МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Кемеровский государственный университет»

Институт цифры

Кафедра цифровых технологий

**ОТЧЕТ**

по производственной практике.

Научно-исследовательская работа.

студента 4 курса

Кузьмина Кирилла Алексеевича

направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

направленность (профиль) подготовки «Информационные системы и базы данных»

Руководитель практики:

канд. физ.-мат. наук, доцент,

доцент кафедры цифровых технологий, Фомина Л.Н.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Работа защищена:

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_

КЕМЕРОВО 2023

Оглавление

[1. Введение 3](#_Toc122719855)

[1.1. Значимость 3](#_Toc122719856)

[1.2. Цель 3](#_Toc122719857)

[1.3. Задачи 4](#_Toc122719858)

[1.4. План разработки на первый семестр 4](#_Toc122719859)

[2. Основная часть 4](#_Toc122719860)

[2.1. Методы решения 4](#_Toc122719861)

[2.2. Описание запросов 5](#_Toc122719862)

[2.2.1. Users 5](#_Toc122719863)

[2.2.2. Tokens 6](#_Toc122719864)

[2.2.3. Covid 7](#_Toc122719865)

[2.2.4. Cabs 8](#_Toc122719866)

[2.2.5. Machine-models 9](#_Toc122719867)

[2.3. Блоки 10](#_Toc122719868)

[2.3.1. Структурное деление 10](#_Toc122719869)

[2.3.2. Функциональное деление 11](#_Toc122719870)

[2.4. Описание классов 11](#_Toc122719871)

[2.4.1. Блок Controller 11](#_Toc122719872)

[2.4.2. Блок Model 13](#_Toc122719873)

[2.4.3. Блок Database 13](#_Toc122719874)

[2.5. Описание методов 13](#_Toc122719875)

[2.5.1. Блок Controller 13](#_Toc122719876)

[2.5.2. Блок Model 14](#_Toc122719877)

[2.5.3. Блок Database 15](#_Toc122719878)

[2.6. Требования 15](#_Toc122719879)

[2.6.1. Требования к системе 15](#_Toc122719880)

[2.6.2. Требования к инструментам разработки 15](#_Toc122719881)

[2.6.3. Требования к техническому обеспечению 16](#_Toc122719882)

[3. Заключение 16](#_Toc122719883)

[4. Список литературы 16](#_Toc122719884)

# Введение

## Значимость

Реализация проекта позволит предсказывать риск смерти при тяжёлом течении COVID-19. Обязанность серверной части системы:

1. принимать запросы клиента;
2. обрабатывать сложные вычислительные алгоритмы;
3. отправлять запросы на Базу Данных;
4. формировать данные и отдавать результаты согласно HTTP протоколу.

## Цель

Цель – разработать серверную часть веб-приложения, поддерживающую необходимый автоматизированный комплекс решения задач, которые, по запросам клиента, сформируют ожидаемый результат.

## Задачи

Задачи серверного приложения:

1. Отвечать на запросы клиента и выдавать ожидаемый результат;
2. Автоматизация задач;
3. Формирование данных;
4. Предоставление точек взаимодействия для клиента;
5. Взаимодействие с Базой Данных;

## План разработки на первый семестр

План разработки проекта на первый семестр:

1. Описание API для взаимодействия сервера с веб-клиентом;
2. Составление ТЗ;
3. Описание архитектуры серверного приложения;
4. Создание и тестирование Базы Данных;
5. Описание API для взаимодействия сервера с микросервисом машинного обучения;
6. Создание прототипа сервера;
7. Тестирование функций сервера;
8. Интеграция микросервиса с машинным обучением.

