

Пенза, 2022

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ П.П. Макарычев
«__» _____ 2022 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Вдовкина София Александровна гр. 18ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Онлайн-сервис верификации рукописной подписи

Тема утверждена приказом ПГУ № 327/О от "18" апреля 2022 г.

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Шибанов С.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Назначение: серверные программные средства для онлайн верификации рукописной подписи*

2) *Среда выполнения: Windows 10*

Языковые средства разработки: Python

Инструментальные средства: VS Code, PostgreSQL 11, DBeaver, SQLAlchemy, Flask.

Технологии разработки ООП, сервисно-ориентированный подход.

3) *Основные функции разрабатываемого программного обеспечения:*

а) регистрация подписи в системе;

б) извлечение данных из рукописной подписи;

в) обучение модели верификации подписи;

г) выбор модели верификации;

д) верификация пользователя по подписи;

е) обновление данных о подписи пользователя;

ж) удаление данных;

з) изменение входного порога для пользователя.

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;
2) План разработки бюджета;
3) Разработка требований и проектирование;
4) Реализация;
5) Контроль качества программного обеспечения;
6) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	10%	13.04.2022 – 22.04.2022	
2	План разработки бюджета	5%	23.04.2022 – 25.04.2022	
3	Проектирование	10%	26.04.2022 – 03.05.2022	
4	Реализация	50%	04.05.2022 – 02.06.2022	
5	Контроль качества программного обеспечения	10%	03.06.2022 – 08.06.2022	
6	Оформление пояснительной записки	15%	09.06.2022 – 17.06.2022	

Дата выдачи задания «22» _____ декабря _____ 2021 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 99 листов, 40 рисунков, 10 таблиц, 46 использованных источника и 4 приложения.

ВЕРИФИКАЦИЯ РУКОПИСНОЙ ПОДПИСИ, RESTFULL СЕРВИС, АЛГОРИТМ ДИНАМИЧЕСКОЙ ТРАНСФОРМАЦИИ ВРЕМЕННОЙ ШКАЛЫ, ОБЗОР МЕТОДОВ ВЕРИФИКАЦИИ, FLASK, POSTGRESQL, SQLALCHEMY.

Объект исследования – методы и алгоритмы онлайн верификации рукописной подписи.

Цель выпускной квалификационной работы – разработать онлайн сервис для верификации рукописной подписи с архитектурой REST.

Технологии разработки – разработка программных средств проводилась на языке программирования Python с использованием фреймворка Flask, ORM SQLAlchemy, СУБД PostgreSQL 11. В качестве среды разработки использовался редактор программного кода VS Code и менеджер баз данных DBeaver.

Среда исполнения – Windows 10.

Результаты работы – в ходе выполнения выпускной квалификационной работы был разработан программный комплекс средств для онлайн верификации рукописной подписи. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08ВР182.02 ПЗ					
Изм.	Лист	№ докум.	Подпись	Дата	Онлайн-сервис верификации рукописной подписи Пояснительная записка			Лит.	Лист	Листов
Разраб.		Вдовкина С.А.								
Пров.		Шибанов С.В.							3	99
								Группа 18ВР2		
Н. Контр.		Такташкин Д.В								
Утв.										

Содержание

Введение.....	8
1 Анализ предметной области и постановка задачи на разработку онлайн-сервиса верификации рукописной подписи.....	10
1.1 Основы верификации рукописной подписи.....	10
1.2 Сравнительный анализ аналогичных программных средств	11
1.2.1 Wacom Ink SDK for verification.....	12
1.2.2 Progress Soft	12
1.2.3 SIGNificant	14
1.3 Постановка задачи на разработку онлайн-сервиса верификации рукописной подписи	15
1.4 Планирование процесса разработки онлайн-сервиса верификации рукописной подписи	17
2 Обоснование выбора алгоритмов верификации рукописной подписи для реализации в рамках онлайн-сервиса.....	19
2.1 Классификация и выбор критериев оценки алгоритмов верификации рукописной подписи	20
2.2 Обзор и теоретическая оценка алгоритмов верификации рукописной подписи.....	21
2.3 Экспериментальная оценка алгоритмов верификации рукописной подписи.....	25
2.4 Результаты оценки и обоснование выбора алгоритмов верификации рукописной подписи	27
3 Разработка требований к онлайн-сервису верификации рукописной подписи.....	31
3.1 Функциональные требования к онлайн-сервису верификации рукописной подписи	31
3.2 Требования к архитектуре онлайн-сервиса верификации рукописной подписи	35

3.2.1 Разработка программного метода оценки качества верифицируемой подписи.....	59
3.3 Концептуальная модель базы данных пользователей и эталонных рукописных подписей.....	38
3.4 Обоснование выбора технологий и средств разработки онлайн-сервиса верификации рукописной подписи	41
3.5 Требования к аппаратно-программному обеспечению онлайн-сервиса верификации рукописной подписи	42
3.5.1 Требования к аппаратно-техническому обеспечению	42
3.5.2 Требования к системному программному обеспечению	42
3.5.3 Требования к прикладному программному обеспечению.....	42
4 Разработка информационных и программных средств онлайн-сервиса верификации рукописной подписи	44
4.1 Разработка физической схемы базы данных.....	44
4.2 Разработка архитектуры развертывания информационных и программных средств онлайн-сервиса верификации рукописной подписи	47
4.3 Разработка программных средств взаимодействия с базой данных пользователей и эталонных рукописных подписей.....	50
4.4 Разработка библиотеки методов верификации рукописной подписи	53
4.4.1 Разработка структуры библиотеки методов верификации рукописной подписи.....	54
4.4.2 Разработка методов верификации подписи.....	59
4.5 Разработка программных средств Restful сервера.....	60
4.5.1 Разработка архитектуры сервера.....	62
4.5.2 Разработка программных средств взаимодействия сервера с клиентскими приложениями	64
4.5.3 Разработка программных средств взаимодействия сервера с базой данных	65
4.6 Разработка клиентского приложения.....	67

5	Контроль качества программных средств онлайн-сервиса верификации рукописной подписи	70
5.1	Планирование процесса контроля качества программных средств	70
5.1.1	Обоснование выбора модели тестирования	70
5.1.2	Обоснование выбора метрик программного кода	70
5.2	Тестирование программных средств	72
5.2.1	Модульное тестирование	72
5.2.2	Функциональное тестирование.....	75
5.2.3	Тестирование сервиса верификации на точность.....	75
5.3	Оценка качества программного кода	76
	Заключение	79
	Список использованных источников	80
	Приложение А. Глоссарий	85
	Приложение Б. Листинг приложения.....	87
	Приложение В. Диаграмма классов	93
	Приложение Г. Результаты тестирования	98

Введение

В последние годы, наряду с растущей потребностью в личной верификации во многих повседневных приложениях, проблема автоматической проверки рукописной подписи рассматривается с новым интересом. Требования безопасности современного общества поставили биометрию в центр широких дискуссий [1, 2].

С практической точки зрения, биометрическая система – это система распознавания шаблонов, которая распознает личность пользователя на основе физиологических или поведенческих характеристик. Таким образом, биометрические системы можно разделить на две основные категории:

- физиологическая биометрия основана на распознавании какой-либо части человеческого тела, таких как отпечаток пальца, сетчатка глаза, сканирование вен и т.д.;
- поведенческая биометрия основана на измерении некоторых характеристик поведения человека, таких как подпись, голос и т.д.

Биометрические системы функционируют в двух режимах: идентификация и верификация. В режиме идентификации определяется, какой именно пользователь среди всего множества известных пользователей предоставляет заведомо верный биометрический параметр. В режиме верификации происходит проверка, предоставлен ли биометрический параметр конкретным известным пользователем или является подделкой. Так как данные процессы тесно связаны друг с другом, а для их решения могут даже использоваться одни и те же алгоритмы, далее данные термины будут использоваться как взаимозаменяемые.

Оценка биометрии – многогранная проблема. Так как биометрический признак должен быть универсальным, т.е. каждый человек должен обладать этим признаком; уникальным, т.е. никакие два человека не должны иметь идентичные признаки; постоянным, т.е. признак не должен ни изменяться, ни быть изменчивым; также признак должен быть легко получаемым. Также нужно учитывать такие характеристики, как точность, экономичность, скорость, приемлемость для пользователей и т.д.

Проверка рукописной подписи является одним из самых распространенных способов идентификации и верификации человека, она активно используется в автоматизации документооборота, например, при обращении в государственные органы, в банковской сфере или судебных разбирательствах. Основное преимущество рукописной подписи для идентификации личности – высокий показатель доверия у людей. После внедрения автоматической проверки собственноручной подписи в любой бизнес привычный рабочий процесс существенно не меняется и легко внедряется в офисную работу. Поэтому актуальна разработка и внедрение в практику постоянно доступных сервисов верификации рукописной подписи, которые помогут автоматизировать процесс распознавания личности для различных сфер применения. Такие сервисы могут быть ориентированы на заинтересованные лица и организации, в задачи которых входят, в том числе регистрация и идентификация личности по рукописной подписи, применительно к клиентам, сотрудникам, контрагентам и другим личностям.

Несмотря на имеющиеся достижения и наработки в данной сфере, проблема создания такого сервиса остается актуальной по ряду причин. К ним можно отнести многообразие подходов и алгоритмов к верификации рукописной подписи, многокритериальность и нетривиальность выбора наиболее эффективного подхода и алгоритма.

Также стоит отметить ограниченность программных продуктов и решений данного класса. Полное отсутствие кроссплатформенных продуктов с открытым программным кодом, способных к развитию и адаптации сторонними разработчиками с учетом собственных целей и задач, особенностей предметной области.

Целью выпускной квалификационной работы является разработка онлайн-сервиса верификации рукописной подписи в различных сферах человеческой деятельности (госуслуги, банковская сфера, судебное производство, подача документов в организациях и пр.). Разработка включает в себя анализ требований, проектирование, реализацию и тестирование программного обеспечения.

1 Анализ предметной области и постановка задачи на разработку онлайн-сервиса верификации рукописной подписи

1.1 Основы верификации рукописной подписи

Верификация подписи может быть выполнена в двух режимах: автономном (статическом) или онлайн (динамическом). В автономном режиме человек расписывается на бумаге, подпись сканируется или фотографируется, система извлекает параметры подписи из ее изображения. В онлайн-режиме человек расписывается на устройстве (графическом планшете), система при этом снимает дополнительную информацию (время, давление, азимут и пр.).

Было доказано, что методы онлайн-верификации подписи по качеству получаемых результатов значительно превосходят методы автономной верификации [3]. Основное преимущество цифровой рукописной подписи состоит в том, что ее экспертизу и в следствие верификацию можно проводить практически мгновенно, используя программный подход. Процесс распознавания данного вида подписи представляет собой сбор информации с электронного устройства (чаще графический планшет и стилус), который регистрирует не только очертания символов подписи, но и другие параметры, которые анализируются в процессе экспертизы подписи: положение конца ручки (стилуса) в определённые моменты времени, угол наклона ручки и оказываемое на планшет давление.

Также статический метод дает меньше информации по сравнению с динамическим методом, так как известны лишь координаты точек.

Так как подпись считается баллистическим движением, время подписи несет важную информацию. Во время совмещения двух подписей значительная разница в скорости и времени двух подписей, даже если они имеют одинаковую форму, приводит к высокой оценке различия между ними. Информацию о форме можно легко скопировать, если фальсификатор получит образец, в то время как информацию о времени труднее имитировать (и заметить).

В пользу приведенных выше аргументов был выбран онлайн подход к верификации подписи.

В онлайн-режиме верификации подписи применяются два основных способа представления рукописной подписи:

- параметрический подход, в котором подпись описывается как вектор, элементы которого соответствуют отдельным характеристикам подписи;
- функциональный подход, при котором подпись описывается функцией времени, значения которой могут быть, например, координаты, скорость, давление и т.д.

Характеристики (параметры) подписи при параметрическом подходе подразделяются на локальные и глобальные. Локальные параметры описывают каждую отдельную точку подписи. Глобальные параметры описывают подпись в целом и процесс ее написания, например, общее затраченное время, среднее давление, средняя скорость и т.д.

Несмотря на то, что некоторые недавние работы утверждают, что подходы, основанные на формах, в некоторых условиях могут составить конкуренцию методам, основанным на функциях, последние в большинстве случаев дают лучшие результаты [4]. Таким образом, проанализировав различные подходы, существующие в онлайн верификации рукописной подписи, было решено выбрать функциональный подход.

Способы представления рукописной подписи, качество и полнота получаемых характеристик подписи оказывают значительное влияние на выбор алгоритма распознавания подписи. Одной из важнейших задач программных средств распознавания подписи является поддержка множества алгоритмов верификации и выбор наиболее эффективного алгоритма.

1.2 Сравнительный анализ аналогичных программных средств

Современные работы по теме исследования условно можно разделить на две основные группы:

- исследование и совершенствование методов верификации рукописной подписи;
- разработка программных средств идентификации и верификации рукописной подписи.

Среди наиболее известных программных средств идентификации и верификации рукописной подписи можно выделить такие, как Wacom Ink SDK for verification [5] от компании Wacom, решения от компании Progress Soft [6], SIGNificant от компании Namirial S.p.A. [7]. Рассмотрим их далее.

1.2.1 Wacom Ink SDK for verification

Программные средства Wacom Ink SDK for verification представляют собой набор платных программных инструментов для проверки подлинности рукописных подписей, которые поддерживает два режима проверки, которые могут использоваться по отдельности и/или в комбинации:

- динамическая проверка подписи путем сравнения с 4 – 12 эталонными вариантами подписи;
- статическая проверка постфактум путем сравнения изображений подписи с изображениями известных образцов.

Модуль проверки SDK сравнивает поступившую подпись с эталонными примерами подписи, которые должны быть ранее зарегистрированы в базе данных. Шаблон эталонной подписи может быть обновлен.

Имеется возможность настраивать «терпимость» программы к ложным подписям, например, увеличить вероятность принятия подписи, пожертвовав растущей ошибкой второго рода.

1.2.2 Progress Soft

Компания Progress Soft предлагает два платных специализированных решения для верификации рукописных подписей, ориентированных на финансовые организации (банки, фондовые компании и т.д.):

- решение «Автоматизированная верификация и управление подписями» (PS-SIG) автоматизирует процесс проверки подлинности

подписей и предоставляет инструментарий для извлечения из БД, управления и хранения неограниченного количества подписей;

– решение «Автоматизированная верификация подписей» (PS-ASV) осуществляет интеллектуальный анализ путем сравнения подписей, извлеченных из электронных чеков или официальных документов, с имеющимися образцами подлинных подписей.

Решение PS-SIG автоматизирует рабочий процесс верификации подписей и обрабатывает все относящиеся к подписи данные, обеспечивая доступ к образцам подлинных подписей. PS-SIG включает в себя модуль клиентских профилей, в котором хранятся подписи и данные клиентов финансовых учреждений. Решение позволяет пользователю-сотруднику банка вести карточки с образцами подписей и управлять подписями клиентов, а также устанавливать лимиты по счетам и платежные правила. Кроме того, во время верификации подписи для установления подлинности имеющейся информации решение автоматически проверяет полномочия подписанта и их действительность по заранее установленным правилам.

PS-SIG сохраняет все изменения данных для каждого экземпляра подписи и определяет дату истечения срока действия для каждого подписавшего. Пользователи решения PS-SIG могут выполнять различные действия в системе, в частности создавать, изменять, закрывать и удалять учетные записи с двумя уровнями авторизации: «создатель» и «проверяющий», и утверждать подпись в соответствии с матрицей полномочий до ее размещения в базе данных.

Решение PS-ASV работает в режиме сквозной автоматизированной обработки и использует алгоритмы распознавания образов, в частности машинное обучение, для сопоставления подписей. Принятие или отклонение извлеченной подписи осуществляется на основании политик и процедур безопасности финансовых учреждений.

1.2.3 SIGNificant

Платный продукт SIGNificant от Namirial S.p.A. предназначена для сбора поведенческих биометрических данных рукописной подписи (скорость, ускорение, ритм, движения в воздухе и давление) и встраивания подписи в электронный документ.

В процессе регистрации (сбора образцов подписей) пользователя требуется до шести подписей, которые собираются с помощью стандартного планшета или специализированного планшета для подписи. Система может обрабатывать множество профилей для каждого пользователя, позволяя, например, использовать один профиль для стандартной подписи, а другой – для подписи только инициалами. При проверке подписи SIGNificant Biometric Server сравнивает подписи с соответствующими профилями.

Собранные данные биометрической подписи автоматически динамически сверяются с образцом подписи, хранящимся в зашифрованном виде в системной базе данных. Каждый раз, когда пользователь обращается к системе для проверки своей подписи, биометрический сервер сравнивает текущую подпись с профилями подписи. При каждой аутентификации сервер продолжает изучать и настраивать профиль пользователя. Это позволяет системе отслеживать постепенные изменения рукописной подписи с течением времени. Кроме того, контрольный журнал на стороне сервера отслеживает каждое действие, позволяя проводить статистический анализ исторических данных или проверять конкретный результат проверки.

Сведем результаты анализа в таблицу 1.

Таблица 1 – Сравнительный анализ продуктов-аналогов

Критериальная оценка	Wacom Ink SDK for verification	Progress Soft	SIGNificant
Наличие профиля пользователя	Нет	Предоставляется программным средством PS-SIG	Да
Возможность выбора применяемого алгоритма верификации	Нет	Нет	Нет

Продолжение таблицы 1

Критериальная оценка	Wacom Ink SDK for verification	Progress Soft	SIGNificant
Возможность обновления шаблона эталонной подписи	Да	Да	Нет
Настройка порогового значения верификации («терпимость»)	Да	Нет	Да
Журнализация обращений к подписи	Нет	Да	Да
Открытый программный код	Нет	Нет	Нет

Можно констатировать, что рассмотренные программные продукты обладают базовыми функциями верификации рукописной подписи. При этом необходимо отметить следующие проблемы и ограничения применения указанных продуктов:

- ограниченные возможности интеграции со сторонними приложениями и системами;
- отсутствие возможностей по развитию приложений сторонними разработчиками для адаптации их под специфику собственных задач и особенностей предметных областей;
- платные лицензии на распространение продуктов и отсутствие открытого программного кода.

После проведения анализа, выделена необходимость разработать продукт, который будет покрывать указанные недостатки.

1.3 Постановка задачи на разработку онлайн-сервиса верификации рукописной подписи

Основная задача сервиса онлайн верификации рукописной подписи состоит в предоставлении удобного API для внедрения в любую информационную систему. Для составления образа подписи, человеку нужно предоставить несколько ее экземпляров (5 – 15 примеров). После этого данный субъект будет зарегистрирован в системе. После его регистрации, организация, которая предоставила данные этого человека, имеет право

направить запрос на его верификацию или идентификацию по подписи. Также организация может запросить шаблон подписи определенного лица.

Необходимо предоставить интуитивно простой интерфейс для взаимодействия с библиотекой.

Таким образом, разрабатываемое приложение должно решать следующие задачи:

- загрузка данных подписи, регистрация человека;
- верификация человека по подписи;
- настройка параметров входа для конкретного человека;
- предоставление данных о подписи человека по запросу от организации.

Сервис предназначен для его встраивания в любую систему, которая хочет поддерживать верификацию по рукописной подписи. Необходимо предоставить API для отправки данных подписи на сервер, без участия пользователя необходимо обработать подпись и внести в базу данных.

Процесс верификации и идентификации необходимо проводить также просто, предоставив API для отправки данных верификации. Также должна быть возможность запросить эталонные данные, введенные при регистрации (образ подписи).

В случае изменения образа подписи необходимо обновить данные о подписи в системе. Например, это может произойти при смене фамилии или просто при смене самого рисунка подписи.

Также нужно предусмотреть наличие своеобразного модератора, пользователя, который может менять параметры входа конкретного лица.

Если у организации возникнет желание удалить данные о том или ином пользователе, нужно предоставить такую возможность.

Сервис должен удовлетворять следующим требованиям:

- код сервиса должен быть легко поддерживаемым и легко расширяемым;
- код должен соответствовать принципам SOLID;

– сервис должен предоставлять удобное API для авторизации и регистрации пользователей.

1.4 Планирование процесса разработки онлайн-сервиса верификации рукописной подписи

Планирование разработки и оценка бюджета была выполнена при помощи программы для управления проектами Microsoft Project. Проект был начат 23 февраля 2022 года.

При планировании разработки было выделено 25 задач. Список задач и их описание представлено на рисунке 1.

Режим задачи	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало проекта	0 дней	Ср 23.02.22	Ср 23.03.22		
2	Анализ требований	22 дней	Ср 23.02.22	Чт 24.03.22		
3	Анализ предметной области	2 дней	Ср 23.02.22	Вт 01.03.22	1	Интернет; Аналитик
4	Сравнительный анализ программ аналогов	5 дней	Ср 02.03.22	Вт 08.03.22	3	Аналитик[30%]; Программист 1; Интернет[100,00 Р]
5	Анализ способов верификации рукописной подписи	5 дней	Вт 01.03.22	Пн 07.03.22	3	Программист 2
6	Анализ методов проектирования сервиса верификации	4 дней	Пн 07.03.22	Чт 10.03.22	5	Программист 2
7	Постановка задачи	4 дней	Чт 10.03.22	Вт 15.03.22	4;6	Аналитик
8	Анализ функциональных требований	7 дней	Ср 16.03.22	Чт 24.03.22	7	Аналитик[70%]; Программист 2[30%]
9	Анализ требований и постановка задачи завершена	0 дней	Чт 24.03.22	Чт 24.03.22	8	
10	Проектирование программных средств	27 дней	Пт 25.03.22	Чт 05.05.22	2	
11	Проектирование структур данных сервиса	10 дней	Пт 25.03.22	Пт 08.04.22	9	Аналитик[50%]; Программист 2[50%]; Интернет
12	Проектирование REST API приложения	10 дней	Пн 11.04.22	Ср 04.05.22	11	Программист 2; Интернет[100,00 Р]
13	Проектирование клиентского приложения	6 дней	Чт 28.04.22	Чт 05.05.22		Программист 1; Интернет[100,00 Р]
14	Проектирование программных средств завершено	0 дней	Ср 04.05.22	Ср 04.05.22	13	
15	Разработка программных средств	19 дней	Чт 05.05.22	Чт 02.06.22		
16	Разработка структур данных	7 дней	Чт 05.05.22	Пт 13.05.22	14	Программист 2[70%]; Программист 1[30%]; Интернет[100,00 Р]
17	Разработка REST API приложения	14 дней	Пн 16.05.22	Чт 02.06.22	16	Программист 2; Интернет[300,00 Р]
18	Разработка клиентского приложения	9 дней	Пн 16.05.22	Чт 26.05.22		Программист 1; Интернет[250,00 Р]
19	Разработка программных средств завершена	0 дней	Чт 02.06.22	Чт 02.06.22	18;17	
20	Тестирование	6 дней	Чт 02.06.22	Чт 09.06.22		
21	Модульное тестирование	3 дней	Чт 02.06.22	Пн 06.06.22	19	Программист 1[80%]; Тестировщик[20%]
22	Функциональное тестирование	3 дней	Чт 02.06.22	Пн 06.06.22	19	Тестировщик[80%]
23	Нагрузочное тестирование	3 дней	Вт 07.06.22	Чт 09.06.22	21;22	Тестировщик; Интернет[100,00 Р]
24	Тестирование завершено	0 дней	Чт 09.06.22	Чт 09.06.22	23	
25	Конец проекта	0 дней	Чт 09.06.22	Чт 09.06.22	24	

Рисунок 1 – Список задач проекта

Для реализации проекта необходимо выделить основные ресурсы, их типа и стоимость. Описание ресурсов представлено на рисунке 2.

