

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

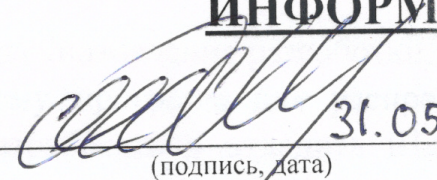
Профиль подготовки: «Программное обеспечение средств ВТ и
автоматизированных систем»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
БАКАЛАВРА**

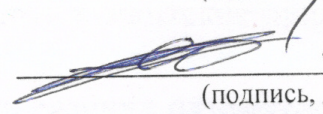
на тему

**ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ
РАСПРЕДЕЛЕННОГО СБОРА ГРАФИЧЕСКОЙ
ИНФОРМАЦИИ**

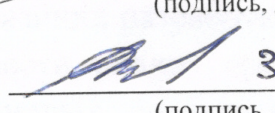
Студент

 31.05.2022 Федяшов М.С.
(подпись, дата)

Руководитель

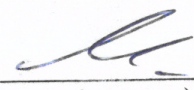
 31.05.2022 Синев М.П.
(подпись, дата)

Нормоконтролёр

 31.05.2022 Бычков А.С.
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 02.06.2022 № 13)

Заведующий кафедрой

 Митрохин М.А.
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от
7.06.2022 № 1)

Секретарь ГЭК

 Гурин Е.И.
(подпись)

Пенза, 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»

Заведующий кафедрой

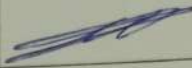
М.А. Митрохин

«26» апреля 2022 г.

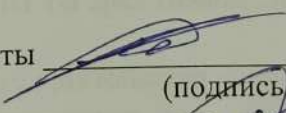
**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

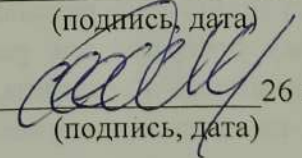
1. Студент Федяшов Марк Сергеевич, гр. 18BB2 факультета вычислительной техники.
 2. Тема работы: «Программное обеспечение для распределенного сбора графической информации».
- Тема утверждена приказом ПГУ № 357/о от «26» апреля 2022 г.
3. Руководитель работы к.т.н., доцент каф. ВТ ПГУ Синев М.П.
 4. Задание на работу (назначение разработки, исходные данные и т.п.).
Разработать систему программ для распределённого сбора графической информации (изображений). Разработать и отладить REST API, спроектировать схему данных для СУБД, разработать клиентское приложение для взаимодействия с REST API.
 5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметных областей: изучить существующие решения проблемы сбора новых данных, подходы к сбору данных, изучить различные типы реляционных СУБД, методологию REST API, подходы к созданию клиентских приложений.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	10	26.04.2022 – 27.04.2022	
2.	Анализ требований	20	28.04.2022 – 30.04.2022	
3.	Выбор технологий, применяемых при проектировании и разработке	10	01.05.2022 – 01.05.2022	
4.	Разработка БД	20	02.05.2022 – 03.05.2022	
5.	Проектирование REST API	20	04.05.2022 – 06.05.2022	
6.	Разработка серверного приложения	80	07.05.2022 – 16.05.2022	
7.	Разработка мобильного приложения	90	17.05.2022 – 27.05.2022	
8.	Оформление пояснительной записки	24	28.05.2022 – 31.05.2022	

Дата выдачи задания «26» апреля 2022 г.

Руководитель бакалаврской работы  26 апреля 2022, Синев М.П.
(подпись, дата)

Задание к исполнению принял студент  26 апреля 2022, Федяшов М.С.
(подпись, дата)

Работу к защите допустить
Декан факультета  06.06.2022 06 июня 2022, Фионова Л.Р.
(подпись, дата)

Реферат

Работа содержит 98 листов, 19 рисунков, 3 таблицы, 8 источников.

АНАЛИЗ ДАННЫХ, КЛИЕНТ-СЕРВЕРНОЕ ВЗАИМОДЕЙСТВИЕ, БД, СУБД, *REST, API, ASP.NET, JETPACK COMPOSE, ANDROID*.

Цель ВКР – разработать систему программ для распределенного сбора графической информации.

Объектом разработки является система программ, обеспечивающее выполнение следующих функций: авторизация пользователя, запечатление объекта с помощью камеры смартфона, возможность классифицировать созданное изображение, управление пользователями.

В результате проведенной работы была разработана и введена в эксплуатацию система программ, отвечающих поставленным функциям. Серверное приложение развёрнуто на виртуальном хостинге и доступно на домене *test-datum.ru*.

Графическая часть выпускной квалификационной работы содержит следующие чертежи:

- *UML*-диаграмма взаимодействия классов-контроллеров и классов-сервисов – 1 лист формата A1;
- *UML*-диаграмма взаимодействия классов-сервисов и классов репозитория – 1 лист формата A1;
- *UML*-диаграмма *view*-классов мобильного приложения – 1 лист формата A1;
- *UML*-диаграмма классов бизнес-логики и *DTO* мобильного приложения – 1 лист формата A1.

Данная система программ позволяет собрать и разметить набор данных, предназначенный для систем на основе машинного обучения.

					ПГУ 1.090301.42.001 ПЗ							
Изм.	Лист	№ докум.	Подпись	Дата	Программное обеспечение для распределённого сбора графической информации	Лит.	Лист	Листов	ФВТ, каф. ВТ гр. 18ВВ2			
Разраб.		Федяшов М.С.						2				98
Провер.		Синев М.П.										
Н. Контр		Бычков А.С.										
												
Утверд.		Митрохин М.А.			Пояснительная записка							

Содержание

Введение.	5
1 Программное обеспечение для распределенного сбора графической информации.	6
1.1 Анализ предметной области	6
1.2 Анализ аналогов.	8
1.3 Выбор решения.	10
1.4 Инструменты.	13
1.4.1 Серверная часть.	13
1.4.2 Клиентская часть.	17
1.5 Архитектура серверной части ПО.	19
1.5.1 Схема базы данных.	19
1.5.2 Описание <i>REST API</i>	20
1.5.3 Описание классов.	23
1.6 Развёртывание серверного ПО.	28
1.7 Архитектура клиентского ПО.	30
1.7.1 Описание основных компонентов мобильного приложения.	30
1.7.2 Пользовательский интерфейс.	31
1.7.2.1 Набор форм для ввода учётных данных.	31
1.7.2.2 Набор форм для создания и разметки обучающего примера	33
1.7.2.3 Набор форм для администрирования.	34
1.7.3 Сервис долговременного хранения сессии.	37
1.7.4 Подсистема бизнес логики.	37
2 Безопасность жизнедеятельности.	39
2.1 Рекомендации по снижению воздействия вредных факторов труда	42
Заключение.	44
Список использованных источников.	45
Приложение А – Листинг программ	46
Приложение Б – <i>UML</i> -диаграммы	90
Приложение В – Презентация	95

Перечень сокращений и обозначений

API (Application Programming Interface) – программный интерфейс приложения

DI (Dependency Injection) – внедрение зависимостей

IoC (Inversion of Control) – инверсия управления

HTML (HyperText Markup Language) – язык гипертекстовой разметки

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста

ID (Identifier) – идентификатор

IT (Informational technologies) – информационные технологии

JSON (JavaScript Object Notation) – текстовый формат обмена данными

REST (Representational State Transfer) – передача состояния представления

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

URI (Uniform Resource Identifier) – унифицированный (единообразный) идентификатор ресурса

XML (eXtensible Markup Language) – расширяемый язык разметки

БД – база данных

ПО – программное обеспечение

РЕПОЗАТОРИЙ – абстракция, способная хранить определенный тип информации

СЕРИАЛИЗАЦИЯ – процесс преобразования объекта в форму, удобную для передачи от одной системы к другой

ТОКЕН – маркер доступа пользователя

СУБД – система управления базами данных

Введение

Наука о данных — раздел информатики, изучающий проблемы анализа, обработки и представления данных в цифровой форме. Объединяет методы по обработке данных в условиях больших объёмов и высокого уровня параллелизма, статистические методы, методы интеллектуального анализа данных и приложения искусственного интеллекта для работы с данными, а также методы проектирования и разработки баз данных.

В настоящее время работа с данными занимает большую часть работы в сфере ИТ. Подъём таких областей, как интернет вещей, автоматическое пилотирование транспорта, а также разнообразные задачи из других сфер привели к высокой ценности данных, имеющих разную природу происхождения. Зачастую Интернет является хорошим источником данных. Тем не менее, при появлении новых задач в области науки о данных может вставать вопрос о сборе совершенно нового набора размеченных данных. Размеченные данные необходимы для многих задач машинного обучения с учителем: от качества разметки и собираемых данных будет зависеть успешность обучения используемой модели.

Поскольку обработка данных является важной частью любого масштабного проекта, существует большое количество сервисов, позволяющих автоматизировать данный процесс. Более того, существуют компании, предоставляющие сбор и обработку данных как отдельную услугу. В дальнейшем эти услуги будут пользоваться большей популярностью, поскольку растёт потребность в данных и в их разметке. Данный факт наталкивает на идею реализации ПО, которое может быть запущено и введено в эксплуатацию собственными силами.

Цель данной работы – создать комплексное ПО для распределённого сбора графической информации.

1 Программное обеспечение для распределенного сбора графической информации

1.1 Анализ предметной области

Основная цель науки о данных - обнаружение закономерностей в данных, извлечение знаний из данных в обобщённой форме. Визуальная информация наиболее важна для человека, с помощью глаз мы получаем до 98% информации об окружающей среде. В машинном обучении данный тип информации также распространён – используются либо отдельные изображения, либо видеопоток. В данной работе мы сосредоточимся на отдельных изображениях, в силу того, что данный тип информации являются исходными данными для большого количества задач, связанных с медициной, агрономией, машиностроением и астрономией – важными отраслями трудовой, научной и экономической деятельности.

Сбор информации представляет из себя комплексный процесс, конечной целью которого является набор данных, наиболее полно характеризующий определенный объект или класс объектов. Для разных задач будет разным и подход к сбору данных. Набор данных может быть собран:

- на основе уже имеющегося большого количества неразмеченных (сырых) данных. Таким образом мы можем собрать необходимое количество информации при помощи сети Интернет или другого иного источника информации, содержащий необходимые для нас данные;

- с помощью обученному персоналу без использования неразмеченных данных. Данный тип задач будет встречаться по мере применения машинного обучения в новых областях для исследования;

- с использованием автоматизированных средств ВТ. В качестве примера можно привести сборочную линию, на которой стоят камеры или датчики, которые в реальном времени отправляют информацию в единую точку сбора для анализа и контроля. В данном случае собираемые данные

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

изначально, вероятнее всего, будут сырыми, поскольку первоначальная разметка для задач, в которой не разработана ни одна подходящая модель, остаётся за человеком.

Отметим, что первая и вторая задача близки по своей постановке, поскольку разметка происходит непосредственно при сборе информации, и данной разметкой занимается человек. На этом этапе мы можем предположить, что решения данных задач по сбору будут также похожи на основе схожести условий. Также отметим, что первая и вторая задача наиболее важны по той причине, что они подразумевают непосредственное участие человека, а, следовательно, и его взаимодействие со средствами ВТ. Более того, вторая задача наиболее интересна, поскольку её решение наиболее масштабно с точки зрения человеческих ресурсов, следовательно – более целесообразно обратить внимание именно на неё.

Исходя из этого, нам следует сконцентрироваться именно на второй задаче.

Машинное обучение предполагает 2 основных типа задач – регрессия (предоставление ответа на поставленный вопрос в непрерывном диапазоне) и классификация – определение класса изображения к одному из предложенных типов.

Существует большое количество известных задач классификации, в которых до сих пор соревнуются решения больших компаний. Целью таких соревнований является выявление наилучших архитектур для моделей, многие из которых будут использоваться в качестве фундамента для решения схожих задач (возможно даже не из смежных областей знаний).

Таковыми известными наборами данных являются:

- *Iris* – набор данных для определения сорта ириса;
- *MNIST* – классификация рукописных цифр;
- *ImageNet* – наиболее известная задача, решения которой принесли хорошие архитектурные решения (*inception block* и *residual block*). Безусловно, разметка этих наборов данных была крайне скрупулёзной.

Исходя из вышеизложенного, мы приходим к выводу, что задача классификации и в настоящее время является фундаментальной, в связи с чем в данной работе ограничимся задачей классификации изображений.

Отметим, что изображение представляет из себя обычный файл, поэтому при определённой архитектуре у нас появится возможность распространить своё решение на другие типы информации (текстовой, звуковой и т.д.).

В конечном счёте изображения должны быть представлены в удобной форме для использования в той или иной библиотеке машинного обучения.

Наиболее распространённой программной библиотекой в академической среде является проект от *Facebook (Meta)* – *PyTorch*. Встроенные классы позволяют крайне просто преобразовывать набор изображений в тензоры, если:

- корневая директория набора данных содержит поддиректории с названиями классов;
- поддиректории содержат изображения, относящиеся к данному классу.

При выборе решения следует отталкиваться от данных особенностей *PyTorch*.

1.2 Анализ аналогов

Были проанализированы наиболее сильные и подходящие аналоги:

- Нейрокор – компания и одноимённый сервис, предоставляющие широкие услуги в сфере Data Science и разработки комплексных средств. Они предоставляют услуги разметки данных для многих задач, в частности – сегментация, детекция и классификация. Так же они продают некоторые уже размеченные наборы данных, предназначенные для новейших задач машинного обучения;
- Яндекс.Толока – крайне популярный сервис для сбора и разметки информации. Он является децентрализованным, что позволяет участвовать в

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

процессе обработки данных большому количеству не вполне подготовленных людей, что может сказаться на конечном качестве набора данных;

- Лейблми – сервис, предоставляющий услуги по сбору, анализу и созданию данных. Наиболее трудоёмкой из них является услуга по созданию данных: по заявлению сервиса, разметчики будут полностью вовлечены в бизнес процессы клиентской компании.

Анализ аналогов показал, что предложения по сбору и разметке существуют уже давно в качестве онлайн сервисов. Наиболее интересный аналог – Лейблми. Данный сервис предлагает также анализ тональностей текстов и транскрибацию речи.

Проанализировав аналоги-конкуренты, мы можем сформировать преимущества данной работы:

- возможность развернуть собственный сервис: поскольку сбором и классификацией занимается другая команда, нет никаких гарантий, что собранные данные не будут проданы через некоторое время или не появятся публично, более того - некоторые данные могут являться конфиденциальными;

- горизонтальное масштабирование: в любой момент мы можем увеличить команду для сбора данных;

- независимость: некоторые задачи требуют сбора данных пребывания в определённых условиях. Маловероятно, что какая-либо команда, кроме наёмных сотрудников будет готова отправиться в сильно отдалённую локацию для сбора данных.

Данный проект будет нацелен на предприятия и организации, которые самостоятельно могут нанять разметчиков или использовать для этих целей штатных сотрудников (что будет наиболее целесообразно, поскольку данные лица уже имеют понятие о бизнес-процессах своего работодателя).

1.3 Выбор решения

Решение предполагает из себя наличие определённого количества клиентов, целью которых является сбор информации и централизованного хранилища, способного хранить и обрабатывать результаты работы клиентов. На данном этапе проектирования мы приходим к идее реализации монолитного серверного ПО, поскольку именно эта архитектура отвечает данной постановке: множество процессов-отправителей и единый процесс-получатель.

Для реализации цели данной задачи нам необходимо реализовать систему взаимодействующих программ, которые смогут предоставить следующий функционал:

- авторизация на основе ролей - каждый член команды может определять те или иные действия в соответствии со своей ролью;
- управление персоналом (пользователями) – руководитель проекта может добавлять или удалять пользователей по своему усмотрению. Учётные записи должны централизованно храниться и обрабатываться с целью возможности управления доступами;
- сбор, разметка и предоставление информации – пользователь может сфотографировать тот или иной объект, назначить ему класс (разметить экземпляр) и отправить его в общее хранилище. Руководитель может просматривать и скачивать уже собранные и размеченные данные. Собранные данные, как и в случае с учётными записями, так же должны храниться и обрабатываться централизованно.

ПО для распределённого сбора графической информации будет предоставлять собой клиент серверное приложение: серверную часть, принимающую запросы от пользователей, и клиентскую часть – мобильное приложение, которое позволит отправлять форму на сервер (изображение и класс сфотографированного объекта). Проектирование мобильного приложения исходит из того, что смартфон в настоящее время обладает большими возможностями по сбору информации: камера, микрофон и т.д.

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

Для данного проекта будет существовать 2 роли – пользователь и учредитель (*user* и *maintainer*). Задача пользователя – собирать и правильно размечать данные, задача учредителя – осуществлять управление пользователями. Только пользователь с ролью *maintainer* может скачивать набор данных.

Основное взаимодействие набора программ будет сводиться к следующим этапам:

- авторизация пользователя: ввод учётных данных для получения доступа к основным возможностям системы;
- аккумуляция информации об исследуемом объекте: в данном случае – фотографирование исследуемого объекта;
- заполнение формы данных: указание класса объекта;
- отправка формы на сервер.

Данные этапы могут повторяться неограниченное количество раз (более точно – для сервера будут существовать пространственные ограничения по памяти), за чем последует финальная выгрузка набора данных.

Исходя из этапов взаимодействия и требуемого функционала мы можем сделать вывод о том, что большую часть времени выполнения система программ будет занята именно вводом-выводом, более точно – межпроцессным взаимодействием (*IPC*) на основе ввода-вывода, поскольку ПО будет располагаться на разных вычислительных машинах.

Основными действиями для серверного ПО будут являться следующие операции:

- приём запроса от клиента;
- сохранение обучающих примеров в файловой системе;
- взаимодействие с СУБД при большинстве случаев обработки запросов (поиск пользователей в СУБД при аутентификации, добавление обучающих примеров, получение классов разметки для текущего набора данных).