# Основная часть

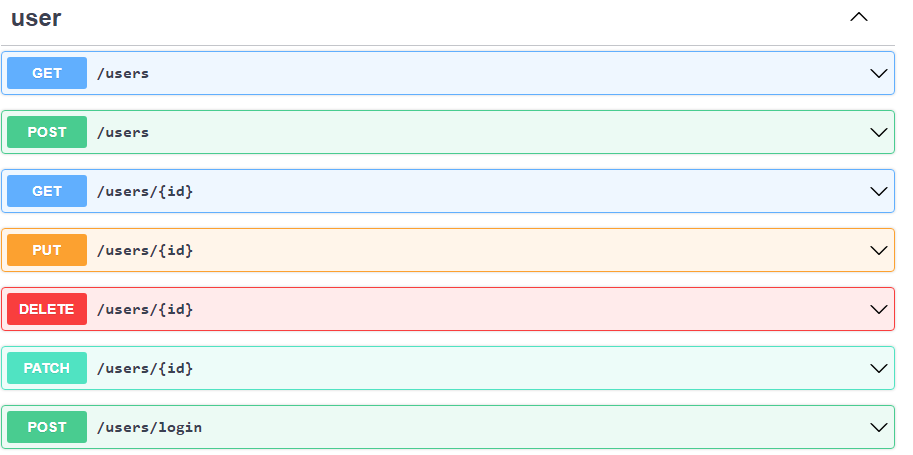
## Методы решения

Целью каждого запроса является ответ клиенту. Этот запрос с задачами, которые необходимо решить и выдать нужный ответ клиенту. Для каждой задачи составлен свой метод решения:

1. Составление множество условий для выдачи нужного ответа клиенту;
2. Разбить серверную часть на блоки согласно паттерну MVC;
3. Согласованность с разработчиком клиента для формирования данных;
4. Список запросов, стандартизация в именах API;
5. Подключение к БД, наличие практической и тестируемой Баз Данных.

## Описание запросов

### Users



*Рисунок 1 - список запросов /users*

Запрос связан с обработкой данных пользователей веб-приложения. Всего имеется 7 запросов по пути **/users**:

1. /users  
   Запрос возвращает список моделей cabs ML.

* Метод запроса: **GET;**
* Возвращаемое значение: **JSON(id, registrationDate, fullName, login, password, role, status, deletionDate);**
* Коды состояния: **200, 401, 403.**

1. /users

Создаёт нового пользователя

* Метод запроса: **POST;**
* Возвращаемое значение: **JSON(id, registrationDate, fullName, login, password, role, status, deletionDate);**
* Параметры: **JSON(accessToken)**
* Коды состояния: **200, 401, 403.**

1. /users/{id}

Возвращает одного пользователя по id

* Метод запроса: **GET;**
* Возвращаемое значение: **JSON(id, registrationDate, fullName, login, password, role, status, deletionDate);**
* Параметры: **JSON(accessToken);**
* Коды состояния: **200, 401, 403.**

1. /users/{id}

Восстанавливает пользователя по id

* Метод запроса: **PUT;**
* Возвращаемое значение: **“user reestablished successfully”;**
* Параметры: **JSON(accessToken);**
* Коды состояния: **204, 401, 403.**

1. /users/{id}

Удаляет пользователя по id

* Метод запроса: **DELETE;**
* Возвращаемое значение: **“user deleted successfully”;**
* Параметры: **JSON(accessToken);**
* Коды состояния: **204, 401, 403.**

1. /users/{id}

Обновляет данные пользователя по id

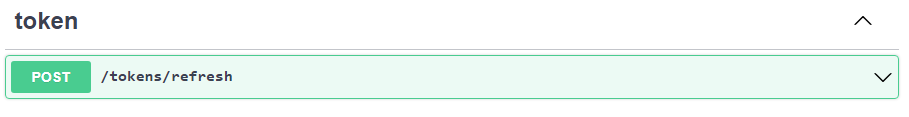
* Метод запроса: **PATCH;**
* Возвращаемое значение: **JSON(id, registrationDate, fullName, login, password, role, status, deletionDate);**
* Параметры: **JSON(accessToken);**
* Коды состояния: **204, 401, 403.**