	Название ресурса	Тип	Единицы измерения материала	Краткое название	Группа	Макс. единица	Стандартная ставка	Ставка сверхурочной	Затраты на исполнителя	Начисление	Базовый календарь	Ко
1	Аналитик	Трудовой		A		100%	000,00 Р/мес	100,00 Р/ч	0,00 Р	Пропорционал	Стандартный	
2	Программист 1	Трудовой		P1		100%	000,00 Р/мес	150,00 Р/ч	0,00 Р	Пропорционал	Стандартный	
3	Программист 2	Трудовой		P2		100%	000,00 Р/мес	200,00 Р/ч	0,00 Р	Пропорционал	Стандартный	
4	Тестировщик	Трудовой		T		100%	000,00 Р/мес	100,00 Р/ч	0,00 Р	Пропорционал	Стандартный	
5	Интернет	Затраты		I						Пропорционал		

Рисунок 2 – Список ресурсов проекта

В результате выделения задач и ресурсов была построена диаграмма Ганта (рисунок 3). На ней представлены продолжительность задач и их взаимосвязи друг с другом.

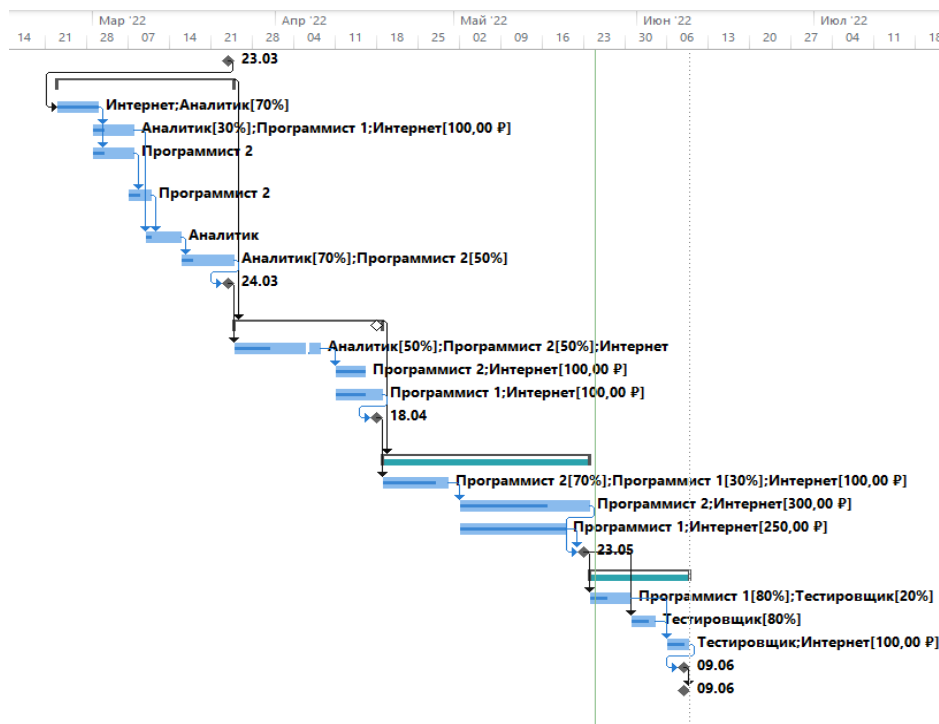


Рисунок 3 – Диаграмма Ганта

На рисунке 4, 5 представлены гистограмма загруженности ресурсов и гистограмма затрат на задачи соответственно.

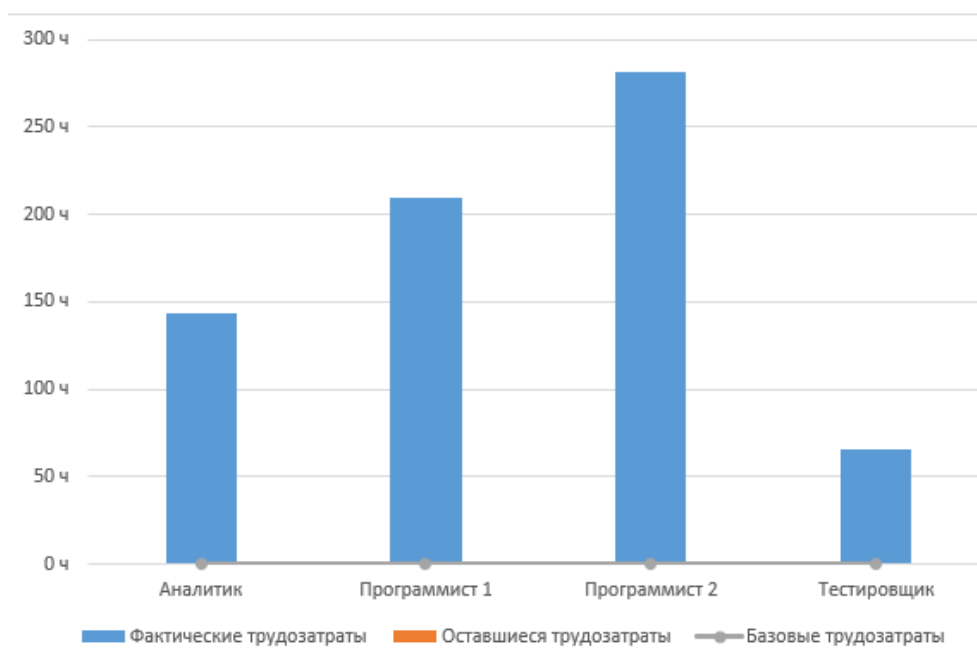


Рисунок 4 – Гистограмма загруженности ресурсов

Таким образом разработка проекта составит 72 полных рабочих дней с 23 февраля 2022 года по 9 июня 2022 года. Трудозатраты составят 700 человеко-часов.

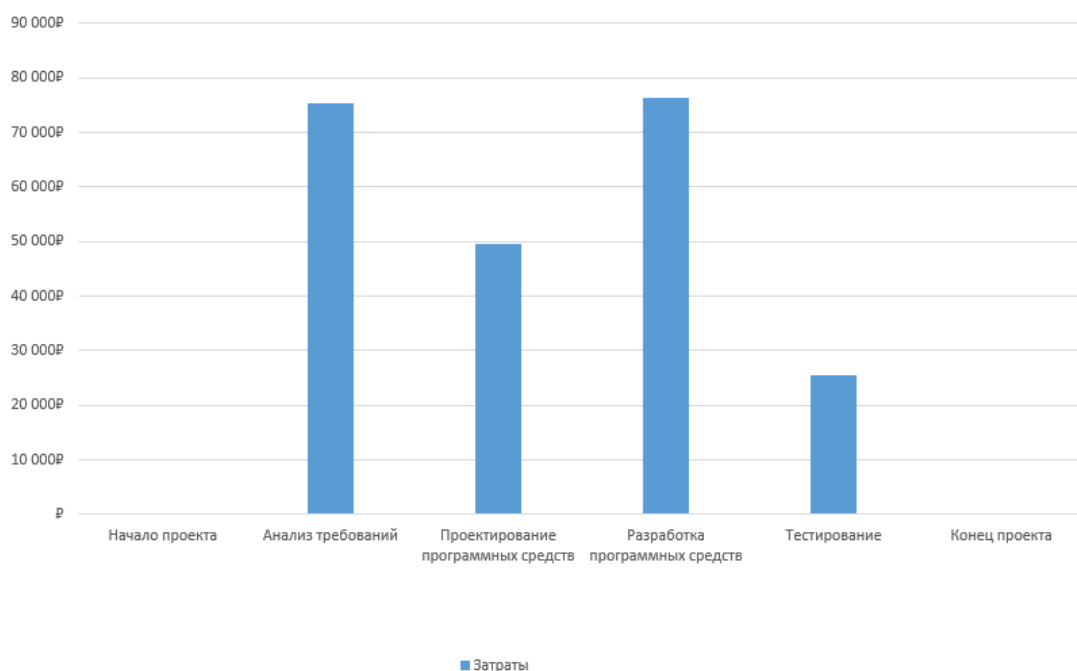


Рисунок 5 – Гистограмма затрат на задачи

Оценка бюджета проекта составляет 226 600 рублей.

2 Обоснование выбора алгоритмов верификации рукописной подписи для реализации в рамках онлайн-сервиса

Онлайн-проверку подписи можно разделить на четыре основных этапа: сбор данных, предварительная обработка, извлечение признаков и проверка. Сбор данных может осуществляться различными электронными устройствами. После сбора данных к данным подписи применяются методы предварительной обработки, чтобы уменьшить шум и определенные различия между подписями. Эти различия между подписями одного и того же человека возникают из-за того, что подписант не может каждый раз в точности повторить подпись, ее масштаб и положение на листе. Таким образом, предварительная обработка является важным шагом для улучшения результатов проверки. Следующий шаг – извлечение признаков. Онлайн-подписи содержат множество характеристик, таких как положение, скорость, давление, ускорение пера и т.д., которые можно использовать во время верификации процесса. Последним шагом является фактическая проверка, при

которой применяются методы измерения сходства и классификации, чтобы отличить подделку от оригинала.

Выбор алгоритмов для последнего шага верификации является нетривиальной задачей. Этой проблеме будет посвящена данная глава выпускной квалификационной работы.

2.1 Классификация и выбор критериев оценки алгоритмов верификации рукописной подписи

Стоит упомянуть, что при оценке качества того или иного метода верификации используются два типа ошибок: ложное отклонение (ошибки I-ого рода) и ложное принятие (ошибки II-ого рода). Ложное отклонение происходит, когда аутентичная подпись отклоняется, а ложное принятие происходит, когда поддельная подпись считается аутентичной.

В зависимости от приложений необходимо определить компромисс между двумя типами ошибок, поскольку любое уменьшение процента ложных отказов (FRR) увеличивает процент ложных принятий (FAR), и наоборот. Когда FRR равен проценту ложных принятий FAR, это называется равным коэффициентом ошибок. Равный уровень ошибок (коэффициент EER) – это коэффициент, при котором оба вида ошибок равны.

Ученые, изучающие технологии верификации подписи, имеют свой взгляд на ошибки I-ого и II-ого рода. Оценка FAR сложна и, как правило, неточна, так как подделка подписи является результатом поведенческой деятельности, она сильно зависит от типа и объема информации, предоставленной фальсификатору, а также от его подготовки и усилий.

В литературе выделяют следующие виды подделок:

- случайные подделки производятся случайным образом без какой-либо информации об имени подписанта, она производится, когда фальсификатор не имеет доступа к изображению или каким-либо данным, получившаяся подпись может отличаться по форме и размеру от подлинной подписи;

- подделка с нулевым усилием может иметь такое же семантическое значение, что и подлинная, но общая форма и размер могут отличаться;
- профессиональная подделка получается, когда фальсификатор владеет достаточным количеством информации для искусного копирования подписи.

Таким образом, для сложного анализа алгоритма верификации необходимо иметь не только базу данных подлинных подписей, но и базу данных подделок.

В качестве критериев оценки можно выбрать теоретическую точность алгоритмов, то есть EER, взятый из литературы. Также в алгоритме в целом необходимо учитывать его обобщающую способность, т.е. возможность алгоритма работать с данными никогда ему не встречавшимися.

2.2 Обзор и теоретическая оценка алгоритмов верификации рукописной подписи

В процессе проверки подлинность тестовой подписи оценивается путем сопоставления ее характеристик с теми, которые хранятся в базе знаний, разработанной на этапе регистрации. Этот процесс выдает один ответ (булево значение), в котором указывается подлинность тестовой подписи. Процесс проверки включает в себя множество критических аспектов, начиная от метода сопоставления подписей и заканчивая стратегией, используемой при разработке базы знаний.

Системные алгоритмы распознавания подписи опираются на алгоритмы обработки образов или математические методы анализа кривых, это обусловлено тем, что подпись может быть представлена набором точек. Несмотря на то, что первые исследования на данную тему были представлены еще в 80-е годы прошлого века, анализ различных методов верификации подписи является до сих пор актуальной темой и в наши дни проводятся исследования новых методов и их комбинаций. Методы онлайн-верификации рукописной подписи можно разделить на три основные категории [8]: методы сопоставления с шаблоном, статистические методы, структурные методы.

В методах сопоставления с шаблоном верифицируемый экземпляр подписи сопоставляется с шаблонами подписей. Алгоритм динамического преобразования временно шкалы (DTW) находит наилучшее нелинейное выравнивание двух временных рядов, при котором общее расстояние между ними минимально. Он показал многообещающие результаты в области проверки подписи [9]. И поскольку он вычисляет расстояние между двумя подписями, обычно данный метод используется в подходах к классификации на основе пороговых значений, например, в [10, 11]. Сравнение полных временных последовательностей приводит к более высокой вычислительной нагрузке, а также к потере информации, связанной со структурной организацией подписей. Чтобы избежать деформации эталонных подписей при сопоставлении с предоставленными образцами, были разработаны асимметричные DTW [12, 13]. Подход, представленный в [13, 14], расширяет DTW за счет использования кодовых векторов, созданных с помощью метода векторного квантования (VQ). Комбинация методов оценки/голосования DTW/VQ показала улучшенный результат проверки подписи. Несмотря на то, что DTW был первый раз предложен для верификации подписи достаточно давно, он до сих пор популярен и его улучшению посвящены многие статьи [11 – 19].

В статистических методах оценивается один или несколько показателей, вычисленных статистическими алгоритмами. При использовании параметрического подхода к представлению подписи применяются статистические методы верификации подписи такие, как нейронные сети, скрытые марковские модели (HMM), метод опорных векторов (SVM), алгоритм случайного леса, метод ближайших соседей (kNN).

Наиболее распространенные статистические алгоритмы используют расстояния Махаланобиса и Евклида. При расчете расстояния Махаланобиса рассматривается полная ковариационная матрица для каждого набора подписей [20]. При использовании Евклидова расстояния рассматривается среднее значение для набора [21]. Подсчет расстояний также применяется в

методе ближайших соседей, где находится расстояние до подписей, которые находятся ближе всего к рассматриваемой в пространстве признаков [22].

В настоящее время реализованы несколько различных топологий скрытых марковских моделей: скрытые марковские модели с лево-правой топологией [23 – 25], эргодической [26], кольцевой, в которой разрешен переход из последнего состояния в первое состояние, но последняя больше применяется в офлайн верификации.

Машины опорных векторов (SVM) содержат алгоритмы обучения моделей с учителем, используемые в классификации. SVM может отображать входные векторы в пространство более высокой размерности, в котором классы определяются при помощи максимальной разделяющей гиперплоскости и достаточно часто применяются для проверки подписи [27, 28]. Преимущества использования SVM заключаются в том, что они имеют выпуклую целевую функцию с эффективными алгоритмами обучения и хорошими свойствами обобщения [29].

Применение нейронных сетей в сфере верификации подписи долгое время было затруднено, так как в свободном доступе для исследователей не было достаточно большой базы данных для обучения моделей, поскольку подписи не являются информацией общедоступной в интернете, как это происходит с другими биометрическими характеристиками, например, лицами [30]. В настоящее время наблюдается большой прогресс в этой области. В [31] для классификации использовалась нейронная сеть вида многослойный перцептрон (MLP) с одним входным слоем, тремя скрытыми слоями и одним выходным слоем. Также для верификации могут использоваться сиамские сети [32] – это особый вид нейронных сетей, которые состоят из двух входных полей, за которыми следуют две идентичные нейронные сети с общими весами. Целью этой сети является вычисление вероятности того, что два входа относятся к одному и тому же классу. В работе [33] была представлена новая рекуррентная нейронная сеть с обратным распространением ошибки (RNN) для повышения эффективности проверки. Также в [30] был предложен новый подход комбинации выравнивания подписи по времени с RNN.

Усовершенствованной архитектурой RNN являются сети с долгой краткосрочной памятью (LSTM), эксперимент по использованию такой сети был проведен в работе [34], в ней же был сделан вывод, из которого следовало, что хотя LSTM работает идеально, если для обучения также использовались подделки исследуемых подписей, в более реалистичных сценариях они обычно терпят неудачу. В реальном случае образцы подделок недоступны, поэтому сеть должна изучать подпись на основе другой информации, оригинальных подписях. При попытке обучить сеть общего назначения отличать подделки от подлинных подписей на большом обучающем множестве или обучить сеть только на подлинных подписях исследователи терпят неудачи. В этом состоит главный недостаток моделей такого рода.

К структурным методам верификации подписи относятся методы сопоставления строк, графов и деревьев, которые обычно используются в сочетании с другими методами. Например, сопоставление строк [35] может использоваться в сочетании с нейросетевым подходом [26]. В отдельных исследованиях граф структурного описания используется для проверки структурной организации сомнительной подписи [36, 37].

В таблице 2 представлено сравнение методов, упомянутых выше на основе информации, предоставленной в статьях. Чаще в работах применяется база SVC2004 [38]. Во время анализа статей, если в работе были упомянуты несколько баз, то для сравнения выбиралась именно SVC2004.

Таблица 2 – Оценка методов верификации из представленных статей

Ссылка на статью	Метод верификации	EER
Song X. et al. [10]	DTW с ограничением кривых подписи	2,89%
Fischer A. et al. [11]	DTW	Для случайных подделок – 1,34%, для профессиональных подделок – 3,58%
Feng H., Wah C. C. [13]	DTW/VQ	Для случайных подделок – 1,32%, для профессиональных подделок – 5,42%
Sharma A., Sundaram S. [14]	DTW/VQ	2,73%
Ansari A. Q. et al. [15]	DTW + нечеткое моделирование подписи	2,46%

Продолжение таблицы 2

Ссылка на статью	Метод верификации	EER
Huang K., Yan H. [16]	DTW	4%
Saleem M., Kovari B. [17]	DTW с индивидуальной частотой дискретизации	1,8%
Parziale A. et al. [19]	DTW	Для случайных подделок – 1,30%, для профессиональных подделок – 3,09%
Nyssen E. et al. [20]	Многоступенчатая верификация с использованием расстояния Махаланобиса	FAR = 0%, FRR = 5,8%
Kashi R. S. et al. [21]	Метод Евклидова расстояния	4,5%
Saleem M., Kovari B. [22]	KNN	4% при наличии 10 эталонных подписей
Kashi R. S. et al. [23]	HMM	2,5%
Igarza J. J. et al. [24]		9,25%
Zou M. et al. [25]		FAR = 2%, FRR = 11,3%
Quan Z. H., Liu K. H. [26]	HMM/ANN	0,12%
Rohilla S., Sharma A. [27]	SVM	FAR = 7%, FRR = 2,26%
Gruber C. et al. [28]	SVM	Для случайных подделок – 0,15%, для профессиональных подделок – 6,91%
Tolosana R. et al. [30]	RNN	Для случайных подделок – 0,6%, для профессиональных подделок – 3,3%
Hefny A., Moustafa M. [31]	MLP	0,49%
Lai S., Jin L [33]	RNN	2,37%
32. Ahrabian K. [32]	Сиамская сеть	FAR = 8,6%, FRR = 8,7%
Otte S et al. [34]	LSTM	23,75%
Chen Y., Ding X. [35]	Сопоставление строк последовательности	4,7%

Методы онлайн верификации подписи заметно усложняются и повышают свое качество. В настоящее время активно применяются новейшие разработки в области машинного обучения. Качество моделей также заметно повышается из-за появления более доступных для исследователей баз данных.

2.3 Экспериментальная оценка алгоритмов верификации рукописной подписи

Способы представления рукописной подписи, качество и полнота получаемых характеристик подписи оказывают существенное влияние на выбор алгоритма распознавания подписи. Одной из наиболее важных задач

программного обеспечения для распознавания подписи является поддержка нескольких алгоритмов проверки и выбор наиболее эффективного алгоритма.

Для обоснования выбора методов верификации был проведен эксперимент, в котором были реализованы некоторые методы верификации.

В качестве платформы для проведения эксперимента была выбрана среда Jupyter Notebook. В качестве базы данных подписей выступила DeepSignDB [29]. База данных DeepSignDB включает в общей сложности 1526 пользователей из пяти различных баз. На рисунке 6 графически представлены состав базы, устройства сбора данных и инструменты захвата подписи.

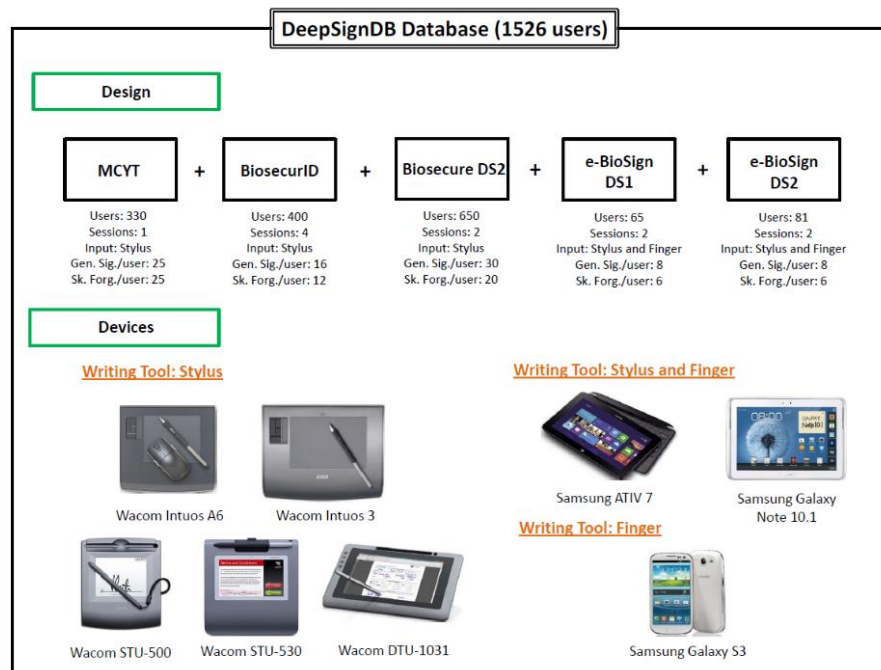


Рисунок 6 – Графическое описание базы данных DeepSignDB

Все данные были нормализованы и интерполированы до 256 точек.

