) {
        try {
            file.delete();
        } catch (SecurityException e) {
            e.printStackTrace();
        }
    }
}

private ResponseEntity createResponseEntity(HttpStatus
status, Object body) {
    return ResponseEntity.status(status).body(body);
}

public class Path {
    public static final String WEB_AUTH = "/web-
authentication";
    public static final String WEB_AUTH_VERIFY =
WEB_AUTH + "/verify";

```

```

        public static final String WEB_AUTH_FACE = WEB_AUTH
+ "/face";
        public static final String WEB_AUTH_VOICE = WEB_AUTH
+ "/voice";
        public static final String WEB_AUTH_VERIFY_VOICE =
WEB_AUTH_VERIFY + "/voice";
    }

    public static class Attribute{
        public static final String VERIFY_PATH =
"verifyPath";
    }

    public class Param {
        public static final String FACE = "source";
        public static final String VOICE = "voice";
        public static final String USER_IP = "user_ip";
    }

    private class Template {
        public static final String WEB_AUTH = "web-auth";
        public static final String WEB_AUTH_VOICE = "web-
auth-voice";
    }
}

```

Приложение А.3

Листинги репозитория

Приложение А.3.1

Файл «AuthorizedDeviceRepository.java»

```

package ru.programsolutions.ubps.repository;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import ru.programsolutions.ubps.entity.AuthorizedDevice;
import ru.programsolutions.ubps.entity.Client;

import java.util.List;

public interface AuthorizedDeviceRepository extends
CrudRepository<AuthorizedDevice, Long>{

```

```

        @Query("select ad from AuthorizedDevice ad where :client
in ad.clients")
        List<AuthorizedDevice> findByClient(@Param("client")
Client client);

        @Query("select ad from AuthorizedDevice ad where
ad.deviceId = :identity")
        AuthorizedDevice findByDeviceIdentity(@Param("identity")
String deviceIdentity);

        @Query("select ad.clients from AuthorizedDevice ad where
ad.deviceId = :deviceId")
        List<Client> findClientsByDeviceId(@Param("deviceId")
String deviceId);
}

```

Приложение A.3.2

Файл «AuthUserRepository.java»

```

package ru.programsolutions.ubps.repository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import ru.programsolutions.ubps.entity.auth.AuthUser;
import ru.programsolutions.ubps.entity.auth.Role;

public interface AuthUserRepository extends
CrudRepository<AuthUser, Long>
{
    @Query("select au.role from AuthUser au where au.email =
:login")
    Role getAuthRole(@Param("login") String login);

    AuthUser findByLogin(String login);
}

```

Приложение A.3.3

Файл «ClientRepository.java»

```

package ru.programsolutions.ubps.repository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import ru.programsolutions.ubps.entity.Client;

```

```

public interface ClientRepository extends
CrudRepository<Client, Long>
{
    @Query("select c from Client c where c.email = :login")
    Client findByLogin(@Param("login") String login);
}

```

Приложение А.3.4

Файл «EnrollFaceRepository.java»

```

package ru.programsolutions.ubps.repository;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import ru.programsolutions.ubps.entity.Client;
import ru.programsolutions.ubps.entity.enroll.EnrollFace;
import ru.programsolutions.ubps.entity.enroll.Gender;
import
ru.programsolutions.ubps.util.enrollcut.EnrollCutList;

import java.util.List;

public interface EnrollFaceRepository extends
CrudRepository<EnrollFace,Long> {

    @Query("delete EnrollFace e where e.id!=:eid")
    void deletExcept(@Param("eid") Long id);

    @Query("select ef from EnrollFace ef where ef.client =
:client")
    List<EnrollFace> findAllByClient(@Param("client") Client
client);

    @Query("select new
ru.programsolutions.ubps.util.enrollcut.EnrollCut(ef.client.
id,ef.path) from EnrollFace ef where ef.client.id in
(:clients)")
    EnrollCutList cutByClients(@Param("clients") List<Long>
clientIds);