Как можно заметить, все эти операции являются операциями ввода-вывода. Данные, переданные клиентом, зачастую имеют элементарную структуру, содержащие несколько полей (пары логин/пароль, файл обучающего примера/класс разметки). При данной структуре будет целесообразно использовать *json*-сериализацию для объектов передачи данных (*DTO*).

В силу того, что для сбора обучающих примеров предполагается использовать ограниченное число участников (клиентов), то и число запросов на единицу времени будет не столь велико. Исходя из этого, в данном случае нам следует использовать *REST* подход к реализации сервера, который отлично справляется с нагрузками данного типа, к тому же является наиболее частым решением для реализации *WebAPI*.

Для хранения данных пользователей и классов разметки следует использовать реляционную базу данных, поскольку протокол взаимодействия, исходя из формализации этапов, в высокой степени стандартизирован. Для хранения обучающих примеров наиболее выгодно использовать файловую систему, поскольку использование СУБД будет нести лишь издержки при формировании единого файла-архива набора данных.

Многие наборы данных собираются с использованием сети Интернет как большой базы знаний. Тем не менее при формировании новых задач необходимых данных может не существовать, поэтому клиент должен иметь возможность самостоятельно создать обучающий экземпляр. Примером такой задачи может служить сбор визуальных данных об объектах естественного или искусственного происхождения, к примеру, если бы известный набор данных *Iris* содержал не параметры сортов растений, а их изображения. Процесс сбора данных будет проходить следующие этапы:

- регистрация пользователя в системе распределённого сбора;

- обучение пользованием ПО, в том числе: понимание процесса правильной разметки (в задачах классификации – понимание того, какой объект относится к классам А, В, С...);
- прибытие в место размещения объектов исследование;
- несколько итераций сбора данных – фотографирование объекта, его классификация, отправка данных.

Исходя из этого клиентом должно являться мобильное приложение, поскольку современные смартфоны имеют всю необходимую аппаратную поддержку для формирования текстовых данных, изображений и аудиоданных, а также просты в эксплуатации. На рисунке 1.1 изображена схема взаимодействия системы.

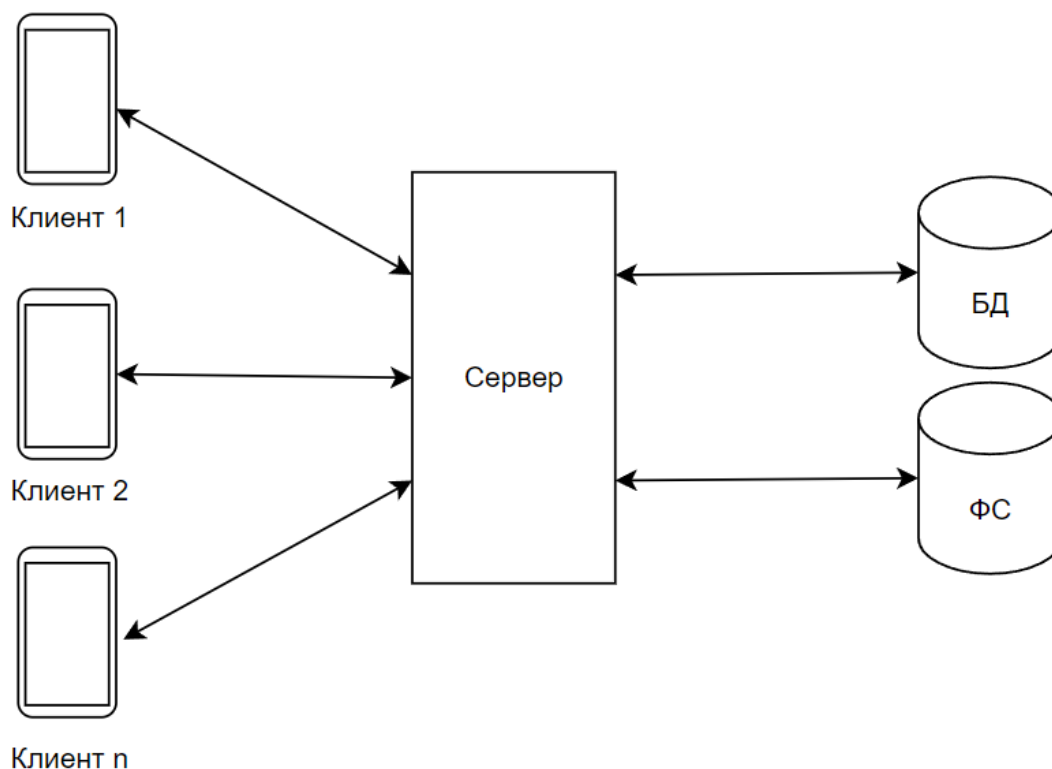


Рисунок 1.1 - Общая схема взаимодействия ПО

1.4 Инструменты

1.4.1 Серверная часть

Программная библиотека для серверной части должен иметь:

- определённую степень известности для потенциального расширения проектов (при её отсутствии библиотека может перестать отвечать новым требованиям);

- поддержку *DI/IoC*, поскольку проект будет содержать компоненты с разным временем жизни;

- поддержку *ORM*, которая обеспечит проект большей читаемостью и скоростью разработки.

Наиболее популярными инструментами для проекта реализации будут два *open-source* конкурента – *Spring Framework* для языка программирования *Java* и *Asp.Net* для платформа *.NET* и языка программирования *C#*. По популярности *Spring* имеет явное преимущество.

Spring Framework – де-факто стандарт для разработки проектов с микросервисной архитектурой, имеет большое количество библиотек и пакетов, облегчающих взаимодействие микросервисов, конфигурацию и эксплуатацию. Конфигурация зависимостей в проекте осуществляется посредством аннотаций: во время старта приложения *Spring* извлекает все классы, фильтрует их по специальными аннотациями, и создаёт необходимые объекты классов. Такой подход станет подходящим в очень больших проектах, где число зависимостей велико[1].

Asp.NET – более легковесное решение, нацеленное на более свободную разработку в плане архитектуры классов и их взаимодействия, стоит лишь более детально изучить инверсию зависимостей, предлагаемую библиотекой [2]. Создание зависимостей в контейнере настраивается в методе *Configure*, где указывается время жизни объекта и его тип. Данный подход делает создание зависимостей более прозрачным и быстрым, поскольку нет необходимости изучать метаданные и атрибуты класса.

Для взаимодействия с СУБД в настоящее время применяют набор библиотек для объектно-реляционного отображения (*ORM*). Преимуществами их использования являются:

- скорость разработки;

- полностью объектно-ориентированный подход при разработке проекта;
- прозрачное использование и поддержка транзакций.

JPA и его реализация *Hibernate* имеют огромный функционал, но вместе с тем и большую сложность, поскольку имеют большое количество тонкостей настройки.

EntityFramework уступает своему конкуренту по функционалу, не имея в своём, требует большее количество кода, но вместе с тем и более явный для использования, при поддержке *LINQ* позволяет писать *SQL* подобные запросы, имеющие большую гибкость в плане вычислений. Более подробный анализ приведён в таблице 1.1[3-4].

Таблица 1.1 - Сравнительная характеристика *Spring* и *.NET*

	<i>Asp.NET</i>	<i>Spring Framework</i>
<i>IoC</i>	Простая и прозрачная конфигурация, быстрота первоначальной инициализации	Гибкость конфигурации
<i>ORM</i>	Прозрачность использования, <i>LINQ</i>	Большое количество аннотаций, позволяющих создавать <i>DAO</i> по именованию
Платформа	Кроссплатформенное решение	Кроссплатформенное решение
Популярность	28.1k на <i>GitHub</i>	47.5k на <i>GitHub</i>

Опираясь на вышеизложенное, в качестве инструмента для реализации *WebAPI* хорошим выбором будет являться *Asp.NET* платформа от *Microsoft*. *ASP.Net* – кроссплатформенная программная библиотека, что означает, что

серверная часть проекта может исполняться как на *Windows*, так и на *Linux* платформах.

Хостинг – важная часть проекта, определяющая надёжность, степень сложности эксплуатации, В качестве хостинга подойдет операционная система, которая легко поддерживается и имеет широкое распространение, а так же поддерживает *Asp.NET* – такой операционной системой является один из дистрибутивов *Linux* – *Debian*.

Для структурированного хранения информации о пользователях, наборе данных и обучающих примерах нам необходима база данных. Сейчас используют как реляционный, так и нереляционный (*NoSQL*) подходы. Важным замечанием является то, что вся информация, поставляемая клиентами, является упорядоченной (все структуры данных являются заранее известными), в силу чего мы будем использовать реляционный подход.

СУБД, используемая в данном проекте, должна иметь следующие характеристики:

- поддержка со стороны ОС на базе ядра *Linux*;
- реализация свойств *ACID*.

С точки зрения *Asp.NET* наилучшим решением было бы использование *MySQL*, поскольку он входит в известный стек технологий *LAMP*. Но набор данных имеет большой объём, в некоторых случаях количество примеров в обучающем наборе может превышать миллион записей. Для хранения большого количества данных хорошо подойдёт известная база данных – *PostgreSQL*. *PostgreSQL* в настоящее время наиболее полно отвечает требованиям *ACID*. Более подробный анализ приведён в таблице 1.2.

Таблица 1.2 - Сравнительная характеристика *PostgreSQL* и *MySQL*

	<i>PostgreSQL</i>	<i>MySQL</i>
Архитектура	Объектно-реляционная	Реляционная
Стандарты	SQL-92, 98, 2003, 2011	Не поддерживает даже SQL-92

	<i>PostgreSQL</i>	<i>MySQL</i>
Лицензия	<i>Open Source</i>	<i>Open Source</i> , принадлежит <i>Oracle</i>
Соответствие ACID	Полное	Частичное
Использование regex	Полное, на любых кодировках	Полное лишь на однобайтовых кодировках
Производительность	Подходит для приложений, использующих большой объём для чтения и записи	Подходит для приложений, использующих большой объём только для чтения

В связи с вышеизложенным для решения поставленной задачи нам следует выбрать *PostgreSQL*.

При публикации приложения на веб-сервере нам потребуется обратный прокси. Цель данного приложения – обеспечить безопасность для внутреннего веб-сервера. Существует по крайней мере два сильных конкурента для данной задачи – *Apache* и *NGINX*.

Как уже говорилось ранее, серверное приложение имеет нагрузки в большей степени на ввод-вывод, и, в связи с тем, что *Nginx* поддерживает *event-based* обработку соединения (неблокирующий ввод-вывод, позволяющие одной нити обрабатывать несколько входящих соединений), то нам следует выбрать именно его.

1.4.2 Клиентская часть

Клиентская часть будет ориентирована на *Android* устройства, поскольку аналитики заявляют, что более 75% мобильных устройств в мире используют операционную систему *Android*, а так же *Android API* предоставляет большие возможности в плане взаимодействия пользователя с устройством[5].

Реализовать такое приложение можно с помощью кроссплатформенных решений (*Flutter, Xamarin, React Native, Apache Cordova*), так и с помощью родных инструментов (*Android SDK*). Поскольку проект предполагает под собой использование аппаратных средств (камеры) и малое время отклика (при длительных задержках сбор информации может быть крайне утомительным, поскольку пользователь занят практически монотонным действием), то нам следует склониться к разработке с помощью средств *Android*, которые используются платформой как базовые компоненты (*native*). Подробный анализ приведён в таблице 1.3

Таблица 1.3 - Сравнительная характеристика мобильных библиотек

	<i>Android Native</i>	<i>Jetpack Compose</i>	<i>Flutter</i>
Кроссплат- форменность	Нет	Нет	Да
Методология	<i>Xml</i> + <i>Java/Kotlin</i>	<i>Kotlin only</i>	<i>Dart only</i>
Скорость разработки	Отдельны вёрстка и код, использование model binding	Код и вёрстка могут быть в рамках одного метода/класса	Код и вёрстка могут быть в рамках одного метода/класса
Модель ис- полнения	<i>ART</i> , натив- ность для плат- формы	<i>ART</i> , нативность для платформы	<i>JNI</i> + <i>Dart minimal runtime</i> + <i>Skia</i> , за- держка при первона- чальной компиляции шейдеров

Для реализации мобильного приложения мы можем использовать стандартный *Android SDK*, который подразумевает под собой *Xml*-разметку для представлений, а так же сложное взаимодействие со списками и другими комплексными объектами пользовательского интерфейса. В настоящее время ему на смену приходит библиотека нового поколения – *Jetpack Compose*,

вдохновлённый успешным проектом *Google* для кроссплатформенной разработки – *Flutter*.

Jetpack Compose – это *SDK* с декларативным подходом к вёрстке, данный библиотека обладает более лаконичным дизайном и стилем кода, что склоняет нас вести разработку клиентской части ПО именно на этом *SDK* [6-7].

1.5 Архитектура серверной части ПО

1.5.1 Схема базы данных

На рисунке 1.2 представлена *ER*-диаграмма таблиц для хранения данных серверного ПО.

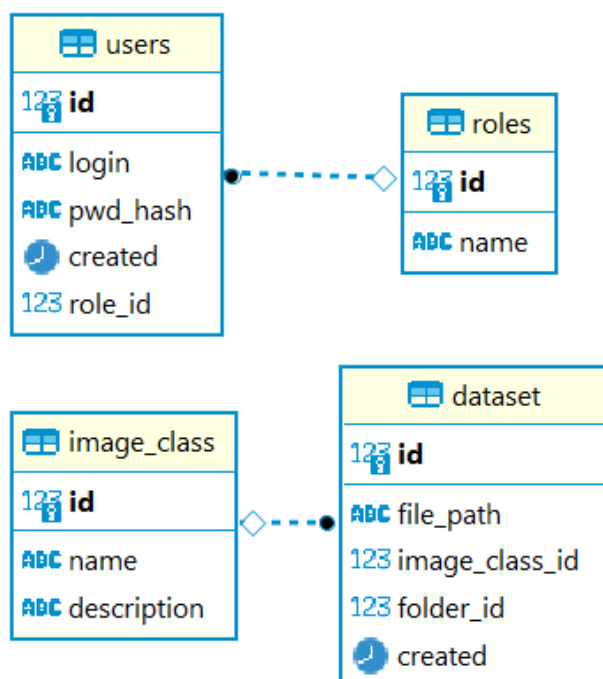


Рисунок 1.2 - *ER* диаграмма таблиц

База данных содержит следующие таблицы:

- *users* – таблица, хранящая список пользователей. Она содержит следующие поля:

- 1) *id* – уникальный идентификатор;
- 2) *pwd_hash* – хэш от пароля и известной строки;

3) *created* – время создания записи;

4) *role_id* – идентификатор роли. Отметим, что в файле конфигурации изначально находится параметр *InitialAccount* – аккаунт первоначального пользователя с ролью *maintainer*;

- *roles* – таблица, хранящая роли. Как говорилось ранее, система предусматривает существование двух ролей – *user* и *maintainer*. Она содержит поля:

1) *id* – уникальный идентификатор;

2) *name* – название роли;

- *dataset_meta* – таблица метаданных для набора данных. Содержит поля:

1) *id* – уникальный идентификатор;

2) *created* – дата создания записи;

3) *train_percent* – часть, попадающая в *train*;

4) *validation_percent* – часть, попадающая в *validation*;

5) *test_percent* – часть, попадающая в *test*.

- *image_class* – таблица, описывающая возможные классы изображений (заполняется пользователем с ролью *maintainer* перед началом сбора данных, не редактируется). Содержит поля:

1) *id* – уникальный идентификатор;

2) *name* – название класса;

3) *description* – описание класса.

Изображения, присылаемые клиентами, будут храниться в файловой системе: в самой базе данных будет лежать путь к файлу.

1.5.2 Описание *REST API*

Исходя из требуемого функционала, описанного в выборе решения, мы можем сформировать контроллеры – логически объединённые методы для взаимодействия с сервером. Клиентское ПО при определённых действиях пользователя отправляет *HTTP* запрос, и получив результат операции,

обрабатывает его. На рисунке 1.3 показан *Swagger*-клиент для взаимодействия с *API*.

Auth – контроллер аутентификации: позволяет получить *jwt*, обновить *jwt* по *GUID* сессии и завершить сессию. Содержит следующие методы:

- *login* – получить пару токенов для взаимодействия с *WebAPI*;
- *refresh* – обновление *bearer* токена без повторной отправки данных пользователя;
- *logout* – отзыв *refresh* токена.

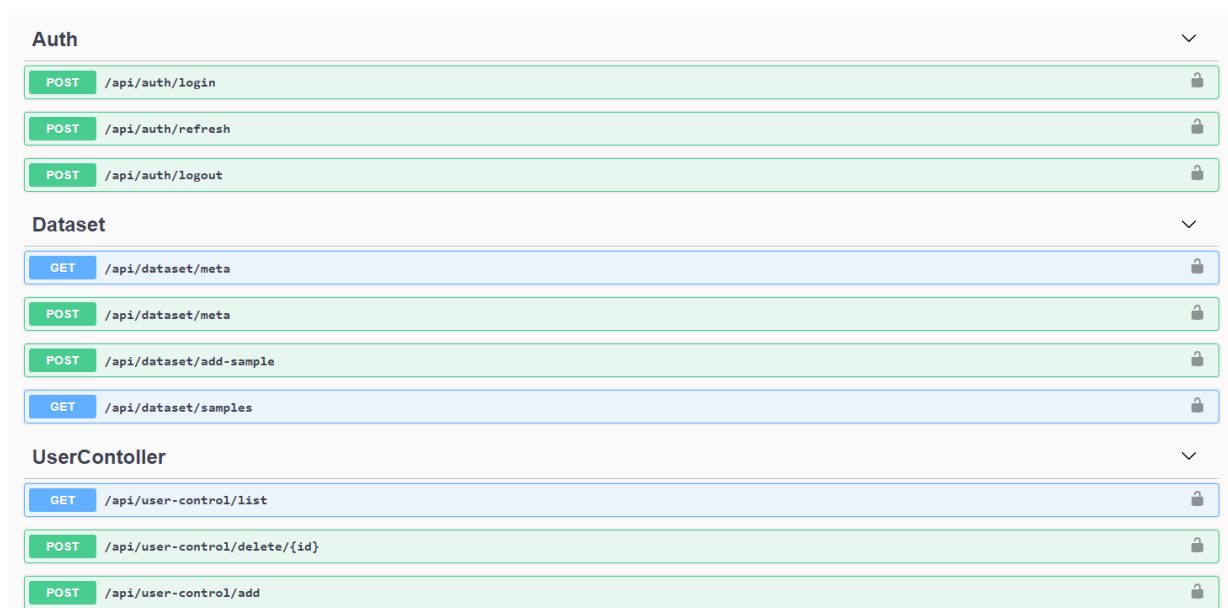


Рисунок 1.3 - *Swagger* для *Rest API*

Dataset – контроллер набора данных: позволяет получить и инициализировать метаданные, а также добавить пример или посмотреть все собранные данные. Содержит следующие методы:

- *GET meta* – получить информацию о наборе данных;
- *POST meta* – инициализировать информацию о наборе данных;
- *add-sample* – получение *multipart/form-data* запроса с изображением;
- *samples* – получить список всех доступных экземпляров.

