RUP Artifacts

for

Doctorinna - medical risk factor analyzer

Prepared by Danis Alukaev, Denis Schegletov,

Maxim Pryanikov, Lada Morozova

25 September 2021

Version History

Editor's Name	Date (DD/MM/YYYY)	Reason for Changes/Sections Updated	Version
The project team	25/08/2021	Business goals and objectives Roles and responsibilities Part of features and user stories Software development plan	1.0.0
The project team	31/08/2021	Non-functional requirements Constraints	1.1.0
The project team	05/09/2021	Functional requirements User stories Low-fidelity prototype Glossary	1.2.0
The project team	25/09/2021	Revision of documentation Design Architecture	1.3.0
The project team	10/10/2021	Revision of documentation	1.4.0

Table of Contents

Business Goals and Objectives **1**

Roles and responsibilities **2**

Requirement Analysis and Specifications **3**

- Features 3
- Functional requirements 3
- Non-Functional Requirements 4
- User Stories 5
- Constraints 6

Design Documentation **7**

- Database design. 7
- Backend design. 7

Architecture **9**

- Static Perspective. 9
- Dynamic Perspective. 10
- Allocation Perspective. 11
- Use-Case diagram 12
- Sequence diagram 13

Software Development plan **14**

Glossary **16**

Technology stack **18**

- Backend 18
- Web frontend 18
- Telegram bot 18

Low fidelity prototypes **19**

- Web application 19
- Telegram bot 20

Links **21**

1. Business Goals and Objectives

Our goal is to build a system for people to find out their risk groups for certain diseases based on their answers to the questionnaire. We want the questionnaire results to **reassure users' health risks** and help them draw the conclusions about how to **improve their lifestyle**. Besides, we hope to **raise users' awareness** about diseases and their factors. So that the process of passing the questionnaire does not cause any difficulties, we aim to make the system **user-friendly** and available for anyone. Since this is the first time for us designing such a complex system, we want to take good from it, namely to **gain development experience**.

Thus, the following goals should be distinguished:

1	Raising awareness about diseases.
2	Reassure people of their health risks.
3	Improve the user's lifestyle.
4	User-friendly interface.
5	Gain development experience.

2. Roles and responsibilities

Stakeholder's Name	Roles	Responsibilities
Developer	Front Developer Backend Developer Database Administrator Machine Learning Engineer Telegram bot Developer	Develop the graphical interface for a web application. Develop backend API. Develop Telegram-bot. Build a pipeline for statistical analysis. Train model for health risk assessment. Deploy applications.
QA engineer	Software Tester Test Analyst	Testing of application. Configuration of test automation. Requirement analysis
User	Web application User Telegram bot User	Use an application. Pass the questionnaire. Give feedback.
Product Owner	Project Manager Business Analytic	Rejecting/accepting project Provide business requirements Task management. Risk identification. Project vision.

3. Requirement Analysis and Specifications

3.1. Features

ID #	Feature Title	Priority	Any Other Label
1.	Questionnaire	Must	
2.	Data analysis	Must	
3.	Graphical Design	Must	
4.	Database Design	Must	
5.	Available on multiple platforms	Could	

3.2. Functional requirements

1	The questionnaire should contain questions that collect necessary information for the machine learning model.
2	When a user takes a questionnaire, data should be entered into the database.
3	The system should analyze user data and predict results with the use of a machine learning model.
4	The system should allow only administrators to access user's data.
5	The software system should be integrated with the Django REST API.
6	The system should illustrate results with graphs.
7	The system should provide a possibility to take the survey with the use of Telegram bot.

3.3. Non-Functional Requirements

ID #	Non-Functional Requirement Title	How to meet Non-Functional Requirement	How to measure Non-Functional requirement
1.	Modularity	Implement the backend following microservices architecture, i.e., divided by loosely-coupled services. Implement the frontend divided into interface components and the global state of the system that is controlled by the state management system (Redux).	Number of modules; The size of each module with respect to the project; Number of connections between modules
2.	Reusability	Implement the backend API allowing to build services on top of it, e.g. web frontend, Telegram bot, mobile app, etc.	How easily service can be integrated
3.	User error protection	Checking correctness of user data and preventing from making errors by user interface.	Number of user's clicks before achieving correct result is less than 3; Number of invalid data imputed by user
4.	Confidentiality	The results of the analysis of responses will be saved to the database without binding to the user's identity. The data requested from the client side of the application cannot be used to determine the identity of other users.	No records in database containing any personal data
5.	Installability	The installation manual is clear and does not require deep knowledge	Number of commands required to install

3.4. User Stories

User Type	User Story Title	User stories
Web application / Telegram bot User	Questionnaire	1.1. As a user, I want to know my health risk group for different diseases, so that I will be able to draw a conclusion about my lifestyle. 1.2. As a user, I want to answer only questions that do not require a medical examination, so that I do not have to waste time going to the doctor.
	Graphical Design	2.1. As a user, I want to interact with a user-friendly web interface, so that I can make fewer misclicks. 2.2. As a user, I want my health group to be illustrated using graphs, so that I can visually see statistics about me.
	Statistics	3.1. As a user, I want to see the statistics about the health risks in different regions of Russia, so that I can compare various regions by the number of disease risks. 3.2. As a user, I want to see my result in the distribution of health risks of all users, so that I can compare it with overall disease risk frequency. 3.3. As a user I want my data to be processed by artificial intelligence, so that the result will be statistically justified
	Telegram bot	4.1. As a user, I want to take a survey through a Telegram bot, so that I can share my results with friends.
	User error protection	5.1. As a user, I want the service to guide me when entering data, so that I did everything right on the first try.
	Confidentiality	6.1. As a user, I want the questionnaire to be anonymous, so that my identity will not be revealed.
	Accessibility	7.1. As a user, I want to have access to the service in the web version, so that I do not need to install new applications on my device. 7.2. As a user, I want the service to be free, so that I could find out my risk group without having free funds.