    @Query("select new
ru.programsolutions.ubps.util.enrollcut.EnrollCut(ef.client.
id,ef.path) from EnrollFace ef where
ef.faceFeature.averageAge between (:age - 10) and (:age +
10) and ef.faceFeature.gender = :gender")

```

```

        EnrollCutList cutByFaceFeature(@Param("gender") Gender
gender, @Param("age") int age);

        @Query("select new
ru.programsolutions.ubps.util.enrollcut.EnrollCut(ef.client.
id,ef.path) from EnrollFace ef")
        EnrollCutList cutAll();
    }

```

Приложение А.3.5

Файл «EnrollFingerRepository.java»

```

package ru.programsolutions.ubps.repository;

import org.springframework.data.repository.CrudRepository;
import ru.programsolutions.ubps.entity.Client;
import ru.programsolutions.ubps.entity.enroll.EnrollFinger;

import java.util.List;

public interface EnrollFingerRepository extends
CrudRepository<EnrollFinger, Long> {
    List<EnrollFinger> findAllByClient(Client client);
}

```

Приложение А.3.6

Файл «EnrollRepository.java»

```

package ru.programsolutions.ubps.repository;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import ru.programsolutions.ubps.entity.Client;
import ru.programsolutions.ubps.entity.enroll.Enroll;
import ru.programsolutions.ubps.entity.enroll.EnrollType;
import
ru.programsolutions.ubps.util.enrollcut.EnrollCutList;

import java.util.List;

public interface EnrollRepository extends
CrudRepository<Enroll, Long>
{
    @Query("select e from Enroll e where e.client.id = :cid
and e.type = :etype")

```

```

        List<Enroll> findManyByAuthUser(@Param("cid") Long id,
@Param("etype") EnrollType enrollType);

        @Query("select e from Enroll e where e.client.id = :cid
and e.type = :etype")
        Enroll findByAuthUser(@Param("cid") Long id,
@Param("etype") EnrollType enrollType);

        @Query("select new
ru.programsolutions.ubps.util.enrollcut.EnrollCut(e.client.i
d,e.path) from Enroll e where e.type =
ru.programsolutions.ubps.entity.enroll.EnrollType.FACE")
        EnrollCutList findFaces();

        @Query("select e from Enroll e where e.client =
:client")
        List<Enroll> findByClient(@Param("client") Client
client);
    }
}

```

Приложение А.3.7

Файл «EnrollVoiceRepository.java»

```

package ru.programsolutions.ubps.repository;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import ru.programsolutions.ubps.entity.Client;
import ru.programsolutions.ubps.entity.enroll.EnrollVoice;

import java.util.List;

public interface EnrollVoiceRepository extends
CrudRepository<EnrollVoice, Long> {
    @Query("delete EnrollVoice e where e.id!=:eid")
    void deletExcept(@Param("eid") Long id);

    @Query("select ev from EnrollVoice ev where ev.client =
:client")
    List<EnrollVoice> findAllByClient(@Param("client")
Client client);
}

```

Приложение А.3.8

Файл «SimpleAuthUserRepository.java»

```
package ru.programsolutions.ubps.repository;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import ru.programsolutions.ubps.entity.auth.AuthUser;
import ru.programsolutions.ubps.entity.auth.SimpleAuthUser;

public interface SimpleAuthUserRepository extends
    CrudRepository<SimpleAuthUser, Long>
{
    @Query("select sau from SimpleAuthUser sau where
sau.authUser.id = :id")
    SimpleAuthUser findByAuthUser(@Param("id") Long id);

    @Query("select sau from SimpleAuthUser sau where
sau.authUser = :au")
    SimpleAuthUser findByAuthUser(@Param("au") AuthUser
authUser);
}
```

Приложение А.3.9

Файл «UserHashRepository.java»

```
package ru.programsolutions.ubps.repository;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import ru.programsolutions.ubps.entity.UserHash;
import ru.programsolutions.ubps.entity.auth.AuthUser;

import java.util.Date;
import java.util.Set;

public interface UserHashRepository extends
    CrudRepository<UserHash, Long>
{
    @Query("select uh from UserHash uh where uh.authUser =
:au and uh.hash like :hash")
    UserHash findByHashAndAuthuserId(@Param("au") AuthUser
authUser, @Param("hash") String hash);
}
```