На вход моделям были поданы подписи пользователей и их идентификаторы. Стоит заметить, что фальсифицированные подписи базы представляют собой квалифицированную подделку и ее определение является нетривиальной задачей. Решение отойти от реализации структурных методов было обусловлено тем, что данные методы чаще применяются комбинации, в дополнение к основному методу. В таблице 3 представлены результаты экспериментов. Определены проценты ложноположительных образов (FAR) и ложноотрицательных (FRR), а также общая точность модели (Acc).

Таблица 3 – Результаты экспериментальной оценки методов верификации

	KNN		SVM		NN		Random Classifier	Forest
	FAR	FRR	FAR	FRR	FAR	FRR	FAR	FRR
	0,0378	0,1322	0,1639	0,0826	0,2689	0,1322	0,1639	0,2025
Acc	0,9146		0,8770		0,8000		0,8167	

Лучше всего в задаче верификации себя показал метод ближайших соседей (KNN).

2.4 Результаты оценки и обоснование выбора алгоритмов верификации рукописной подписи

Был проведен сравнительный анализ методов и алгоритмов верификации рукописной подписи. Проведенный анализ позволил выявить спектр алгоритмов, подходящих для реализации в рамках библиотеки.

Было решено реализовать статистические методы (метод опорных векторов, градиентный бустинг) и метод сопоставления с шаблоном (алгоритм динамической трансформации временной шкалы с применением kNN). Подробнее о них далее.

Метод опорных векторов или SVM (от англ. Support Vector Machines) – это набор методов машинного обучения с учителем, используемых в задачах классификации, регрессии и для обнаружения выбросов в данных. Таким образом, Основной задачей SVM как классификатора является нахождение уравнения разделяющей гиперплоскости:

$$\omega_1 x_1 + \omega_2 x_2 + \dots + \omega_n x_n + \omega_0 = 0$$

в пространстве R^n , которая бы разделила два класса неким оптимальным образом.

Преимущества метода опорных векторов:

- эффективен в пространствах больших размеров;
- может быть эффективен в случаях, когда количество измерений больше, чем количество образцов, то есть количество признаков может оказаться больше количества предоставленных пользователем подписей;

– использует подмножество обучающих точек в функции принятия решений (называемых опорными векторами), поэтому метод эффективен с точки зрения памяти.

К недостаткам опорных векторных машин можно отнести:

– если количество функций намного превышает количество выборок, обязательным становится применение регуляризации, а также следует соблюдать осторожность при выборе ядра во избежание переобучения;

– SVM не предоставляет оценки вероятностей, они рассчитываются с использованием дорогостоящей перекрестной проверки [41].

Если обучение является линейным разделимым, то можно выбрать две параллельные гиперплоскости таким образом чтобы они разделяли множество точек на два класса и между плоскостями не было точек.

Сначала алгоритм тренируется на объектах из обучающей выборки, для которых заранее известны метки классов. Далее уже обученный алгоритм предсказывает метку класса для каждого объекта из отложенной/тестовой выборки. Пусть метки классов могут принимать значения $Y = \{-1, +1\}$. Объект – вектор с N признаками $x = (x_1, x_2, \dots, x_n)$ в пространстве R^n . При обучении алгоритм должен построить функцию $F(x) = y$, которая принимает в себя аргумент x – объект из пространства R^n и выдает метку класса y .

Общий вид преобразования F объекта X в метку класса y :

$$F(x) = \text{sign}(w^T x - b),$$

где $w = (w_1, w_2, \dots, w_n)$;

$b = -w_0$ из формулы выше.

После настройки весов алгоритма w и b (обучения), все объекты, попадающие по одну сторону от построенной гиперплоскости, будут предсказываться как первый класс, а объекты, попадающие по другую сторону как второй класс.

В основе XGBoost, применяемого в сервисе лежит алгоритм градиентного бустинга деревьев решений. Градиентный бустинг – это техника машинного обучения для задач классификации и регрессии, которая строит

модель предсказания в форме ансамбля слабых предсказывающих моделей, обычно деревьев решений. Обучение ансамбля проводится последовательно. На каждой итерации вычисляются отклонения предсказаний уже обученного ансамбля на обучающей выборке. Следующая модель, которая будет добавлена в ансамбль будет предсказывать эти отклонения. Таким образом, добавив предсказания нового дерева к предсказаниям обученного ансамбля мы можем уменьшить среднее отклонение модели, которое является целью оптимизационной задачи. Новые деревья добавляются в ансамбль до тех пор, пока ошибка уменьшается, либо пока не выполняется одно из правил «ранней остановки» [42]. Ответы в XGBoost суммируются по всем деревьям ансамбля:

$$F(x) = \sum_{k=1}^K f_k(x).$$

Суммарная функция потерь в XGBoost выглядит следующим образом:

$$total_loss = \sum_{i=1}^N loss(y_i, F(x_i)) + \gamma \sum_{k=1}^K T_k + \frac{1}{2} \lambda \sum_{k=1}^K \|w_k\|^2,$$

где γ, λ – гиперпараметры;

T_k – количество листьев в k-м дереве ансамбля;

w_k – вектор, составленный из выходных значений на всех листьях k-го дерева.

Алгоритм динамической трансформации временной шкалы (DTW) вычисляет оптимальную последовательность трансформации (деформации) времени между двумя временными рядами. Алгоритм вычисляет оба значения деформации между двумя рядами и расстоянием между ними [43].

Предположим, что у нас есть две числовые последовательности $(a_1, a_2, \dots, a_n)$ и $(b_1, b_2, \dots, b_m)$. Длина двух последовательностей может быть различной. Алгоритм начинается с расчета локальных отклонений между элементами двух последовательностей, использующих различные типы отклонений. Самый распространенный способ для вычисления отклонений является метод, рассчитывающий абсолютное отклонение между значениями двух элементов (Евклидово расстояние). В результате получаем матрицу отклонений, имеющую n строк и m столбцов общих членов:

$$d_{ij} = |a_i - b_j|, i = \overline{1, n}, j = \overline{1, m}.$$

Минимальное расстояние в матрице между последовательностями определяется при помощи алгоритма динамического программирования и следующего критерия оптимизации:

$$a_{ij} = d_{ij} + \min(a_{i-1,j-1}, a_{i-1,j}, a_{i,j-1}),$$

где a_{ij} – минимальное расстояние между последовательностями $(a_1, a_2, \dots, a_n)$ и $(b_1, b_2, \dots, b_m)$.

Путь деформации – это минимальное расстояние в матрице между элементами a_{11} и a_{nm} , состоящими из тех a_{ij} элементов, которые выражают расстояние до a_{nm} .

Глобальные деформации состоят из двух последовательностей и определяются по следующей формуле:

$$GC = \frac{1}{p} \sum_{i=1}^p w_i,$$

где w_i – элементы, которые принадлежат пути деформации.

3 Разработка требований к онлайн-сервису верификации рукописной подписи

3.1 Функциональные требования к онлайн-сервису верификации рукописной подписи

Основная задача сервиса онлайн верификации рукописной подписи состоит в предоставлении удобного API для внедрения в любую информационную систему. После этого данный субъект зарегистрирован в системе. После его регистрации, организация, которая предоставила данные этого человека, имеет право направить запрос на его верификацию или на идентификацию заинтересованного лица по подписи. Организация может запросить шаблон подписи определенного лица.

Также существует администратор, данный пользователь имеет право добавлять образы подписи к индивиду, либо удалять их. Администратор имеет право изменять пороговые значения для входа человека в систему, например увеличить максимальное количество попыток входа или уменьшить пороговый процент схожести (если подпись совпадает больше чем на 70%, то вход разрешен, если процент меньше, то вход отклонен).

Так, можно выделить следующие категории пользователей (таблица 4).

Таблица 4 – Категории пользователей

№ п/п	Категория пользователей	Наименование приложения (подсистемы)	Фрагмент структуры ПрОбл	Информационные задачи
1	Организация	Сервис регистрации	Образ подписи	Регистрация подписи
		Сервис обработки данных	Образ подписи	Идентификация, верификация подписи
2	Администратор	Сервис базы данных	Образ подписи	Запрос подписи
		Сервис базы данных	Образ подписи	Изменить порог входа, удалить образ, добавить образ к пользователю

То есть, человек, которому принадлежит подпись не является пользователем системы как таковым. Он является ее действующим лицом, его данные содержатся в базе.

На основе сформулированных задач были выделены бизнес-процессы для сервиса верификации рукописной подписи (таблица 5). Сформулированные процессы позволят позже составить концептуальную модель базы данных и сущности предметной области. Бизнес-процессы описывают ограничения, налагаемые на базу данных, что важно при дальнейшей ее разработке. Также в их содержании мы видим будущие поля таблиц базы данных.

Таблица 5 – Бизнес-процессы сервиса

Бизнес-процесс	Исполнитель	Входные данные		Выходные данные		Ограничения
		Поставщик	Содержание	Потребитель	Содержание	
Регистрация	Организация	Пользователь	Данные пользователя (ФИО, паспортные данные, подпись)	Система	Образ подписи	Каждому пользователю должен быть назначен уникальный id
Изменение порога пользователя	Администратор	Администратор	id пользователя	Система	Измененный порог	Пользователь должен быть зарегистрирован
Идентификация	Организация	Организация	Образ подписи	Сервис обработки данных	Id пользователя	-
Верификация	Организация	Пользователь	Образ подписи, id пользователя	Сервис обработки данных	Подтверждение или отклонение подписи	Пользователь должен быть зарегистрирован
Регистрация	Организация	Пользователь	Данные пользователя (ФИО, паспортные данные, подпись)	Система	Образ подписи	Каждому пользователю должен быть назначен уникальный id
Удаление образа подписи	Администратор	Администратор	id пользователя, id подписи	Система	Список доступных образов	Пользователь должен быть зарегистрирован и иметь образ подписи
Добавление образа подписи	Администратор	Администратор	id пользователя, образ подписи	Система	Список доступных образов	Пользователь должен быть зарегистрирован

Данные бизнес-процессы являются глобальными для всего приложения и реализованы на стороне сервера БД и сервера приложений.

Согласно выделенным категориям пользователей и определенных для них задач, были выделены следующие функциональные требования:

- регистрация человека в системе;
- возможность обновить данные подписи;
- получение эталона подписи;
- изменение порога входа для человека в системе с помощью администратора;
- удаление данных о подписи или пользователе;
- идентификация человека;
- верификация человека.

На основе поставленных задач была составлена диаграмма вариантов использования [39], представленная на рисунке 7.



Рисунок 7 – Диаграмма вариантов использования REST сервиса

На диаграмме вариантов использования представлены функциональные требования к разрабатываемой системе. Подробное описание основных вариантов использования (прецедентов) представлено далее.

На рисунке 8 приведен сценарий регистрации человека в системе.

Прецедент: Зарегистрировать подпись заинтересованного лица в БД.
Краткое описание: Организация регистрирует в системе подпись клиента.
Главные актеры: Система, организация.
Основной поток: <ol style="list-style-type: none"> 1. Организация посылает запрос на регистрацию 2. Если пользователь существует <ol style="list-style-type: none"> 2.1. Система выдает предупреждение. 2.2. Предлагается обновить данные о подписи. 3. Иначе <ol style="list-style-type: none"> 3.1. Человек добавляется в систему.
Постусловия: <ol style="list-style-type: none"> 1. Образ подписи сохранен в БД системы.

Рисунок 8 – Сценарий варианта использования «Зарегистрировать подпись заинтересованного лица в БД»

В сценарии регистрации введена возможность обновления данных о пользователе в случае, если подпись сменилась.

На рисунке 9 приведен сценарий идентификации.

Прецедент: Идентифицировать человека по подписи.
Краткое описание: Загружается подпись для ее анализа и идентификации по ней человека.
Главные актеры: Система, организация.
Основной поток: <ol style="list-style-type: none"> 1. Организация посылает запрос с неизвестной подписью. 2. Модель, сохраненная в системе, определяет ближайший похожий образ подписи с вычислением процента уверенности. 3. Если процент больше или равен установленному порогу <ol style="list-style-type: none"> 3.1. Выводятся данные пользователя. 4. Иначе <ol style="list-style-type: none"> 4.1. Система выдает предупреждение о том, что человек не найден.
Постусловия: <ol style="list-style-type: none"> 1. Подпись подверглась анализу и много классовой классификации

Рисунок 9 – Сценарий варианта использования «Идентифицировать человека по подписи»

Сценарий идентификации имеет смысл применять только в крупных системах, где больше вероятность найти в базе данных нужного человека.

На рисунке 10 приведен сценарий верификации, который используется гораздо чаще.

Прецедент: Верифицировать человека по подписи.
Краткое описание: Загружается подпись и данные нужного лица.
Главные актеры: Система, организация.
Предусловие: Верифицируемый человек зарегистрирован в системе.
Основной поток: <ol style="list-style-type: none"> 1. Организация посылает запрос с подписью и данными верифицируемого человека. 2. Модель подвергает подпись анализу и выдает процент сходства поданной подписи с образом, зарегистрированным в системе. 3. Если процент сходства меньше заявленного в системе <ol style="list-style-type: none"> 3.1. Подпись отклоняется как сфальсифицированная. 4. Иначе <ol style="list-style-type: none"> 4.1. Человек верифицирован.

Рисунок 10 – Сценарий варианта использования «Верифицировать человека по подписи»

Данные варианты использования являются основными бизнес-процессами в сервисе по верификации рукописной подписи.

В результате анализа функциональных требований были выделены варианты использования программы. Для основных вариантов использования были представлены их спецификации, которые позволяют отобразить последовательность действий пользователя и системы при выполнении определенных функций.

3.2 Требования к архитектуре онлайн-сервиса верификации рукописной подписи

Для уменьшения трудозатрат на разработку сложного программного обеспечения необходимо использовать готовые унифицированные решения.

Паттерн (англ. design pattern) – повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста [40].

Model-View-Controller (MVC) – это фундаментальный паттерн, который нашел применение во многих технологиях.

Под Моделью, обычно понимается часть, содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта, данный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться.

В обязанности Представления входит отображение данных, полученных от Модели. Однако, представление не может напрямую влиять на модель. Можно сказать, что представление обладает доступом «только на чтение» к данным. В некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику.

Рассмотрим и сравним некоторые виды MVC паттерна.

Model-View-Presenter позволяет создавать абстракцию представления. Для этого выделяется интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель. Один экземпляр презентера связан с одним отображением.

Логика представления должна иметь ссылку на экземпляр презентера. Все события представления передаются для обработки в презентер и практически никогда не обрабатываются логикой представления (в т.ч. создания других представлений).

Model-View-ViewModel подход позволяет связывать элементы представления со свойствами и событиями View-модели. Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя. View-модель не имеет ссылки на интерфейс представления. Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку

используется механизм связывания данных. Один экземпляр View-модели связан с одним отображением.

Основная идея паттерна Model-View-Controller состоит в том, что и контроллер, и представление зависят от модели, но модель никак не зависит от этих двух компонент. Контроллер определяет, какие представление должно быть отображено в данный момент; События представления могут повлиять только на контроллер. Контроллер может повлиять на модель и определить другое представление. Возможно несколько представлений только для одного контроллера;

Контроллер перехватывает событие извне и в соответствии с заложенной в него логикой, реагирует на это событие изменяя Модель посредством вызова соответствующего метода.

Был выбран паттерн MVC из-за того, что он имеет больше возможностей по управлению представлением, а также в случае разработки сервиса верификации связь между представлением и другими частями приложения невозможна.

Частым примером использования может служить ASP.NET MVC.

Для работы сервиса необходимо клиентское приложение. Его главная функция – отправка данных подписи на сервер.

Слои Model, View, Controller естественно ложится на структуру web сервиса. На рисунке 11 представлена схема взаимодействия приложений с API по MVC архитектуре.

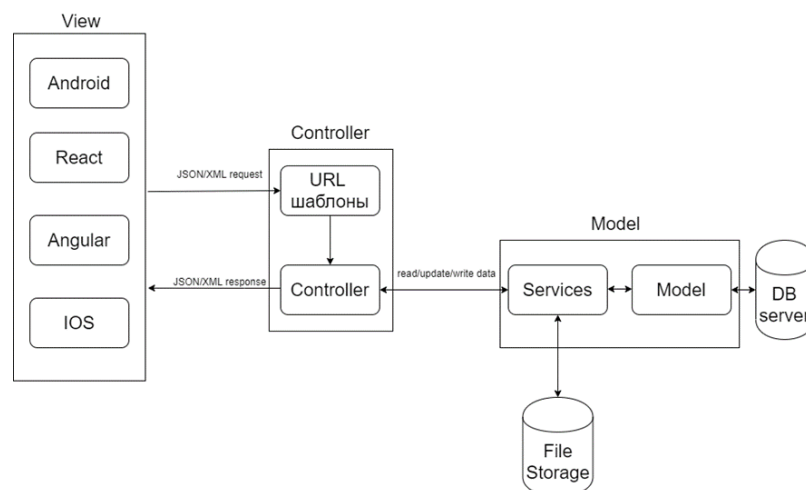


Рисунок 11 – Схема взаимодействия приложений с API в MVC архитектуре

View – это клиентская часть, которая разрабатывается frontend, IOS или Android разработчиками. Взаимодействия с API осуществляется посредством JSON или XML.

JSON и XML – это стандартизированный формат данных, который web приложение получает, обрабатывает и отправляет обратно. В данном случае web приложение – это API, построенное на web сервисах. В качестве обмена данными был выбран JSON формат из-за простоты его создания и наглядно понятной структуры.

3.3 Концептуальная модель базы данных пользователей и эталонных рукописных подписей

Концептуальная модель – это модель, представленная множеством понятий и связей между ними, определяющих смысловую структуру рассматриваемой предметной области или её конкретного объекта.

Графическое представление концептуальной модели базы данных в виде диаграммы «сущность-связь» для предметной области информационной системы представлено на рисунке 12.

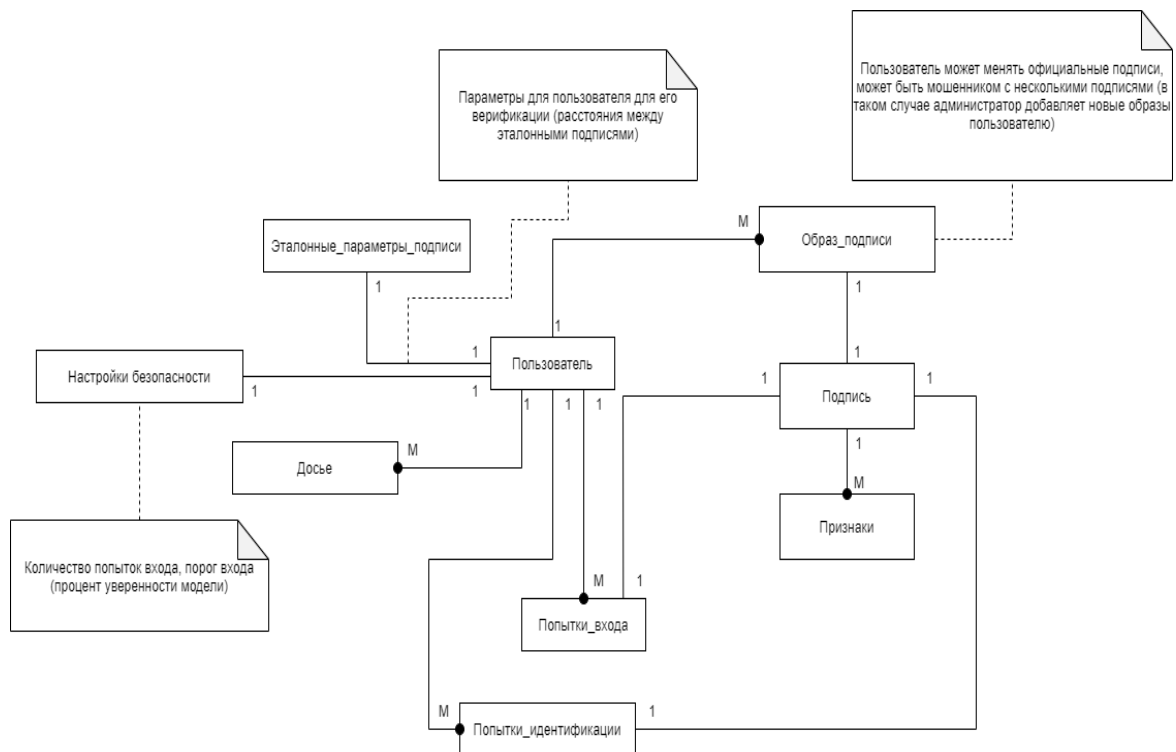


Рисунок 12 – Графическое представление концептуальной модели

Основная структура БД была описана в концептуальной модели и далее будет лишь дополняться.

В таблице 6 представлены сущности, сформированные на основании входных и выходных данных бизнес-процессов.

Таблица 6 – Описание сущностей предметной области

№	Наименование	Описание	Категория	Количество экземпляров	Режим обновления
1	Пользователь	Списочный состав пользователей, зарегистрированных в системе	Стержневая	10000	Редко
2	Образ подписи	Образы подписей для пользователя	Вспомогательная	20000	Очень редко
3	Подпись	Подписи пользователей	Стержневая	200000	5% в неделю
4	Признаки	Признаки, извлеченные из подписей	Вспомогательная	2000000	5% в неделю
5	Эталонные параметры подписи	Параметры, извлеченные из активной зарегистрированной подписи человека	Вспомогательная	50	Очень редко
6	Настройки безопасности	Количество допустимых попыток входа, допустимых процент совпадения подписи	Стержневая	10000	Очень редко
7	Досье	Дополнительные данные о пользователе	Вспомогательная	1000	Очень редко
8	Попытки входа	Попытки верификации пользователей	Вспомогательная	3500000	5% в день
9	Попытки идентификации	Попытки идентификации подписей	Вспомогательная	2000000	5% в день

Предполагается, что БД создается для приложения, где предусмотрено малое количество зарегистрированных пользователей, эти обусловлены данные в столбце «Режим обновления».