UserController – контроллер пользователей: данный контроллер доступен только для пользователей с ролью *maintainer*. Содержит методы:

- *List* – получить список пользователей;

- *Delete/{id}* – удалить пользователя по идентификатору;
- *Add* – добавить логин и пароль пользователя в БД отправить приглашение (письмо на почту с логином и паролем для входа в систему).

Существует множество архитектурных паттернов и приёмов для реализации *REST API*, однако наиболее подходящим, и де-факто стандартом разработки является *MVC*, поскольку *REST API* зачастую имеет куда меньше параметров состояния, в отличие от клиентского ПО, которое занимается обработкой навигации, *UI*-событий и т.д.

Архитектура серверной части строится будет строиться по принципу *MVC(Model-View-Controller)*.

Model – слой логики приложения, отвечающий за тот или иной функционал. Он состоит из сервисов, которые обращаются в классы-репозитории – слой, предназначенный для взаимодействие с данными (которые могут храниться как и в оперативной памяти, так и в базе данных).

View – это классы передачи данных (*DTO*), которые необходимы для согласованной работы клиента и сервера.

Controller – классы контроллеры, ответственные за валидацию данных, передачу их в слой логики и формирование ответа на поступивший запрос.

Определившись со слоями взаимодействия, перейдём к проектированию классов серверного ПО. Деление проекта на классы мы произведём в соответствии с набором рекомендаций по *Clean Architecture P. Мартина*, поскольку данные советы позволяют достигнуть более простого масштабирования и сопровождения проекта. Классы логически разбиты на модули, взаимодействие которых представлено на рисунке 1.4.



Рисунок 1.4 - Схема взаимодействия модулей

1.5.3 Описание классов

JwtConfig – статический класс, предназначенный для хранения конфигурации *Jwt*.

Program – класс, являющийся точкой входа в приложение.

Startup – класс конфигуратор приложения. В данном классе происходит создание зависимостей, конфигурация аутентификации и авторизации, настройка доступа к статическим ресурсам.

UserController – контроллер для управления пользователями. К методам данного контроллера разрешено обращаться только пользователям с ролью *mainainer*.

AuthController – контроллер аутентификации. Данный контроллер содержит методы для получения пары токенов, обновления *access*-токена, а также метод для выхода пользователя (поскольку *jwt* нельзя отозвать, то потенциально пользователь может пользоваться сервисами до тех пор, пока *access* токен не перейдёт в состояние «таймаут» - данный вопрос решается клиентской частью проекта: нам будет необходимо удалить токен из внутреннего хранилища и перекинуть пользователя на экран ввода логина и пароля).

DatasetController – контроллер для взаимодействия с набором данных. Данный контроллер поддерживает методы по получению и изменению настроек набора данных, загрузки обучающего примера и получение списка всех собранных экземпляров.

DTO классы являются классами передачи данных, согласующих протокол взаимодействия клиентского и серверного ПО.

Классы сущностей прежде всего предназначены для взаимодействия с *Entity Framework*. *Entity Framework* – наиболее популярная *ORM* для *C#*. В настоящее время практически все веб-приложения используют *ORM* поскольку:

- отпадает необходимость ручного написания *SQL*-запросов;
- более прозрачное участие в процессе инъекции зависимостей.

Сущности спроектированы следующим образом. Каждая сущность является реализацией интерфейса *Identifiable* <*Comparable*>. Если сущность имеет первичный ключ типа *int* – то она реализует интерфейс *IEntityBase* : *Identifiable*<*int*>, но если первичный ключ это *UUID* – *IguidIdentifiable*<*Guid*>.

Классы репозитории спроектированы с целью наименьшего повторного использования кода. Абстрактный класс *CrudRepository* проводит операции над сущностями *IEntityBase*. Поскольку все эти операции однотипны, то для реализации необходимого функционала нам можно прибегнуть к паттерну стратегия: каждый из классов репозитория должен лишь предоставлять ссылку на таблицу, над которой необходимо произвести операции. Например, класс репозиторий для набора данных будет выглядеть следующим образом (рисунки 1.5 и 1.6):

```
public class DatasetRepository : CrudRepository<int, DatasetSample>
{
    public DatasetRepository(DatumDbContext context): base(context)
    {}

    protected override DbSet<DatasetSample> Table => _context.Dataset;
}
```

Рисунок 1.5 – Код класса *DatasetRepository*

```
public abstract class CrudRepository<K, T> : IRepositoryBase<K, T>
where T : class, IIdentifiable<K> where K : IComparable
{
    protected readonly DatumDbContext _context;

    abstract protected DbSet<T> Table { get; }

    public CrudRepository(DatumDbContext context)
    {
        _context = context;
    }

    public virtual IList<T> ReadAll()
    {
        return Table.ToList();
    }
    public virtual T Read(K id)
    {
        return Table.Where(t => t.Id.Equals(id)).FirstOrDefault();
    }
    public virtual T Create(T entity)
    {
        entity = Table.Add(entity).Entity;
        _context.SaveChanges();
        return entity;
    }
}
```

Рисунок 1.6 – Код класса *CrudRepository*

Классы сервисы не имеют интерфейсов, поскольку для них не предусмотрена множественная реализация. Сервисы взаимодействуют с

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						24
Изм.	Лист	№ докум.	Подпись	Дата		

классами репозитория, и совершив набор *CRUD* операций, получают результат. Затем, преобразовав результат в *DTO*, передают обратно в контроллер, после чего сформированный ответ возвращается клиенту.

UserManagementService – класс, ответственный за управление пользователями. Он содержит в себе объект *MailingService*, основная задача которого состоит в отправке приглашения тому или иному пользователю на почту с логином и паролем для входа в систему. Для полноценного функционирования данного сервиса был создан специальный аккаунт *Gmail*, в котором разрешено производить аутентификацию клиента на *SMTP* сервере посредством логина и пароля (пароля приложения).

DatasetService – сервис, чья роль заключается в создании, обработке и хранении информации об экземплярах набора. Данный сервис также сохраняет экземпляры изображений в нужную папку в файловой системе.

AuthenticationService – класс, ответственный за аутентификацию пользователей. Данный сервис содержит ссылку на особый репозиторий – *RefreshTokenRepository*, данные которого находятся не в БД, а лежат в словаре с многопоточным доступом (*ConcurrentDictionary*). Данный функционал позволяет сделать следующее – обнулить все сессии (*refresh* токены) при перезапуске серверного ПО.