3.5. Constraints

1. Number of servers: 1 per frontend, backend, bot, database.
2. Server constraints: amount of RAM, number of cores, throughput capacity.
3. Number of developers: 4 people.
4. Time of development: 5 weeks.

4. Design Documentation

4.1. Database design.

The data in the Doctorinna application is represented by the UML Entity-Relationship diagram (see Figure 1).

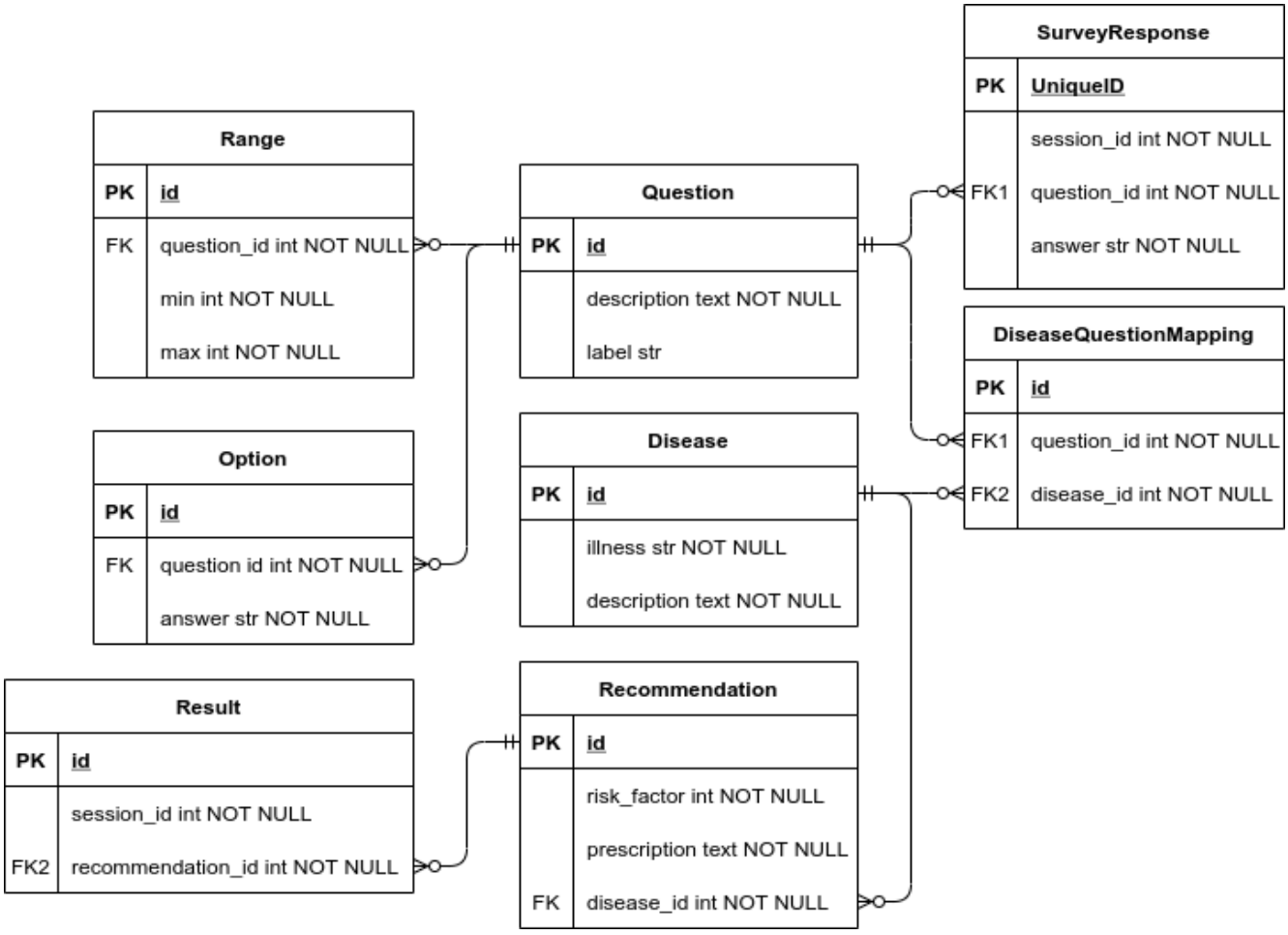


Figure 1. Entity-Relationship diagram

4.2. Backend design.

The backend side of the application is implemented in Django REST Framework. A significant number of design patterns are used in this framework. Therefore, in the course of our work, we indirectly use them.

For instance, the Factory design pattern is used for creation of fields in forms. The form module provides the possibility to create various types of fields and customizations. Apparently, one can add to the model CharField and specify its maximal length and whether it is required when the object is created.