Для сущностей предметной области были сформированы связи (таблица 7), которые позже помогут сформировать связи между таблицами, которые будут основаны на сущностях предметной области, столбец таблицы «Мощность» определяет характер связей в системе, например, один к одному, один ко многим и т.д.

Таблица 7 – Описание типов связей между сущностями

№ п/ п	Тип сущности 1	Тип сущности 2	Описание	Мощность	Обязательность	Ограничения
1	Пользователь	Образ подписи	Пользователь может иметь несколько образов подписи Образ подписи ассоциируется с одним пользователем	1 : М	Обязательная	У пользователя может быть только один активный (официальный) образ подписи
2	Пользователь	Эталонные параметры подписи	У пользователя одни эталонные параметры / Эталонные параметры индивидуальны для каждого пользователя	1 : 1	Обязательная	-
3	Пользователь	Настройки безопасности	Настройки безопасности индивидуальны для каждого пользователя	1 : 1	Обязательная	-
4	Пользователь	Досье	У пользователя может быть несколько досье, например из нескольких организаций	1 : М	Не обязательная (на пользователя может не быть досье)	-
5	Пользователь	Попытки входа	Может быть несколько попыток входа, ассоциированных с одним пользователем / Одна попытка входа производится с участием только одного пользователя	1 : М	Не обязательная (у пользователя может не быть попыток входа)	-
6	Пользователь	Попытки идентификации	Может быть несколько попыток идентификации, где был идентифицирован один и тот же пользователь / При попытке идентификации на выходе получается только один пользователь	1 : М	Не обязательная (у пользователя может не быть попыток)	-
7	Образ подписи	Подпись	В образе подписи содержатся уникальные подписи / Подпись может принадлежать одному образу	1 : 1	Обязательная	-

Продолжение таблицы 7

№ п/ п	Тип сущности 1	Тип сущности 2	Описание	Мощн ость	Обязательность	Ограничения
8	Подпись	Признаки	Из подписи может быть извлечено несколько признаков / Признаки индивидуальны для каждой подписи	1 : M	Обязательная	-

Данная концептуальная модель будет отражена в физическую модель БД. В физической модели будут добавлены атрибуты в таблицах.

3.4 Обоснование выбора технологий и средств разработки онлайн-сервиса верификации рукописной подписи

Для разработки серверной части приложения был выбран язык Python, так как он обладает простым синтаксисом и множеством математических библиотек машинного обучения, что значительно упрощает имплементацию методов верификации и идентификации.

Для кодирования приложения был использован редактор исходного кода Visual Studio Code, благодаря его поддержке нескольких языков и форматов файлов. Также в нем интуитивно понятный и удобный интерфейс.

В качестве системы управления базами данных (СУБД) была выбрана PostgreSQL так как она является не только бесплатной, но и объектно-реляционной СУБД, что дает ей некоторые преимущества над другими бесплатными СУБД, в большинстве являющимися реляционными.

Для работы с БД необходимо использовать ORM (англ. object-relational mapping, рус. объектно-реляционное отображение), ORM – это технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии. В ВКР решено было использовать SQLAlchemy, это библиотека на языке Python для работы с реляционными СУБД с применением технологии ORM. Служит для синхронизации объектов Python и

записей реляционной базы данных. SQLAlchemy позволяет описывать структуры баз данных и способы взаимодействия с ними на языке Python без использования SQL. SQLAlchemy имеет несколько основных преимуществ из-за чего была выбрана эта библиотека. Во-первых, параметры запросов экранируются, что делает атаки типа внедрение SQL-кода маловероятными. Во-вторых, повышается вероятность повторного использования запроса к серверу базы данных, что может позволить ему в некоторых случаях применить повторно план выполнения запроса.

3.5 Требования к аппаратно-программному обеспечению онлайн-сервиса верификации рукописной подписи

3.5.1 Требования к аппаратно-техническому обеспечению

Минимальные системные требования включают в себя:

- при использовании архитектуры x86 или AMD64/x64 требуется процессор с тактовой частотой не ниже 1,6 ГГц;
- 3 ГБ оперативной памяти;
- 2 ГБ доступного пространства на жестком диске;
- разрешение экрана не ниже 1024 на 768.

Если минимальные системные требования будут удовлетворены, этого будет достаточно для стабильной работы сервиса.

3.5.2 Требования к системному программному обеспечению

Необходима операционная система Windows 7 и выше, интерпретатор языка Python версии 3.7 и выше. Необходима СУБД PostgreSQL, если пользователю сервиса необходим локальный доступ к содержанию БД, то возможно использование DBeaver, который является удобным инструментом для администрирования баз данных.

3.5.3 Требования к прикладному программному обеспечению

Серверная часть приложения выполнена на языке Python с использованием фреймворка Flask, который является микрофреймворком для

создания вебсайтов на языке Python. Понятие «микрофреймворк» означает, что в комплекте нет набора инструментов и библиотек, программист может сам установить их в зависимости от задач. Он. Позволяет быстро создать веб-приложение, используя только один файл Python. Ядро фреймворка можно масштабировать под разные задачи. Разработчик должен сам выбрать библиотеки и инструменты, которые хочет использовать.

В проекте также используются следующие библиотеки:

- NumPy – это расширение языка Python, добавляющее поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых математических функций для операций с этими массивами, NumPy написан на языке C, что является большим плюсом в производительности;
- SciPy предоставляет алгоритмы для оптимизации, интегрирования, интерполяции, статистики и многих других классов задач, а также расширяет NumPy, предоставляя дополнительные инструменты для вычисления массивов и обеспечивая специализированные структуры данных, такие как разреженные матрицы и k-мерные деревья;
- Pandas – это библиотека Python для обработки и анализа структурированных данных, которая также расширяет NumPy, в экосистеме Python, Pandas является наиболее продвинутой и быстроразвивающейся библиотекой для обработки и анализа данных;
- Модуль Pickle реализует мощный алгоритм сериализации и десериализации объектов Python, в нем «pickling» – процесс преобразования объекта Python в поток байтов, а «unpickling» – обратная операция, в результате которой поток байтов преобразуется обратно в Python-объект, так как поток байтов легко можно записать в файл, модуль pickle широко применяется для сохранения и загрузки сложных объектов в Python;
- Sklearn – это один из наиболее широко используемых пакетов Python для Data Science и Machine Learning, он содержит функции и алгоритмы для машинного обучения: классификации, прогнозирования или разбивки данных на группы.

4 Разработка информационных и программных средств онлайн-сервиса верификации рукописной подписи

4.1 Разработка физической схемы базы данных

Физическая модель – логическая модель базы данных, выраженная в терминах языка описания конкретной СУБД. Физические модели баз данных определяют способы размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне. Способы физической организации данных в различных СУБД, как правило, различны и определяются типом используемой ЭВМ, инструментальными средствами разработки СУБД, а также критериями, которыми руководствуются разработчики СУБД при выборе методов размещения данных и способов доступа к этим данным. В данном курсовом проекте используется СУБД PostgreSQL.

Плюсами данной СУБД является то, что она свободно распространяемая и имеет реализацию для множества UNIX-подобных платформ. Также данная СУБД имеет поддержку множества различных типов данных, такие как JSON, XML, массивы, uuid идентификаторы и т.д.

Также в PostgreSQL есть механизм наследования таблиц. При этом данные, добавленные в порождённую таблицу, автоматически будут участвовать (если это не указано отдельно) в запросах к родительской таблице. Одной из сильных сторон PostgreSQL является ее архитектура. Как и многие коммерческие СУБД, PostgreSQL может применяться в среде клиент-сервер, что дает массу преимуществ как пользователям, так и разработчикам. Основу PostgreSQL составляет серверный процесс базы данных. Доступ из приложений к данным базы осуществляется посредством процесса базы данных.

В данной ВКР потребуется механизм СУБД работы с массивами.

На основании концептуальной и логической модели была разработана физическая модель базы данных (рисунок 13). Физическая модель построена в среде Visual Paradigm в нотации IDEF1X.

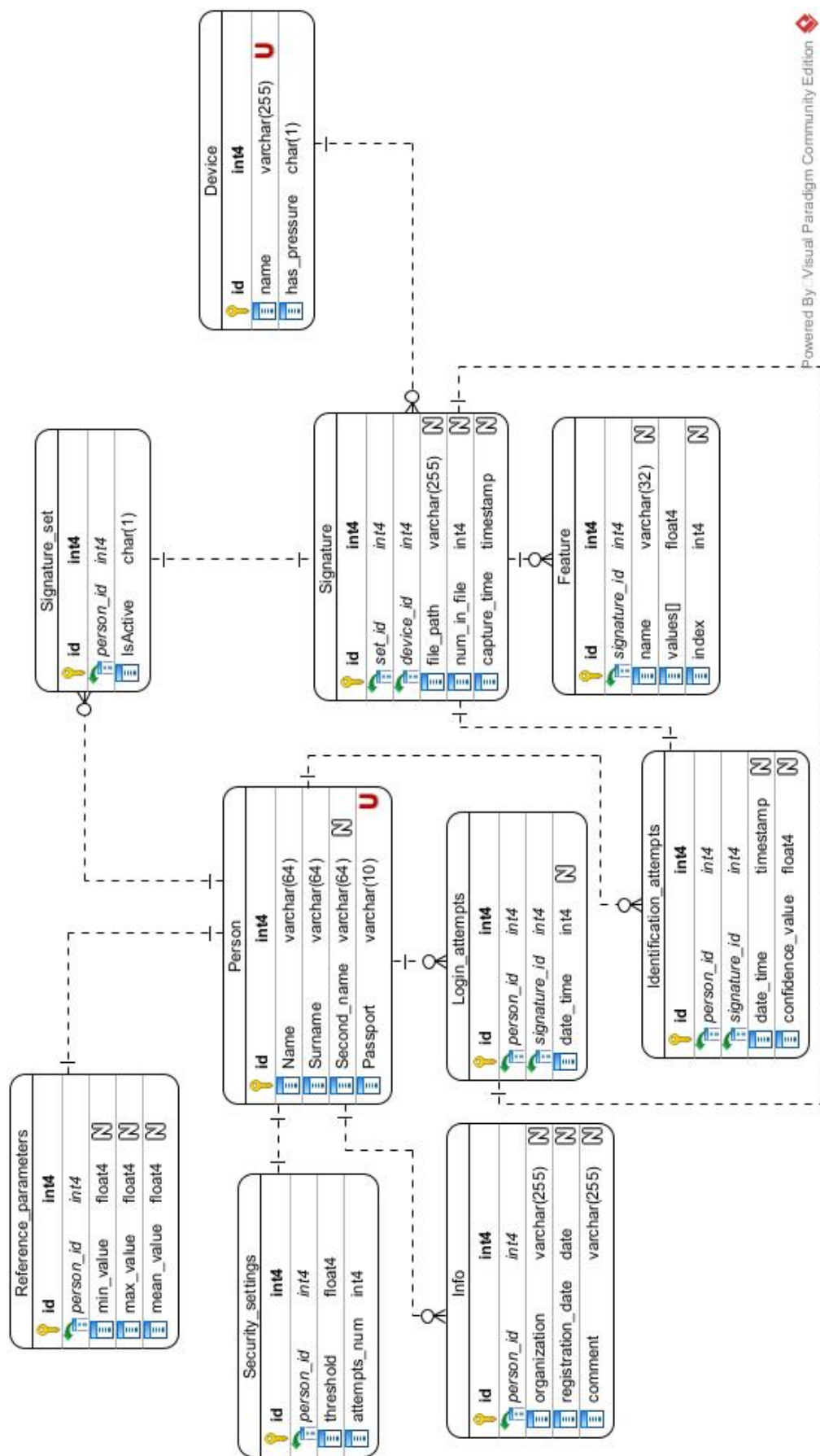


Рисунок 13 – Физическая модель базы данных

В процессе создания физической модели было принято решение использовать для создания модели БД миграции, а для описания БД ORM SQLAlchemy.

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Так как ORM использует объектный подход, то необходимо перенести физическую модель в объектную модель.

Рассмотрим описание таблицы Person с помощью ORM SQLAlchemy. Описание таблицы представлено в листинге 1.

Листинг 1 – Описание таблицы Person в терминах SQLAlchemy

```
class Person(Base):
    __tablename__ = "person"
    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid.uuid4)
    name = Column(String(64), nullable=False)
    surname = Column(String(64), nullable=False)
    second_name = Column(String(64), nullable=True)
    passport = Column(String(10), unique=True, nullable=False)

    signature_sets = relationship("SignatureSet", back_populates="person")
    params = relationship("ReferenceParams", back_populates="person")
    security = relationship("SecuritySettings", back_populates="person")
    info = relationship("Info", back_populates="person")
    login_attempts = relationship("LoginAttempts", back_populates="person")
    identification_attempts = relationship("IdentificationAttempts",
back_populates="person")
```

Данная таблица в терминах СУБД PostgreSQL описана в листинге 2.

Листинг 2 – Описание таблицы Person в терминах PostgreSQL

```
CREATE TABLE public.Person (
    id          SERIAL NOT NULL,
    Name        varchar(64) NOT NULL,
    Surname     varchar(64) NOT NULL,
    Second_name varchar(64),
    Passport    varchar(10) NOT NULL UNIQUE,
    PRIMARY KEY (id));
```

Таким образом, все таблицы были описаны в терминах SQLAlchemy на языке Python. Полная их спецификация представлена в приложении Б.2.

4.2 Разработка архитектуры развертывания информационных и программных средств онлайн-сервиса верификации рукописной подписи

После запуска сервера необходимо создать БД, где будут созданы таблицы приложения. БД создается через модуль, написанный на Python. Изначально происходит подключение к серверу, а затем выполняется команда создания БД (CREATE DATABASE application). Программный код для инициализации базы данных представлен в листинге 3.

Листинг 3 – Программный код для инициализации базы данных

```
class InitialDBCommand(ConsoleCommand):
    """
    Команда инициализации базы данных
    """

    def execute(self):
        env.configure_app()

        try:
            run_sql([f"CREATE DATABASE {os.getenv('APPLICATION_DB')}"])
        except psycopg2.errors.lookup('42P04'):
            print(
                f"Databse {os.getenv('APPLICATION_DB')} already exist and will not be
recreated"
            )

    def run_sql(statements):
        conn = psycopg2.connect(
            dbname =os.getenv("POSTGRES_DB"),
            user   =os.getenv("POSTGRES_USER"),
            password=os.getenv("POSTGRES_PASSWORD"),
            host    =os.getenv("POSTGRES_HOSTNAME"),
            port    =os.getenv("POSTGRES_PORT"),
        )

        conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
        cursor = conn.cursor()

        for statement in statements:
            cursor.execute(statement)

        cursor.close()
        conn.close()
```

Полный программный код представлен в приложении Б.4.

Миграция – обновление структуры базы данных от одной версии до другой (обычно более новой). Механизм миграций необходим для версионирования БД и поддержания ее полноты. То есть этот механизм схож с работой систем контроля версий, только в отличии от контроля

версий он синхронизирует структуру БД. В данной ВКР работе использовалась библиотека Alembic для миграции БД.

Для создания файла миграции выполняется команда `./manage.py migrate(alembic) revision --autogenerate -m «имя миграции»`. В этом файле описаны операции, которые будут выполнены целевой СУБД [40].

Миграция выполняется с помощью команды `(./manage.py migrate upgrade head)`.

Результат выполнения миграции представлен на 14.

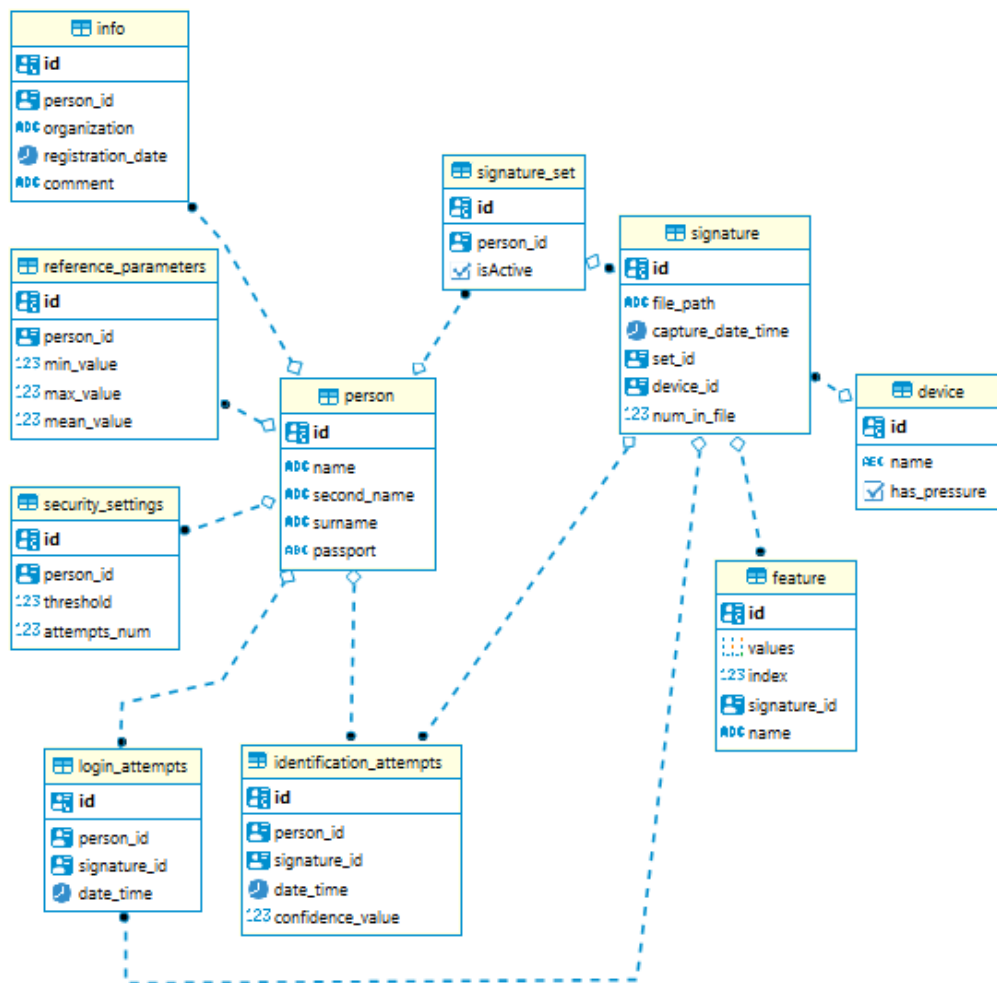


Рисунок 14 – Схема БД в среде DBeaver

Данная БД является основой для разработки сервера приложения.

Сам сервис верификации представляет собой совокупность модулей, ответственных за разные подзадачи сервиса. Диаграмма классов программы представлена на рисунках В.1 – В.4.

Для взаимодействия с БД используется ORM. А для создания запросов к БД использован шаблон проектирования Facade для того, чтобы инкапсулировать логику обращения к БД.

В соответствии с шаблоном MVC в структуру сервиса входит слой Controller, за реализацию логики которого ответственен одноименный модуль, слой View, который представляет собой пользовательское приложение, а слой model поделен на несколько модулей:

- Модуль create_db_records содержит в себе бизнес-логику по занесению различных данных в БД, с классами внутри этого модуля взаимодействует слой Controller;
- Модуль db_services содержит в себе бизнес-логику по скачиванию, получению информации из базы данных;
- Модуль extract_features реализует извлечению параметров из подписи;
- Модуль facades содержит в себе логику обращения к БД [45].

Диаграмма компонентов сервиса изображена на рисунке 15.

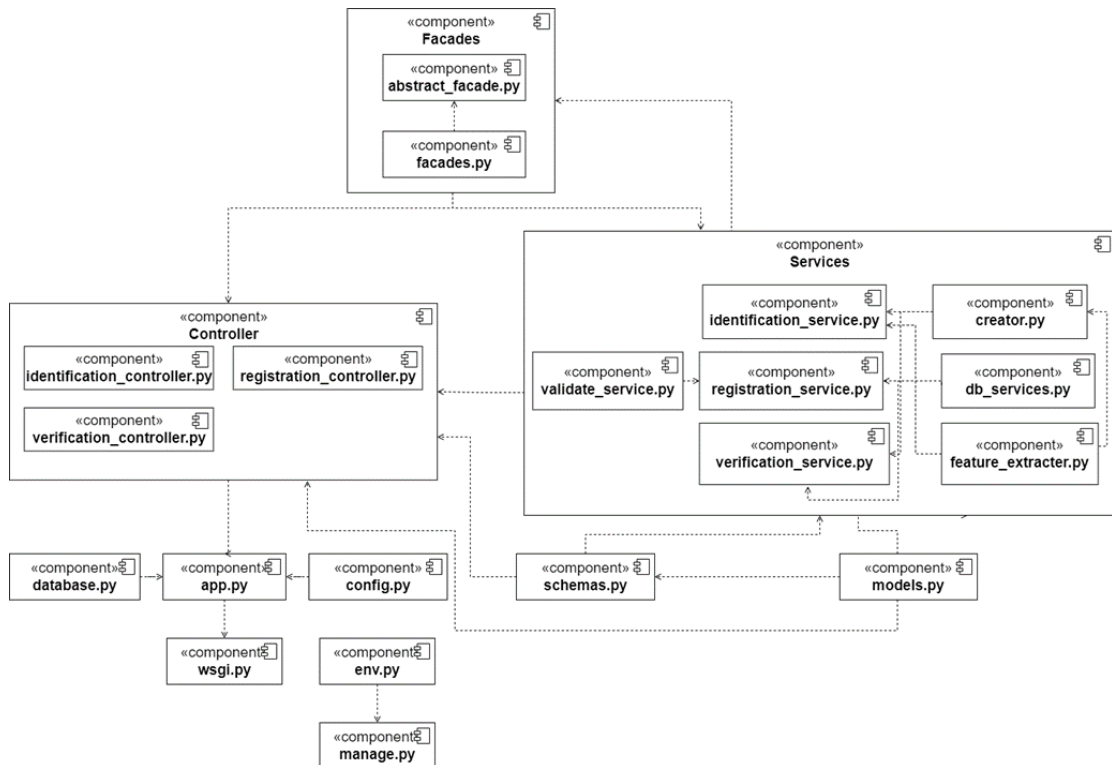


Рисунок 15 – Диаграмма компонентов сервиса

Программные средства были реализованы на языке Python, с помощью микрофреймворка для создания web-приложений Flask. Для взаимодействия с сервером посылаются GET и POST запросы.