1. /users/login

Возвращает токен пользователя

* Метод запроса: **POST;**
* Возвращаемое значение: **JSON(login, password);**
* Параметры: **JSON(accessToken)**
* Коды состояния: **200, 401.**

### Tokens



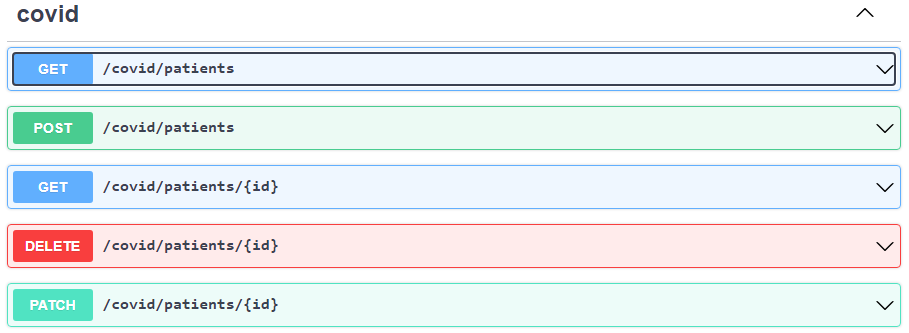
**Рисунок 2 - список запросов /tokens**

1. /tokens/refresh

Возвращает обновлённый токен

* Метод запроса: **POST;**
* Возвращаемое значение: **JSON(accessToken, refreshToken);**
* Параметры: **JSON(refreshToken)**
* Коды состояния: **200, 401.**

### Covid



**Рисунок 3 - список запросов /covid**

1. /covid/patients

Возвращает список пациентов с covid

* Метод запроса: **GET;**
* Возвращаемое значение: **JSON(patient);**
* Параметры: **JSON(accessToken);**
* Коды состояния: **200, 401, 403.**

1. /covid/patients

Добавляет одного пациента с covid

* Метод запроса: **POST;**
* Возвращаемое значение: **“covid patient added successfully”;**
* Параметры: **JSON(accessToken), JSON(patientData);**
* Коды состояния: **200, 401, 403.**

1. /covid/patients/{id}

Возвращает пациента с covid по id

* Метод запроса: **GET;**
* Возвращаемое значение: **JSON(patientData);**
* Параметры: **JSON(accessToken);**
* Коды состояния: **200, 401, 403.**

1. /covid/patients/{id}

Удаляет данные пациента с covid по id

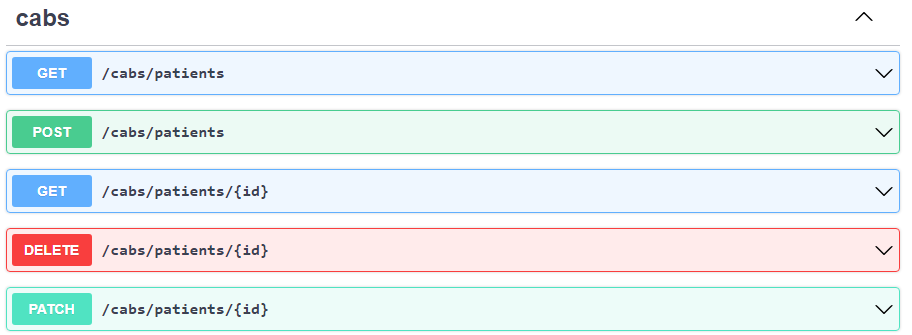
* Метод запроса: **DELETE;**
* Возвращаемое значение: **“covid patient deleted successfully”;**
* Параметры: **JSON(accessToken);**
* Коды состояния: **204, 401, 403.**

1. /covid/patients/{id}

Обновляет данные пациента с covid по id

* Метод запроса: **PATCH;**
* Возвращаемое значение: **“covid patient updated successfully”;**
* Параметры: **JSON(accessToken), JSON(patientData);**
* Коды состояния: **204, 401, 403.**