Also Django utilizes the concept of *middleware* that is similar to decorator and does the same job. Specifically, they add new effects either from the top or from the bottom in the chain. Still, they are different in implementation. The interface of middleware hides the next middleware object in the chain in contrast to the decorator. However, Django provides tools to make decorators out of middleware.

Another example of design pattern is Observer. There is a third-party Django package called *django-observer* used to register callback functions executed when fields in models were changed.

The framework is rapidly developing and some design patterns began to occur commonly. These design patterns are called Django Design Patterns (DDP) and not strictly related to classes, but instead include the best practices followed by Django developers.

Any Django REST application includes four essential components: Models, Forms, Views, and Serializers. **Models** are presented in *models.py* and describe entities stored in the database. They allow preserving data in an OOP manner. **Forms** are used to collect information from users. **Views** are controllers treating user requests and connecting business logic with the database. **Serializers** used to translate objects for sending to clients.

One DDP actively used in our project is *CRUD Views*. The idea is to logically group all actions with a set of objects in one method. Thus, we can create only one route for each model to perform GET, POST, UPDATE, DELETE requests.

Another DDP used is handling reverse relationships in the serializers. For example, consider the Question and Answer models: Answer model has auxiliary foreign key to Question as the question can have several possible options. However, when the client side requests for a list of questions, we want to send correspondent options or possible ranges. To do so, in the Answer model we defined a related name in a foreign key field, and configured Question serializer to include objects with the related name to serialization.

5. Architecture

5.1. Static Perspective

Static view (see Figure 2) presents the main modules of the system and their hierarchy in an abstract way. It allows to clearly grasp what the system is composed of in overall, helps to outline the blueprint of the project's code, and distribute the responsibilities among the team members.

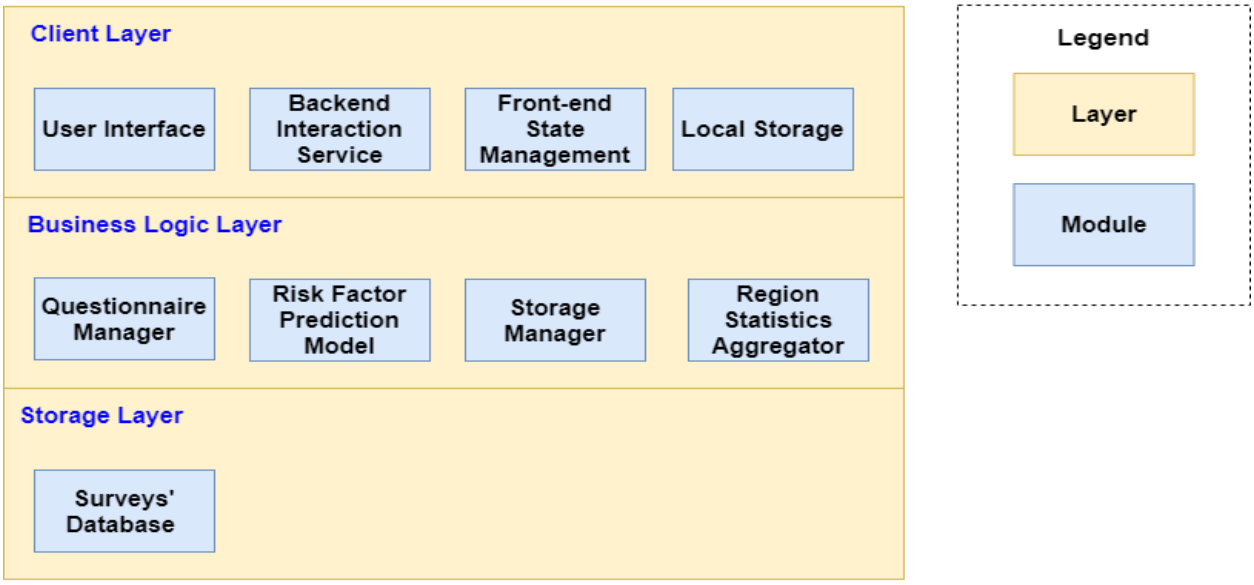


Figure 2. Static architectural view

5.2. Dynamic Perspective

Dynamic view (see Figure 3) extends the static view by adding the interactions between different components. It helps to understand how the system should behave, points out control and data flows in it.