4.3 Разработка программных средств взаимодействия с базой данных пользователей и эталонных рукописных подписей

При использовании ORM SQLAlchemy таблицы БД соотносятся к моделям, написанным на языке Python. Перед работой с БД необходимо создать сессию для работы с БД.

Сессия устанавливает связь с БД и представляет собой «зону хранения» для всех объектов, которые были загружены или ассоциированы с сессией во время её жизненного цикла. Сессия предоставляет интерфейс, в котором выполняются SELECT и другие запросы, которые возвращают и изменяют объекты, отображаемые ORM. Код для создания сессии и связывания с ней моделей БД представлен в листинге 4.

Листинг 4 – Программный код для создания сессии для подключения к БД

```
engine = create_engine(Config.SQLALCHEMY_DATABASE_URI)
db_session = scoped_session(sessionmaker(autocommit=False,
                                          autoflush=False,
                                          bind=engine))

Base = declarative_base()
Base.query = db_session.query_property()
```

Изначально мы получаем подключение к серверу и связываем сессию с данным подключением. Потом мы связываем все объекты Base с данной сессией. Все модели БД являются наследниками от объекта Base.

Пример запроса с использованием механизма сессий представлен в листинге 5.

Листинг 5 – Запрос по добавлению новой записи в БД

```
def create(entity: Base):
    db_session.add(entity)
    db_session.commit()
    db_session.refresh(entity)
```

Таким образом, ORM представляет возможность использовать собственный механизм запросов, а также стандартный синтаксис для

выполнения SQL запросов. К тому же ORM предоставляет мощный механизм сессий. Код сервера приложений представлен в приложении Б.6.

Для заполнения БД был разработан программный модуль client.py, который обращается к REST серверу регистрации и передает туда json с параметрами пользователя и его подписи, фрагмент класса пользователя представлен в листинге 6.

Листинг 6 – Клиентское приложение, посылающее запросы к серверу

```
from preprocessing.signature_preprocessing import read_features
import os
import requests
json = {
    "name": "Ivan",
    "surname": "Ivanov",
    "second_name": "Ivanovich",
    "passport": "5640322252",
    "organization": "ПГУ",
    "registration_date": None,
    "comment": "hello world",
    "device_name": "stylus",
    "device_pressure": True,
    'set_isActive': True,
    'signatures': []
}
if __name__ == "__main__":
    dir = r'..\\DeepSignDB\\Development\\stylus'
    sig = None
    name = 0
    i = 0
    for file in os.listdir(dir):
        sig = []
        if name != file[:5]:
            json["name"] = names.get_first_name()
            json["surname"] = names.get_last_name()
            json["second_name"] = names.get_first_name()
            json["passport"] = str(int(json["passport"])+1)
            if i>0:
                requests.post('http://127.0.0.1:5000/api/v1/register/person',
                    json=json)
                requests.post('http://127.0.0.1:5000/api/v1/register/sig', json=json)
                json['signatures'] = []
            i += 1
            name = file[:5]
        with open(os.path.join(dir, file), 'r') as ofile:
            sig = read_features(ofile)
            json['signatures'].append(sig)
```

Результат выполнения кода по наполнению БД представлен на рисунке 16. Данные о пользователях занесены в соответствующую таблицу, также как и данные подписи также будут заполнены в соответствующие таблицы: signature_set, signature, device, а извлеченные сервером параметры будут храниться в таблице feature.

Далее сервер извлекает нужные данные из соответствующих таблиц, а данные не изменяются, лишь дополняются, либо удаляются.

	id	abc name	abc second_name	abc surname	abc passport
1	d57f1a1d-2eee-42be-9938-b4a2e50d3b2e	James	Sue	Reagan	5640322254
2	ddff5144-8ae8-4908-a10e-4335bd29eb64	Norman	Chase	Paschall	5640322255
3	0c169bfc-c8f1-45eb-9fe3-630ecce66863	Sarah	Jeffrey	Wheeler	5640322256
4	2e061014-6faa-47ae-88f6-07bc899bed8f	Phillip	James	Lee	5640322257
5	474423fa-a1c2-4436-95c5-1c70fe2b4298	Mark	Michael	Bohnen	5640322258
6	c109a8fc-2e86-4915-91b8-e222ebefea18	Philip	Mark	Raggs	5640322259
7	af8ae7e7-e7b8-4237-a748-fe05cba0ea6f	Douglas	Felix	Depuy	5640322260
8	c0e4a758-b95b-493d-9d11-6b695aaacf00	Myrtle	Wilma	Lewis	5640322261
9	9a0fbe44-b739-4f5a-91aa-a82b33f3bf7e	Myrna	John	Long	5640322262
10	4cfb36cd-b178-49a8-84c2-05bb975f378b	William	Charles	Costello	5640322263
11	fe4a6bf4-620e-406f-881f-ea82658ce637	Steven	Danielle	Mckenzie	5640322264
12	3e366bd3-09f6-49ce-b374-f7b0a023a158	Sharon	Michael	Mcentegart	5640322265
13	f7553827-7372-485f-b43e-49f084534337	Katy	Tommy	Burleson	5640322266
14	401d785e-710f-4984-96de-4ecda643e323	Isabel	Babette	Harner	5640322267
15	c9c03224-c207-4a38-8b11-cbb3c5256a58	Kam	Mary	Petty	5640322268
16	10c4ff92-fff8-4e58-9905-c335c70431dd	Alice	Nora	Ray	5640322269
17	ca02c057-41c4-419f-9eb7-b5fb77fd57b7	Sarah	Margaret	Bailey	5640322270
18	2af88cf3-d317-44c5-b978-ff8cfede5e8a	Russell	Melissa	Trainor	5640322271
19	58eb8337-9bb3-4f24-8c6f-6daa83468df2	Julie	Sara	Robinson	5640322272
20	caab8b74-9300-4fa2-9106-0e925e598d20	Joyce	Anna	Reid	5640322273
21	9ee6c53e-017b-4500-8acd-38ac0f3d1b8f	George	Theresa	Golden	5640322274
22	9b05d2e7-e5bd-49d9-a869-1f7ede36f35a	Carrie	Mildred	Bodi	5640322275
23	59c80a1a-b8cf-414e-a719-4f359bd66340	William	Veronica	Saldana	5640322276
24	7c026bc6-e0d8-4a08-b995-212bf0ee2c6f	Angelina	Sandra	Boyd	5640322277
25	11b73dda-e621-4712-86d1-5f009617f9b9	Celestine	Brenda	Hill	5640322278
26	43a44535-d0ba-4f4f-89d7-a70b1f971f0f	Brooke	Ada	Amado	5640322279
27	437877e2-e384-4e72-9c99-88816d2f5f8e	Freddie	Thomas	Keller	5640322280
28	41270a50-8b0e-403f-bde0-0c722b7a513f	Dominique	Maria	Hayes	5640322281
29	19e9bd1a-b94c-4cce-ab67-d23098f54b7f	Joyce	Yvonne	Gill	5640322282

Рисунок 16 – Наполнение таблицы person

Для взаимодействия с базой данных и создания в ней новых записей о пользователях был использован модуль «Creator», диаграмма классов этого модуля представлена на рисунке В.3. За создание же классов ответственен модуль на рисунке В.4, выполняющий обращение к модулю на рисунке В.5. Модуль Creator преобразует передаваемые ему данные в объекты классов моделей из модуля models и передает в соответствующий класс модуля фасадов.

Рассмотрим абстрактный класс, от которого происходит наследование классов для создания записей

Листинг 6 – Абстрактный класс фасада для работы с базой данных

```

from abc import ABC
from application.database import db_session, Base

class AbstractFacade(ABC):
    def __init__(self, cls):
        self.cls = cls

```

```

def create(self, entity: Base):
    db_session.add(entity)
    db_session.commit()
    db_session.refresh(entity)

def change(self, entity: Base):
    db_session.merge(entity)
    db_session.commit()
    db_session.refresh(entity)

def remove(self, entity: Base):
    db_session.delete(entity)
    db_session.commit()

def get_entity(self, id: int):
    return self.cls.query.filter_by(id=id).first()

def get_all(self):
    return self.cls.query.order_by(self.cls.id).all()

def get_in_range(self, offset: int = 0, limit: int = 20):
    return self.cls.query.offset(offset).limit(limit).all()

```

Полный листинг всех классов модуля Facades представлен в приложении Б.4.

Для тестирования были сгенерированы имена с использованием библиотеки names. С помощью нее были сгенерированы имена, второе имена (surname или при русскоязычном использовании базы можно принимать за отчество) и фамилии. Для генерации оригинальных номеров паспорта к исходному числу (5640322256) добавлялась единица.

Для наполнения базы данных подписями были использованы данные DeepSignDB [29].

4.4 Разработка библиотеки методов верификации рукописной подписи

Для разработки методов верификации необходимо создать классы, реализующие методы обучения моделей, посылающие запросы верификации и идентификации в эти модели, а также класс, реализующий извлечение нужных параметров из переданных ему данных подписей.

Диаграмма классов библиотеки методов верификации представлена на рисунке 17. На ней можно видеть модули, реализующие логику верификации, идентификации и извлечение признаков из подписей, данные модули могут быть использованы только после регистрации пользователя в базе данных.

Для стабильной и точной работы сервисов верификации и идентификации предпочтительно занесение в базу данных нескольких подписей для того, чтобы модели могли обучаться на различных данных.

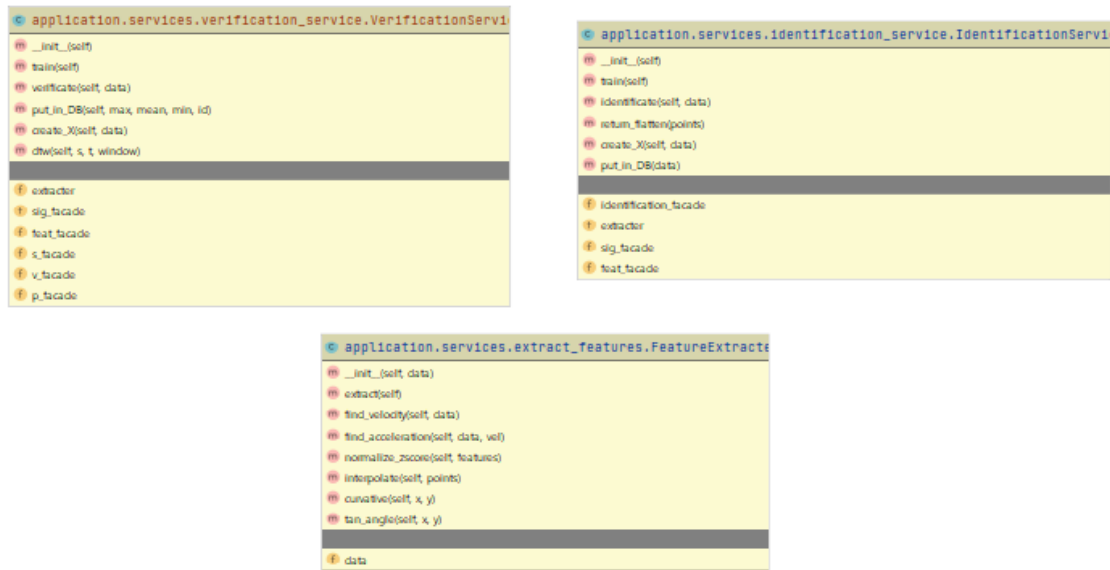


Рисунок 17 – Диаграмма классов библиотеки верификации рукописной подписи

Далее опишем подробнее структуру библиотеки верификации рукописной подписи и разберем подробнее входящие в нее модули.

4.4.1 Разработка структуры библиотеки методов верификации рукописной подписи

Пользователь библиотеки верификации может выбрать метод верификации из предложенных. То есть сервис рассчитан на его применение программистами, которые хотят встроить в проект сервис верификации. Благодаря открытому исходному коду существует возможность менять и дополнять сервис верификации.

Стоит заметить, что для достижения предсказуемого результата методы `train()` и `verify()` должны быть вызваны с одинаковыми параметрами выбора функции верификации. Если метод `verify()` вызван, но переданной модели не существует, будет выдана ошибка.

Библиотека предназначена для внедрения в проекты с целью осуществления аутентификации с использованием рукописной подписи или

для использования рукописной подписи в качестве основного пароля человека. Приведем спецификацию методов классов разработанной библиотеки на рисунках ниже. Для удобства, сигнатура методов представлена не в формате, реализованном в языке python, а с указанием возвращаемого типа и передаваемых параметров.

На рисунке 18 указана спецификация класса идентификации.

Методы класса идентификации (<i>IdentificationService</i>)
Назначение метода: Создание модели машинного обучения для идентификации подписи.
Сигнатура: int train (DataFrame data, Methods method)
Краткое описание: Создается модель для дальнейшей идентификации подписей. Данный метод в некоторых случаях вызывается в одноименном методе класса VerificationService.
Входные данные: Данные всех подписей в формате из (2) и объект класса методов (выбранный алгоритм верификации). Далее в спецификации данные формата DataFrame подразумевают формат (2).
Выходные данные: Код ответа (200 при удачном завершении обучения)
Назначение метода: Идентификация человека по подписи.
Сигнатура: int, UID, float identify (DataFrame data, Methods method)
Краткое описание: Происходит идентификация подписи с использованием ранее тренированной модели.
Входные данные: Данные подписи и объект класса методов (выбранный алгоритм верификации).
Выходные данные: int – код ответа, UID – уникальный идентификатор пользователя в формате, который идентичен 'person_id', переданному в метод создания модели, float – уверенность идентификатора.

Рисунок 18 – Спецификация класса идентификации

Идентификация пользователя может использоваться, например, при обнаружении в базе данных неизвестных экземпляров подписей. Или же при подозрении в случайной подделке. При вызове метода для создания модели машинного обучения формируется модель наивного байесовского классификатора для многомерных моделей Бернулли из библиотеки scipy.

Преобразование UID человека в вид обычного целочисленного типа происходит при помощи модуля preprocessing библиотеки sklearn. Во время обучения самой модели также проходит обучение и преобразователь, позже он хранится в системе.

На рисунке 19 представлена спецификация класса верификации.

Методы класса верификации (<i>VerificationService</i>)
Назначение метода: Создание шаблона для верификации.
Сигнатура: int train (dict data, Methods method)
Краткое описание: Создаются шаблоны эталонных подписей для дальнейшей верификации по ним.
Входные данные: Данные подписей в исходном формате (1) и объект класса методов (выбранный алгоритм верификации).
Выходные данные: Код ответа (200 при удачном завершении обучения)
Назначение метода: Верификация человека по подписи.
Сигнатура: int, bool verify(dict data, Methods method)
Краткое описание: Происходит верификация человека по подписи с использованием шаблонов
Входные данные: Данные подписей в исходном формате (1) и объект класса методов.
Выходные данные: Код ответа и булево значение, которое оповещает принадлежит подпись верифицируемому человеку или нет.

Рисунок 19 – Спецификация класса верификации

Перед регистрацией и непосредственно верификацией подпись проходит этапы предобработки и извлечения признаков, спецификация класса для извлечения признаков представлена на рисунке 20. А выбор метода происходит на основе класса Methods. Библиотека ориентирована на использования разработчиками, которым нужна надежная библиотека верификации «из коробки». Нужно лишь представить данные подписи в нужном формате и обучит модель верификации. Стоит отметить, что библиотека хранит все данные пользователей локально, тем самым уменьшая

вероятность их «утечки» при перемещении по каналам связи, таким образом увеличивая безопасность всего сервиса в целом.

На рисунке 20 представлена спецификация вспомогательного класса извлечения признаков.

Методы класса извлечения признаков (<i>FeatureExtractor</i>)	
Назначение метода:	Инициализация объекта.
Сигнатура:	<code>__init__(dict data)</code>
Краткое описание:	Инициализируется класс извлечения признаков.
Входные данные:	Данные подписей в исходном формате (1).
Выходные данные:	Объект класса <code>FeatureExtractor</code> .
Назначение метода:	Извлечение признаков.
Сигнатура:	<code>dict extract()</code>
Краткое описание:	Извлечение признаков (2) из подписи.
Входные данные:	Берутся из поля <code>data</code> .
Выходные данные:	Словарь извлеченных признаков.

Рисунок 20 – Спецификация класса извлечения признаков

Класс извлечения признаков скрыт от пользователя, вызывается автоматически при верификации и регистрации. подписи.

Получаемые параметры представлены на рисунке 21.

Скорость	$\frac{\sqrt{(\Delta x^2 + \Delta y^2)}}{\Delta t}$
Ускорение	$\frac{\Delta v}{\Delta t}$
Угол касательной к траектории в точке	$\arctan x/y$
Радиус кривизны в точке	$\frac{ x'' * y' - x' * y'' }{((x')^2 + (y')^2)^{3/2}}$

Рисунок 21 – Извлекаемые из подписи параметры

Далее происходит стандартизация параметров и линейная интерполяция до 256 точек.

Таким образом, полученные параметры должны быть представлены в формате dataframe библиотеки pandas со следующими колонками:

$\{x, y, p, v, acceleration, angle, radius, person_id\},$

где x – координата x введенной подписи;
 y – координата y введенной подписи;
 p – значения давления введенной подписи;
 v – скорость;
 $acceleration$ – ускорение;
 $angle$ – угол касательной к траектории в точке;
 $radius$ – радиус кривизны в точке;
 $person_id$ – уникальный идентификатор пользователя.

Класс `Methods` типа `Enum` на рисунке 22 позволяет легко выбрать желаемый метод верификации.

Перечисление алгоритмов (<i>Methods (enum.Enum)</i>)	
SVM	Метод опорных векторов
DTW	Алгоритм динамической трансформации временной шкалы
XGB	Градиентный бустинг

Рисунок 22 – Спецификация класса алгоритмов

Применение библиотеки при может описано в следующей диаграмме на рисунке 23.

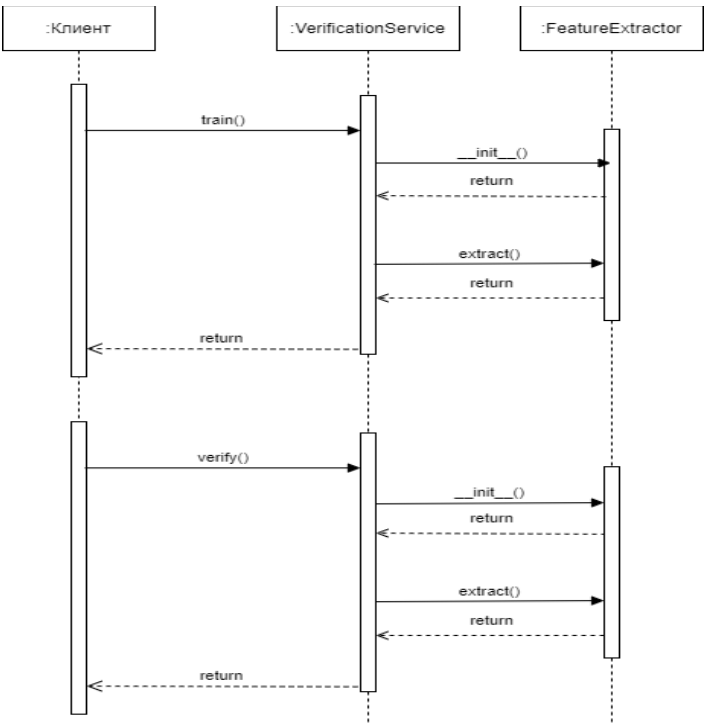


Рисунок 23 – Диаграмма последовательности

Для подключения библиотеки достаточно импортировать класс библиотеки, объявить объект и использовать методы по назначению.

Библиотека функционирует в составе онлайн REST сервиса по верификации подписи.

4.4.2 Разработка программного метода оценки качества верифицируемой подписи

После получения нужного количество примеров подписи для регистрации можно выявить качество данных примеров и отбросить некачественные образы. Качество для полученных признаков получается по формуле:

$$q = coef_{i,j} - \frac{mean_j}{std_j},$$

где $coef_{i,j}$ – j-ый признак i-го примера подписи;

$mean_j$ – математическое ожидание для j-го признака;

std_j – стандартное отклонение для j-го признака.

После накопления качества по формуле выше полученное число делится на общее количество признаков и переводится в проценты. Так получается качество для одного примера подписи.

Как только накапливается вектор качества для всех подписей, этот вектор сортируется по убывания. Распределение качества всех подписей будем считать нормальным. По правилу трех сигм 95% признаков должно лежать на расстоянии не больше одного стандартного отклонения от математического ожидания. А значит подписи, чье качество лежит дальше, мы можем отбросить.

4.4.3 Разработка методов верификации подписи

Выбранные методы верификации были подробно описано в разделе 2.4. Теперь подробнее остановимся на их реализации. Методы определены в одном классе и алгоритм верификации зависит от выбранного метода. Процессы тренировки и верификации являются схожими для всех методов: в процессе тренировки извлекаются данные всех пользователей базы данных и

передаются в модель регистрации, после обучения модель локально сохранена и готова к использованию.

Разберем подробнее один из алгоритмов – алгоритм динамической трансформации временной шкалы. Его блок схема представлена на рисунке 24.

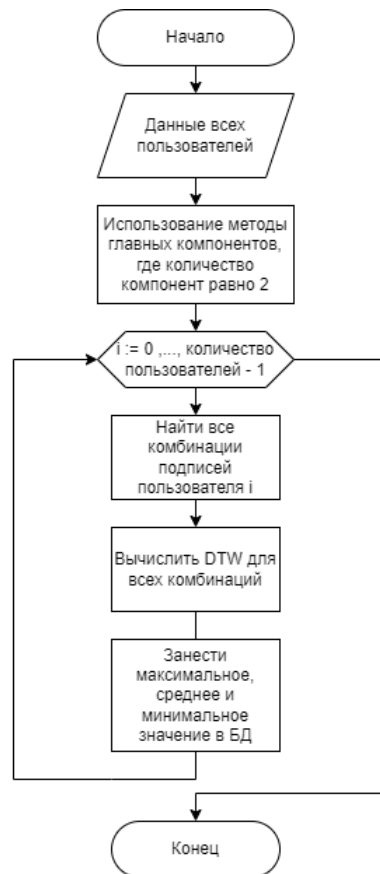


Рисунок 24 – Блок схема обучения алгоритма DTW

После обучения алгоритм применяется в верификации, где между введенной подписью и эталонами также вычисляется расстояние DTW и сравнивается со значениями в БД, на основе этого принимается решение верификации.