### Cabs



**Рисунок 4 - список запросов /cabs**

1. /cabs/patients

Возвращает список пациентов, перенёсших операцию коронарного шунтирования

* Метод запроса: **GET;**
* Возвращаемое значение: **JSON(patient);**
* Параметры: **JSON(accessToken);**
* Коды состояния: **200, 401, 403.**

1. /cabs/patients

Создаёт пользователя, перенёсшего коронарное шунтирование

* Метод запроса: **POST;**
* Возвращаемое значение: **“cabs patient added successfully”;**
* Параметры: **JSON(accessToken), JSON(patientData);**
* Коды состояния: **201, 401, 403.**

1. /cabs/patients/{id}

Возвращает пациента, перенёсшего коронарное шунтирование, по id

* Метод запроса: **GET;**
* Возвращаемое значение: **JSON;**
* Параметры: **JSON(accessToken);**
* Коды состояния: **200, 401, 403.**

1. /cabs/patients/{id}

Удаляет данные пациента, перенёсшего коронарное шунтирование, по id

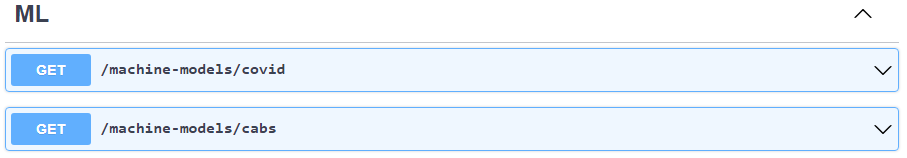
* Метод запроса: **DELETE;**
* Возвращаемое значение: **“cabs patient deleted successfully”;**
* Параметры: **JSON(accessToken);**
* Коды состояния: **204, 401, 403.**

1. /cabs/patients/{id}

Обновляет данные пациента, перенёсшего коронарное шунтирование, по id

* Метод запроса: **PATCH;**
* Возвращаемое значение: **“covid patient updated successfully”;**
* Параметры: **JSON(accessToken), JSON(patientData);**
* Коды состояния: **204, 401, 403.**

### Machine-models



**Рисунок 5 - список запросов /machine-models**

1. /machine-models/covid

Возвращает список моделей машинного обучения по covid

* Метод запроса: **GET;**
* Возвращаемое значение: **JSON;**
* Параметры: **JSON(accessToken);**
* Коды состояния: **200, 401, 403.**