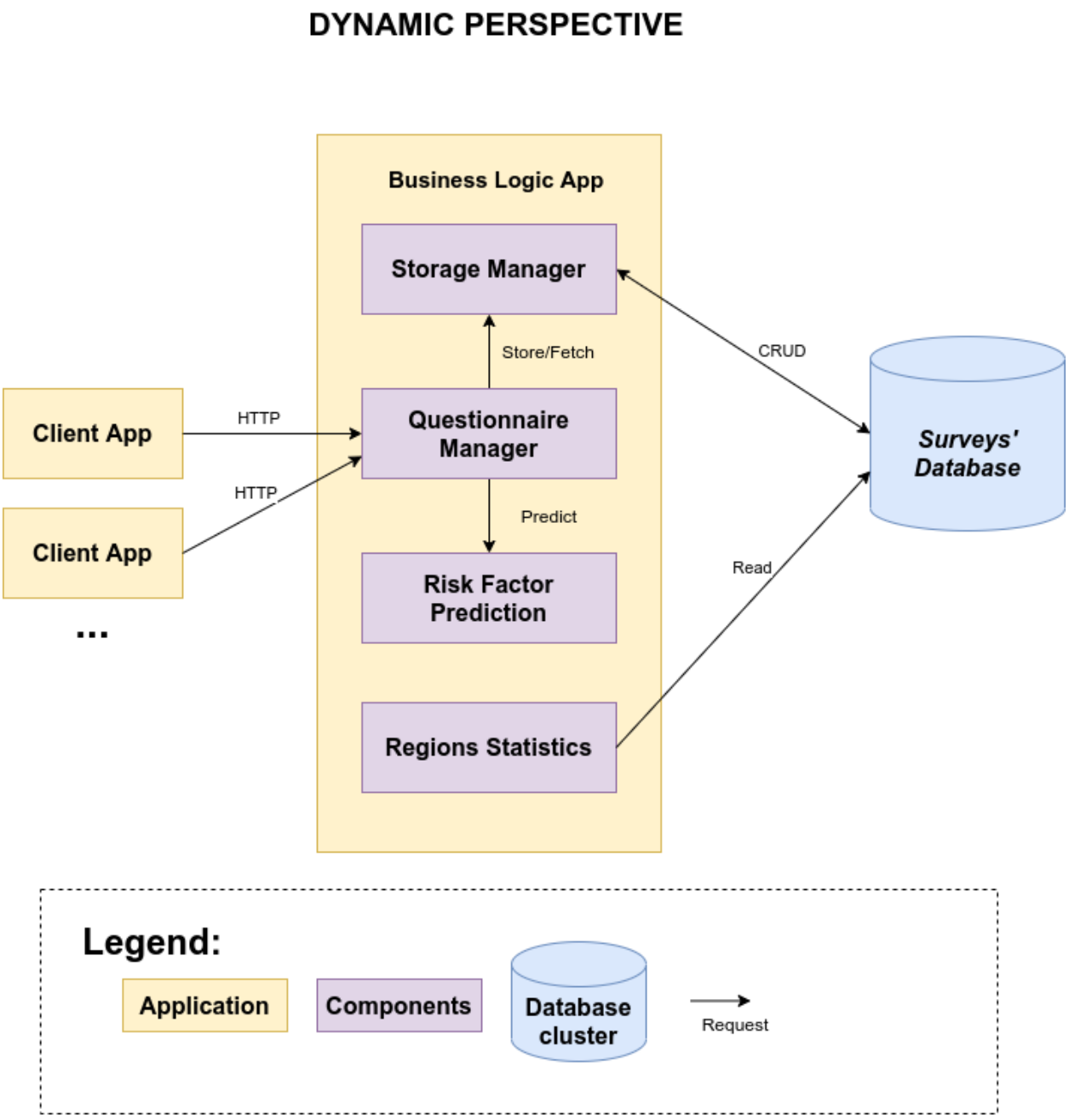


Figure 3. Dynamic architectural view

5.3. Allocation Perspective

Allocation view (see Figure 4) shows more specifics of the system, providing hardware resources and more detailed data flow.