```

        @Query("select uh from UserHash uh where uh.authUser =
:au and uh.deviceId like :devid")
        UserHash findByUserAndDeviceId(@Param("au") AuthUser
authUser, @Param("devid") String deviceId);

        @Query("select uh from UserHash uh where uh.lastModified
<= :dbefore and uh.expired = false")
        Set<UserHash> findNotExpiredBefore(@Param("dbefore")
Date before);

        @Query("select uh from UserHash uh where uh.authUser.id
= :uid and uh.hash = :hash and uh.deviceId = :devid")
        UserHash find(@Param("uid") Long userId, @Param("hash")
String hash, @Param("devid") String deviceId);
    }

```

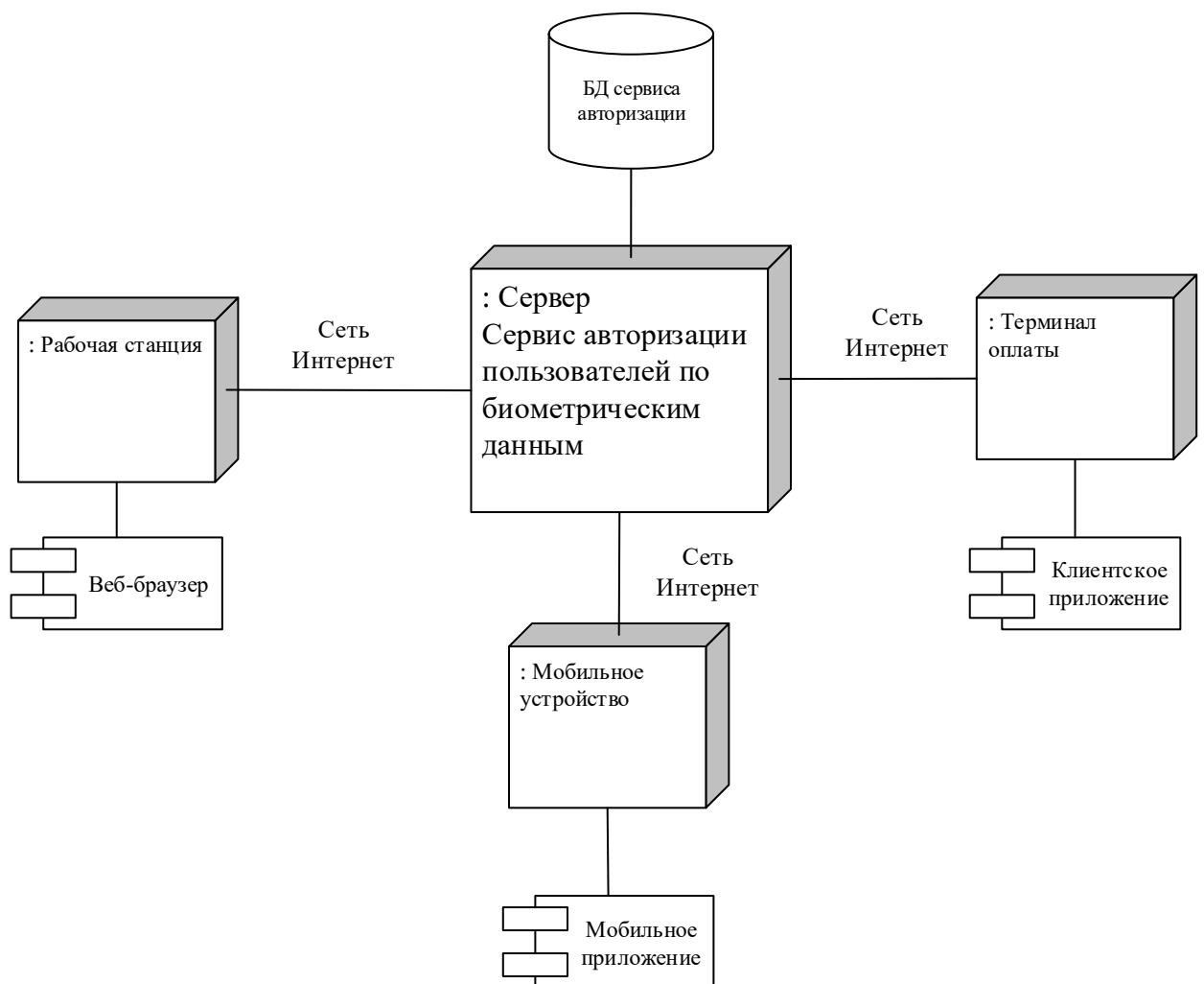
ПРИЛОЖЕНИЕ Б

ДИАГРАММЫ UML

(рекомендуемое)

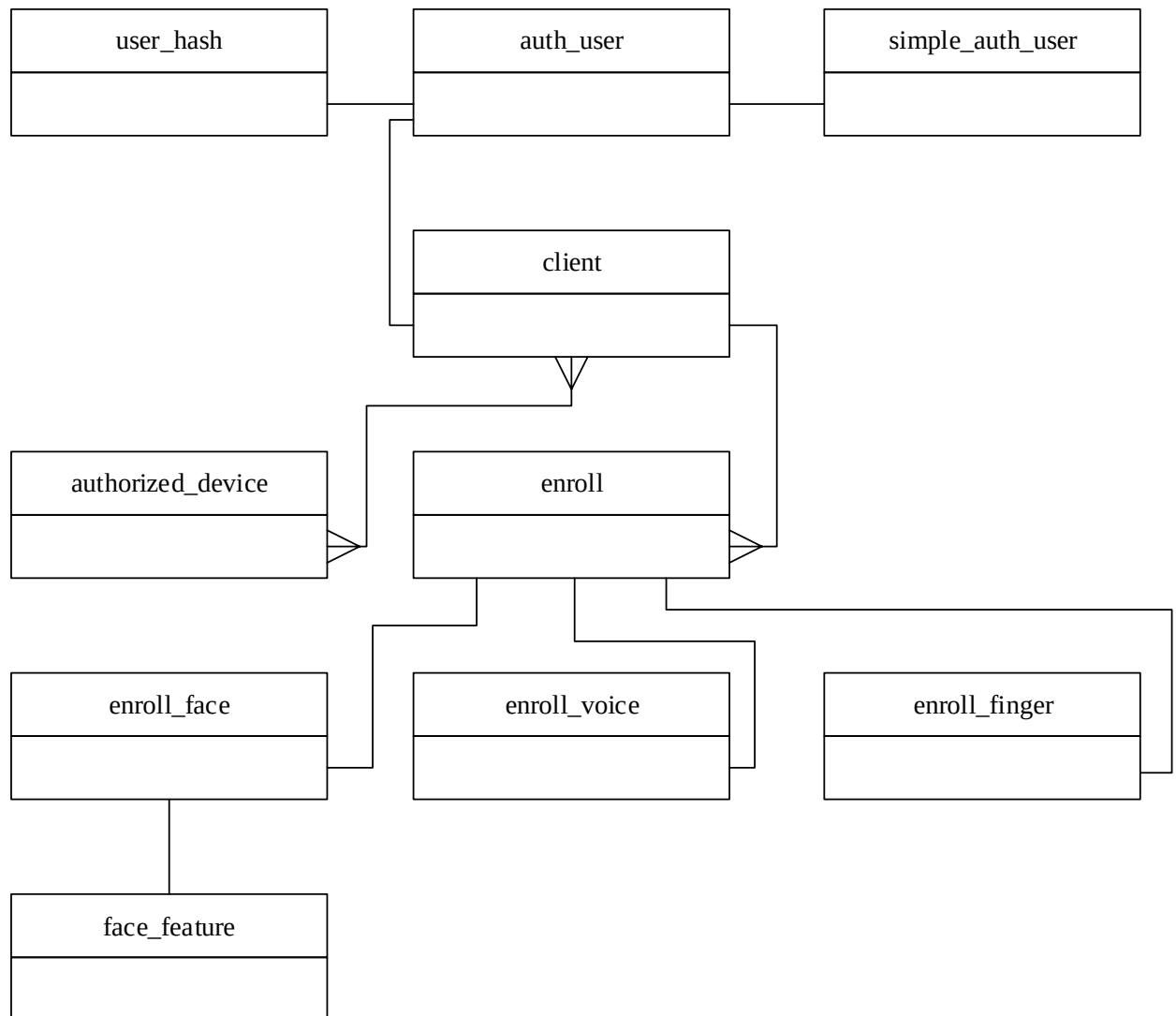
Приложение Б.1

Диаграмма развертывания



Приложение Б.2

Диаграмма сущность-связь