Диаграммы классов представлены на рисунках 1.7 – 1.12

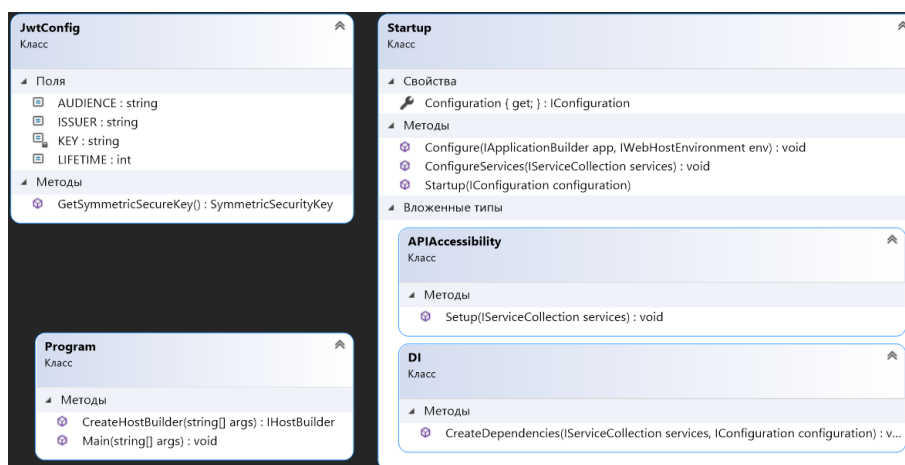


Рисунок 1.7 - Диаграмма классов для инициализации приложения

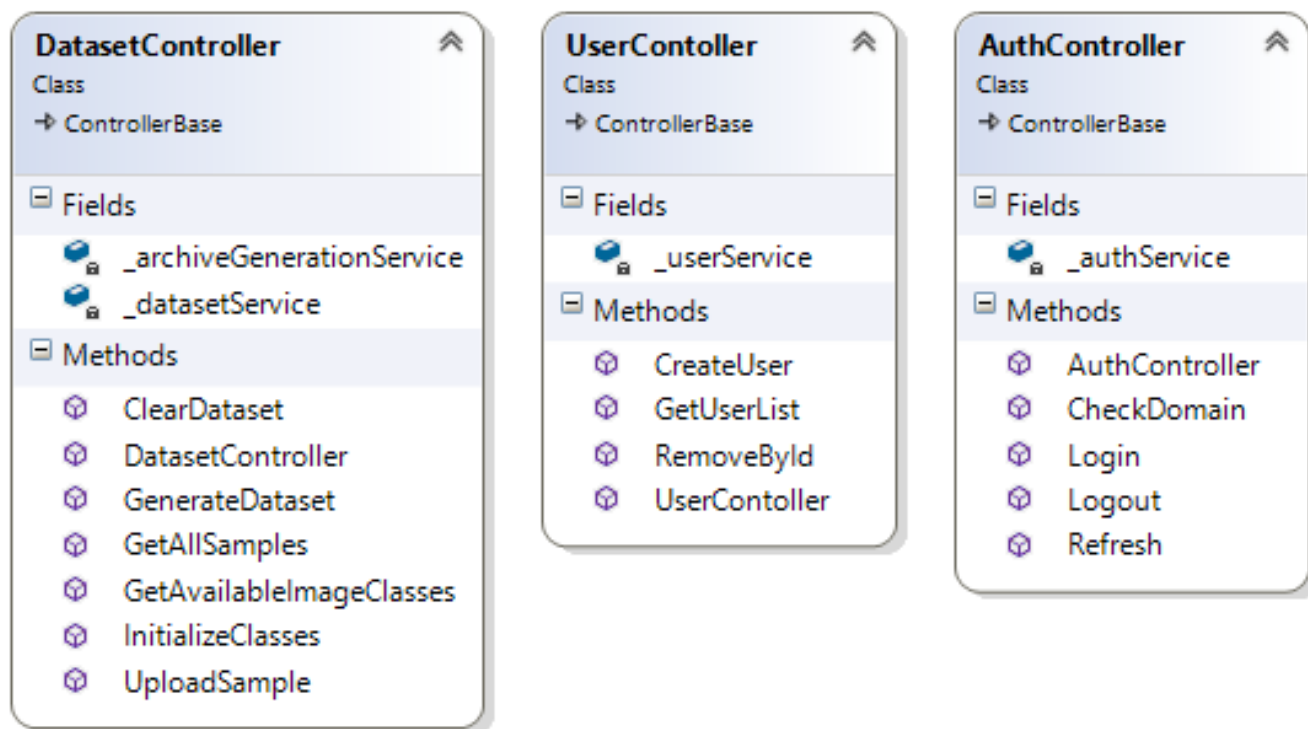


Рисунок 1.8 - Диаграмма классов-контроллеров

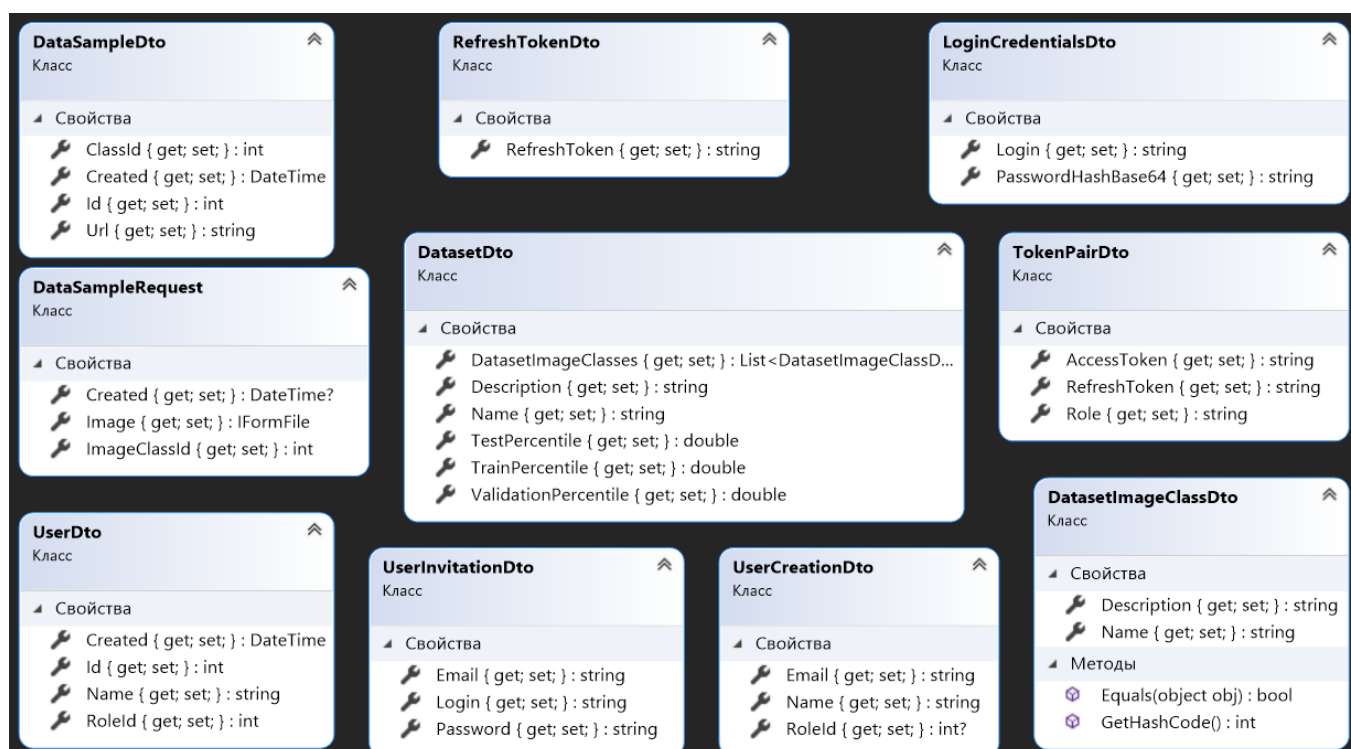


Рисунок 1.9 - Диаграмма классов для передачи данных

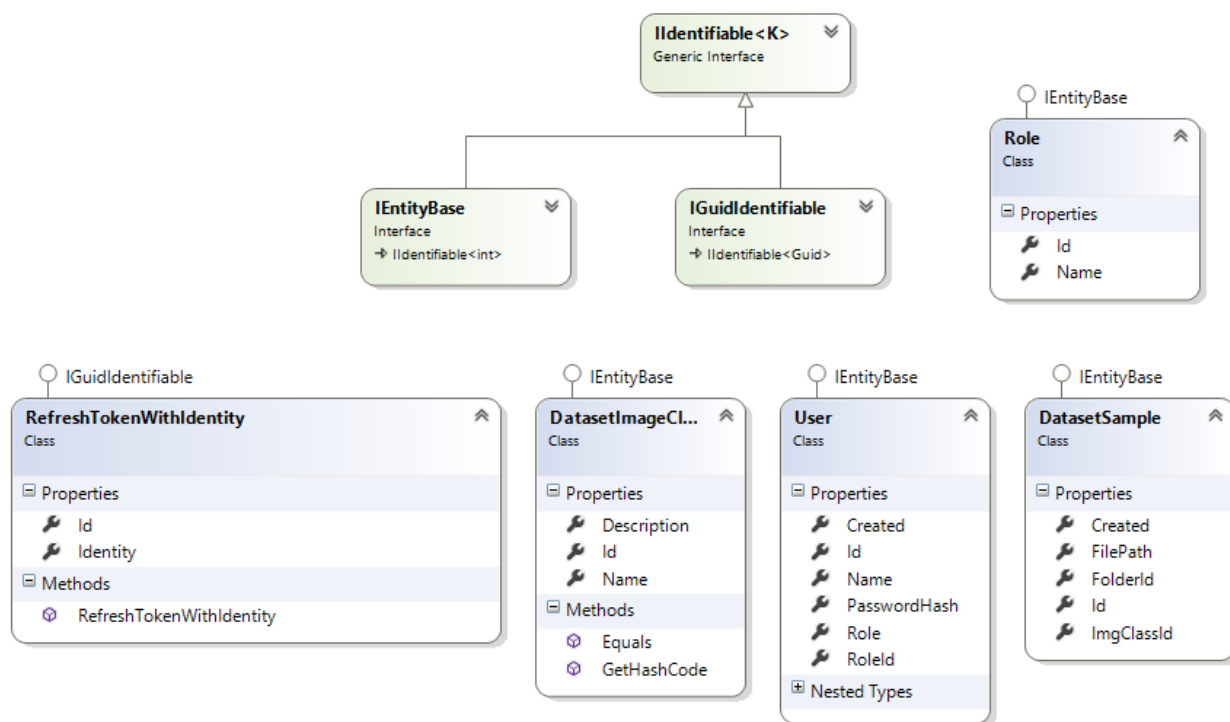


Рисунок 1.10 - Диаграмма классов-сущностей

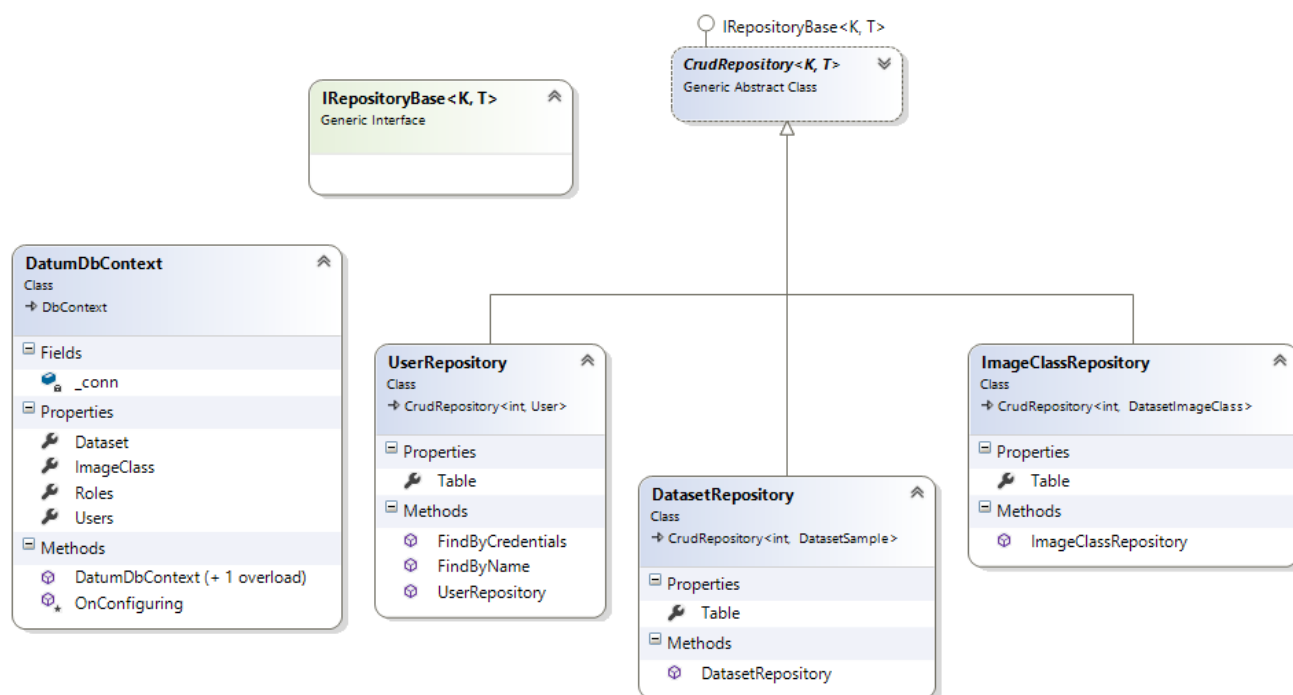


Рисунок 1.11 - Диаграмма классов для доступа к данным (DAO)

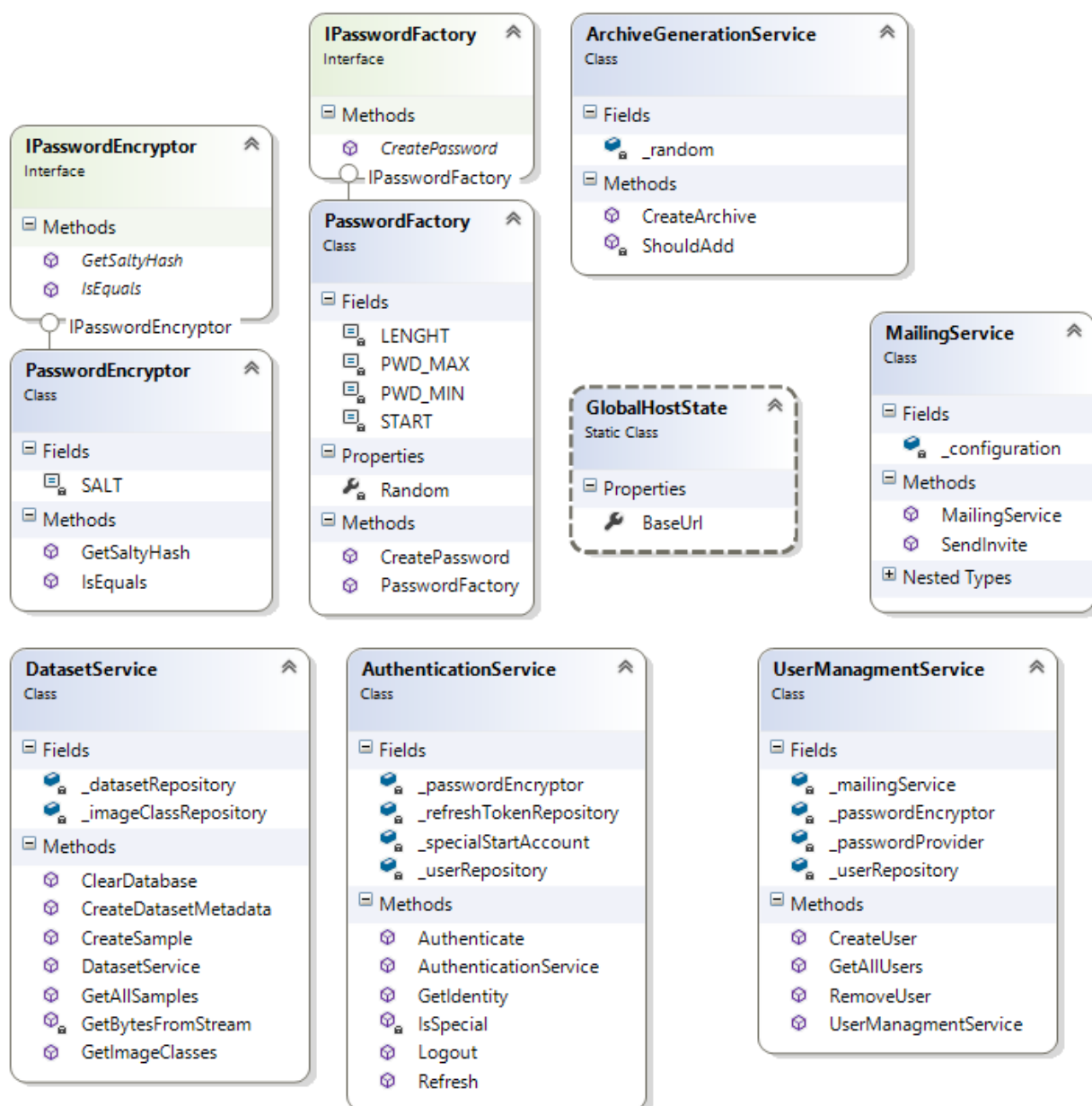


Рисунок 1.12 - Диаграмма классов-сервисов

1.6 Развёртывание серверного ПО

Для публикации серверного ПО нам необходимо выполнить следующие шаги:

- собрать артефакты получившегося проекта (форма для сборки артефактов представлена на рисунке 1.13;
- запаковать артефакты в одну папку для отправки (.zip);

Изм.	Лист	№ докум.	Подпись	Дата

- отправить полученный файл на удалённую машину. Для этого мы можем использовать утилиту `rscp` или любую другую программу, позволяющую передать файл по сети между двумя машинами;
- установить *dotnet* 5.0 на целевой машине. Для этого существует отдельный скрипт (<https://dot.net/v1/dotnet-install.sh>). Необходимо выдать ему права на исполнение и запустить его;
- затем, распаковав артефакты с помощью утилиты *unzip*, необходимо запустить приложение на *localhost:5000* (стандартный адрес для запуска);
- настроить *Nginx* в качестве *reverse proxy* (рисунок 1.14).