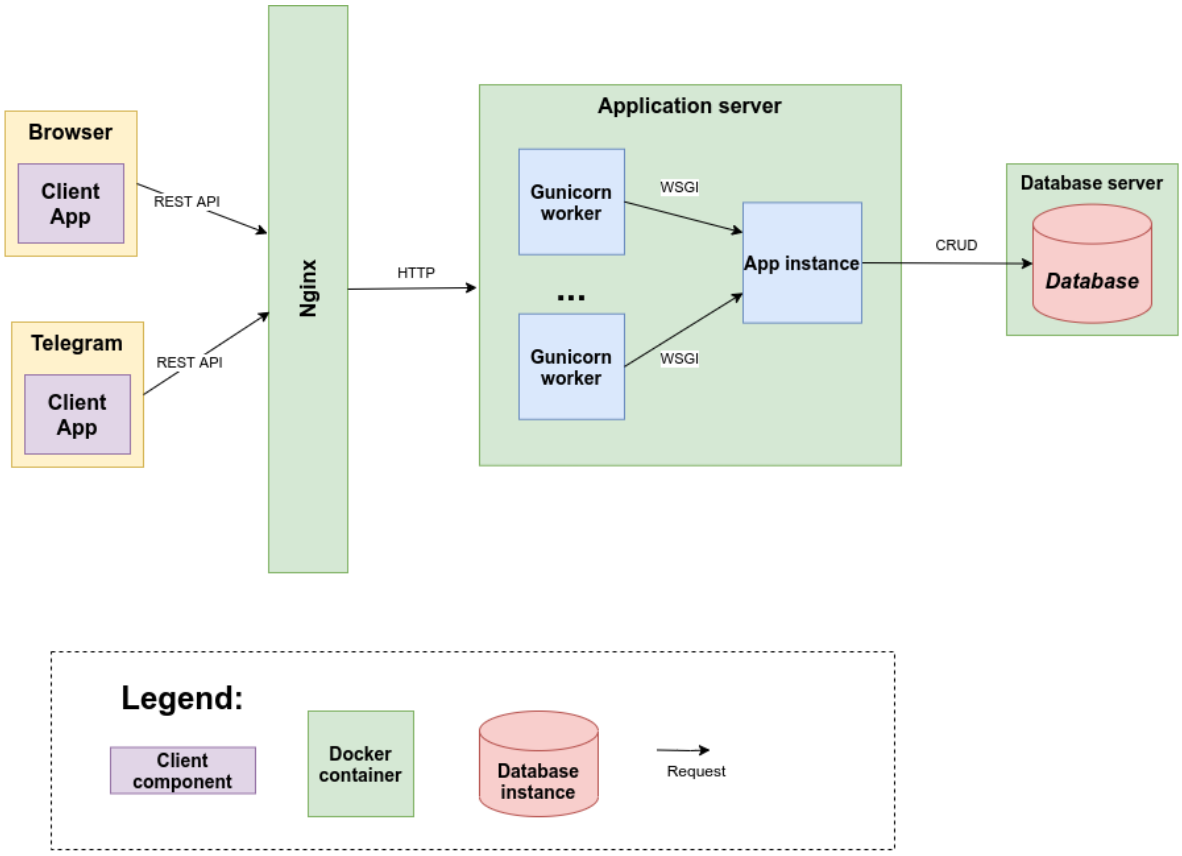


Figure 4. Allocation architectural view

5.4. Use-Case diagram

Use-case diagram (see Figure 5) shows the interaction between the user and the system, by specifying certain user actions towards it. The diagram allows to define what the system should provide to the users, and therefore is a useful tool to assess the system on how it follows the requirements.

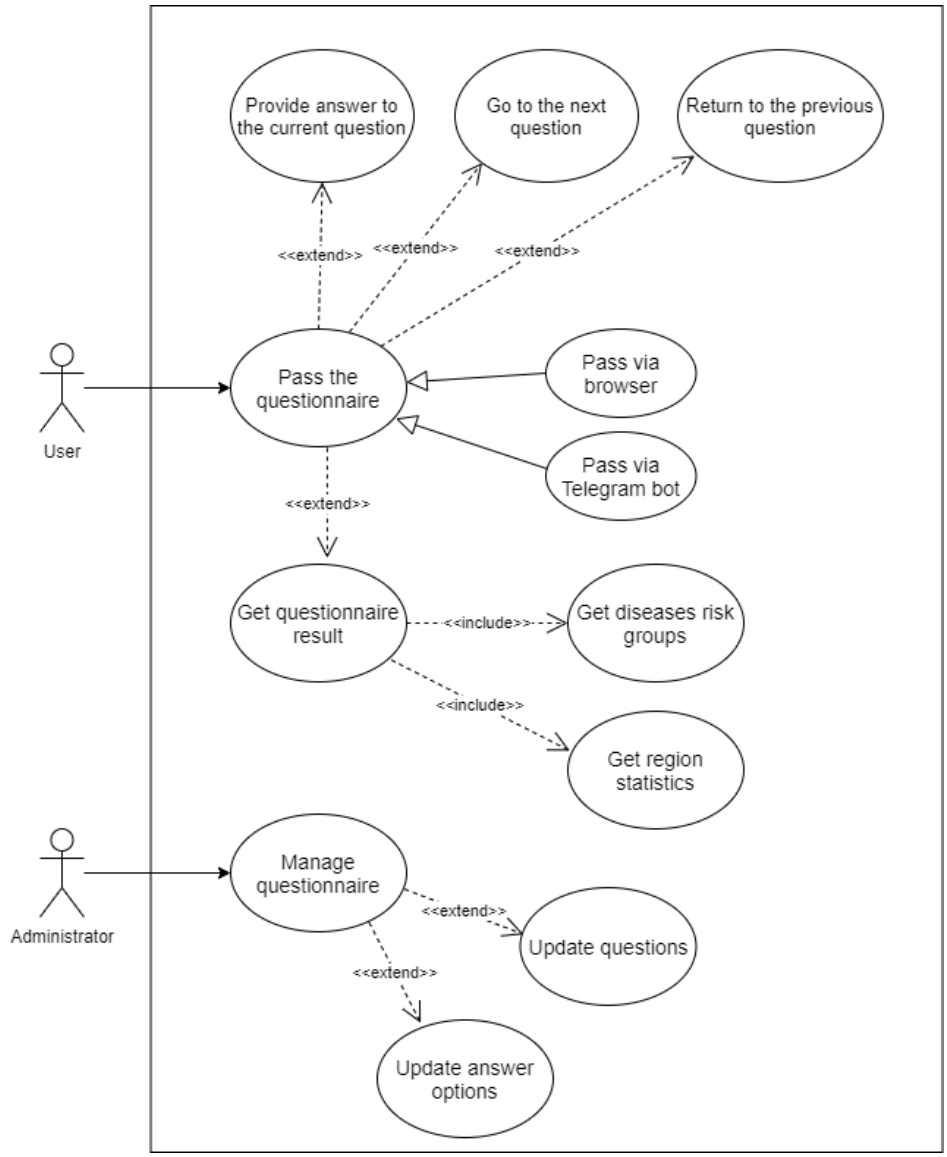


Figure 5. Use-case diagram

5.5. Sequence diagram

Sequence diagram (see Figure 6) presents the sequence of events occurring in the system, interactions between entities and their lifespans. It shows how the entities group together to perform particular functions. This diagram allows to document the processes in the system.