ПРИЛОЖЕНИЕ В
ПРЕЗЕНТАЦИЯ
(рекомендуемое)



**СЕРВИС АВТОРИЗАЦИИ
ПОЛЬЗОВАТЕЛЕЙ ПО
БИОМЕТРИЧЕСКИМ ДАННЫМ**

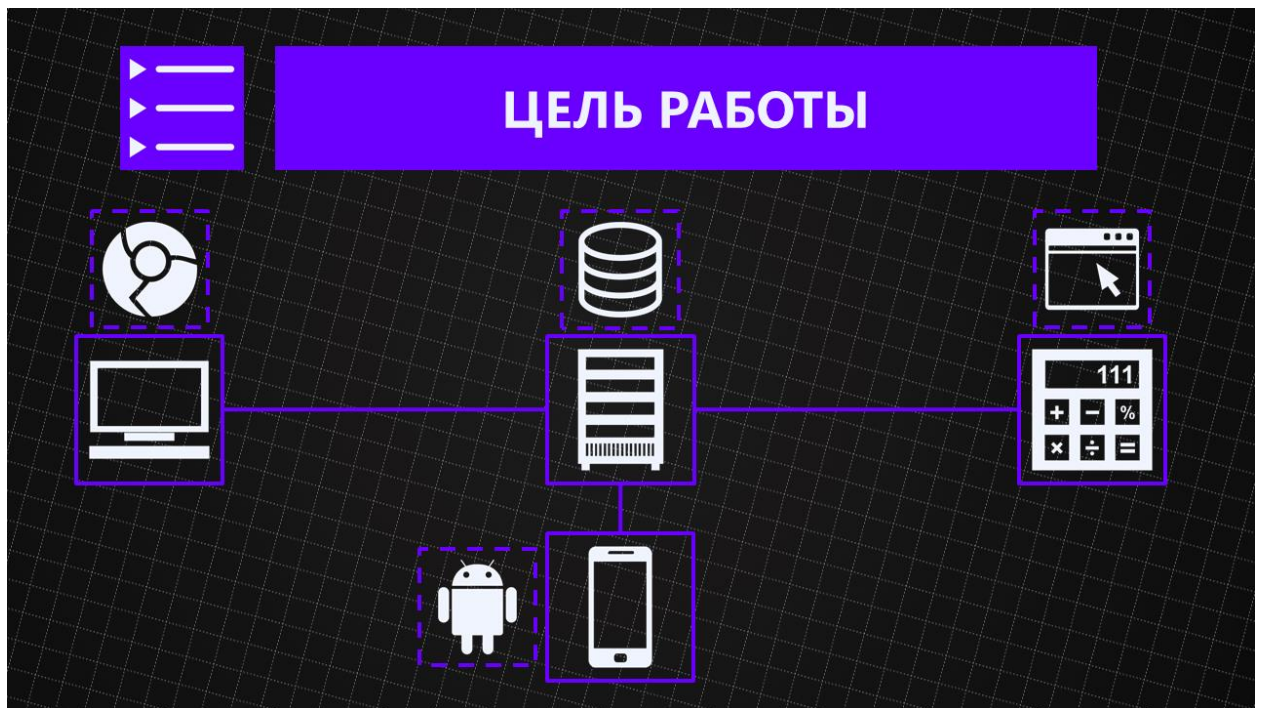
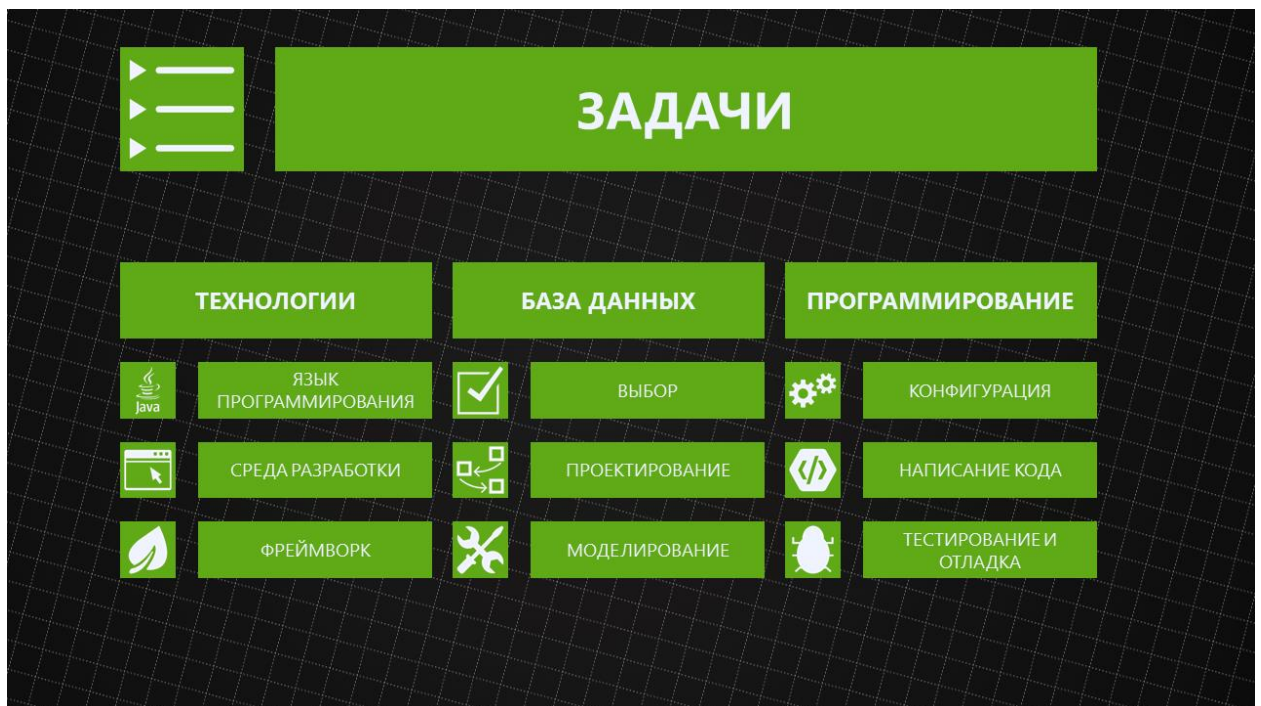
 **МИРОНОВ А.А.**

 **ДТН, ПРОФЕССОР ЗИНКИН С.А.**



ВИДЫ БИОМЕТРИИ

	СТАТИЧЕСКИЕ	ДИНАМИЧЕСКИЕ	