4.5 Разработка программных средств Restful сервера

В качестве сервера приложений будет разработан REST сервис.

REST веб-сервис является CRUD приложением (Create, Read, Update, Delete). Основой REST приложения является HTTP, так вся работа с REST сервисами происходит через HTTP запросы.

Взаимодействие с RESTful API происходит через следующие запросы: GET, POST, UPDATE, DELETE.

Особенность REST в том, что сервер не запоминает состояние пользователя между запросами – в каждом запросе передаётся вся информация, необходимая для выполнения операции.

В качестве фреймворка для создания REST сервиса будет использоваться фреймворк Flask. Для взаимодействия с БД будет использоваться ORM SQLAlchemy, а для подключения к БД будет использоваться библиотека psycopg2 – адаптер для работы с СУБД PostgreSQL.

Как было описано выше, взаимодействие с сервером происходит по средствам http запросов. Тело запросов будут представлять из себя JSON объекты, которые и будет обрабатывать сервер приложений.[44]

В качестве архитектуры будет использован MVC. Он позволяет поделить приложение на следующие слои:

- бизнес-логика или сервисы: в этом слое содержится обращение к функциям, представлениям БД, также в этом слое описаны некоторые запросы к БД с помощью ORM, было принято решение убрать со стороны БД некоторую валидацию данных, поэтому проверка имени пользователя, его паспорта и т.д. была размещена в этом слое сервера приложений;
- контроллер: связующий слой между действиями пользователя и бизнес-логикой, содержит в себе последовательный вызов функций бизнес-логики приложения;
- представление – слой UI, сам сервер приложений не обладает UI, слоем представления можно назвать JSON ответы сервера, которые он отправляет пользователю в ответ на его запросы, далее JSON обрабатывается клиентским приложением [44].

В REST архитектуре каждому сервису соответствует свой URI. Поэтому в таблице 8 представлена спецификация сервисов REST приложения.

Таблица 8 – Спецификация REST сервисов web приложения

№	Назначение сервиса	URI сервиса	Класс контроллера
1	Сервис регистрации пользователя	/api/v1/register/person	PersonRegistration
2	Сервис регистрации подписи	/api/v1/register/sig	SignatureRegistration
3	Сервис идентификации подписи	/api/v1/identificate	SignatureIdentification
4	Сервис верификации подписи	/api/v1/verificate	SignatureVerification

Далее рассмотрим эти сервисы подробнее.

4.5.1 Разработка архитектуры сервера

Классы контроллера представляют собой слой Controller в архитектуре MVC. Слой Model представляют классы модуля services.

Спецификации классов модуля приведена далее. Спецификация классов, ответственных за регистрацию данных представлена на рисунках 25 – 27.

Наименование класса: PersonRegistrationService
Назначение класса: содержит в себе бизнес логику регистрации пользователя
Поля класса: json: dict – Json в теле запроса пользователя при обращении к контроллеру регистрации p_facade: PersonFacade – Фасад для обращения и взаимодействия с таблицей Person в БД
Методы класса: registrate – метод регистрации пользователя

Рисунок 25 – Спецификация класса «PersonRegistrationService»

При регистрации подписи вызывается именно класс PersonRegistrationService. Из этого класса вызываются методы классов InformationService для инициализации записей о зарегистрированном человеке, SignatureRegistrationService для регистрации подписи и также в нем происходит валидация введенных данных: проверяется уникальность паспорта, правильность имени, фамилии и отчества. Если паспорт не

уникален, то важно узнать нужно ли обновить данные о подписи или же возникла ошибка

Наименование класса: InformationService
Назначение класса: содержит в себе бизнес логику создания записей информации о человеке
Поля класса: json: dict – Json в теле запроса пользователя при обращении к контроллеру регистрации (класс InformationService вызывается из класса PersonRegistrationService) user: UUID – ID пользователя при регистрации которого инициализируется класс InformationService i_facade: InfoFacade – Фасад для обращения и взаимодействия с таблицей Info в БД
Методы класса: registrate – метод инициализации информации о человеке

Рисунок 26 – Спецификация класса «InformationService»

Класс InformationService – это вспомогательный класс для занесения информации о пользователе в БД.

Наименование класса: SignatureRegistrationService
Назначение класса: содержит в себе бизнес логику регистрации подписи
Поля класса: json: dict – Json в теле запроса пользователя при обращении к контроллеру регистрации подписи sig_s_facade: SignatureSetFacade – Фасад для обращения и взаимодействия с таблицей SignatureSet в БД dev_facade: DeviceFacade – Фасад для обращения и взаимодействия с таблицей Device в БД sig_facade: SignatureFacade – Фасад для обращения и взаимодействия с таблицей Signature в БД feat_facade: FeatureFacade – Фасад для обращения и взаимодействия с таблицей Feature в БД
Методы класса: registrate – метод инициализации информации о человеке

Рисунок 27 – Спецификация класса «SignatureRegistrationService»

Класс SignatureRegistrationService вызывается классом PersonRegistrationService и в нем обрабатывается информация, связанная с подписью непосредственно. На рисунках 28, 29 представлены спецификации классов верификации и идентификации.

Наименование класса: IdentificationService
Назначение класса: содержит в себе бизнес логику идентификации подписи
Поля класса: json: dict – Json в теле запроса пользователя при обращении к контроллеру идентификации подписи
Методы класса: train – метод обучения модели идентификации identificate – метод по идентификации подписи return_flatten – метод, возвращающий параметры в «плоском» формате put_in_DB – метод для занесения попытки идентификации в БД

Рисунок 28 – Спецификация класса «IdentificationService»

В классе IdentificationService содержатся методы, необходимые для создания и эксплуатации модели идентификации.

Наименование класса: VerificationService
Назначение класса: содержит в себе бизнес логику верификации подписи
Поля класса: json: dict – Json в теле запроса пользователя при обращении к контроллеру верификации подписи
Методы класса: train – метод обучения модели верификации verify – метод по верификации подписи put_in_DB – метод для занесения попытки верификации в БД

Рисунок 29 – Спецификация класса «VerificationService»

В классе IdentificationService содержатся методы, необходимые для создания и эксплуатации модели идентификации.

4.5.2 Разработка программных средств взаимодействия сервера с клиентскими приложениями

В случае, когда пользователь хочет обратиться к серверу, то он посылает POST (в случае регистрации и обучения) или GET запрос (в случае верификации). Данные запросы направляются на классы Controller, которые ассоциируются с нужными адресами при помощи Flask и его метода add_resource.

На рисунке 30 представлена спецификация модуля контроллера регистрации.

Наименование класса: PersonRegistration
Назначение класса: перенаправление запроса на регистрацию пользователя
Методы класса: post: int, str – Обращается к методу регистрации личности в системе, возвращает полученный код и сообщение
Наименование класса: SignatureRegistration
Назначение класса: перенаправление запроса на регистрацию данных подписи
Методы класса: post: int, str – Обращается к методу регистрации эталонных подписей в системе, возвращает полученный код и сообщение

Рисунок 30 – Спецификация модуля «RegistrationController»

На рисунке 31 представлена спецификация модуля контроллера идентификации.

Наименование класса: SignatureIdentificationTrain
Назначение класса: перенаправление запроса обучения модели идентификации
Методы класса: get: int – Обращается к методу обучения модели идентификатора, возвращает полученный после обучения код
Наименование класса: SignatureIdentificationPredict
Назначение класса: перенаправление запроса идентификацию
Методы класса: post: int, str – Обращается к методу идентификации, возвращает полученный код и сообщение идентификатора

Рисунок 31 – Спецификация модуля «IdentificationController»

На рисунке 32 изображена спецификация модуля контроллера верификации.

Наименование класса: SignatureVerificationTrain
Назначение класса: перенаправление запроса на обучения модели верификации
Методы класса: get: int – Обращается к методу обучения модели верификации
Наименование класса: SignatureVerificationVerify
Назначение класса: перенаправление запроса на верификацию
Методы класса: post: dict – Обращается к методу верификации личности, возвращает код результата, результат в виде булевого значения и сообщение верификации

Рисунок 32 – Спецификация модуля «VerificationController»

От модуля контроллера запросы перенаправляются в модуль сервисов.

4.5.3 Разработка программных средств взаимодействия сервера с базой данных

Для взаимодействия с базой данных, создания и модифицирования данных был разработан модуль Creator, диаграмма классов данного модуля представлена на рисунке 33.

В него входят классы, соответствующие всем таблицам базы данных, классы имеют соответствующие атрибутам поля.

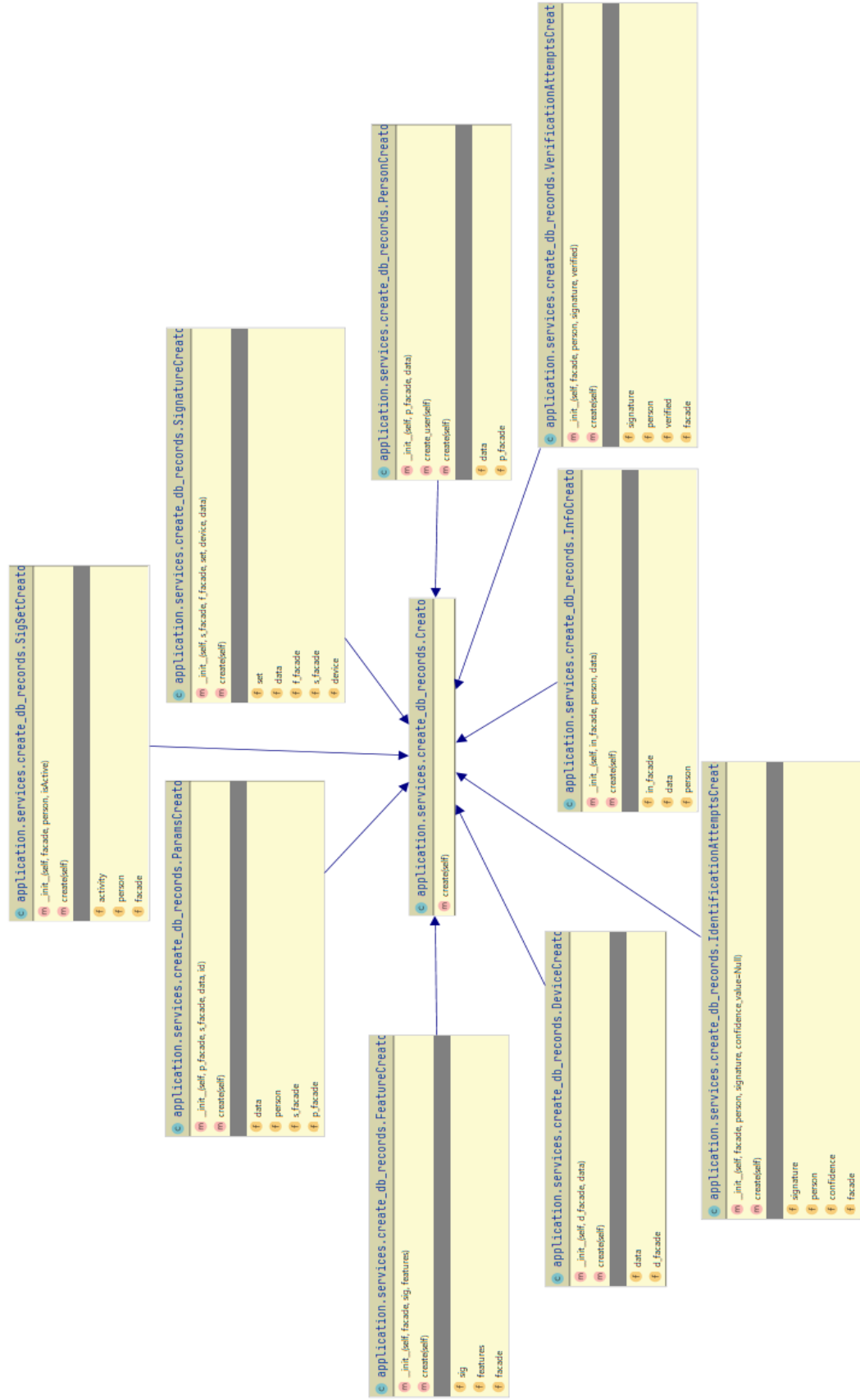


Рисунок 33 – Диаграмма классов модуля Creator

Классы этого модуля вызываются методами модуля сервисов каждый раз, когда необходимо каким-либо образом модифицировать БД.

4.6 Разработка клиентского приложения

Основная задача клиентского приложения – получение данных подписи, ее ввод и передача этих данных в сервис.

Для написания клиентского приложения был выбран язык C++ в сочетании с фреймворком для разработки кроссплатформенного программного обеспечения Qt. Разработка велась в среде Qt Creator.

Для клиентского приложения был спроектирован графический интерфейс. Пользовательский интерфейс должен обеспечивать простое и эффективное выполнение основных функций.

На рисунке 34 представлен интерфейс ввода подписи.

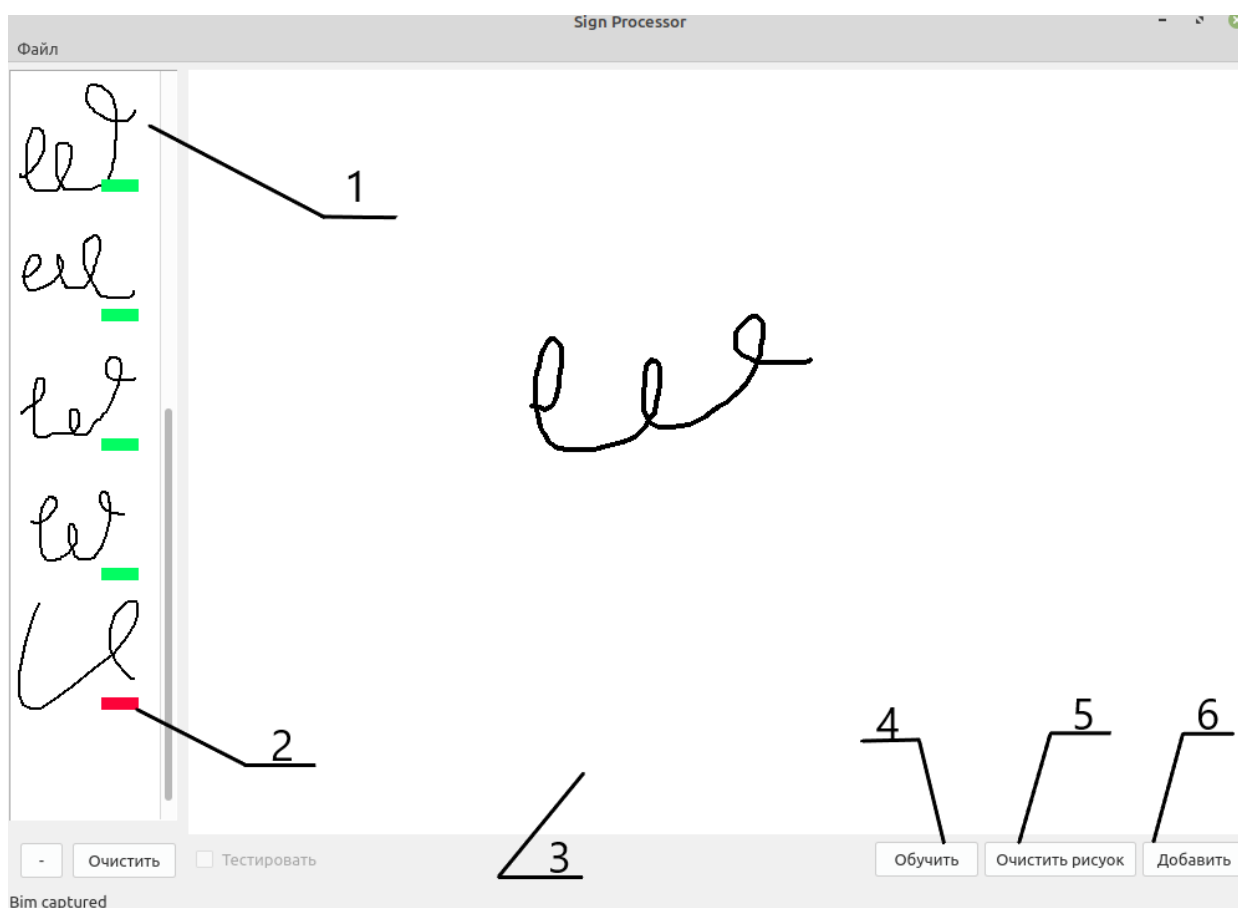


Рисунок 34 – Окно ввода подписи

Основные элементы интерфейса клиентского приложения для обработки рукописной подписи, представленного на рисунке 34:

- 1 – миниатюра введенной подписи;
- 2 – значение качества, если цвет зеленый – качество хорошее, красный – рекомендуется удалить элемент;
- 3 – виджет для ввода рукописной подписи;
- 4 – кнопка для регистрации подписи и обучения модели;
- 5 – очищение виджета от введенной подписи;
- 6 – досрочное добавление подписи в панель миниатюр.

При отсутствии ввода в течение 4 секунд подпись автоматически добавляется в панель. При желании, можно удалить некоторые подписи или же полностью очистить панель.

При нажатии кнопки «Обучить» образы из панели миниатюр передаются на сервер, где они обрабатываются.

Рассмотрим диаграмму классов приложения по сбору рукописных подписей на рисунке 35.

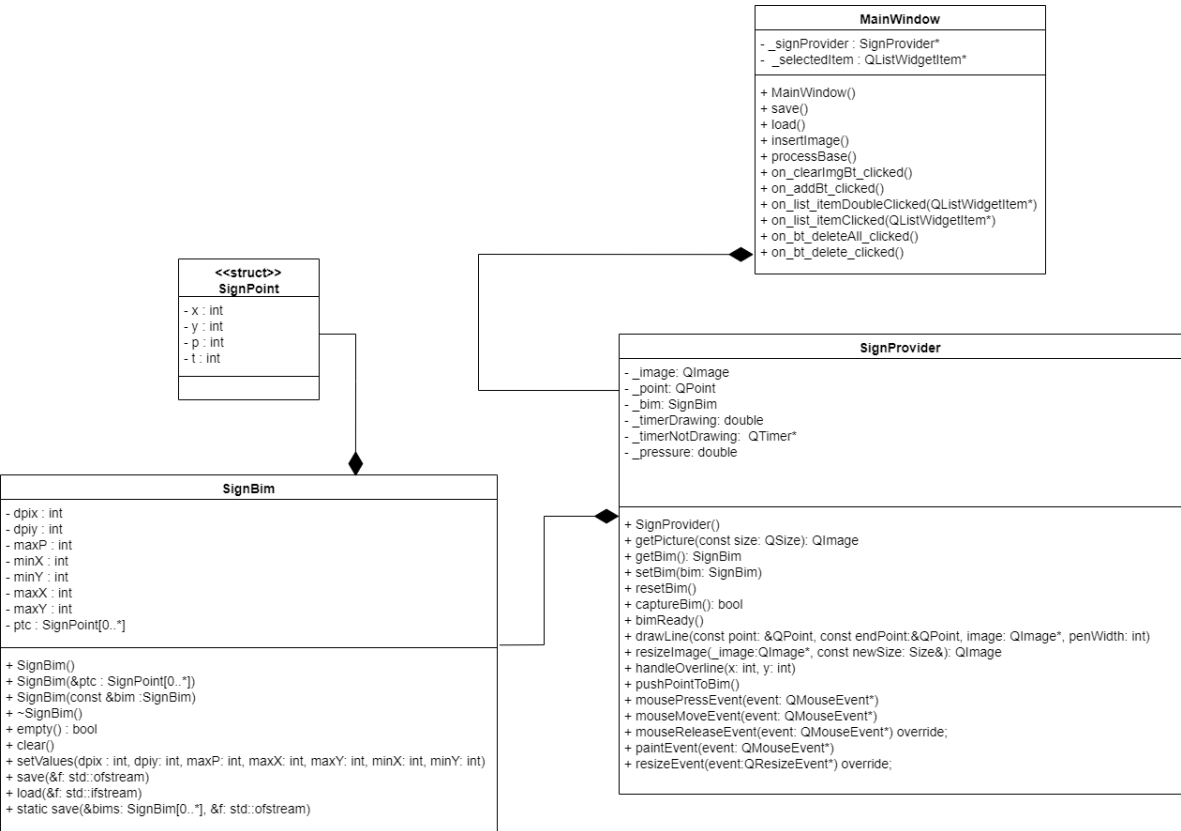


Рисунок 35 – Диаграмма классов

Класс SignBim отвечает за описание подписи как класса, ее создание, сохранение, очистку. Класс SignProvider ответственен за окно сбора подписи

и его функциональность. Класс `MainWindow` обрабатывает события, связанные с меню и общими действиями над окном, посылая нужные запросы в `SignProvider`.

5 Контроль качества программных средств онлайн-сервиса верификации рукописной подписи

5.1 Планирование процесса контроля качества программных средств

5.1.1 Обоснование выбора модели тестирования

Существует достаточно много видов тестирования программного продукта. Были рассмотрены виды тестирования, описанные ниже.

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям. Функциональное тестирование на сегодняшний день является одним из наиболее часто применяемых видов тестирования, так как позволяет наиболее полно проверить программу на соответствие требованиям.

Нефункциональное тестирование позволяет проверить соответствие свойств программного обеспечения с поставленными нефункциональными требованиями.

Каждая сложная программная система состоит из отдельных частей – модулей, выполняющих ту или иную функцию в составе системы. Для того, чтобы удостовериться в корректной работе всей системы, необходимо вначале протестировать каждый модуль системы по отдельности. В случае возникновения проблем при тестировании системы в целом это позволяет проще выявить модули, вызвавшие проблему, и устранить соответствующие дефекты в них.

Для данной ВКР были реализованы все виды тестирования.

5.1.2 Обоснование выбора метрик программного кода

Существуют несколько стандартных метрик измерения качества программного кода.

Метрика устойчивости для компоненты отражает количественную меру работы, которую необходимо произвести при изменении других зависимых компонент.

Для вычисления позиционной устойчивости кода используется следующая формула:

$$I = \frac{Ce}{Ca+Ce},$$

где I – мера неустойчивости (данная величина изменяется в диапазоне от $[0, 1]$, если $I = 0$, то компонент максимально устойчив, если $I = 1$, то компонент максимально неустойчив);

Ce – количество классов вне данного компонента, зависящих от классов внутри компонента;

Ca – количество классов внутри данного компонента, зависящих от классов вне компонента.