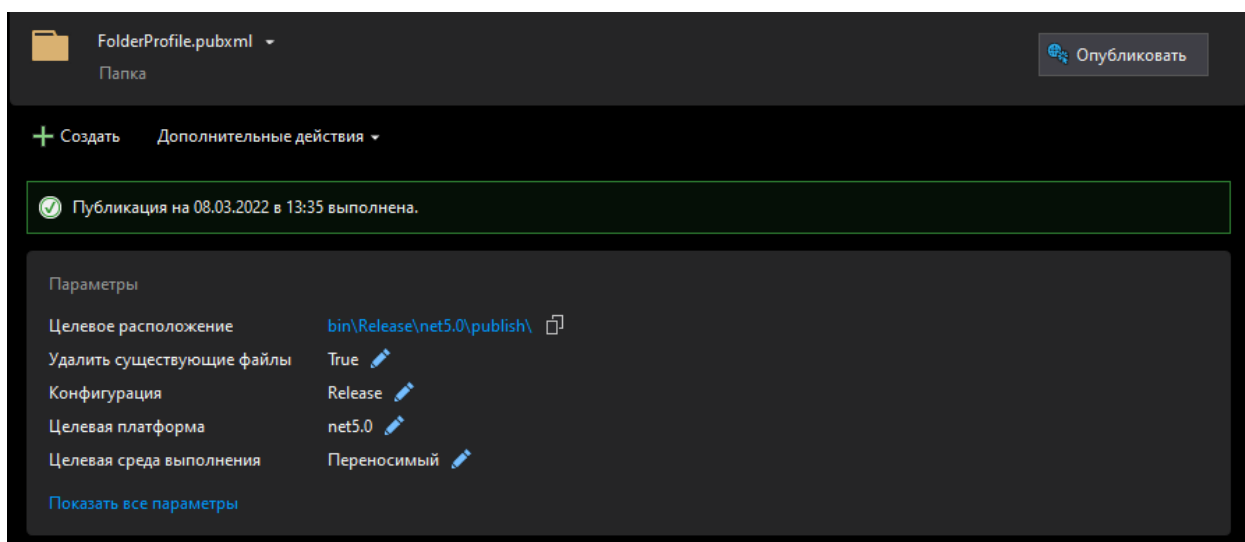


Рисунок 1.13 - Форма в *Visual Studio* 2019 для настройки параметров сборки

```
http {
    charset utf-8;
    server {
        listen      443 ssl;
        server_name test-datum.ru;
        ssl_certificate /etc/ssl/test-datum.crt;
        ssl_certificate_key /etc/ssl/test-datum.key;
        location / {
            proxy_pass http://127.0.0.1:5000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection keep-alive;
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

Рисунок 1.14 – Файл настройки *Nginx*

После данных шагов серверное ПО будет полностью готово к эксплуатации.

1.7 Архитектура клиентского ПО

1.7.1 Описание основных компонентов мобильного приложения

Мобильное приложение является клиентским программным обеспечением с архитектурой *MVP* (данная архитектура неявно «навязывается» библиотекой *Jetpack Compose*, по аналогии с тем, как *SwiftUI* навязывает *MVP*, а *Spring Framework* и *Asp.NET* навязывают *MVC*), содержащим следующие компоненты:

- пользовательский интерфейс, являющийся *view* компонентом паттерна;
- сетевой взаимодействие;
- взаимодействие с данными, предназначенными для долговременного хранения;
- слой бизнес логики, формирующий *model* компонент архитектуры.

В особый компонент следует вынести бизнес логику, поскольку она является связующим звеном между вышеперечисленными компонентами.

Для создания компонентов нам следует использовать набор библиотек:

- *Jetpack compose* – завоевывающая популярность библиотека для построение *UI* в декларативном стиле, используя *code-only* подход;
- *Ktor* – набор библиотек, разработанный для языка *Kotlin*. Существует как серверная версия, позволяющая создавать *WebAPI*-приложения, так и клиентская версия, которая упрощает работу с протоколом *HTTP*;
- долговременное хранение необходимо для сохранения последней сессии пользователя, а именно – *refresh*-токена и адреса сервера: объем таких данных крайне мал. Поэтому целесообразно будет использовать *shared preferences* – *xml* файл, структурой которого являются пары «ключ-значение».

1.7.2 Пользовательский интерфейс

Каждая отдельная экранная форма является потомком абстрактного класса *Page*, который содержит в себе свойства *navController* для навигации и *backStackEntry* для передачи данных между экранными формами. На рисунке 1.15 представлена *UML*-диаграмма абстрактного класса *Page*.

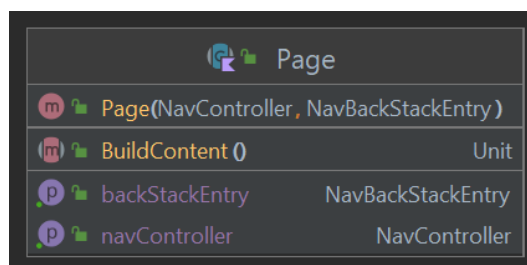


Рисунок 1.15 - *UML*-диаграмма класса *Page*

Все экранные формы расширяют данный абстрактный класс, явно вызывают базовый конструктор и реализуя абстрактный метод *BuildContent* (нотация *UpperCamelCase* используется в качестве индикации того, что функция помечена аннотацией *@Composable*, то есть напрямую строит пользовательский интерфейс).

1.7.2.1 Набор форм для ввода учётных данных

Мобильное приложение реализует основные функциональные требования как для обычного пользователя, так и для владельца набора данных (*user* и *maintainer*).

Набор форм для ввода учётных данных устроен таким образом, чтобы пользователь мог самостоятельно менять обслуживающий сервер (и, следовательно, сам набор данных). Для этого первой экранной формой является форма ввода адреса обслуживающего сервера (*EnterDomainPage*). После ввода доменного имени адреса сервера инициируется запрос по протоколу *https* (*http* в данной системе не предусмотрен, поскольку не обеспечивает шифрование трафика) по пути */api/auth/check*. При получении ответа 200 с сервера мы убеждаемся, что на указанной машине развёрнуто необходимое серверное программное обеспечение.

Пользователь должен иметь учётные данные (в англоязычной литературе - *credentials*), которые, предъявляя серверному ПО, проходит следующие этапы:

- идентификация – определение того, имеется ли такой пользователь в данной системе (определяется по логину);
- аутентификация – процесс доказательства того, что субъект, предоставляющий учётные данные, является тем за кого себя выдаёт (определяется по паре логин/пароль);
- авторизация – определение полномочий пользователя в системе (на основе роли).

Данные этапы реализуются методом *POST api/auth/login*, который в качестве аргумента принимает логин и пароль, при успешной проверке возвращая *JWT*, *refresh*-токен (используемый в качестве восстановления сессии без запроса у пользователя повторного ввода данных) и идентификатор роли (1 – *user*, 2 – *maintainer*).

Таким образом формируется «длительная сессия» - в её процессе учётные данные отправляются единожды: в дальнейшем, при каждом *REST* запросе будет использоваться заголовок, содержащий *JWT*, который инкапсулирует в себе всю необходимую для авторизации информацию.

Учётные данные имеют два источника:

- база данных при участии серверного ПО и мобильного приложения. Пользователь с ролью *maintainer* вводит логин и электронную почту для пользователя, выбирает роль, после чего отправляет *POST* запрос */api/user-controll/add*. Затем сервер генерирует случайный пароль для пользователя, создаёт сущность *User* и добавляет в таблицу *user*, после чего инициирует создание и отправку письма на указанный в запросе эл. адрес с логином и паролем;

- файл конфигурации. Для первоначального добавления пользователей необходима особая учётная запись, которая существует вне таблицы *users* (данное требование не позволит другим пользователям с ролью *maintainer*

удалить её), а также доступна в любой момент времени. Стандартный файл конфигурации для веб-приложений на *.NET* - *appsettings.json* - содержит запись *InitialAccount*, которая соответствует данным требованиям.

В общем виде конечный автомат алгоритма авторизации с точки зрения представлен на рисунке 1.16.

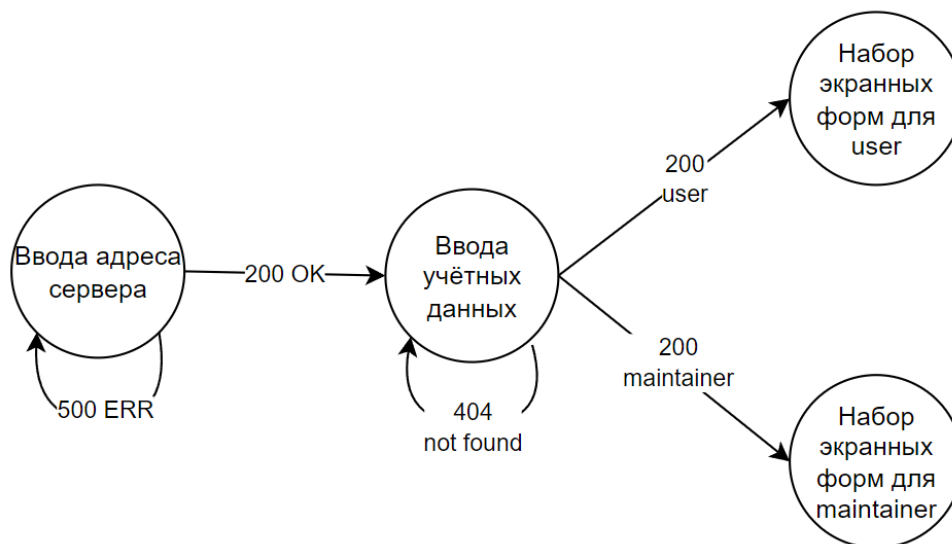


Рисунок 1.16 - Конечный автомат алгоритма аутентификации

На основе роли, которая присылается сервером в ответ на запрос *login*, клиентское приложение решает, к какому набору форм следует перейти. При роли *user* – набор форм для создания фотографии и её дальнейшей разметки, при роли *maintainer* – набор форм для инициализации и добавления новых классов, управления пользователями и генерации архива, содержащего обучающие примеры.

1.7.2.2 Набор форм для создания и разметки обучающего примера

Этот набор форм является основным для формирования набора данных. Прежде, чем перейти к данным формам, необходимо получить доступный для назначения набор классов разметки. Набор классов предоставляется сервером посредством запроса *GET api/dataset/image-classes*.

Поскольку изменяться изображения классов будут крайне редко, целесообразно кешировать результат выполнения запроса и хранить его во

время работы клиентского приложения. Таким образом, классы разметки запрашиваются с сервера в процессе:

- аутентификации (вход в приложение посредством учётных данных);
- обновления сессии по *refresh*-токену (повторный запуск приложения / истечение срока действия *JWT*).

После получения классов для разметки мы переходим на основной экран для пользователя с ролью *user*, целью которого является единственная функция – запечатлеть исследуемый объект. После того, как изображение получено с помощью камеры, мы частично сжимаем изображение (примерно до 70% от исходного) и переходим на экран *SendForm*.

Данный экран позволяет нам просмотреть полученное изображение, выбрать класс этого изображения (из выпадающего списка) и отправить на сервер в качестве обучающего примера через запрос */api/dataset/add-sample*, который принимает 2 обязательных аргумента – класс обучающего примера и сам обучающий пример. После отправки на сервер мы вновь переходим на основной экран пользователя.

Также отметим, что на этом экране находится кнопка для выхода из учётной записи, при этом приложение «забывает» сервер, на котором работал пользователь, и *refresh*-токен для обновления сессии.

1.7.2.3 Набор форм для администрирования

Набор форм для администрирования содержит следующие формы

- форма *Project control* для выбора области администрирования: содержит пункты *Manage users* и *Manage dataset*;
- форма администрирования пользователями: содержит список пользователей (при длительном нажатии на элемент списка появляется всплывающее окно [*alert dialog*] с действием удаления пользователя), боковую кнопку (*floating point button*) для добавления пользователя (при нажатии появляется всплывающее окно с полями, необходимыми для создания пользователя, а так же кнопка подтверждения);

- форма администрирования набора данных – содержит список вложенных действий:

1) форма управления разметкой – содержит список существующих (уже созданных и используемых) классов разметки, список желаемых классов для добавления (ещё не созданных для сервера, существуют в контексте сессии приложения и на данной странице), кнопка для создания нового класса разметки (при нажатии на неё появляется всплывающее окно для создания класса);

2) форма генерации архива на основе имеющихся данных: при нажатии на действие возникает всплывающее окно с элементом «*slider*», характеризующим процент пополнения архива уже существующими данными (на случай, если пользователю необходимо получить только часть собранных данных), при подтверждении действия – отправляется запрос POST `api/dataset/archive` с целью получения будущей ссылки на скачивание архива. Будущая она по той причине, что при большом количестве данных архивация может занять продолжительное время, поэтому сервер заранее генерирует имя файла, и, возвращая его клиенту, в фоновом потоке добавляет файлы в архив. По окончании задачи архив будет сформирован и доступен для скачивания;

3) Кнопка «*Delete dataset*», удаляющая строки в таблице *image_classes*, *dataset* и файлы обучающих примеров.

1.7.2.3 Сетевое взаимодействие

Сетевое взаимодействие разбито на два уровня:

- низкоуровневое взаимодействие формирования запроса и получение результата обработки;

- взаимодействие на уровне описания действия.

Низкоуровневое взаимодействие реализуется объектом класса *NetworkClient*, конструктор которого принимает лямбда функцию *tokenProvider*, вызвав которую данный объект сможет получить *JWT*, а также

1.7.3 Сервис долговременного хранения сессии

Для сохранения данных сессии (диаграмма представлена на рисунке 1.181) мы создадим интерфейс *SettingsRepository* с двумя свойствами – *endpoint* и *refreshToken*, а затем создадим его реализацию *SettingsRepository* через использование *SharedPreferences*.

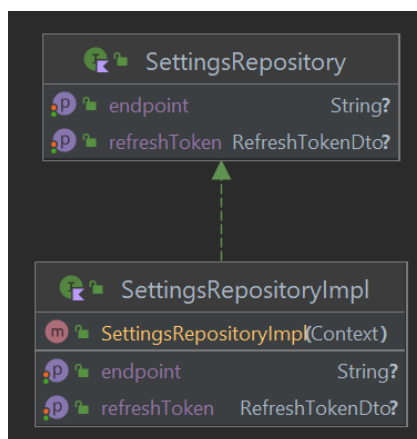


Рисунок 1.18 - UML-диаграмма классов, реализующих долговременное хранение данных

У данных свойств переопределены методы геттеры и сеттеры на вызов приватных встраиваемых методов *getByKey* и *setByKey*, которые обращаются к объекту класса *SharedPreferences*.

1.7.4 Подсистема бизнес логики

Подсистема бизнес логики описывается классом *BusinessLogicService* (рисунок 1.19). Объект данного класса создаётся в момент создания основного *activity*. Данный объект является синглтоном. Во многих случаях методы класса просто «перенаправляют» вызов метода в *apiService*; в случаях, когда метод потенциально может закончиться неудачно, выполняются более сложные действия, обрабатываются потенциальные исключения. Некоторые операции включают в себя несколько *REST* вызовов, как, например, метод добавления нового пользователя: сначала выполняется вызов *api/users/add*, после чего выполняется *api/users/list*, затем результат возвращается методу обработчику пользовательского ввода.

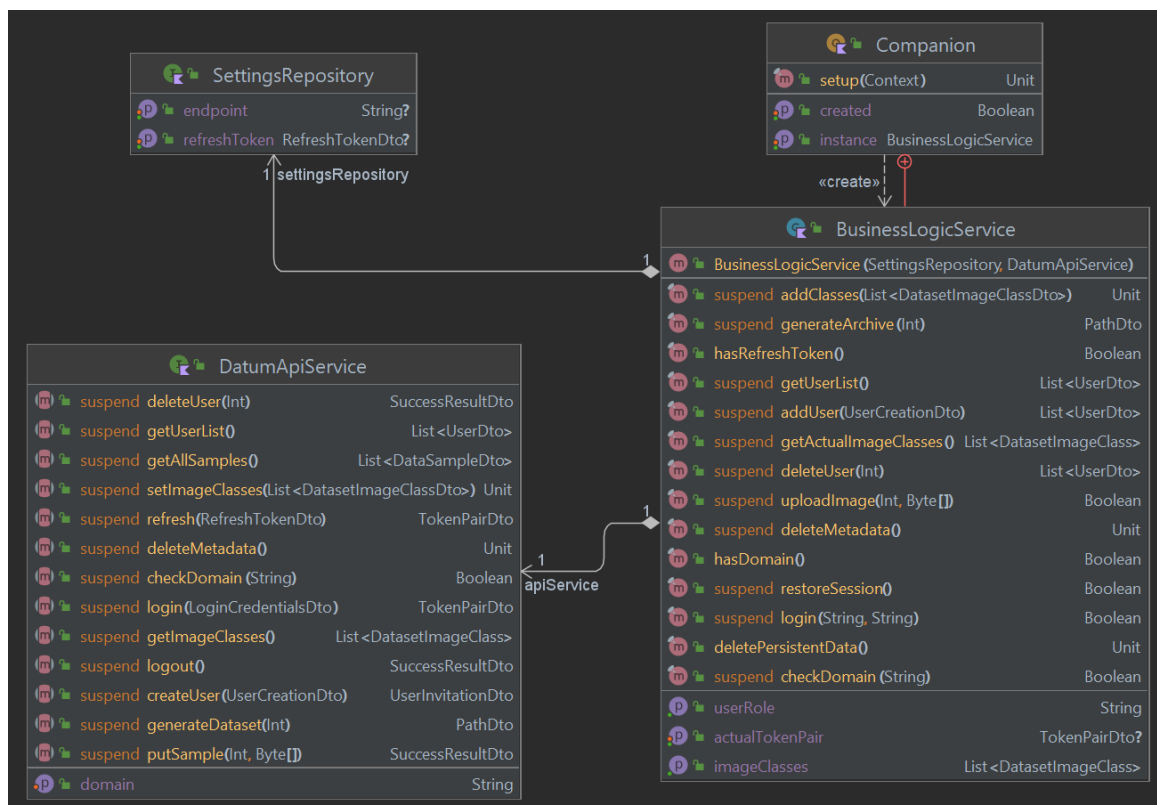


Рисунок 1.19 - UML-диаграмма слоя, реализующего бизнес-логику

Многие операции требуют того, чтобы пользователь не мог прервать данную операцию или начать исполнять ещё одну. Для этого существует особый объект – *ProgressIndicator*. Данный объект содержит метод *blockOperation*, принимающую в качестве аргумента параметризованную лямбду: вначале операции *ui* (пользовательский интерфейс) блокируется, а после её завершения *ui* становится вновь доступен для ввода.

2 Безопасность жизнедеятельности

В настоящее время мобильные устройства глубоко проникли в нашу жизнь, почти каждый житель города имеет смартфон. Вместе с тем такое внедрение может иметь негативное воздействие на здоровье человека, связанное со следующими факторами:

- несоответствие устройства требованиям технического регламента;
- неблагоприятная среда для пользования устройством;
- использование устройства свыше нормативного срока эксплуатации;
- ненадлежащий уход за устройством.

Использовать следует лишь устройства с оформленным сертификатом соответствия требованиям технических регламентов таможенного союза:

- «О безопасности низковольтного оборудования» (ТР ТС 004/2011);
- «Электромагнитная совместимость технических средств» (ТР ТС 020/2011).

Пользователи мобильного приложения наиболее подвержены риску, поскольку контактируют с мобильным устройством непосредственно. Предполагается ряд мер, снижающих негативные воздействия, связанные с использованием смартфона в неблагоприятных средах:

- использование мобильного устройства в температурном диапазоне от 0 до +40С;
- хранение мобильного устройства в температурном режиме 0С - +40С;
- исключение использования устройства при сильном дожде или влажности воздуха выше 95%.

Мобильное устройство необходимо использовать в соответствии со следующими требованиями:

- исключить сильное механическое воздействие на устройство (сжатие, перегибы, падения);
- использовать устройство необходимо вдали от источника тепла;
- исключить перегибы USB кабеля для зарядки устройства;
- для очистки поверхности телефона использовать сухую мягкую ткань;

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						39
Изм.	Лист	№ докум.	Подпись	Дата		

- в случае утечки жидкости аккумуляторной батареи не допускать её попадания на одежду и открытые участки тела;

- необходимо периодически дезинфицировать устройство (особенно актуально при использовании смартфона несколькими лицами). Для этого следует протереть экран и корпус устройства обычными дезинфицирующими салфетками либо салфетками, пропитанными 70% изопропиловым спиртом;

- исключить использования устройства в процессе его зарядки.

Сбор данных потенциально монотонный и трудоёмкий процесс, требующий концентрации внимания и зрения, в следствие чего требуется:

- при ощущении усталости глаз прекратить использование устройства;
- использовать устройство необходимо на достаточном расстоянии от глаз (порядка 40 сантиметров);

- не использовать устройство в тёмном помещении.

Существуют упражнения и рекомендации, позволяющие уменьшить утомление зрительного аппарата:

- частое открытие/закрытие глаз в течение 2-3 минут;
- открывать и закрывать глаза с периодичностью 3-5 секунд;
- массирующими движениями надавливайте на веки;
- перевод взгляда на удалённые объекты, затем на близлежащие, с фокусировкой на протяжении нескольких секунд;
- использование офтальмологических капель для профилактики и снижения усталости глаз.

Процесс сбора данных является трудом, характеризующимся как физической деятельностью (дойти до места дислокации изучаемых объектов, при необходимости принять удобную позу для фотографирования объекта), так и умственной (например, если два изучаемых объекта фактически относятся к разным классам разметки, но визуально их трудно отличить, либо требуется провести определённую аналитику). Процесс работы состоит из большого количества итераций, состоящих из следующих фаз:

- визуальное исследование объекта;

- предварительная классификация;
- запечатление объекта через мобильное приложение;
- окончательная классификация и отправка данных на сервер.

Произведём определение класса условий труда по монотонности трудового процесса:

- плотность сигналов и сообщений крайне мала – трудящийся получает единственное указание приступить к сбору данных (1 сигнал) – оптимальный;

- работа с оптическими приборами и сенсорные нагрузки (мобильное устройство) включает большую часть времени труда (порядка 35-70% в зависимости от удалённости объектов изучения, планируемых для исследования) – оптимальный/вредный;

- нагрузка на голосовой аппарат отсутствует – оптимальный;
- монотонность нагрузок крайне велика – вредный, поскольку:

- 1) число элементарных операций необходимых для реализации простого задания – 4;

- 2) время пассивного наблюдения за ходом технологического процесса – 5-20% в зависимости от сложности классификации.

Таким образом, данный процесс можно отнести классу 3, вредный, подклассу 3.1.

Произведём определение тяжести трудового процесса:

- физическая динамическая нагрузка отсутствует – 1, оптимальный;
- количество стереотипных движений менее 20 000 – 1, оптимальный;
- статическая нагрузка отсутствует – 1, оптимальный;
- рабочая поза относительно рабочей смены – периодическое нахождение в неудобной позе – 2, допустимый;
- наклоны корпуса вынужденные более 30 градусов – зависит от места дислокации объекта, от 0 до 400, следовательно, вариация от «оптимального» до «вредного»;

- перемещения в пространстве, обусловленные технологическим процессом в течение рабочей смены, также зависят от дислокации объектов, от «оптимального» до «вредного».

Таким образом, тяжесть труда варьируется от класса 1 до класса 3 в зависимости от дислокации объектов исследования.

2.1 Рекомендации по снижению воздействия вредных факторов труда

Снизить влияние монотонности труда на здоровье и работоспособность трудящегося предлагается:

- короткими перерывами 5-7 минут через каждые час-полтора;
- включать мероприятия по повышению работоспособности человека (физические упражнения, функциональная музыка, сторонняя информация и раздражители);
- в течение регламентных перерывов предоставить возможность выпить горячие/холодные напитки[8].

Заключение

В ходе выполнения выпускной квалификационной работы была разработана система программ для распределенного сбора графической информации. Разработанная система программ предназначена для использования в сферах искусственного интеллекта и машинного обучения в качестве инструмента по сбору и классификации обучающих примеров. Полученный набор данных в формате *zip*-архива может быть скачан с машины-сервера.

В ходе выполнения выпускной квалификационной работы были разработаны:

- схема базы данных;
- клиентское мобильное приложение;
- серверное веб-приложение (*REST API*);
- последовательность действий и скрипты для публикации веб-приложения на *VPS/VDS*.

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						43
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Гради Б., Объектно-ориентированный анализ и проектирование с примерами приложений. – СПб.: Невский Диалект, 2017. – С. 135–177
2. Марк Дж. Прайс. С# 7 и .NET Core. Кросс-платформенная разработка для профессионалов – СПб.: Питер – 432 С.
3. Бертран М., Объектно-ориентированное конструирование программных систем. – М.: Русская Редакция, 2005. – С. 302–309
4. Spring Boot vs ASP.NET Core: A Showdown // Информационный портал Medium [Электронный ресурс]. Режим доступа: <https://medium.com/@putuprema/spring-boot-vs-asp-net-core-a-showdown1d38b89c6c2d>, свободный
5. Android vs. iOS// Информационный портал gdetraffic[Электронный ресурс]. - Режим доступа: https://gdetraffic.com/Analitika/Android_vs_iOS, свободный
6. Jetpack Compose// Документация Android [Электронный ресурс]. - Режим доступа: <https://developer.android.com/jetpack/compose/documentation>, свободный
7. Таненбаум Э., Уэзеролл Д. Компьютерные сети, изд.5. – СПб.: Изд. Питер, 2012. – С. 18–21.
8. Рекомендации по устранению и предупреждению неблагоприятного влияния монотонии на работоспособность человека в условиях современного производства //Библиотека нормативно-правовых актов СССР [Электронный ресурс] .- Режим доступа: http://www.libussr.ru/doc_ussr/usr_10478.htm, свободный

Приложение А

(обязательное)

Программное обеспечение для распределенного сбора графической
информации

Листинг программ

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						45
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – Листинг серверного приложения

Файл «AuthController.cs»

```
using Datum.Dto;
using Datum.Service;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System;

namespace Datum.Controllers
{
    [Route("api/auth")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private AuthenticationService _authService;

        public AuthController(AuthenticationService authenticationService)
        {
            _authService = authenticationService;
        }

        [HttpGet("check")]
        public IActionResult CheckDomain()
        {
            return Ok();
        }

        [HttpPost("login")]
        public IActionResult Login(LoginCredentialsDto credentials)
        {
            var identity = _authService.GetIdentity(credentials);
            if (identity != null)
            {
                return Ok(identity);
            }
            return Unauthorized();
        }
    }
}
```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		46

```

        return new
JsonResult(_authService.Authenticate(identity));
    }
    else
    {
        HttpContext.Response.StatusCode = 400;
        return new JsonResult(new Exception("No such user"));
    }
}

[AllowAnonymous]
[HttpPost("refresh")]
public IActionResult Refresh(RefreshTokenDto refreshTokenDto)
{
    bool success = Guid.TryParse(refreshTokenDto.RefreshToken,
out Guid key);
    if (success)
    {
        try
        {
            var result = _authService.Refresh(key);
            return new JsonResult(result);
        } catch (Exception e)
        {
            return new JsonResult(e.ToString());
        }
    }
    return new JsonResult(new Exception("Invalid refresh
token"));
}