1. /machine-models/cabs

Возвращает список моделей машинного обучения по коронарному шунтированию

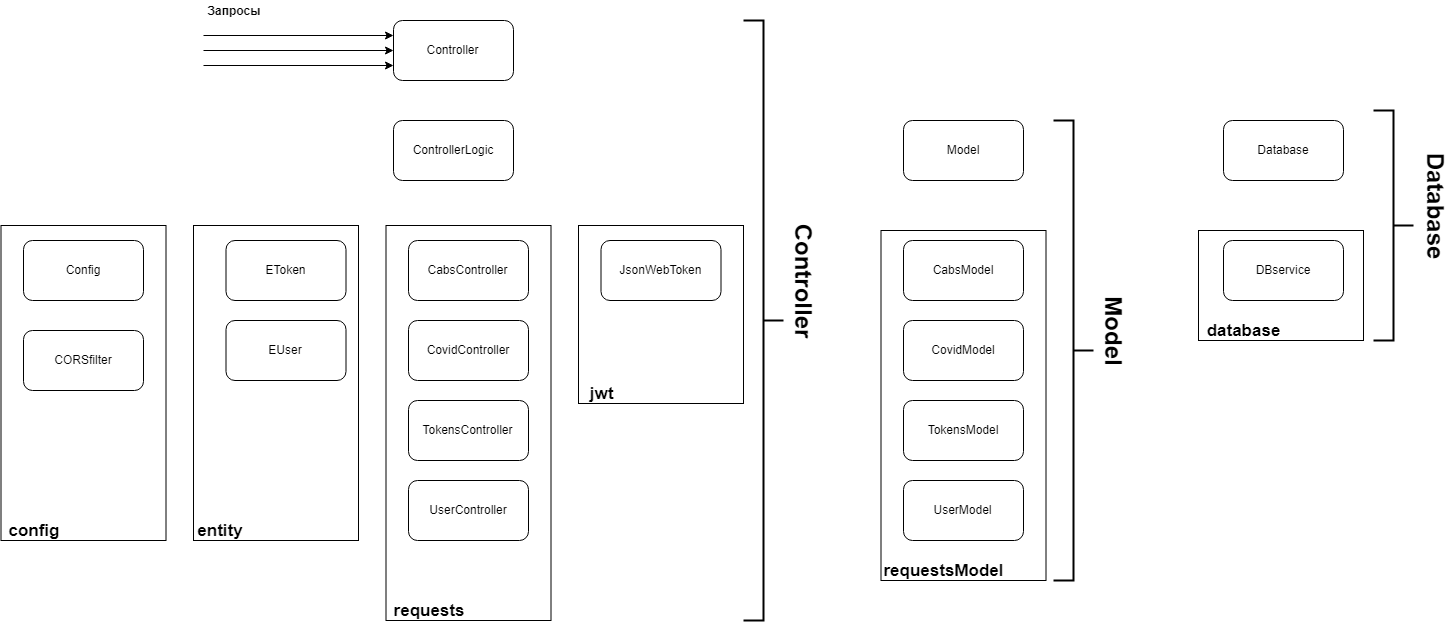
* Метод запроса: **GET;**
* Возвращаемое значение: **JSON;**
* Параметры: **JSON(accessToken);**
* Коды состояния: **200, 401, 403.**

## Блоки

Графическое описание серверной части приложения в виде структурного и функционального делений:

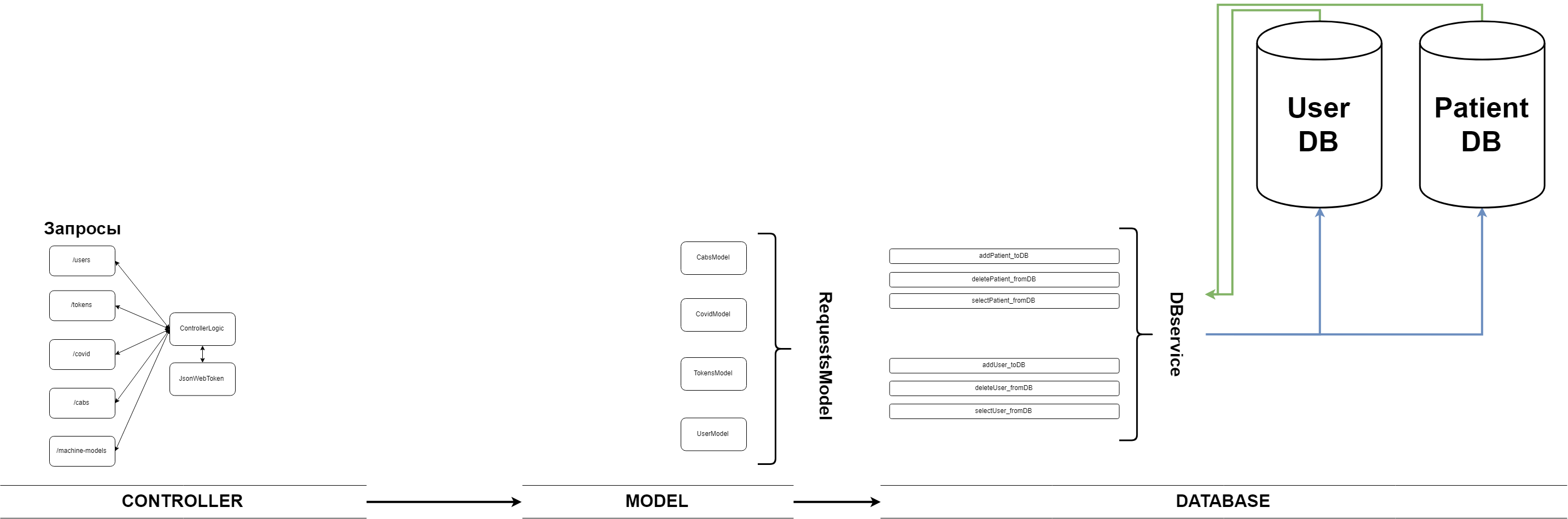
* Структурный блок описывает наличие классов;
* Функциональный блок описывает взаимодействие логических частей.