Метрика A , являющаяся мерой абстрактности компонента, равна отношению количества абстрактных классов к общему количеству классов в компоненте. Абстрактностью A называется величина:

$$A = \frac{Na}{Nc},$$

где Nc – количество классов в компоненте;

Na – количество абстрактных классов в нем же.

Метрика A изменяется от 0 до 1, причем 0 означает, что в компоненте вообще нет абстрактных классов, а 1 – что компонент содержит только абстрактные классы.

Вычислив меру устойчивости компоненты и ее абстрактность можно отобразить их на графике. На вертикальной оси необходимо отложить меру абстрактности A , а на горизонтальной оси – меру неустойчивости I .

Компонент, который располагаются в верхнем левом углу или в нижнем правом являются максимально качественным. На практике часто случается, так что полученная точка лежит на диагонали между точкой $(1,0)$ и точкой $(0,1)$. Такая диагональ называется главная последовательность (рисунок 36).

Для вычисления количественной оценки используется формула вычисления расстояние до главной последовательности:

$$D = \left| \frac{(A+I-1)}{\sqrt{2}} \right|,$$

где A – мера абстрактности;

I – мера неустойчивости.

Чем ближе точка располагается к главной последовательности, тем лучше.

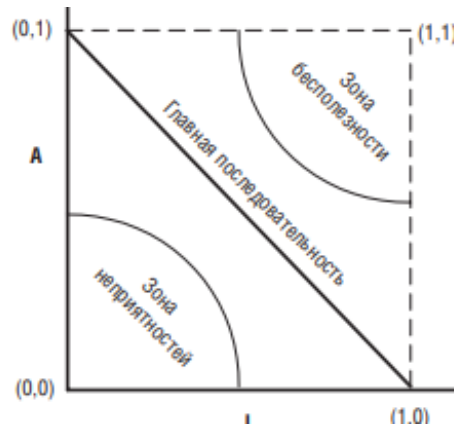


Рисунок 36 – Зоны исключения

Для тестирования кода с помощью метрик также необходимо будет построить график с рисунка 36.

5.2 Тестирование программных средств

5.2.1 Модульное тестирование

Модульное тестирование позволяет проверить корректность работы отдельных модулей программы с помощью программных средств. При помощи модульного тестирования можно в автоматическом режиме проверить корректно ли работают все функции программы, а также убедиться в том, что изменение кода не привело к ошибкам.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Также при модульном тестировании необходимо знать метрику покрытия кода. Она показывает процент исходного кода программы, который был выполнен в процессе тестирования.

Для проведения модульного тестирования была выбрана библиотека `pytest`. Плюсом данной библиотеки является то, что в ней присутствует механизм создания соответствующего окружения для проведения тестирования, что облегчает, например, работу с БД.

Результаты тестирования программных средств подсистемы верификации и регистрации приведены в таблице 9.

Таблица 9 – Описание модульных тестов

Тестовый метод	Описание	Ожидаемый результат
<code>test_register_user</code>	Регистрация при всех валидных данных	Вернется код успеха 200, пользователь будет добавлен в БД
<code>test_register_name</code>	Регистрация при не валидном имени	Вернется код неудачи 400 и сообщение об ошибке
<code>test_register_surname</code>	Регистрация при не валидном фамилии	Вернется код неудачи 400 и сообщение об ошибке
<code>test_register_secname</code>	Регистрация при не валидном отчестве	Вернется код неудачи 400 и сообщение об ошибке
<code>test_register_passport</code>	Регистрация при не валидном паспорте	Вернется код неудачи 400 и сообщение об ошибке
<code>test_register_user_is_already_exist</code>	Регистрация в случае, если такой пользователь уже был зарегистрирован	Вернется код неудачи 400 и сообщение об ошибке
<code>test_verify_user</code>	Проверка верификации пользователя	Вернется код успеха 200, вернется значение <code>True</code>
<code>test_ident_user</code>	Проверка идентификации пользователя	Вернется код успеха 200 и имя пользователя

Все тесты были пройдены успешно. Результаты прохождения тестов представлены ниже. Код модульных тестов представлен в приложении Б.7.

На рисунке 37 приведены результаты модульного тестирования. Видно, что все тесты, заданные в модульном тестировании, были пройдены успешно.

```
platform win32 -- Python 3.8.12, pytest-6.2.5, py-1.11.0, pluggy-1.0.0 -- d:\anaconda3\envs\sign_env\python.e
cachedir: .pytest_cache
rootdir: C:\Users\sophi\Projects\SignatureService
plugins: cov-3.0.0
collected 7 items

tests/test_sig.py::test_verify_user PASSED
tests/test_sig.py::test_register_name 400 {'error': 'Name should contain only letters'}
PASSED
tests/test_sig.py::test_register_surname 400 {'error': 'Name should contain only letters'}
PASSED
tests/test_sig.py::test_register_secname 400 {'error': 'Name should contain only letters'}
PASSED
tests/test_sig.py::test_register_passport 400 {'error': 'Name should contain only letters'}
PASSED
tests/test_sig.py::test_register_user_is_already_exist 400 {'error': 'Name should contain only letters'}
PASSED
tests/test_sig.py::test_ident_user PASSED
```

Рисунок 37 – Информация о пройденных модульных тестах

Основной из характеристик модульного тестирования является покрытие кода. Она показывает на сколько тесты покрывают все возвращаемые результаты из функций исходного кода. Если проверены все возможные ветви кода, то результат покрытия будет 100%. Результат покрытия представлен ниже на рисунке 38.

Name	Stmts	Miss	Cover
application__init__.py	0	0	100%
application\api__init__.py	0	0	100%
application\api\identification_controller.py	15	15	0%
application\api\registration_controller.py	28	28	0%
application\api\verification_controller.py	14	14	0%
application\app.py	18	18	0%
application\config.py	12	0	100%
application\database.py	8	0	100%
application\facades__init__.py	0	0	100%
application\facades\abstract_facade.py	22	8	64%
application\facades\db_facades.py	43	4	91%
application\models.py	100	0	100%
application\services__init__.py	0	0	100%
application\services\create_db_records.py	110	26	76%
application\services\db_services.py	43	19	56%
application\services\extract_features.py	57	16	72%
application\services\identification_service.py	60	19	68%
application\services\registration_services.py	50	13	74%
application\services\validate_service.py	41	4	90%
application\services\verification_service.py	67	37	45%
TOTAL	688	221	68%

Рисунок 38 – Результаты проверки покрытия тестами

Было проведено модульное тестирование, на основе которого были выявлены и исправлены недостатки системы.

5.2.2 Функциональное тестирование

Для основных вариантов использования, взаимодействующих с СУБД, были выполнены функциональные тесты.

В таблице 10 представлено описание тестов хранимых процедур.

Таблица 10 – Описание тестов хранимых процедур и функций

Бизнес-процесс	Описание теста	Параметры запроса	Ожидаемый результат	Результат
Регистрация человека в БД	Посылаем запрос к серверу регистрации человека	JSON с данными пользователя	Данные занесены в БД	Тест пройден (рисунок В.1)
Регистрация подписи по БД	Посылаем запрос к серверу регистрации подписи	JSON с данными пользователя и его подписей	Данные занесены в БД	Тест пройден (рисунок В.2)
Идентификация подписи	Посылаем данные подписи	JSON с данными подписи	id пользователя	Тест пройден (рисунок В.3, В.4)
Верификация подписи	Посылаем данные подписи и пользователя	JSON с данными пользователя и его подписей	Булевая переменная	Тест пройден (рисунок В.5)

Было проведено функциональное тестирование, на основе которого были выявлены и исправлены недостатки системы.

5.2.3 Тестирование сервиса верификации на точность

При тестировании API в базу данных были загружены подписи из DeepSignDB [29]. На сервер передаются точки каждой подписи, получаемые при рисовании подписи в формате

$$[\{x_1, y_1, p_1, t_1\}, \{x_2, y_2, p_2, t_2\}, \dots, \{x_n, y_n, p_n, t_n\},],$$

после чего из них извлекаются признаки и заносятся в базу данных. Также все подписи интерполируются до 256 точек, нормируются и стандартизируются.

При проведении эксперимента в базу данных предварительно были загружены тренировочные подписи базы DeepSignDB.

После чего, для всех подписей каждого пользователя были посчитаны параметры входа по следующему алгоритму:

- к каждой подписи был применен метод главных компонент, где количество компонент было равно двум;
- полученные параметры всех подписей пользователя были проанализированы при помощи алгоритма деформации временной шкалы, были получены попарные расстояния между всеми эталонными подписями;
- были выделены минимальное, среднее и максимальное расстояние и занесены в базу данных.

Таким образом, сервис был готов к эксперименту по верификации.

При верификации тестовые данные базы DeepSignDB были отправлены на сервер и получены результаты, представленные на рисунке 39.

Среднее время отклика сервера	2.9 сек
Качество верификации	98%

Рисунок 39 – Результаты эксперимента

Качество верификации было подсчитано по формуле

$$Q = \frac{k}{all},$$

где k – количество верно верифицированных пользователей;

all – все пользователи, подвергшиеся верификации.

Таким образом, можно сделать вывод о том, что эксперимент был проведен успешно. Алгоритм также можно дорабатывать, делая его более устойчивым к подделкам.

5.3 Оценка качества программного кода

В созданном приложении можно выделить 7 логических компонент:

- слой Controller;
- компонент работы с БД (фасады);
- компонент сервисов для анализа данных (регистрация, верификация, идентификация);
- компонент моделей БД;

- компонент создания объектов моделей БД;
- компонент для извлечения признаков из подписей;
- компонент для построения сложных аналитических запросов к БД.

Для каждой компоненты была просчитана величина I – метрика неустойчивости:

- $I_1 = \frac{0}{0+2} = 0;$
- $I_2 = \frac{2}{2+1} = 0.7;$
- $I_3 = \frac{1}{1+4} = 0.2;$
- $I_4 = \frac{4}{4+0} = 1;$
- $I_5 = \frac{1}{1+2} = 0.3;$
- $I_5 = \frac{1}{1+0} = 1;$
- $I_6 = \frac{1}{1+1} = 0.5.$

Вычислим также метрику абстрактности:

- $A_1 = \frac{0}{7} = 0;$
- $A_2 = \frac{1}{10} = 0.1;$
- $A_3 = \frac{2}{9} = 0.22;$
- $A_4 = \frac{1}{10} = 0.1;$
- $A_5 = \frac{1}{10} = 0.1;$
- $A_6 = \frac{0}{1} = 0;$
- $A_7 = \frac{0}{1} = 0.$

Отообразим меры устойчивости логических компонент приложения и их метрики абстрактности на графике (рисунок 40). Координаты точек соответствуют метрикам неустойчивости и абстрактности.

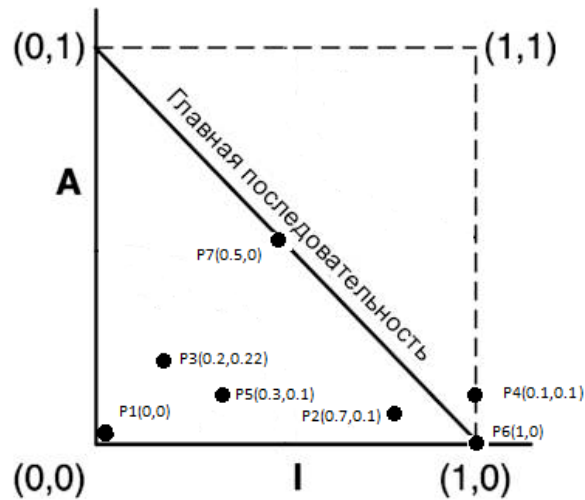


Рисунок 40 – Отображение метрик на графике

Было вычислено расстояние до главной последовательности:

- $D_1 = \left| \frac{0+0-1}{\sqrt{2}} \right| = 0.707;$
- $D_2 = \left| \frac{0.1+0.7-1}{\sqrt{2}} \right| = 0.141;$
- $D_3 = \left| \frac{0.22+0.2-1}{\sqrt{2}} \right| = 0.41;$
- $D_4 = \left| \frac{0.1+0.1-1}{\sqrt{2}} \right| = 0.566;$
- $D_5 = \left| \frac{0.1+0.3-1}{\sqrt{2}} \right| = 0.424;$
- $D_5 = \left| \frac{0+1-1}{\sqrt{2}} \right| = 0;$
- $D_6 = \left| \frac{0+0.5-1}{\sqrt{2}} \right| = 0.354.$

Первая компонента достаточно удалена от главной последовательности и находится в зоне неприятностей. Это обусловлено тем, что данный компонент является слоем контроллера и его задача только вызывать методы бизнес-логики, поэтому этот компонент имеет устойчивость. Нулевая абстрактность обусловлена малым количеством классов в данной компоненте.

Остальные компоненты лежат достаточно близко к главной последовательности. Это значит, что отношение абстрактности к устойчивости этих компонент является правильным.

Заключение

В ходе выполнения выпускной квалификационной работы разработан онлайн сервис верификации рукописной подписи.

Проведен анализ методов верификации рукописной подписи. Данные методы подробно описаны и в каждом из них выделены основные особенности. Был проведен сравнительный анализ методов верификации на основе изученных статей, а также был проведен эксперимент для выбора оптимальных методов верификации подписи. Проведен сравнительный анализ сервисов для онлайн верификации подписи.

После изучения средств аналогов и выделения их основных особенностей были сформулированы функциональные и нефункциональные требования к системе.

Для решения задачи создания сервиса онлайн верификации подписи выбрана архитектура системы и основной шаблон проектирования. Также спроектирована логическая и физическая схема БД, отображающая основные сущности и связи в выбранной предметной области. Для основных модулей построены диаграммы классов.

Разработка программных средств проводилась в среде разработки VS Code на языке программирования Python. Разработанный сервис может взаимодействовать с клиентским приложением, функционирующим на любой платформе и написанное на любом языке программирования

Разработанное программное обеспечение протестировано. Выполнено функциональное и модульное тестирование. Все тесты пройдены успешно.

Также проведено планирование и оценка бюджета проекта при помощи программы для управления проектами Microsoft Project. Затраты на проект составили 226 600 рублей. Время разработки 72 рабочих дня.

Таким образом, в результате выполнения выпускной квалификационной работы разработан сервис онлайн верификации рукописной подписи, отвечающий всем требованиям технического задания.

Список использованных источников

1. Prabhakar S. et al. Introduction to the special issue on biometrics: Progress and directions //IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2007. – Т. 29. – №. 4. – С. 513 – 516.
2. Teoh A. B. J., Leng L. Special issue on advanced biometrics with deep learning //Applied Sciences. – 2020. – Т. 10. – №. 13. – С. 4453.
3. Impedovo D., Pirlo G. Automatic signature verification: The state of the art //IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews). – 2008. – Т. 38. – №. 5. – С. 609 – 635.
4. Fierrez-Aguilar J. et al. Fusion of local and regional approaches for on-line signature verification //International Workshop on Biometric Person Authentication. – Springer, Berlin, Heidelberg, 2005. – С. 188 – 196.
5. Wacom Ink SDK for verification // Wacom [Электронный ресурс]. – URL: <https://www.wacom.com/ru-ru/for-business/products/will-sdk-for-verification> (дата обращения: 20.04.2022).
6. Верификация и распознавание подписи // Корпорация ProgressSoft [Электронный ресурс]. – URL: <https://www.progresssoft.com/ru/products/signature-verification-recognition> (дата обращения: 20.04.2022).
7. Automatic Biometric Signature Verification in Real Time // Namirial GmbH [Электронный ресурс]. – URL: <https://www.xyzmo.com/e-signature-products/signature-verification> (дата обращения: 20.04.2022).
8. Вдовкина С. А., Шибанов С. В. Обзор методов онлайн-верификации рукописной подписи // Аналитические и численные методы моделирования естественно-научных и социальных проблем : сб. ст. по материалам XVI Междунар. науч.-техн. конф. (г. Пенза, Россия, 1–4 декабря 2021 г.) / под ред. проф. И. В. Бойкова. – Пенза : Изд-во ПГУ, 2021. – С. 108 – 113.
9. Malik M. I. et al. ICDAR2015 competition on signature verification and writer identification for on-and off-line skilled forgeries (SigWlcomp2015)

//2015 13th International Conference on Document Analysis and Recognition (ICDAR). – IEEE, 2015. – C. 1186 – 1190.

10. Song X., Xia X., Luan F. Online signature verification based on stable features extracted dynamically //IEEE Transactions on Systems, Man, and Cybernetics: Systems. – 2016. – T. 47. – №. 10. – C. 2663 – 2676.

11. Fischer A. et al. Robust score normalization for DTW-based on-line signature verification //2015 13th international conference on document analysis and recognition (ICDAR). – IEEE, 2015. – C. 241 – 245.

12. Feng H., Wah C. C. Online signature verification using a new extreme points warping technique //Pattern Recognition Letters. – 2003. – T. 24. – №. 16. – C. 2943 – 2951.

13. Faundez-Zanuy M. On-line signature recognition based on VQ-DTW //Pattern Recognition. – 2007. – T. 40. – №. 3. – C. 981 – 992.

14. Sharma A., Sundaram S. An enhanced contextual DTW based system for online signature verification using vector quantization //Pattern Recognition Letters. – 2016. – T. 84. – C. 22 – 28.

15. Ansari A. Q. et al. Online signature verification using segment-level fuzzy modelling //IET biometrics. – 2014. – T. 3. – №. 3. – C. 113 – 127.

16. Wu X. et al. Deep dynamic time warping: end-to-end local representation learning for online signature verification //2019 International Conference on Document Analysis and Recognition (ICDAR). – IEEE, 2019. – C. 1103 – 1110.

17. Huang K., Yan H. Stability and style-variation modeling for on-line signature verification //Pattern Recognition. – 2003. – T. 36. – №. 10. – C. 2253 – 2270.

18. Saleem M., Kovari B. Online signature verification using signature down-sampling and signer-dependent sampling frequency //Neural Computing and Applications. – 2021. – C. 1 – 13.

19. Parziale A. et al. Sm-dtw: Stability modulated dynamic time warping for signature verification //Pattern Recognition Letters. – 2019. – T. 121. – C. 113 – 122.

20. Nyssen E., Sahli H., Zhang K. A multi-stage online signature verification system //Pattern Analysis & Applications. – 2002. – T. 5. – №. 3. – C. 288 – 295.
21. Kashi R. S., Turin W., Nelson W. L. On-line handwritten signature verification using stroke direction coding //Optical Engineering. – 1996. – T. 35. – №. 9. – C. 2526 – 2533.
22. Saleem M., Kovari B. Performance Evaluation of the JK-nearest Neighbor Online Signature Verification Parameters //2021 4th International Conference on Data Storage and Data Engineering. – 2021. – C. 1 – 5.
23. Kashi R. S. et al. On-line handwritten signature verification using hidden Markov model features //Proceedings of the fourth international conference on document analysis and recognition. – IEEE, 1997. – T. 1. – C. 253 – 257.
24. Igarza J. J. et al. Online handwritten signature verification using hidden markov models //Iberoamerican Congress on Pattern Recognition. – Springer, Berlin, Heidelberg, 2003. – C. 391 – 399.
25. Zou M. et al. On-line signature verification using local shape analysis //Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings. – IEEE, 2003. – C. 314 – 318.
26. Quan Z. H., Liu K. H. Online signature verification based on the hybrid hmm/ann model //International Journal of Computer Science and Network Security. – 2007. – T. 7. – №. 3. – C. 313 – 322.
27. Rohilla S., Sharma A. SVM based online signature verification technique using reference feature vector //Proceedings of the National Academy of Sciences, India Section A: Physical Sciences. – 2017. – T. 87. – №. 1. – C. 125 – 136.
28. Gruber C. et al. Online signature verification with support vector machines based on LCSS kernel functions //IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics). – 2009. – T. 40. – №. 4. – C. 1088 – 1100.
29. El-Henawy I. et al. Online signature verification: State of the art //International Journal of Computers and Technology. – 2013. – T. 4. – №. 2. – C. 277 – 3061.

30. Tolosana R. et al. DeepSign: Deep on-line signature verification //IEEE Transactions on Biometrics, Behavior, and Identity Science. – 2021. – T. 3. – №. 2. – C. 229 – 239.
31. Hefny A., Moustafa M. Online signature verification using deep learning and feature representation using Legendre polynomial coefficients //International conference on advanced machine learning technologies and applications. – Springer, Cham, 2019. – C. 689 – 697.
32. Ahrabian K., BabaAli B. Usage of autoencoders and Siamese networks for online handwritten signature verification //Neural computing and applications. – 2019. – T. 31. – №. 12. – C. 9321 – 9334.
33. Lai S., Jin L., Yang W. Online signature verification using recurrent neural network and length-normalized path signature descriptor //2017 14th IAPR international conference on document analysis and recognition (ICDAR). – IEEE, 2017. – T. 1. – C. 400 – 405.
34. Otte S., Liwicki M., Krechel D. Investigating long short-term memory networks for various pattern recognition problems //International Workshop on Machine Learning and Data Mining in Pattern Recognition. – Springer, Cham, 2014. – C. 484 – 497.
35. Chen Y., Ding X. On-line signature verification using direction sequence string matching //Second International Conference on Image and Graphics. – SPIE, 2002. – T. 4875. – C. 744 – 749.
36. Dimauro G., Impedovo S., Pirlo G. Component-oriented algorithms for signature verification //International Journal of Pattern Recognition and Artificial Intelligence. – 1994. – T. 8. – №. 03. – C. 771 – 793.
37. Bovino L. et al. Multi-Expert Verification of Hand-Written Signatures //ICDAR. – 2003. – T. 3. – C. 932 – 936.
38. Yeung D. Y. et al. SVC2004: First international signature verification competition //International conference on biometric authentication. – Springer, Berlin, Heidelberg, 2004. – C. 16 – 22.

39. Арлоу Д. UML 2 и Унифицированный процесс, 2-е издание. Практический объектно-ориентированный анализ и проектирование. / Джим Арлоу, Айла Нейштадт – Санкт Петербург: Символ Плюс, 2007. – С. 624
40. Сейдур С. Паттерны для новичков: MVC vs MVP vs MVVM / Сейдур Сергей // Хабр [Электронный ресурс]. – URL: <https://habr.com/ru/post/215605/> (дата обращения 16.05.2022).
41. Корчак А. Разработка web API / Андрей Корчаков // Хабр [Электронный ресурс]. – URL: <https://habr.com/ru/post/181988/> (дата обращения 10.12.2021).
42. Platt J. Probabilistic outputs for SVMs and comparisons to regularized likelihood methods //Advances in Large Margin Classifiers. – MIT Press, 1999.
43. Chen T., Guestrin C. Xgboost: A scalable tree boosting system //Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. – 2016. – С. 785 – 794.
44. Salvador S., Chan P. Toward accurate dynamic time warping in linear time and space //Intelligent Data Analysis. – 2007. – Т. 11. – №. 5. – С. 561-580.
45. Баргетсов С. Введение в JSON / Сергей Баргетсов // Медиум [Электронный ресурс]. – URL: <https://medium.com/@stasonmars/введение-в-json-c798d2723107> (дата обращения 10.12.2021).
46. Гамма Э. Приемы объектно-ориентированного программирования. Паттерны программирования. / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. – Санкт Петербург: Питер, 2001. – 368 с.