[Authorize]
[HttpPost("logout")]
public IActionResult Logout()
{
    string name = User.Identity.Name;

```

```

        return new JsonResult(new { success =
        _authService.Logout(name) });
    }
}

```

Файл «DatasetController.cs»

```

using Datum.Dto;
using Datum.Service;
using Datum.Validator;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Datum.Controllers
{
    [Authorize]
    [Route("api/dataset")]
    [ApiController]
    public class DatasetController : ControllerBase
    {
        private DatasetService _service;
        private ArchiveGenerationService
        _archiveGenerationService;

        public DatasetController(DatasetService service,
        ArchiveGenerationService archiveGenerationService)
        {
            _service = service;
            _archiveGenerationService =
        archiveGenerationService;
        }

        [Authorize(Roles = "maintainer")]
        [HttpPost("image-classes")]

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

```

        public IActionResult
InitializeClasses(List<DatasetImageClassDto> initializationDto)
        {

            return new JsonResult(new { success =
_service.CreateDatasetMetadata(initializationDto) });

        }

        [Authorize]
        [HttpPost("add-sample")]
        public IActionResult UploadSample([FromForm]
DataSampleRequest dataSample)
        {

            return new JsonResult(new { success =
_service.CreateSample(dataSample) });

        }

        [Authorize]
        [HttpGet("image-classes")]
        public IActionResult GetAvailableImageClasses()
        {

            var classes = _service.GetImageClasses();

            return new JsonResult(classes);

        }

        [Authorize]
        [HttpGet("samples")]
        public IActionResult GetAllSamples()
        {

            return new JsonResult(new { data =
_service.GetAllSamples() });

        }

        [HttpPost("archive")]
        [PercentValidatorFilter]

```

```

        public IActionResult GenerateDataset([FromQuery] int
percent = 100)
        {
            string filePath =
_archiveGenerationService.CreateArchive(percent);
            return new JsonResult(new { path = filePath });
        }

        [HttpPost("clear")]
        public async Task<IActionResult> ClearDataset()
        {
            await _service.ClearDatabase();
            return NoContent();
        }
    }
}

```

Файл «UserController.cs»

```

using Datum.Dto;
using Datum.Service;
using Microsoft.AspNetCore.Mvc;

namespace Datum.Controllers
{
    [Route("api/user-control")]
    [ApiController]
    public class UserController : ControllerBase
    {
        private UserManagmentService _userService;

        public UserController(UserManagmentService
userManagmentService)
        {
            _userService = userManagmentService;
        }
    }
}

```

```

        [HttpGet("list")]
        public IActionResult GetUserList()
        {
            return new JsonResult(_userService.GetAllUsers());
        }

        [HttpPost("delete/{id}")]
        public IActionResult RemoveById([FromRoute(Name = "id")]
int id)
        {
            var success = _userService.RemoveUser(id);
            return new JsonResult( new { success = success });
        }

        [HttpPost("add")]
        public IActionResult CreateUser(UserCreationDto
userCreation)
        {
            return new
JsonResult(_userService.CreateUser(userCreation));
        }
    }
}

```

Файл «ArchiveGenerationService.cs»

```

using System;
using System.IO;
using System.IO.Compression;
using System.Linq;
using System.Threading.Tasks;

namespace Datum.Service
{
    public class ArchiveGenerationService
    {
        private Random _random = new Random();

        private bool ShouldAdd(int percent) => _random.Next(1,
101) < percent;
    }
}

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

```

public string CreateArchive(int percent)
{

    var srcFolder = "./res";

    var files = Directory.GetFiles(srcFolder);
    foreach(var f in files)
    {
        File.Delete(f);
    }

    var classFolders =
Directory.GetDirectories(srcFolder);

    var dstFolder = Guid.NewGuid().ToString() + ".zip";
    Task t = Task.Run(() =>
    {
        using var archive = ZipFile.Open(srcFolder + "/" +
dstFolder, ZipArchiveMode.Create);
        foreach (var folder in classFolders)
        {
            var files =
Directory.GetFiles(folder).Select((t) =>
Path.GetRelativePath(srcFolder, t)).ToList();

            foreach (var f in files)
            {
                if (ShouldAdd(percent))
                {
                    archive.CreateEntryFromFile(Path.Combine(srcFolder, f), f);
                }
            }
        }
    });
}

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

```

        return $"{GlobalHostState.BaseUrl}/res/{dstFolder}";
    }
}

```

Файл «AuthenticationService.cs»

```

using Datum.Dto;
using Datum.Entity;
using Datum.Repository;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;

namespace Datum.Service
{
    public class AuthenticationService
    {
        private UserRepository _userRepository;
        private RefreshTokenRepository _refreshTokenRepository;
        private User _specialStartAccount;
        private IPasswordEncryptor _passwordEncryptor;

        public AuthenticationService(UserRepository
userRepository, RefreshTokenRepository refreshTokenRepository,
IPasswordEncryptor passwordEncryptor, User specialStartAccount)
        {
            _userRepository = userRepository;
            _refreshTokenRepository = refreshTokenRepository;
            _specialStartAccount = specialStartAccount;
            _passwordEncryptor = passwordEncryptor;
        }
    }
}

```

```

        private bool IsSpecial(LoginCredentialsDto credentials)
        {
            return credentials.Login ==
            _specialStartAccount.Name && credentials.PasswordHashBase64 ==
            _specialStartAccount.PasswordHash;
        }

        public ClaimsIdentity GetIdentity(LoginCredentialsDto
credentials)
        {

            var encryptedCredentials = new LoginCredentialsDto
            {
                Login = credentials.Login,
                PasswordHashBase64 =
                _passwordEncryptor.GetSaltyHash(credentials.PasswordHashBase64)
            };

            User user = IsSpecial(credentials) ?
            _specialStartAccount :
            _userRepository.FindByCredentials(encryptedCredentials);

            ClaimsIdentity identity = null;
            if(user != null)
            {
                var claims = new List<Claim>
                {
                    new Claim(ClaimTypes.Name, user.Name),
                    new Claim(ClaimTypes.Role, user.Role.Name)
                };

                identity = new ClaimsIdentity(claims, "Token",
JwtBearerDefaults.AuthenticationScheme,
ClaimsIdentity.DefaultRoleClaimType);
            }
        }
    }

```

```

        return identity;
    }

    public TokenPairDto Authenticate(ClaimsIdentity
identity)
    {
        var timestamp = DateTime.UtcNow;

        var jwtSrc = new JwtSecurityToken(issuer:
JwtConfig.ISSUER, audience: JwtConfig.AUDIENCE, claims:
identity.Claims,

            expires:
timestamp.Add(TimeSpan.FromMinutes(JwtConfig.LIFETIME)),
            signingCredentials: new
SigningCredentials(JwtConfig.GetSymmetricSecureKey(),
SecurityAlgorithms.HmacSha256)

        );

        var jwt = new
JwtSecurityTokenHandler().WriteToken(jwtSrc);

        var refreshToken = new
RefreshTokenWithIdentity(identity);

        _refreshTokenRepository.Create(refreshToken);

        string role = identity.Claims.Where(x => x.Type ==
ClaimTypes.Role).FirstOrDefault().Value;

        return new TokenPairDto { AccessToken = jwt,
RefreshToken = refreshToken.Id.ToString(), Role = role };
    }

    public TokenPairDto Refresh(Guid refreshToken)
    {
        var refreshEntity =
_refreshTokenRepository.Read(refreshToken);

        _refreshTokenRepository.Delete(refreshToken);

        return Authenticate(refreshEntity.Identity);
    }

    public bool Logout(string userName)
    {
        _refreshTokenRepository.RemoveAllTokensForUser(userName);
    }

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

```

        return true;
    }
}

}

Файл «AuthenticationService.cs»

using Datum.Dto;
using Datum.Entity;
using Datum.Repository;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;

namespace Datum.Service
{
    public class DatasetService
    {
        private DatasetRepository _datasetRepository;
        private ImageClassRepository _imageClassRepository;

        public DatasetService(ImageClassRepository imgClass,
            DatasetRepository datasetRepository)
        {
            _imageClassRepository = imgClass;
            _datasetRepository = datasetRepository;
        }

        public bool
        CreateDatasetMetadata(List<DatasetImageClassDto> dto)
        {
            if (dto.Count != 0)
            {
                try
                {

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						56
Изм.	Лист	№ докум.	Подпись	Дата		

```

        ISet<DatasetImageClassDto> classes = new
HashSet<DatasetImageClassDto>(); //добавляем в сет чтобы
убедиться что класс уникален по имени

        dto.ForEach(t => classes.Add(t));

        foreach(var imgClassDto in classes)
        {
            DatasetImageClass imgClass = new
DatasetImageClass
            {
                Description =
imgClassDto.Description,
                Name = imgClassDto.Name
            };

            if (!_imageClassRepository.Table.Any((c)
=> c.Name == imgClass.Name))
            {
                _imageClassRepository.Create(imgClass);
            }
        }

        return true;

    }

    catch (Exception e)
    {
        return false;
    }
}
else
{
    return false;
}
}

```

```

    }

    private byte[] GetBytesFromStream(Stream stream)
    {
        long size = stream.Length;
        byte[] arr = new byte[size];
        stream.Read(arr, 0, (int)size);
        return arr;
    }

    public bool CreateSample(DataSampleRequest sampleDto)
    {
        if (_imageClassRepository.ReadAll().Count != 0)
        {

            string subDirectory =
            _imageClassRepository.Table.Where(it => it.Id ==
            sampleDto.ImageClassId).FirstOrDefault().Name;

            if (!Directory.Exists(subDirectory))
            {

                Directory.CreateDirectory(subDirectory);

            }

            string name =
            $"{subDirectory}/{Guid.NewGuid()}.{sampleDto.Image.FileName.Split(
            ".").Last()}";

            DatasetSample entity = new DatasetSample
            {

                FolderId = 0,
                FilePath = name,
                ImgClassId = sampleDto.ImageClassId
            };

            _datasetRepository.Create(entity);

```

```

        Stream stream =
sampleDto.Image.OpenReadStream();

        byte[] arr = GetBytesFromStream(stream);

        try
        {
            File.WriteAllBytes(name, arr);
        } catch (DirectoryNotFoundException)
        {
            return false;
        }
        return true;
    }
    else
    {
        return false;
    }
}

public IList<DataSampleDto> GetAllSamples()
{
    List<DataSampleDto> samples = new
List<DataSampleDto>();
    foreach(var s in _datasetRepository.ReadAll())
    {
        samples.Add(new DataSampleDto
        {
            ClassId = s.ImgClassId,
            Created = s.Created,
            Url = GlobalHostState.BaseUrl + "/" +
s.FilePath,
            Id = s.Id
        });
    }
    return samples;
}

```

```

    }

    public IList<DatasetImageClass> GetImageClasses()
    {
        return _imageClassRepository.ReadAll();
    }

    public async Task ClearDatabase()
    {
        await _imageClassRepository.DeleteAll();
        await _datasetRepository.DeleteAll();
    }
}

```

Файл «PasswordEncryptor.cs»

```

using System;
using System.Security.Cryptography;
using System.Text;

namespace Datum.Service
{
    public class PasswordEncryptor : IPasswordEncryptor
    {
        private const string SALT = "*31sg2zXX1";

        public string GetSaltyHash(string password)
        {
            byte[] saltedPwdBytes =
Encoding.ASCII.GetBytes(password + SALT);

            SHA256 sha = SHA256.Create();

            byte[] hash = sha.ComputeHash(saltedPwdBytes);

            return Convert.ToBase64String(hash);
        }

        public bool IsEquals(string password, string
saltyHashBase64)

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						60
Изм.	Лист	№ докум.	Подпись	Дата		

```

        {
            return GetSaltyHash(password) == saltyHashBase64;
        }
    }
}

```

Файл «PasswordFactory.cs»

```

using System;
using System.Security.Cryptography;
using System.Text;

namespace Datum.Service
{
    public class PasswordFactory : IPasswоrdFactory
    {
        private const int PWD_MAX = 12;
        private const int PWD_MIN = 8;

        private const int START = 48;
        private const int LENGHT = 42;

        private RNGCryptoServiceProvider Random { get; set; }

        public PasswordFactory()
        {
            Random = new RNGCryptoServiceProvider();
        }

        public string CreatePassword()
        {
            StringBuilder builder = new StringBuilder();
            int pwdLenght = (new Random()).Next(PWD_MIN, PWD_MAX
+ 1);

            byte[] pwdBytes = new byte[pwdLenght];
            Random.GetBytes(pwdBytes);

```

```

        foreach (byte b in pwdBytes)
        {
            char c = (char) (START + (b % LENGHT));
            if (char.IsLetter(c))
            {
                c = (new Random()).Next(0, 2) == 1 ?
char.ToLower(c) : char.ToUpper(c);
            }
            builder.Append(c);
        }
        return builder.ToString();
    }
}

```

Файл «UserManagmentService.cs»

```

using Datum.Dto;
using Datum.Entity;
using Datum.Repository;
using System;
using System.Collections.Generic;
using System.Linq;

namespace Datum.Service
{
    public class UserManagmentService
    {
        private UserRepository _userRepository;
        private IPasswordFactory _passwordProvider;
        private IPasswordEncryptor _passwordEncryptor;
        private MailingService _mailingService;

        public UserManagmentService(UserRepository
userRepository, IPasswordFactory passwordProvider,
IPasswordEncryptor passwordEncryptor, MailingService
mailingService)
    {

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		62

```

        _userRepository = userRepository;
        _passwordProvider = passwordProvider;
        _passwordEncryptor = passwordEncryptor;
        _mailingService = mailingService;
    }

    public IList<UserDto> GetAllUsers()
    {
        return _userRepository.ReadAll().Select(x =>
        {
            return new UserDto
            {
                Created = x.Created,
                Id = x.Id,
                RoleId = x.RoleId,
                Name = x.Name
            };
        }).ToList();
    }

    public bool RemoveUser(int userId)
    {
        return _userRepository.Delete(userId) != null;
    }

    public UserInvitationDto CreateUser(UserCreationDto
userCreationDto)
    {
        if (_userRepository.FindByName(userCreationDto.Name)
!= null)
        {
            throw new Exception("User with same login
already exist");
        }
    }

```

```

        else
        {
            string password =
                _passwordProvider.CreatePassword();

            string passwordEncrypted =
                _passwordEncryptor.GetSaltyHash(password);

            User user = new User
            {
                Name = userCreationDto.Name,
                PasswordHash = passwordEncrypted,
                RoleId = userCreationDto.RoleId ??
                User.Permission.UserRole.RoleId
            };

            var ininvite = new UserInvitationDto { Email =
                userCreationDto.Email, Login = userCreationDto.Name, Password =
                password };

            user = _userRepository.Create(user);
            _mailingService.SendInvite(ininvite);

            return ininvite;
        }
    }
}

```

Файл «BaseRepository.cs»

```

using Datum.Entity;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

```

```

namespace Datum.Repository

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

```

{
    public interface IRepositoryBase<K, T> where T:
    IIdentifiable<K> where K : IComparable
    {
    }

    public abstract class CrudRepository<K, T> :
    IRepositoryBase<K, T> where T: class, IIdentifiable<K> where K:
    IComparable
    {
        protected readonly DatumDbContext _context;

        abstract public DbSet<T> Table { get; }

        public CrudRepository(DatumDbContext context)
        {
            _context = context;
        }

        public virtual IList<T> ReadAll()
        {
            return Table.ToList();
        }

        public virtual T Read(K id)
        {
            return Table.Where(t =>
t.Id.Equals(id)).FirstOrDefault();
        }

        public virtual T Create(T entity)
        {
            entity = Table.Add(entity).Entity;
            _context.SaveChanges();
            return entity;
        }

        public virtual T Update(T entity, Action<T>
updateCallback)
        {

```

```

        _context.Attach(entity);
        updateCallback(entity);
        _context.SaveChanges();
        return entity;
    }

    public virtual T Delete(K id)
    {
        var obj = Table.Where(t =>
t.Id.Equals(id)).FirstOrDefault();
        _context.Entry(obj).State = EntityState.Deleted;
        _context.SaveChanges();
        return obj;
    }

    public async virtual Task DeleteAll()
    {
        await Table.ForEachAsync(t => Table.Remove(t));
        _context.SaveChanges();
    }
}
}

```

Приложение А.2 – Листинг мобильного приложения

Файл «MainActivity.kt»

```
package com.datum.client

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.ExperimentalMaterialApi
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.navigation.*
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.datum.client.service.BusinessLogicService
import com.datum.client.service.Role
import com.datum.client.types.AlertBinder
import com.datum.client.types.createAlertState
import com.datum.client.types.hide
import com.datum.client.types.show
import com.datum.client.ui.custom.ActionSubmitAlert
import com.datum.client.ui.custom.AlertConfiguration
import com.datum.client.ui.custom.ProgressIndicator
import com.datum.client.ui.page.archive_page.ArchivePage
import com.datum.client.ui.page.archive_page.ArchivePageNavHelper
import com.datum.client.ui.theme.DatumTheme
import kotlinx.coroutines.runBlocking
import com.datum.client.ui.page.dataset_control.DatasetControlNavHelper
import com.datum.client.ui.page.dataset_control.DatasetControlPage
```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		67