### Структурное деление



**Рисунок 1 – схема структурного деления**

### Функциональное деление



**Рисунок 1 – схема функционального деления**

## Описание классов

Архитектура серверного приложения адаптируется под шаблон MVC. Все классы относятся к одному из трёх блоков:

1. **Controller –** блок маршрутизации. *Обязанности*:

* Обработка запросов;
* Проверка полученных данных;
* Передача данных блоку **Model.**

1. **Model –** вычислительный блок. *Обязанности*:

* Получение данных от **Controller;**
* Выполнение необходимых вычислительных операций;
* Передача обработанных данных в **Database.**

1. **Database** – блок Базы Данных. *Обязанности*:

* Получение обработанных данных от **Model;**
* Запросы к БД;
* Передача результата **Controller.**

В каждом блоке есть класс, который относится к одному из трёх блоков схемы MVC.

### Блок Controller

В блоке **Controller** имеются следующие пакеты с классами:

1. Controller:

* **Controller.**Класс **Controller** является маршрутизатором между блоками ControllerиModel**;**
* **ControllerLogic.**Класс ControllerLogic выполняет маленькие вычисления с помощью сторонних библиотек.

1. Controller.Config:

* **Config.**Класс **Config** определяет путь приложения.
* **CORSfilter.**Класс **CORSfilter** является настройкой для доступа к запросам сервера с разных доменов.

1. Controller.Entity

* **EToken.**Класс **EToken** определяет сущность **token** для работы с JSON-объектами.
* **EUser.**Класс **EUser** определяет сущность **user** для работы с JSON-объектами.

1. Controller.JWT

* **JsonWebToken.**Класс **JsonWebToken** настроен на работу с JWT(json-web-token).

1. Controller.Requests

* **CabsController.**Класс **CabsController** принимает запросы клиента и возвращает нужное значение. Обрабатывает запросы по коронарного шунтирования.
* **CovidController.**Класс **CovidController** принимает запросы клиента и возвращает нужное значение. Обрабатывает запросы по covid.
* **TokensController.**Класс **TokensController** принимает запросы клиента и возвращает нужное значение. Обрабатывает запросы по токенам.
* **UserController.**Класс **UserController** принимает запросы клиента и возвращает нужное значение. Обрабатывает запросы по пользователям.

### Блок Model

В блоке **Model** имеются следующие пакеты с классами:

1. Model.RequestsModel

* **CabsModel.**Класс **CabsModel** выполняет сложные вычислительные операции по коронарному шунтированию.
* **CovidModel.**Класс **CovidModel** выполняет сложные вычислительные операции по ковиду.
* **TokensModel.**Класс **TokensModel** выполняет сложные вычислительные операции по работе с токенами.
* **UserModel.**Класс **UserModel** выполняет сложные вычислительные операции по работе с пользователями.

### Блок Database

В блоке **Database** имеются следующие пакеты с классами:

1. Database

* **DBservice.**Класс **DBservice** выполняет запросы к Базе Данных, проверяет данные.

## Описание методов

### Блок Controller

#### Класс Controller

* **userCheckData**Метод принимает данные от класса **UserController**, парсит JSON-объект и передаёт в класс **UserModel.** Метод проверяет данные пользователя.