Приложение А
ГЛОССАРИЙ
(справочное)

Процент ложных отказов (FRR) – это мера вероятности того, что биометрическая система безопасности ошибочно отклонит попытку доступа авторизованного пользователя.

Процент ложных принятий (FAR) – это единица, используемая для измерения среднего числа ложных принятий в биометрической системе.

Равный уровень ошибок (EER) – это коэффициент, при котором оба вида ошибок равны.

Алгоритм динамического преобразования временно шкалы (DTW) – алгоритм, находит наилучшее нелинейное выравнивание двух временных рядов, при котором общее расстояние между ними минимально.

База данных (БД) – это информативная модель, позволяющая организованно поддерживать данные о группе объектов, имеющих один и тот же набор параметров.

Unified Modeling Language (UML) – графический унифицированный язык моделирования, использующий объектно-ориентированный подход для моделирования в сфере разработки программного обеспечения.

ORM – технология, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Приложение Б
ЛИСТИНГ ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг Б.1 – Файл app.py

```

from flask import Flask
from flask_restful import Api
import application.database
import application.api.registration_controller as a_test
import application.api.identification_controller as i_test
import application.api.verification_controller as v_test

def create_app(config_name):

    app = Flask(__name__)

    config_module = f"application.config.{config_name.capitalize()}Config"
    app.config.from_object(config_module)

    api = Api(app)
    api.add_resource(a_test.PersonRegistration, "/api/v1/register/person")
    api.add_resource(a_test.SignatureRegistration, "/api/v1/register/sig")
    api.add_resource(i_test.SignatureIdentification, "/api/v1/identify")
    api.add_resource(v_test.SignatureVerification, "/api/v1/verify")
    return app

```

Листинг Б.2 – Файл database.py

```

from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker
from application.config import Config

engine = create_engine(Config.SQLALCHEMY_DATABASE_URI)
db_session = scoped_session(sessionmaker(autocommit=False,
                                         autoflush=False,
                                         bind=engine))

Base = declarative_base()
Base.query = db_session.query_property()

```

Листинг Б.3 – Файл registration_controller.py

```

from flask import request, jsonify
from flask_restful import Resource
from application import models
from application.services import registration_services

class PersonRegistration(Resource):

    def post(self):
        json = request.get_json()
        user_register_context =
registration_services.PersonRegistrationService(json)
        code, message = user_register_context.register()
        return message, code

class SignatureRegistration(Resource):
    def post(self):
        json = request.get_json()
        user_register_context =
registration_services.SignatureRegistrationService(json)
        code, message = user_register_context.register()
        return message, code

```

```

class RegistrationController(Resource):

    def get(self):
        json = {"hello": "world"}
        result = models.Feature.query.all()
        return jsonify(result)

    def post(self):
        json = request.json
        print(json['name'])

    def put(self):
        pass

    def delete(self):
        pass

```

Листинг Б.4 – Файл identification_controller.py

```

from flask import request, jsonify
from flask_restful import Resource
from application import models
from application.services import identification_service

class SignatureIdentification(Resource):

    def post(self):
        json = request.get_json()
        user_register_context =
identification_service.IdentificationService()
        code, message = user_register_context.train()
        return message, code

```

Листинг Б.5 – Файл create_db_records.py

```

from abc import ABC, abstractmethod
import numpy
from application import models
from application.services.extract_features import FeatureExtractor

class Creator(ABC):
    """
    Abstract create records of datasets based on ther json
    """

    @abstractmethod
    def create(self):
        pass

class PersonCreator(Creator):

    def __init__(self, p_facade, data):
        self.data = data
        self.p_facade = p_facade

    def create_user(self):
        new_person = models.Person(
            name = self.data["name"],
            surname = self.data["surname"],
            second_name = self.data["second_name"],
            passport = self.data["passport"]

```

```

    )
    return new_person

def create(self):
    user = self.create_user()
    self.p_facade.create(user)
    return user

class InfoCreator(Creator):

    def __init__(self, in_facade, person, data):
        self.data = data
        self.in_facade = in_facade
        self.person = person

    def create(self):
        info = models.Info(
            person_id = self.person,
            organization = self.data["organization"],
            registration_date = self.data["registration_date"],
            comment = self.data["comment"]
        )
        self.in_facade.create(info)
        return info

class SignatureCreator(Creator):

    def __init__(self, s_facade, f_facade, set, device, data):
        self.s_facade = s_facade
        self.f_facade = f_facade
        self.set = set
        self.device = device
        self.data = data

    def create(self):
        signature = models.Signature(
            set_id = self.set,
            device_id = self.device
        )
        self.s_facade.create(signature)
        features = FeatureExtractor(self.data).extract()
        FeatureCreator(self.f_facade, signature.id, features).create()

class DeviceCreator(Creator):

    def __init__(self, d_facade, data):
        self.d_facade = d_facade
        self.data = data

    def create(self):
        name = self.data['device_name']
        device = self.d_facade.get_device_by_name(name)
        if device:
            return device
        device = models.Device(
            name = name,
            has_pressure = self.data["device_pressure"],
        )
        self.d_facade.create(device)
        return device

class SigSetCreator(Creator):

    def __init__(self, facade, person, isActive):

```

```

        self.facade = facade
        self.person = person,
        self.activity = isActive

    def create(self):
        set = models.SignatureSet(
            person_id = self.person,
            isActive = self.activity
        )
        self.facade.create(set)
        return set

class FeatureCreator(Creator):

    def __init__(self, facade, sig, features):
        self.facade = facade
        self.features = features
        self.sig = sig

    def create(self):
        i = 0
        for key in self.features.keys():
            feature = models.Feature(
                signature_id = self.sig,
                values = self.features[key].tolist(),
                name = str(key),
                index = i
            )
            self.facade.create(feature)
            i = i + 1
        return 200

```

Листинг Б.6 – Файл abstract_facade.py

```

from abc import ABC
from application.database import db_session, Base
class AbstractFacade(ABC):

    def __init__(self, cls):
        self.cls = cls

    def create(self, entity: Base):
        db_session.add(entity)
        db_session.commit()
        db_session.refresh(entity)

    def change(self, entity: Base):
        db_session.merge(entity)
        db_session.commit()
        db_session.refresh(entity)

    def remove(self, entity: Base):
        db_session.delete(entity)
        db_session.commit()

    def get_entity(self, id: int):
        return self.cls.query.filter_by(id=id).first()

    def get_all(self):
        return self.cls.query.order_by(self.cls.id).all()

    def get_in_range(self, offset: int = 0, limit: int = 20):
        return self.cls.query.offset(offset).limit(limit).all()

```

Листинг Б.7 – Файл db_facade.py

```

from application.database import db_session, engine
from application import models
from application.facades.abstract_facade import AbstractFacade

class PersonFacade(AbstractFacade):

    def __init__(self):
        super().__init__(models.Person)

    def get_person_by_passport(self, passport: str):
        return models.Person.query.filter_by(passport=passport).first()

    def get_all(self):
        return models.Person.query.all()

class ReferenceParamsFacade(AbstractFacade):

    def __init__(self):
        super().__init__(models.ReferenceParams)

class SecuritySettingsFacade(AbstractFacade):

    def __init__(self):
        super().__init__(models.SecuritySettings)

class InfoFacade(AbstractFacade):

    def __init__(self):
        super().__init__(models.Info)

class SignatureFacade(AbstractFacade):

    def __init__(self):
        super().__init__(models.Signature)

class DeviceFacade(AbstractFacade):

    def __init__(self):
        super().__init__(models.Device)

    def get_device_by_name(self, name: str):
        return models.Device.query.filter_by(name=name).first()

class FeatureFacade(AbstractFacade):

    def __init__(self):
        super().__init__(models.Feature)

class LoginAttempts(AbstractFacade):

    def __init__(self):
        super().__init__(models.LoginAttempts)

class IdentificationAttempts(AbstractFacade):

    def __init__(self):
        super().__init__(models.IdentificationAttempts)

```

Полный код содержится в репозитории <https://github.com/sophia-vdovkina/SignApp>

Приложение В
ДИАГРАММА КЛАССОВ
(рекомендуемое)

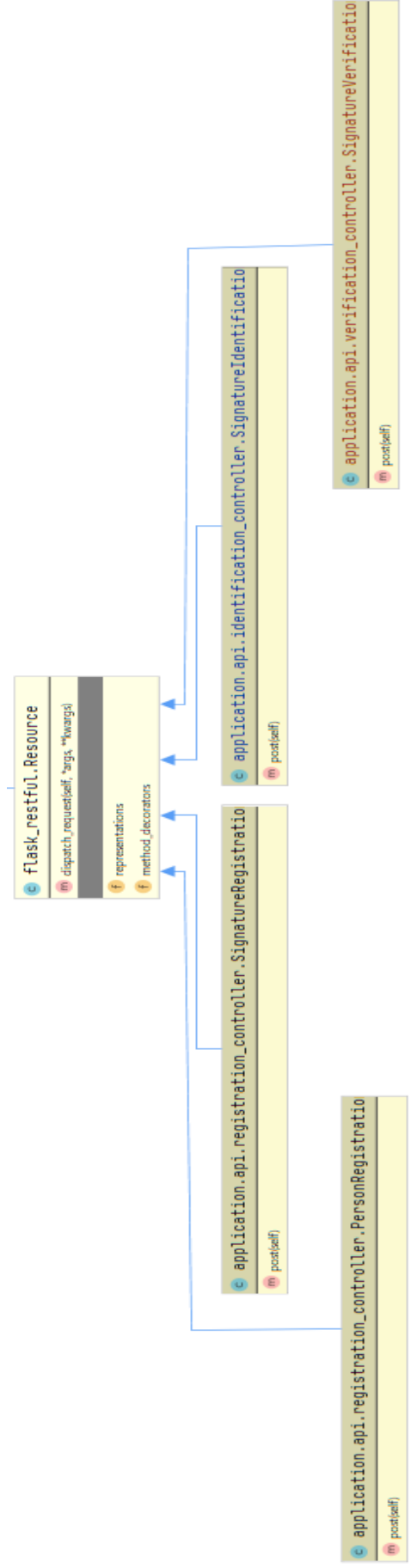


Рисунок В.1 – Диаграмма классов сервисов обработки

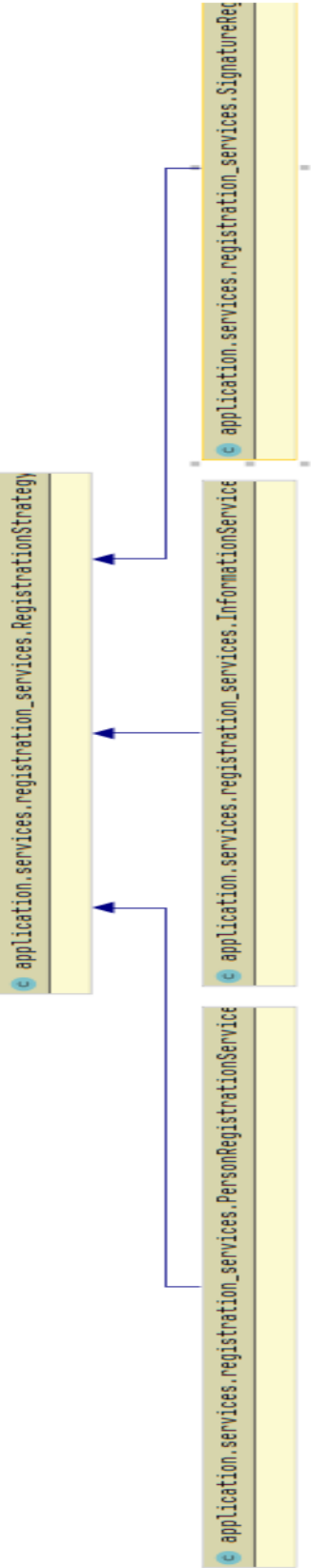


Рисунок В.2 – Диаграмма классов сервиса регистрации

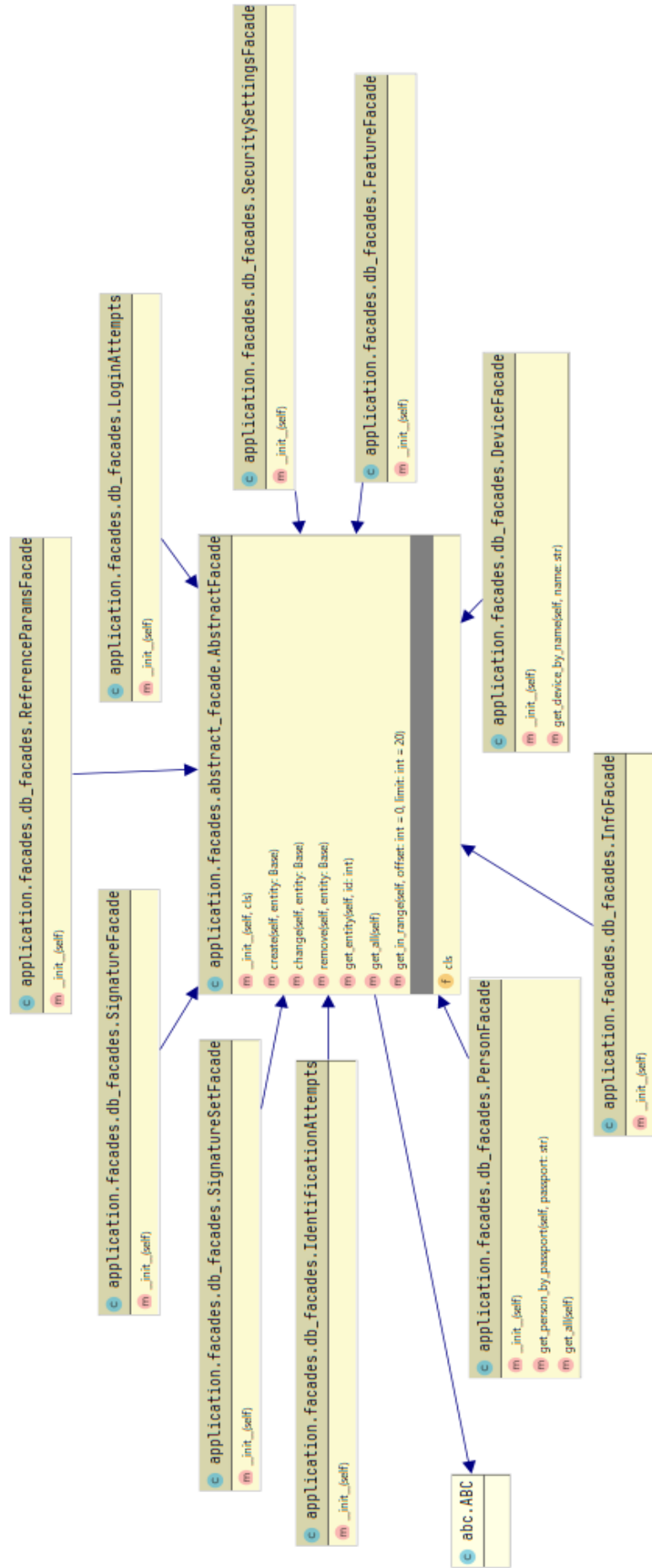


Рисунок В.3 – Диаграмма класса фасадов в БД

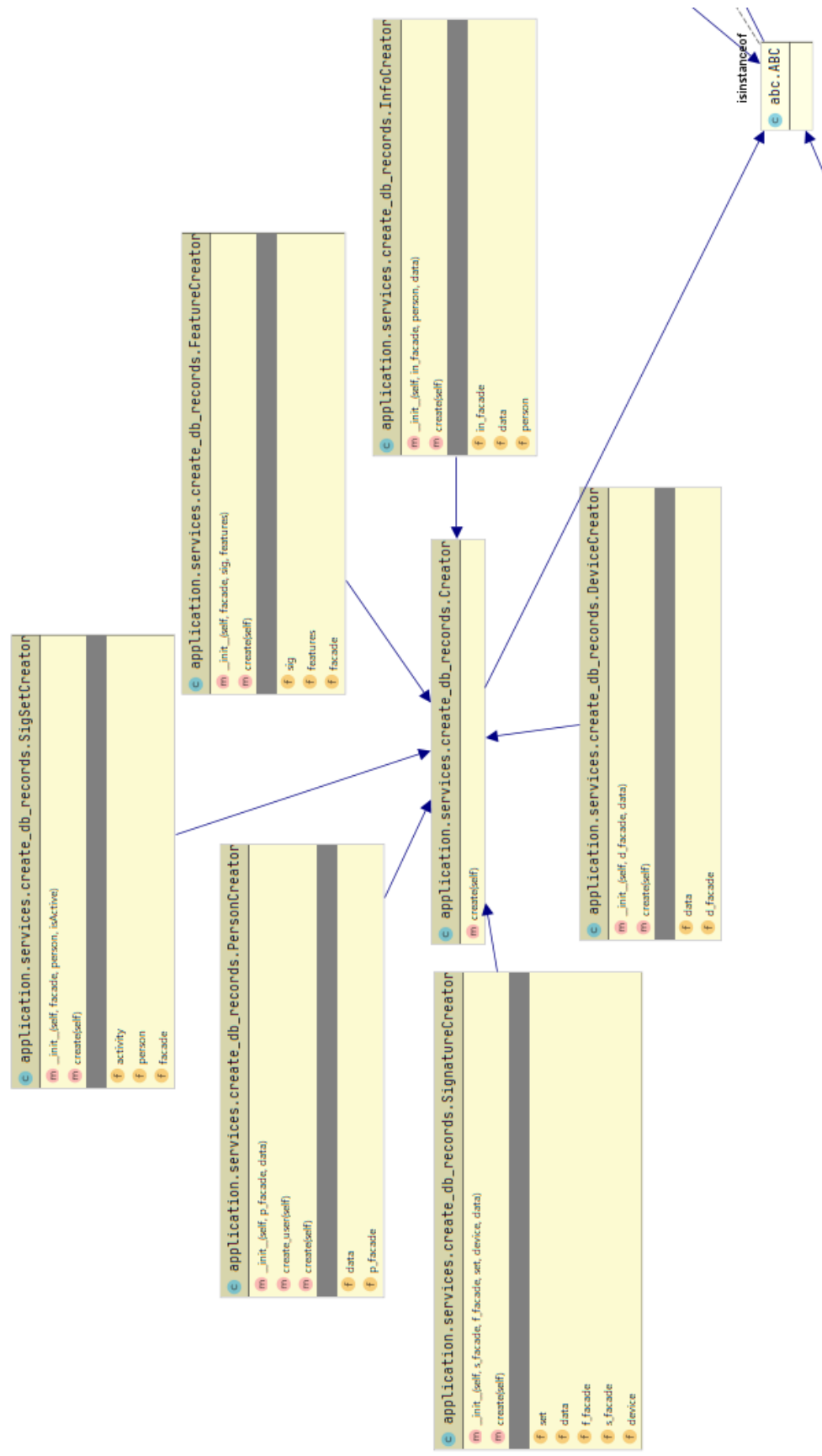


Рисунок В.4 – Диаграмма классов модуля создания записей в БД

Приложение Г
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(рекомендуемое)

2	ddff5144-8ae8-4908-a10e-4335bd29eb64	Norman	Chase	Paschall	5640322255
3	0c169bfc-c8f1-45eb-9fe3-630ecce66863	Sarah	Jeffrey	Wheeler	5640322256

Рисунок Г.1 – Данные людей занесены в БД

	id	values	123 index	signature_id	abc name
1	fb9c78da-0bc6-4bc1-984f-e09e0477a494	{1240.0,1266.0,1266.0,1266.0,1287.0,1321.0,1367.0,1417.0,	0	4036b96c-51c7-4d40-85e7-6eee65c99105	x
2	93112ece-55d1-48f6-9537-95324ba874e7	{7836.0,7830.0,7844.0,7855.0,7886.0,7930.0,7987.0,8066.0,	1	4036b96c-51c7-4d40-85e7-6eee65c99105	y
3	f91373d7-2bcf-484c-baa3-05a93bfabc5b	{274.0,404.0,420.0,433.0,447.0,483.0,499.0,527.0,561.0,616	2	4036b96c-51c7-4d40-85e7-6eee65c99105	p
4	4f157f95-1d0a-4ea3-b98e-ed23c88661cf	{0.0,2.668332812825267,1.4,1.1,3.744329045369811,5.560	3	4036b96c-51c7-4d40-85e7-6eee65c99105	v
5	e2dd9122-2af5-45e7-a016-3a8b1067b5e1	{0.0,0.2668332812825267,-0.12683328128252672,-0.02995	4	4036b96c-51c7-4d40-85e7-6eee65c99105	acceleration
6	4329a180-6fdc-46ad-bfc9-d08415903d09	{-12.994616791916506,17.102728969052375,90.0,63.43494	5	4036b96c-51c7-4d40-85e7-6eee65c99105	angle
7	559db805-8f9b-4de9-84db-573e777381a5	{0.0,0.06845444251657211,0.008,0.012169077428570284,0	6	4036b96c-51c7-4d40-85e7-6eee65c99105	radius

Рисунок Г.2 – Параметры извлеченные из подписи в БД

	id	person_id	signature_id	date_time	123 confidence_value
1	038ed54b-9c86-486a-afc5-874bc1e5cfee	d57f1a1d-2eee-42be-9938-b4a2e50d3b2e	64d199ad-59bf-43d0-9467-b607364b6c95	2021-12-30 12:39:31.645	1

Рисунок Г.3 – Данные идентификации

message: 'Person James Sue Reagan identified'

Рисунок Г.4 – Сообщение об идентификации

	id	person_id	signature_id	date_time	is_verified
1	74be14d7-ee18-4277-ac31-42f80082a3f6	d57f1a1d-2eee-42be-9938-b4a2e50d3b2e	0a88c67c-4748-4ecd-a211-0f2d18034303	2021-12-31 15:29:08.207	[v]

Рисунок Г.5 – Данные верификации

message: 'Person James Sue Reagan verified'

Рисунок Г.6 – Сообщение о верификации