```

import
com.datum.client.ui.page.domain_enter.DomainEnterNavHelper
import com.datum.client.ui.page.domain_enter.DomainEnterPage
import
com.datum.client.ui.page.image_classes.ImageClassNavHelper
import com.datum.client.ui.page.image_classes.ImageClassPage
import com.datum.client.ui.page.login.LoginPage
import com.datum.client.ui.page.login.LoginPageNavHelper
import com.datum.client.ui.page.maintainer.MaintainerPage
import
com.datum.client.ui.page.maintainer.MaintainerPageNavHelper
import com.datum.client.ui.page.send_form.SendForm
import com.datum.client.ui.page.send_form.SendFormNavHelper
import com.datum.client.ui.page.user.UserPage
import com.datum.client.ui.page.user.UserPageNavHelper
import com.datum.client.ui.page.user_list.UserListNavHelper
import com.datum.client.ui.page.user_list.UserListPage

class MainActivity : ComponentActivity() {

    lateinit var navController: NavHostController
    companion object {
        val state = createAlertState()
    }

    @OptIn(ExperimentalMaterialApi::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        BusinessLogicService.setup(this)
        setContent {
            DatumTheme {
                Surface(color = MaterialTheme.colors.background)
            }
            App()
        }
    }
}

```

```

        }

    }

}

@Composable
fun WaitMetaInitAlert() {
    ActionSubmitAlert(state = state, config =
AlertConfiguration(
        "Error", "Metadata not initialized by maintainer.
Wait for it.",
        onOk = state::hide, onCancel = state::hide
    ))
}

private fun defineStartPath(): String {
    val service = BusinessLogicService.instance
    return if (service.hasDomain()) {
        runBlocking {
            if (service.hasRefreshToken()) {
                try {
                    if (service.restoreSession()) {
                        val role = service.getUserRole()
                        if (Role.isUser(role))
UserPageNavHelper().substituteArgument() else
MaintainerPageNavHelper().substituteArgument()
                    } else {

LoginPageNavHelper().substituteArgument()
                }
            } catch (e:
DatasetNotInitializedException) {
                state.show()

LoginPageNavHelper().substituteArgument()
            }
        } else {
LoginPageNavHelper().substituteArgument()

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		69

```

        }

    }
} else {
    DomainEnterNavHelper().substituteArgument()
}
}

@ExperimentalMaterialApi
@Composable
fun App() {
    val startPath = defineStartPath()
    navController = rememberNavController()
    ProgressIndicator.Build()
    val state = remember { state }
    AlertBinder(state = state) {
        WaitMetaInitAlert()
    }

    NavHost(navController = navController, startDestination
= startPath) {

        composable(DomainEnterNavHelper().substituteArgument()) {
            DomainEnterPage(navController,
it).BuildContent()
        }

        composable(LoginPageNavHelper().templateUrl()) {
            LoginPage(navController, it).BuildContent()
        }

        composable(UserPageNavHelper().templateUrl()) {
            UserPage(navController, it).BuildContent()
        }

        composable(MaintainerPageNavHelper().templateUrl()) {
            MaintainerPage(navController, it).BuildContent()
        }
    }
}

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }

    val sendForm = SendFormNavHelper()
    composable(sendForm.templateUrl(), arguments =
sendForm.getNavArguments()) {
        SendForm(navController, it).BuildContent()
    }

    val userList = UserListNavHelper()
    composable(userList.templateUrl(), arguments =
userList.getNavArguments()) {
        UserListPage(navController, it).BuildContent()
    }

    val datasetControl = DatasetControlNavHelper()
    composable(datasetControl.templateUrl(), arguments =
datasetControl.getNavArguments()) {
        DatasetControlPage(navController,
it).BuildContent()
    }

    val imageClass = ImageClassNavHelper()
    composable(imageClass.templateUrl(), arguments =
imageClass.getNavArguments()) {
        ImageClassPage(navController, it).BuildContent()
    }

    val archivePage = ArchivePageNavHelper()
    composable(archivePage.templateUrl(), arguments =
archivePage.getNavArguments()) {
        ArchivePage(navController, it).BuildContent()
    }
}
}
}

```

Файл «AdditionalType.kt»

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

```

package com.datum.client.types

import androidx.compose.runtime.Composable
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.mutableStateOf

typealias AlertState = MutableState<Boolean>

fun createAlertState(): AlertState = mutableStateOf(false)

fun AlertState.hide() {
    this.value = false
}

fun AlertState.isActive() = value

fun AlertState.show() {
    this.value = true
}

@Composable
fun AlertBinder(state: AlertState, AlertBuilder: @Composable
(state: AlertState) -> Unit) {
    if(state.isActive()){
        AlertBuilder(state)
    }
}

interface NamedEntity{
    val name: String
}

Файл «UserListPage.kt»

package com.datum.client.ui.page.user_list

```

```

import android.content.Context
import android.widget.Toast
import androidx.compose.foundation.clickable
import androidx.compose.foundation.combinedClickable
import androidx.compose.foundation.gestures.detectTapGestures
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.lazy.rememberLazyListState
import androidx.compose.foundation.rememberScrollState
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.vector.rememberVectorPainter
import androidx.compose.ui.input.key.KeyEvent
import androidx.compose.ui.input.key.KeyEvent.Type
import androidx.compose.ui.input.key.onPreviewKeyEvent
import androidx.compose.ui.input.pointer.pointerInput
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.ui.window.DialogProperties
import androidx.navigation.NavBackStackEntry
import androidx.navigation.NavController
import com.datum.client.dto.UserCreationDto
import com.datum.client.dto.UserDto
import com.datum.client.service.BusinessLogicService
import com.datum.client.service.Role
import com.datum.client.types.AlertBinder
import com.datum.client.types.AlertState

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		73

```

import com.datum.client.types.createAlertState
import com.datum.client.types.hide
import com.datum.client.ui.Page
import com.datum.client.ui.custom.*
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import java.time.Instant
import java.time.LocalDate
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter
import java.time.format.FormatStyle
import java.util.*
import java.util.logging.Level
import java.util.logging.Logger

class UserListPage(navController: NavController, backStackEntry:
NavBackStackEntry): Page(navController, backStackEntry) {

    companion object {

        private val newData: MutableState<List<UserDto>??> =
mutableStateOf(null)

    }

    private fun getCurrentUserList(): List<UserDto> {

        return newData.value ?:
UserListNavHelper().getList(backStackEntry)

    }

    @Composable
    @ExperimentalMaterialApi
    override fun BuildContent() {

        val argument = getCurrentUserList()

        val scrollState = rememberLazyListState()

        val showDeleteDialogState = remember {
createAlertState() }

```

```

        val showAddUserDialogState = remember {
mutableStateOf(false) }

        val deleteUserRef = remember {
mutableStateOf(argument.firstOrNull()) }

        val scope = rememberCoroutineScope()
        val context = LocalContext.current

deleteUserRef.value?.apply {
            AlertBinder(state = showDeleteDialogState) {
                DeleteUserDialog(user = this, showAlert = it,
scope = scope)
            }
        }

AlertBinder(showAddUserDialogState) {
    AddUserDialog(showAddUserDialogState) {
        scope.launch {
            onCreate(it, context)
        }
    }
}

Scaffold(
    floatingActionButton = {
        AddUserFloating(showAddUserDialogState)
    }) {
        LazyColumn(state = scrollState, modifier =
Modifier
            .fillMaxSize()
            .padding(horizontal = 10.dp)) {
            items(argument) {
                UserRow(it) {
                    showDeleteDialogState.value = true
                    deleteUserRef.value = it
                }
            }
        }
    }
}

```

```

        }

        if(argument.last() != it){
            Separator(color = Color.Gray)
        }
    }

}

}

}

@Composable
private fun AddUserFloating(show: MutableState<Boolean>){
    FloatingActionButton(onClick = {
        show.value = true
    }) {
        Icon(painter =
rememberVectorPainter(Icons.Default.Add), "Add User")
    }
}

@Composable
private fun UserRow(user: UserDto, onLongTap: () -> Unit){
    Box(Modifier.pointerInput(Unit){
        detectTapGestures(
            onLongPress = {
                onLongTap()
            }
        )
    }) {
        Row(
            horizontalArrangement =
Arrangement.SpaceBetween,
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier
                .fillMaxWidth()

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

```

        .padding(vertical = 10.dp)
    ) {
        Column {
            Text(user.name, fontSize = 20.sp)
            Text(
                DateTimeFormatter
                    .ofLocalizedDateTime(FormatStyle.SHORT)
                    .withLocale(Locale.UK)
                    .format(LocalDate.parse(user.created)))
        }
        Text(Role.by(user.roleId).name, fontSize =
20.sp)
    }
}

@Composable
private fun AlertDialogButtons(onOk: () -> Unit, onCancel:
() -> Unit){
    Row(horizontalArrangement = Arrangement.End, modifier =
Modifier.fillMaxWidth()) {
        Button(onClick = onOk) {
            Text("OK")
        }
        Box(Modifier.width(20.dp))
        Button(onClick = onCancel) {
            Text("Cancel")
        }
        Box(Modifier.width(10.dp))
    }
}

@Composable

```

```

        private fun DeleteUserDialog(user: UserDto, showAlert:
AlertState, scope: CoroutineScope){

            ActionSubmitAlert(state = showAlert, config =
AlertConfiguration(

                title = "Warning",

                text = "Delete ${user.name}? This user will cannot
upload images!",

                onOk = {

                    scope.launch {

                        withContext(Dispatchers.IO) {

                            withContext(Dispatchers.Main){

                                showAlert.hide()

                            }

                            ProgressIndicator.blockOperation {

                                newData.value =
BusinessLogicService.instance.deleteUser(user.id)

                            }

                        }

                    }

                },

                onCancel = {

                    showAlert.hide()

                }

            ))

        }

@ExperimentalMaterialApi

@Composable

        private fun AddUserDialog(showAlert: MutableState<Boolean>,
onUserAdd: (UserCreationDto) -> Unit){

            val newUserNmae = remember { mutableStateOf ("")}

            val currentRole = remember {
mutableStateOf(Role.ROLES.last())}

            val newUserEmail = remember { mutableStateOf("") }

            val assign = { value: MutableState<String> -> { it:
String -> value.value = it} }

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

```

        AlertDialog(onDismissRequest = { showAlert.value = false
    },

        title = { Text ("Add user") },

        text = {
            Column() {
                Text("Name")

                TextField(value = newUserName.value,
onValueChange = assign(newUserName))

                Box(Modifier.height(20.dp))

                Text("E-mail")

                TextField(value = newUserEmail.value,
onValueChange = assign(newUserEmail))

                Box(Modifier.height(20.dp))

                Text("Role")

                ComplexDropDownMenu(selectedClass =
currentRole, options = Role.ROLES)
            }
        },

        buttons = {
            AlertDialogButtons(onOk = {
                val userCreationDto =
UserCreationDto(newUserName.value, newUserEmail.value,
currentRole.value.id)

                showAlert.value = false

                onUserAdd(userCreationDto)
            }) {
                showAlert.value = false
            }
        },

        properties = DialogProperties(dismissOnBackPressed =
true, dismissOnClickOutside = true)
    )
}

private suspend fun onUserCreate(userCreationDto:
UserCreationDto, context: Context){

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		79

```

        withContext(Dispatchers.IO) {
            val result = when (ProgressIndicator.blockOperation
{
                newData.value =
BusinessLogicService.instance.addUser(userCreationDto)
                newData.value
            } != null) {
                true -> "User created"
                false -> "Bad"
            }
            withContext(Dispatchers.Main) {
                Toast.makeText(context, result,
Toast.LENGTH_SHORT).show()
            }
        }
    }
}

```

Файл «UserListPageNavHelper.kt»

```

package com.datum.client.ui.page.user_list

import androidx.navigation.NamedNavArgument
import androidx.navigation.NavBackStackEntry
import androidx.navigation.NavType
import androidx.navigation.navArgument
import com.datum.client.dto.UserDto
import com.datum.client.repository.ArgumentRepository
import com.datum.client.ui.page.NavHelper
import kotlin.reflect.KClass

class UserListNavHelper: NavHelper() {

    companion object{
        const val ID = "id"
    }
}

```

```

        override fun getNavArguments(): List<NamedNavArgument> {
            return listOf(
                navArgument(ID) {
                    type = NavType.IntType
                }
            )
        }

        override fun templateUrl(): String {
            return "user-list/{id}"
        }

        override fun substituteArgument(vararg arr: Any): String {
            val id = ArgumentRepository.putArgument(arr[0])
            return "user-list/$id"
        }

        fun getList(backStackEntry: NavBackStackEntry):
        List<UserDto> {
            val id = backStackEntry.arguments!![ID] as Int
            return ArgumentRepository.getArgument(id)
        }

        override fun typeOf(): KClass<*> {
            return UserListPage::class
        }
    }
}

```

Файл «BusinessLogicService.kt»

```
package com.datum.client.service
```

```

import android.content.Context
import com.datum.client.DatasetNotInitializedException
import com.datum.client.dto.*
import com.datum.client.repository.SettingsRepository

```

```

import com.datum.client.repository.SettingsRepositoryImpl
import com.datum.client.service.api.DatumApiService
import com.datum.client.service.api.DatumApiServiceImpl

class BusinessLogicService(private val settingsRepository:
SettingsRepository,

                                private val apiService:
DatumApiService = DatumApiServiceImpl()) {

    var actualTokenPair: TokenPairDto? = null

    private lateinit var imageClassesCache:
List<DatasetImageClass>

    companion object {
        lateinit var instance: BusinessLogicService
        var created = false
        fun setup(context: Context){
            if(!created) {
                instance =
BusinessLogicService(SettingsRepositoryImpl(context))
                instance.apply {
                    if (hasDomain()) {
apiService.setDomain(settingsRepository.endpoint!!)
                    }
                }
                created = true
            }
        }

        fun getUserRole(): String = actualTokenPair!!.role
    }

```

```

    fun hasDomain(): Boolean = settingsRepository.endpoint !=
null

    fun hasRefreshToken(): Boolean =
settingsRepository.refreshToken != null

suspend fun checkDomain(domain: String): Boolean {
    val result = apiService.checkDomain(domain)
    if(result){
        apiService.setDomain(domain)
        settingsRepository.endpoint = domain
    }
    return result
}

suspend fun login(login: String, password: String): Boolean
{
    val credentials = LoginCredentialsDto(login, password)
    return try {
        val tokenPair = apiService.login(credentials)
        actualTokenPair = tokenPair
        settingsRepository.refreshToken =
RefreshTokenDto(tokenPair.refreshToken)
        imageClassesCache = apiService.getImageClasses()
        if(imageClassesCache.isEmpty() and
Role.isUser(tokenPair.role)){
            throw DatasetNotInitializedException()
        }
        true
    } catch (e: DatasetNotInitializedException){
        throw e
    } catch (e: Exception) {
        throw e
    }
}

suspend fun restoreSession(): Boolean{

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

```

        val refreshTokenDto = settingsRepository.refreshToken
        return if (refreshTokenDto != null) {
            try {
                val tokenPair =
apiService.refresh(refreshTokenDto)

                settingsRepository.refreshToken =
RefreshTokenDto(tokenPair.refreshToken)

                actualTokenPair = tokenPair

                imageClassesCache = apiService.getImageClasses()

                if (imageClassesCache.isEmpty() and
Role.isUser(tokenPair.role)) {

                    throw DatasetNotInitializedException()

                }

                true
            } catch (e: DatasetNotInitializedException) {

                throw e

            } catch (e: Exception) {

                settingsRepository.refreshToken = null

                false

            }

        } else {

            false

        }

    }

}

fun getImageClasses() = imageClassesCache

suspend fun getActualImageClasses() =
apiService.getImageClasses()

suspend fun uploadImage(id: Int, image: ByteArray) =
apiService.putSample(id, image)?.success ?: false

suspend fun getUserList() = apiService.getUserList()

```

```

suspend fun addUser(user: UserCreationDto): List<UserDto>? {
    return try {
        if(apiService.createUser(user) != null) {
            getUserList()
        }
        else null
    } catch (_: Exception) {
        return null
    }
}

suspend fun deleteUser(userId: Int): List<UserDto>? {
    return if(apiService.deleteUser(userId).success){
        getUserList()
    } else return null
}

fun deletePersistentData(){
    settingsRepository.apply {
        endpoint = null
        refreshToken = null
    }
}

suspend fun deleteMetadata() = apiService.deleteMetadata()

suspend fun generateArchive(percent: Int) =
apiService.generateDataset(percent)

suspend fun addClasses(classes: List<DatasetImageClassDto>)
= apiService.setImageClasses(classes)
}

Файл «DatumApiServiceImpl.kt»

package com.datum.client.service.api

import com.datum.client.dto.*
import com.datum.client.service.BusinessLogicService
import com.datum.client.service.network.NetworkClient

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

```

import com.google.gson.Gson
import io.ktor.client.request.forms.*
import io.ktor.http.*

class DatumApiServiceImpl : DatumApiService {

    private val networkClient: NetworkClient = NetworkClient {
        BusinessLogicService.instance.actualTokenPair?.accessToken }

    override fun setDomain(domain: String){
        networkClient.host = "https://$domain"
    }

    override suspend fun login(credentialsDto:
LoginCredentialsDto): TokenPairDto {
        try {
            return
networkClient.sendAndGetResponseBodyOfType<TokenPairDto>(ApiPath
.Auth.LOGIN, HttpMethod.Post, credentialsDto)
        } catch (e: Exception) {
            throw e
        }
    }

    override suspend fun refresh(refreshTokenDto:
RefreshTokenDto): TokenPairDto {
        try {
            return
networkClient.sendAndGetResponseBodyOfType<TokenPairDto>(ApiPath
.Auth.REFRESH, HttpMethod.Post, refreshTokenDto)
        } catch (e: Exception) {
            throw e
        }
    }

    override suspend fun logout(): SuccessResultDto {
        TODO("Not yet implemented")
    }
}