	ДНК	РЕЧЬ	
	ЛИЦО	ПОХОДКА	
	ПАЛЬЦЫ	ПОЧЕРК	
	ЛАДОНЬ		





ТЕХНОЛОГИИ

ЯЗЫК
ПРОГРАММИРОВАНИЯ



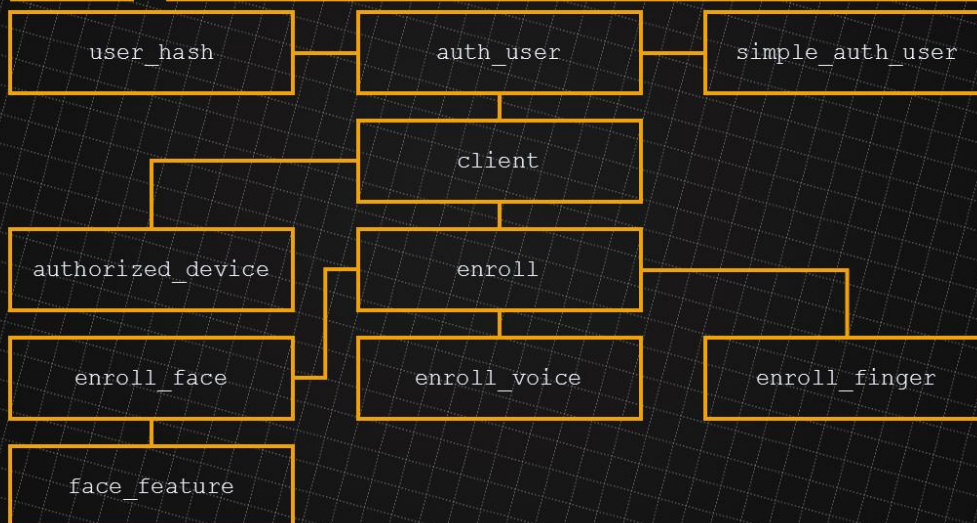
СРЕДА РАЗРАБОТКИ



ФРЕЙМВОРК

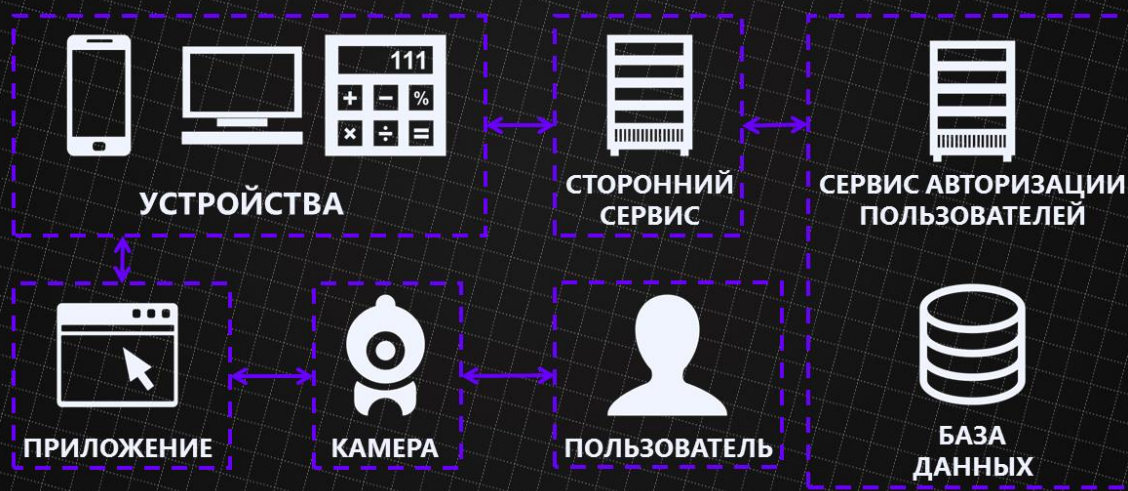


БАЗА ДАННЫХ





АЛГОРИТМ РАБОТЫ



1 ЭТАП АВТОРИЗАЦИИ



ПОЛУЧЕНИЕ ЗАПРОСА