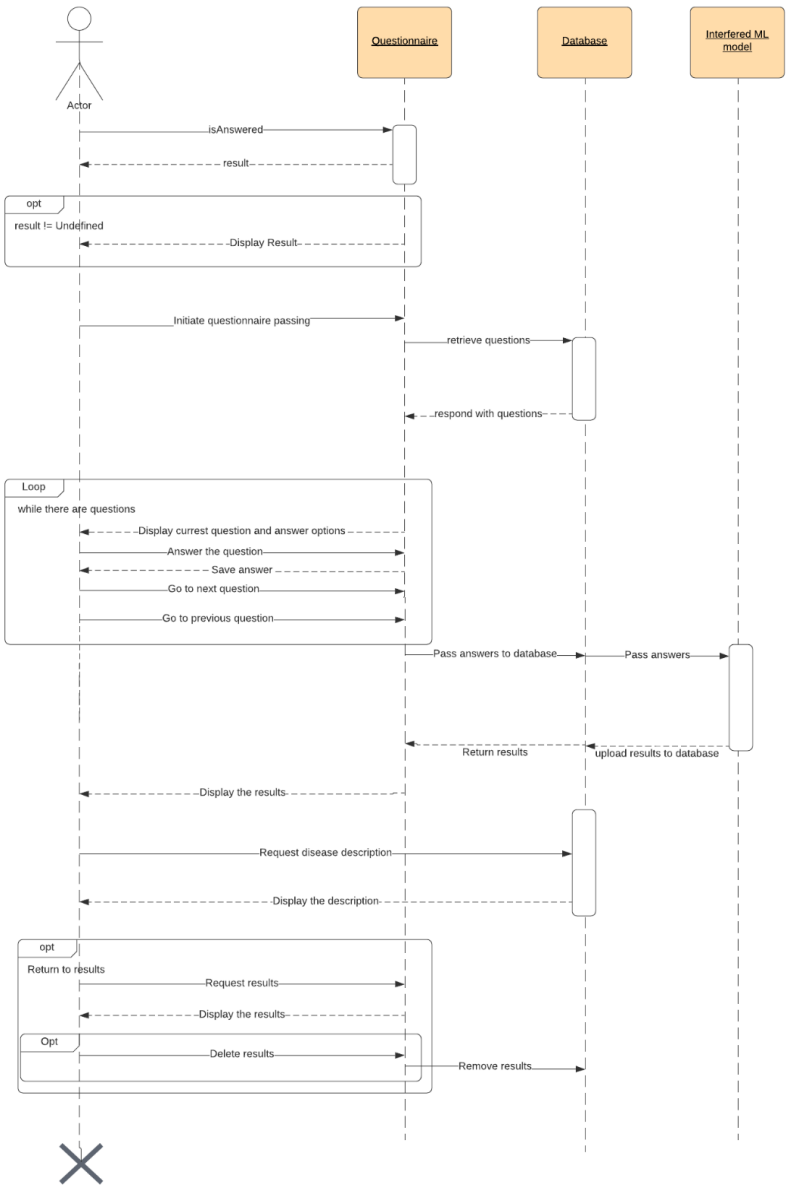


Figure 6. Sequence diagram

6. Software Development plan

Inception Phase				
#Iteration	**Timeline**	**Stakeholders**	**Activities**	**Artifacts**
#1	25/08/2021 - 29/08/2021	Product Owners	Determine Business goals and objectives with valid justification Identify the stakeholders Establish roles and responsibilities	Deliver the documentation of achieved milestones
#2	30/08/2021 - 06/09/2021	Developers, Product Owners	Requirement engineering (20% user stories) Identify Risks	Update the documentation of achieved milestones with User stories and Risk Lists

Elaboration Phase				
#Iteration	**Timeline**	**Stakeholders**	**Activities**	**Artifacts**
#1	06/09/2021 - 11/09/2021	Developers, Product Owners	Revise User Stories (100%)	Document 100% user stories
#2	12/09/2021 - 16/09/2021	Product Owners, Developers	Software development planning	Iteration Plan
#3	17/09/2021 - 20/09/2021	Product Owners, Developers	Software Architecture Test Plan	Software architecture document Test Plan Document

Construction Phase				
#Iteration	**Timeline**	**Stakeholders**	**Activities**	**Artifacts**
#1	21/09/2021 - 28/09/2021	Developer	Implement Feature 1: determine the target list of diseases, prepare set of questions Implement Feature 2: choose machine learning model, discover data for training, train the model	Working feature 1, 2 branches Unit test results
#2	29/09/2021 - 04/10/2021	Developer	Implement Feature 3: develop graphical interface Implement Feature 4: develop REST API,	Working feature 2 branch Unit test results

https://github.com/Doctorinna/overview

			design database Implement Feature 5: develop a telegram bot Unit test cases for feature 3, 4, 5	

Transition Phase				
#Iteration	**Timeline**	**Stakeholders**	**Activities**	**Artifacts**
#1	04/10/2021 - 06/10/2021	Developer, Product Owner, User	Integration, End to end testing Training for Users and Developers	Github repository Merged branches Integration and ended to end test results Final README for developers and Users
#2	06/10/2021	Product Owners, Developers, Users	Final product release	Working Product