```

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

```

        private fun buildUrl(domain: String, path: String) =
            "https://$domain$path"

        override suspend fun checkDomain(domain: String): Boolean {
            val url = buildUrl(domain, ApiPath.Auth.CHECK)
            return try {
                val result = networkClient.send(url, HttpMethod.Get)
                result.status == HttpStatusCode.OK
            } catch (e: Exception) {
                false
            }
        }

        override suspend fun setImageClasses(datasetMetadata:
            List<DatasetImageClassDto>) {
            return
            networkClient.sendAndGetResponseBodyOfType(ApiPath.Dataset.IMAGE
                _CLASSES, HttpMethod.Post, datasetMetadata)
        }

        override suspend fun deleteMetadata() {
            networkClient.sendAndGetResponseBodyOfType<Any>(ApiPath.Dataset.
                CLEAR, HttpMethod.Post, "")
        }

        override suspend fun putSample(imageClassId: Int, image:
            ByteArray): SuccessResultDto? {
            val parts = listOf(
                FormPart("ImageClassId", imageClassId),
                FormPart("Image", image, Headers.build {
                    append(HttpHeaders.ContentType, "image/jpeg")
                    append(HttpHeaders.ContentDisposition,
                        "filename=file.jpeg")
                })
            )

```

```

        )

        return try{

            val data =
networkClient.sendMultipart(ApiPath.Dataset.ADD_SAMPLE, parts)

            Gson().fromJson(data, SuccessResultDto::class.java)

        } catch(_: Exception){

            null

        }

    }

    override suspend fun getAllSamples(): List<DataSampleDto> {

        TODO("Not yet implemented")

    }

    override suspend fun getImageClasses():
List<DatasetImageClass> {

        return
networkClient.sendAndGetResponseBodyOfType(ApiPath.Dataset.IMAGE
_CLASSES, HttpMethod.Get, "")

    }

    override suspend fun generateDataset(percent: Int): PathDto
{

        return
networkClient.sendAndGetResponseBodyOfType(ApiPath.Dataset.archi
ve(percent), HttpMethod.Post, "")

    }

    override suspend fun getUserList(): List<UserDto> {

        return
networkClient.sendAndGetResponseBodyOfType(ApiPath.User.LIST,
HttpMethod.Get, "")

    }

}

```

Приложение Б

(обязательное)

Программное обеспечение для распределенного сбора графической
информации
UML-диаграммы

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.1 – Диаграмма классов серверного приложения

					ПГУ 1.09.03.42.001 Д11			
Изм.	Лист	№ докум.	Подпись	Дата	Программное обеспечение для распределенного сбора графической информации Серверное приложение	Лит.	Масса	Масштаб
Разраб.	Федяшов М.С.							
Прое.	Синев М.П.							
Т.контр.						Лист	Листов	
Н.контр.	Бычков А.С.							
Утверд.	Митрохин М.А.				ФВТ, ар. 18ВВ2			

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.42.001 ПЗ

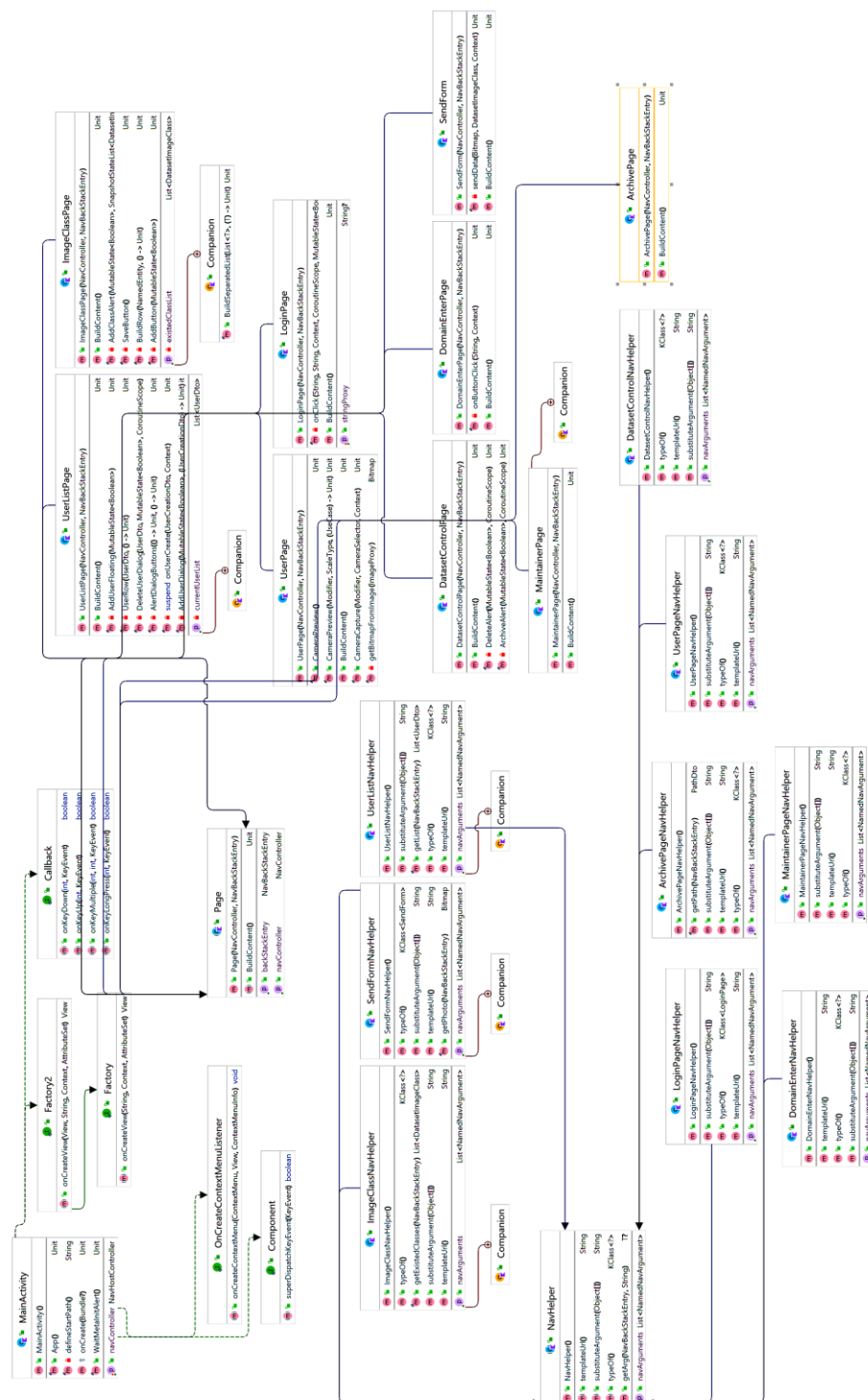
Лист

90

Изм.	Лист	№ докум.	Подпись	Дата
Разраб.	Федяшов М.С.			
Пров.	Синев М.П.			
Т.контр.				
Н.контр.	Бычков А.С.			
Утверд.	Митрохин М.А.			

ПГУ 1.09.03.42.001 Д11			
Программное обеспечение для распределенного сбора графической информации Серверное приложение			
Лист		Листов	
ФБТ, ар. 18ВВ2			

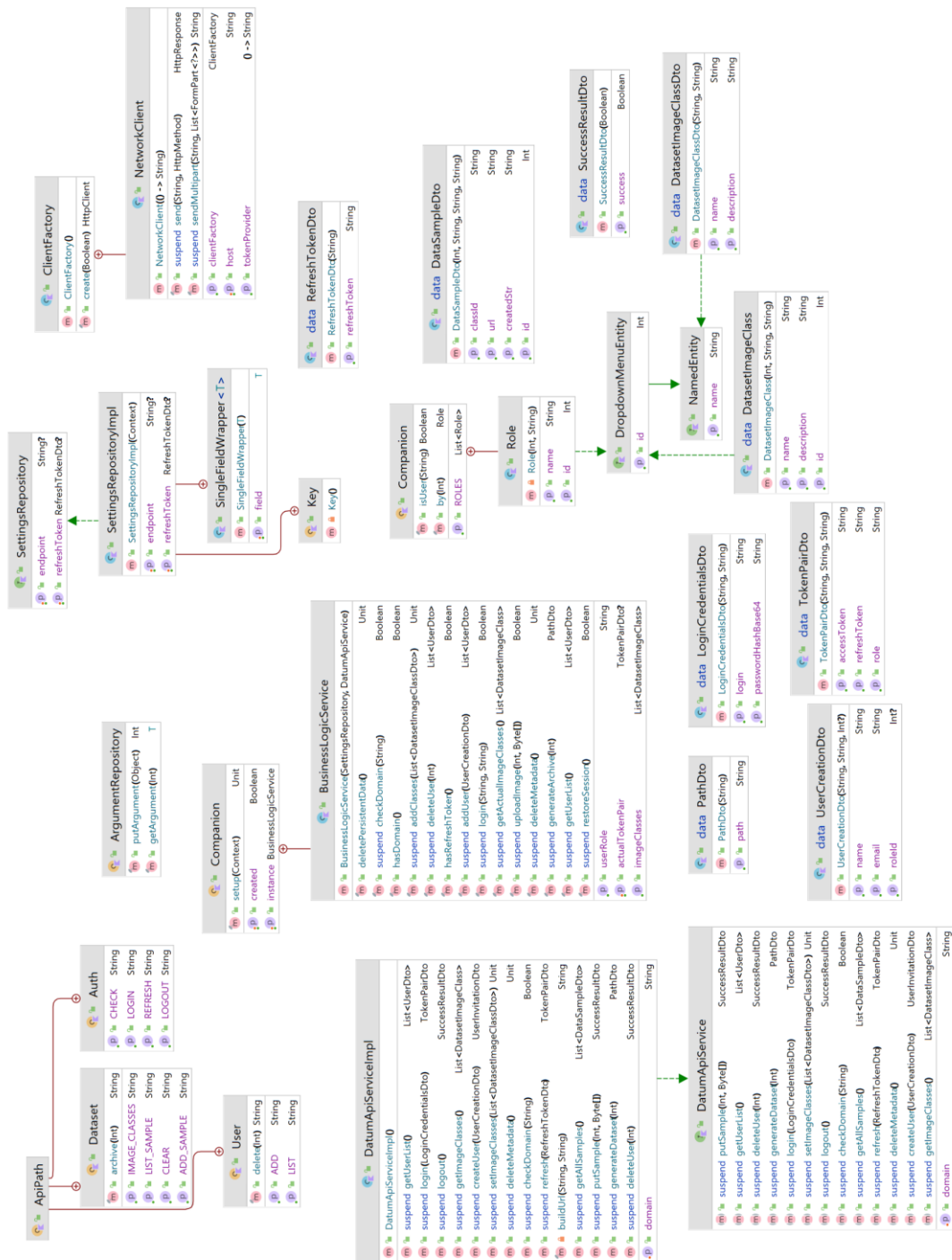
Приложение Б.2 – Диаграмма классов клиентского приложения



					ПГУ 1.09.03.02.001 Д11			
					Программное обеспечение для распределенного сбора графической информации Клиентское приложение	Лит.	Масса	Масштаб
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Федосов М.С.						
Пров.		Синев М.П.						
Т.контр.						Лист	Листов	
И.контр.		Бычков А.С.				ФСТ, стр. 188Б2		
Утверд.		Митрохин М.А.						

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ 1.09.03.01.42.001 ПЗ



Изм.	Лист	№ докум.	Подпись	Дата	
Разр.	Федяев М.С.				
Пров.	Синев М.П.				
Т.контр.					
Н.контр.	Бычков А.С.				
Утверд.	Митрохин М.А.				

ПГУ 1.09.03.42.001 Д11					
Программное обеспечение для распределенного сбора графической информации Клиентское приложение	Лит.		Масса		Масштаб
	Лист		Листов		
	ФБТ, ар. 18ББ2				

Изм.	Лист	№ докум.	Подпись	Дата

Приложение В

(обязательное)

Программное обеспечение для распределенного сбора графической
информации
Презентация

					ПГУ 1.09.03.01.42.001 ПЗ	Лист
						94
Изм.	Лист	№ докум.	Подпись	Дата		

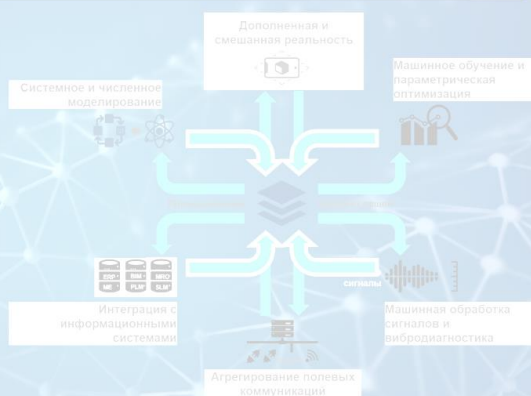
Программное обеспечение для распределенного сбора графической информации

АВТОР: ФЕДЯШОВ МАРК, 18882

НАУЧНЫЙ РУКОВОДИТЕЛЬ: СИНЕВ М.П.

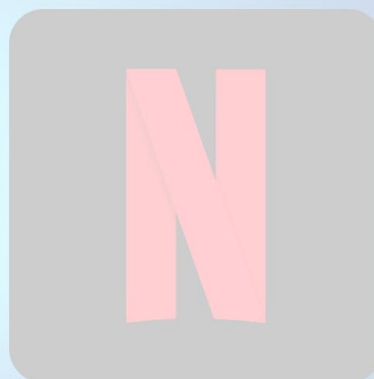
Актуальность

Системы искусственного интеллекта на основе ML проникли в большое количество сфер нашей жизни, от индустрии развлечений до военной промышленности



На примере Netflix

- A/B тестирование
- Персонализация контента и обложек
- Анализ («ходовых») тематик фильмов



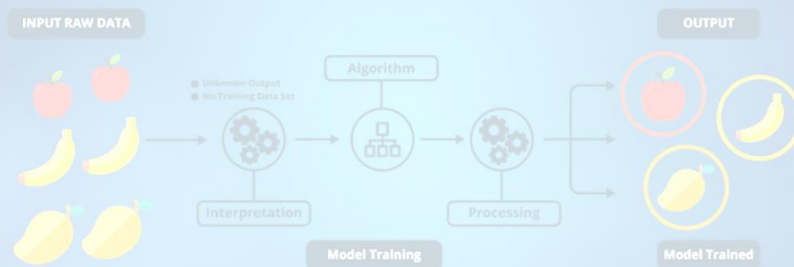
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.42.001 ПЗ

Лист

95

Обучение с учителем



Существующие решения

LabelMe 

 Toloka

NEURO
CORE

Решение

Решением является создание системы программ с клиент-серверным взаимодействием, позволяющее собирать и размечать обучающие примеры, управлять пользователями и датасетом

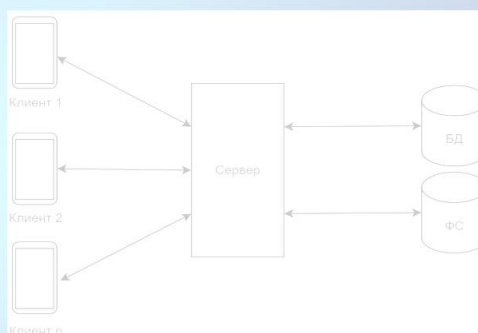


Схема взаимодействия компонентов

Архитектура
устроена по
классическому
паттерну MVC для
WebAPI приложений

Role-based
авторизация

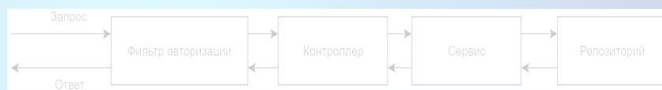
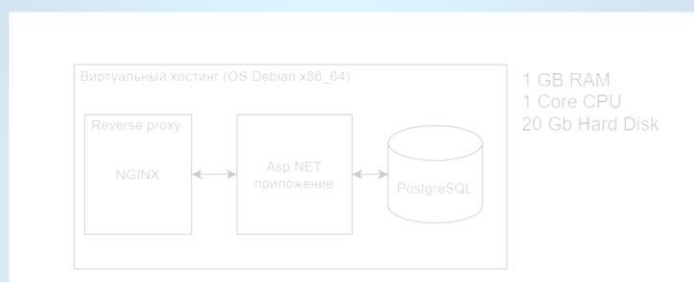


Схема развёртки Rest API



Инструменты мобильного приложения

При реализации мобильного приложения
применялись

- 1) Jetpack Compose для создания UI
- 2) Ktor для взаимодействия с сетью



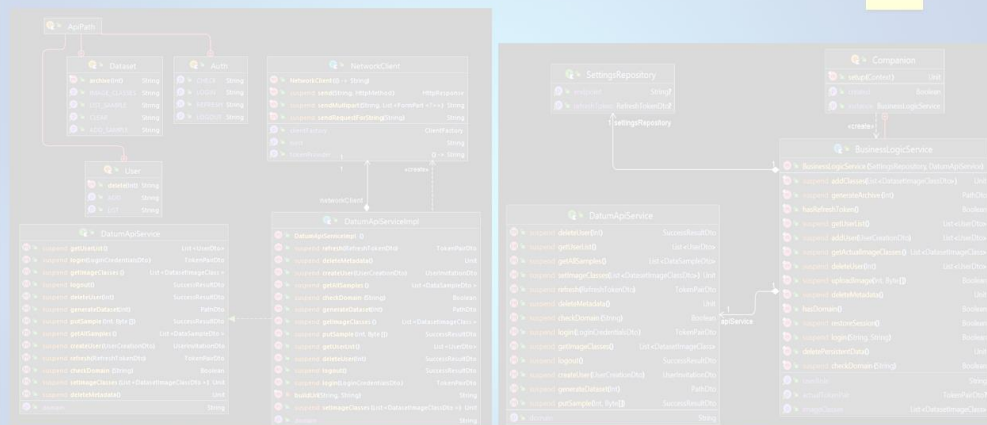
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.42.001 ПЗ

Лист

97

Архитектура фронтенда



Спасибо за
внимание!

MARKO2042001@MAIL.RU

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.42.001 ПЗ

Лист

98