```
@RequestParam(value = Param.ENROLL) MultipartFile multipartFile,  
@RequestParam(value = Param.DEVICE_ID) String deviceId,  
@RequestParam(value = Param.AGE) Integer avgAge,  
@RequestParam(value = Param.MALE) Boolean isMale,  
@RequestParam(name = Param.ENCODED, defaultValue = "false", required =  
false) Boolean encoded
```



ПОИСК ПОДХОДЯЩЕГО КАНДИДАТА



ГЕНЕРАЦИЯ, ПРИВЯЗКА И СОХРАНЕНИЕ ТОКЕНОВ В БАЗЕ ДАННЫХ



ВОЗВРАТ ПУБЛИЧНОЙ ЧАСТИ ТОКЕНА



2 ЭТАП АВТОРИЗАЦИИ



ПОЛУЧЕНИЕ ЗАПРОСА

```
@RequestParam(name = Param.AUTH_ID) Long authId,  
@RequestParam(name = Param.DEVICE_ID) String deviceId,  
@RequestParam(name = Param.LOGIN_TOKEN) String token,  
@RequestParam(name = Param.ENROLL, required = true) MultipartFile  
multipartFile,  
@RequestParam(name = Param.ENROLL_TYPE, required = true) String enrollType
```



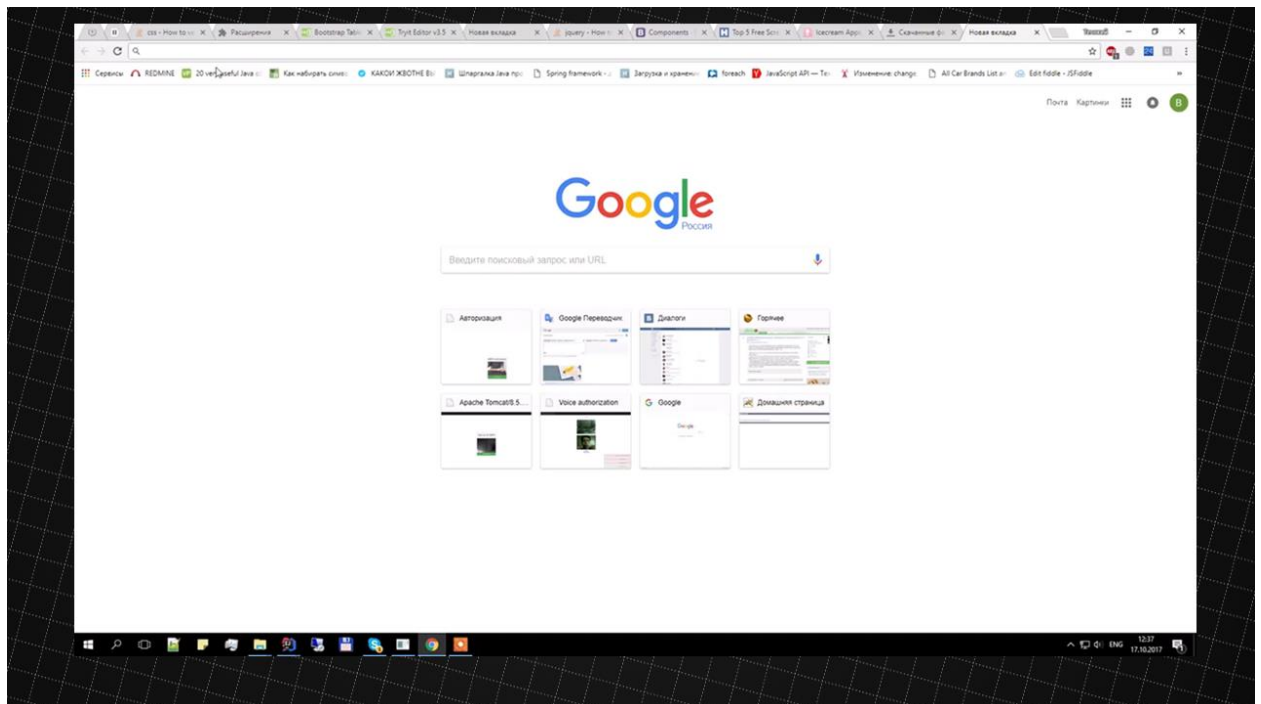
ПРОВЕРКА ТОКЕНА



ПРОВЕРКА ПО ПЕРЕДАННОМУ БИОМЕТРИЧЕСКОМУ ИДЕНТИФИКАТОРУ



ВОЗВРАТ РЕЗУЛЬТАТА





КОНЕЦ