7. Glossary

Word	Description
Cardiovascular Disease	Disease is a general term for conditions affecting the heart or blood vessels.
Diabetes	Disease which is caused by a high level of blood glucose.
Stroke	Stroke occurs when the blood supply to part of your brain is interrupted or reduced, preventing brain tissue from getting oxygen and nutrients.
Lung cancer	Type of cancer that begins in the lungs.
REST	Architecture that are typically loosely based on HTTP methods to access resources via URL-encoded parameters and the use of JSON or XML to transmit data.
Microservice	Architecture that arranges an application as a collection of loosely-coupled services. Opposed to monolithic architecture.
CRUD	Create, Read, Update, Delete - 4 basic operations of persistent storage.
Serializer / DTO	Data structure in the backend that can be converted to JSON and back, and then sent to the frontend.
Entity / Model (Backend)	Data structure used in the backend that is mapped to tables in SQL databases.
Repo	Github repository of organization Doctorinna.
Component (Frontend)	Group of HTML elements, that can be reused multiple times and structuralized code.
Logger (Backend)	Log the information about all activities to retrieve bugs more easily.
Risk group	Medical risk group of user. There are 4 types of risk groups: no risk, low risk, medium risk, high risk.
Component (UML)	Modular part of the system that encapsulates the state and behavior of a number classifiers.
State manager	Object that controls state in all applications and sends this state to

(Frontend)	components.
Service (Backend)	Part of the Backend that contains business logic.
Controller / urls (Backend)	Part of the Backend that contains sending and receiving information using HTTP methods.
Local Storage	Persistent storage in browsers that can store key-value pairs.
RUP	Rational Unified Process.
SSR	Server Side Rendering - technique to render web page in server and deliver to client already rendered(but it can be rerendered later on client side).
Questionnaire	Set of questions that are answered by users, processed by the system and results are displayed on the user interface.
Bot	Telegram bot.
User	Web Users of the web application or Telegram bot.
ER diagram	Entity-relationship diagram

8. Technology stack

Backend

1. Django REST framework
2. Python
3. Docker
4. PostgreSQL
5. Nginx
6. Gunicorn
7. RabbitMq
8. Celery
9. Scikit-learn

Web frontend

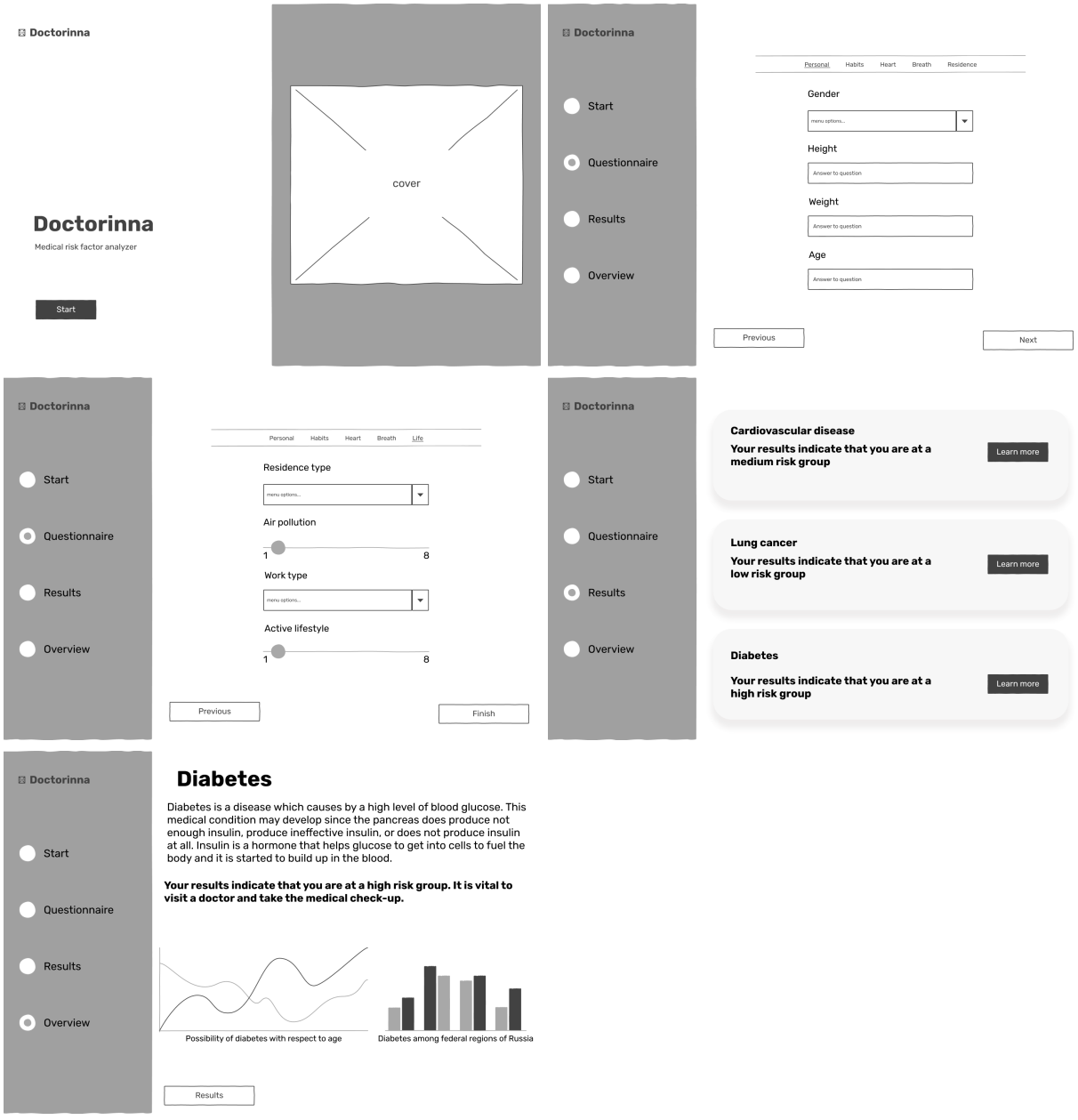
1. React
2. Next.js
3. Redux
4. Typescript
5. Npm
6. HTML
7. SCSS
8. Jest + Enzyme
9. Material-ui