#### Класс ControllerLogic

* **userGetToken**Метод принимает данные от класса **UserController**, парсит JSON-объект и передаёт в класс **JsonWebToken.** Метод возвращает accessToken и refreshToken;
* **userCheckToken**Метод принимает данные от класса **UserController**, отдаёт данные классу **JsonWebToken.** Метод проверяет токен.

#### Класс JsonWebToken

* **generateToken**Метод генерирует новый токен;
* **checkToken**Метод проверяет токен.

#### Класс CabsController

В классе **CabsController** находятся методы, обрабатывающие запросы клиента по работе с коронарным шунтированием.

#### Класс CovidController

В классе **CovidController** находятся методы, обрабатывающие запросы клиента по работе с covid.

#### Класс TokensController

В классе **TokensController** находятся методы, обрабатывающие запросы клиента по работе с токенами.

#### Класс UserController

В классе **UserController** находятся методы, обрабатывающие запросы клиента по работе с пользователями.

### Блок Model

#### Класс CabsModel

Методы класса **CabsController** выполняют вычисления по запросам с работой коронарного шунтирования.

#### Класс CovidModel

Методы класса **CovidModel** выполняют вычисления по запросам с работой covid.

#### Класс TokensModel

Методы класса **TokensModel** выполняют вычисления по запросам с работой токенов.

#### Класс UserModel

Методы класса **UserModel** выполняют вычисления по запросам с работой с пользователями.

### Блок Database

#### Класс DBservice

* **userLogin**Метод проверяет входные данные с данными из Базы Данных;

## Требования

### Требования к системе

1. Система должна обрабатывать логику входящих запросов, обращаться к БД и возвращать результат;
2. Система должна соответствовать архитектуре MVC;
3. Тестирование запросов на back-end планируется в декабре;
4. Серверное приложение необходимо для обработки больших данных пациентов, пользователей, вычислений алгоритмов обучения моделей.
5. Срок – лето 2023.

### Требования к инструментам разработки

1. Инструменты разработки для написания серверного приложения:

* **Среда разработки** InteliJ IDEA;
* **Сервер приложений** GlassFish;
* **Язык программирования** Java;
* **Текстовый редактор** Visual Studio Code;
* **СУБД** MongoDB/MySQL/SQLdeveloper;

1. На сервере приложений **GlassFish** будет развёртано приложение. **InteliJ IDEA** необходимо настроить под **GlassFish**. Текстовый редактор **VsC** будет служить для локального тестирования API-запросов. БД должна хранить данные пользователей и пациентов, СУБД даст возможность добавлять, изменять, удалять, получать данные;
2. Настройка инструментов и тестирование запросов планируется в декабре;
3. Срок – конец декабря 2022.

### Требования к техническому обеспечению

Система должна быть реализована с использованием выделенных серверов Заказчика.

# Заключение

На первый семестр была поставлена задача создать рабочий прототип серверного приложения, с которым разработчик веб-приложения и разработчик машинного обучения могут работать и получать фиктивные данные. Реализована часть задач:

1. Создание БД;
2. Описание архитектуры;
3. Составление ТЗ;
4. Описание API для взаимодействия сервера с веб-клиентом.

В процессе находятся задачи:

1. Создание прототипа серверного приложения;
2. Описание API для взаимодействия сервера с микросервисом машинного обучения;
3. Тестирование функций сервера.

Задачи, срок которых продлён до января:

1. Описание API для взаимодействия сервера с микросервисом машинного обучения;

# Список литературы

* 1. Документация Jakarta EE [Электронный ресурс] / URL: <https://jakarta.ee/resources/#documentation>
  2. Документация Glassfish [Электронный ресурс] / URL: <https://glassfish.org/docs/>
  3. Документация MySql [Электронный ресурс] / URL: <https://dev.mysql.com/doc/>
  4. Туториал Java EE [Электронный ресурс] / URL: <https://docs.oracle.com/javaee/7/tutorial/>