Telegram bot

1. aiogram framework
2. Python
3. Docker
4. Redis

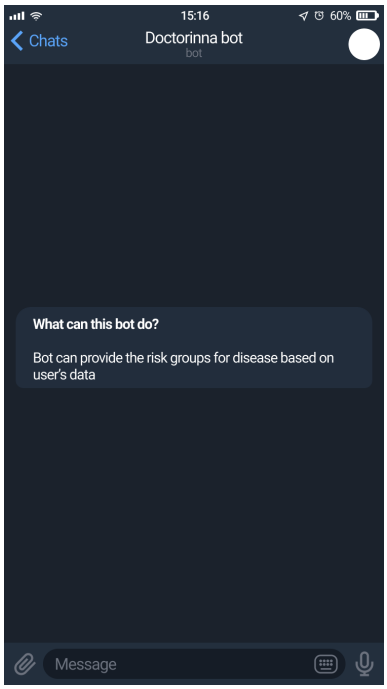
9. Low fidelity prototypes

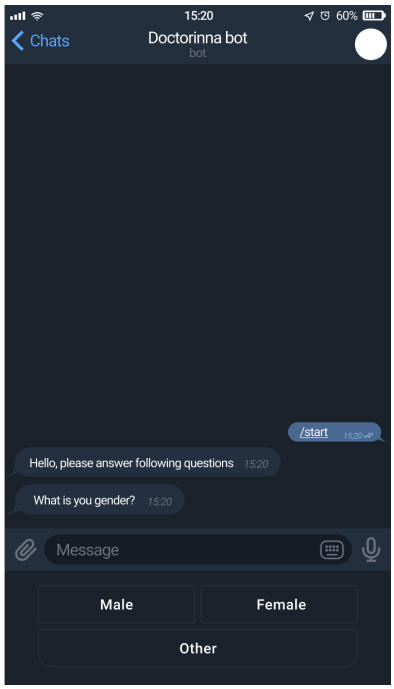
9.1. Web application

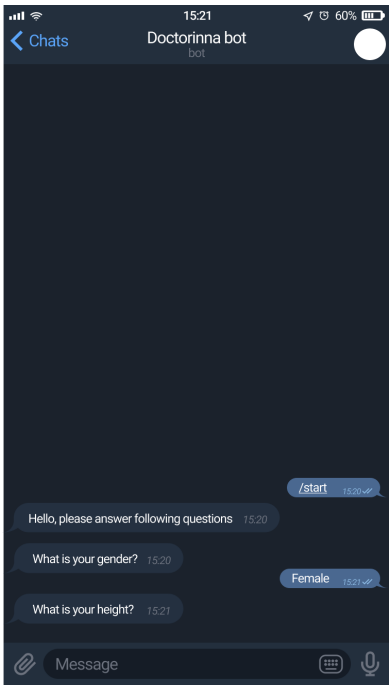


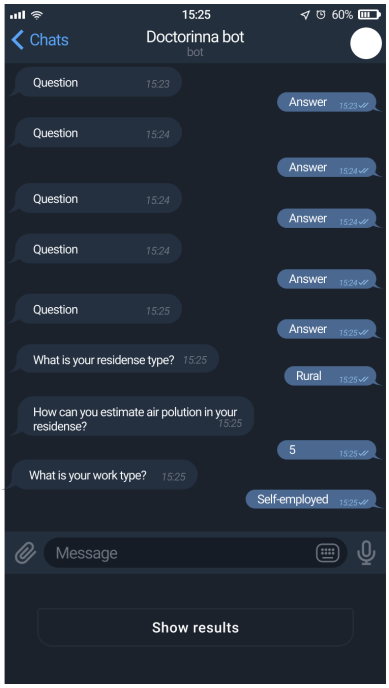
https://github.com/Doctorinna/overview

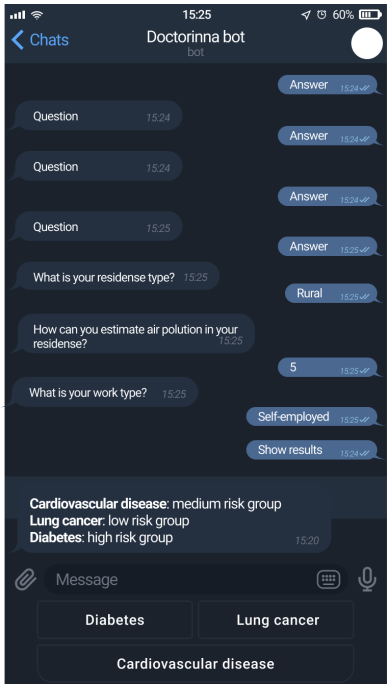
9.2. Telegram bot











https://github.com/Doctorinna/overview

10. Links

The deliveries of the software process for Doctorinna will be stored in the overview repository. Specifically, this is the place for the main README for the project describing the future solution, mentioning collaborators and problem solved.

The code is distributed into 3 repositories:

1. Backend - built on Django REST Framework.
2. Web Frontend - built on Next.js.
3. Telegram bot - utilizes aiogram.