UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 1862

Early Screening for Preeclampsia

Marina Rupe

Zagreb, June 2019

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
MASTER THESIS COMMITTEE

Zagreb, 7 March 2019

MASTER THESIS ASSIGNMENT No. 1862

Student: **Marina Rupe (0036483027)**
Study: Computing
Profile: Software Engineering and Information Systems

Title: **Early Screening for Preeclampsia**

Description:

Preeclampsia, which affects 3 -5 % of pregnancies, is a major cause of maternal and perinatal morbidity and mortality. The prevalence of preeclampsia could be significantly reduced by early identification of high-risk groups and pharmacological intervention. Risk models are based on maternal demographic characteristics and medical history (maternal factors) and biomarkers.
Develop a web application for an early identification of risk for preeclampsia with the following functionalities: user authentication and authorization, entering and editing patient data, calculation of preeclampsia risk, PDF report generation and management of generated reports, visualization of statistics related to biochemistry measurement. Application should enable easy exchange of models for risk calculation.
Develop application in Javascript or Python, thoroughly test and comment code.

Issue date: 15 March 2019
Submission date: 28 June 2019

Mentor:
Full Professor Mile Šikić, PhD

Committee Chair:
Associate Professor Igor Mekterović, PhD

Committee Secretary:
Associate Professor Boris Milašinović, PhD

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 7. ožujka 2019.

DIPLOMSKI ZADATAK br. 1862

Pristupnik: **Marina Rupe (0036483027)**
Studij: Računarstvo
Profil: Programsko inženjerstvo i informacijski sustavi

Zadatak: **Rani probir za otkrivanje preeklampsije**

Opis zadatka:

Preeklampsija, koja se pojavljuje u 3-5 % trudnoća, je glavni uzrok oboljenja te smrti majke i djeteta prije poroda. Učestalost preeklampsije može se značajno reducirati ranim utvrđivanjem rizičnih grupa i davanjem lijekova. Modeli rizika su temeljeni na majčinim demografskim podacima i povijesti bolesti te biomarkerima.

Potrebno je razviti web aplikaciju za ranu identifikaciju rizika od preeklampsije. Aplikacija treba imati slijedeće funkcionalnosti: autentikacija i autorizacija korisnika, unošenje i uređivanje podataka vezanih za pacijenta, računanje rizika za preeklampsiju, generiranje PDF izvještaja i upravljanje izvještajima, vizualizacija statistike vezane uz biokemijska mjerenja. Aplikacija treba omogućiti jednostavnu izmjenu modela za računanje rizika.

Aplikaciju razviti u jeziku Javascript ili Python, iscrpno testirati i komentirati kod.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 28. lipnja 2019.

Mentor:

Prof. dr. sc. Mile Šikić

Djelovođa:

Izv. prof. dr. sc. Boris Milašinović

Predsjednik odbora za diplomski rad profila:

Izv. prof. dr. sc. Igor Mekterović

I would like to thank my professor and mentor Mile Šikić for his patience and guidance through my university years.

Additionally, I would like to thank Ivana Zec and Domagoj Marijančević from the Sisters of Charity Hospital in Zagreb for providing me the necessary data and domain knowledge.

Finally, I would like to thank my family for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.

CONTENTS

1. Introduction

Preeclampsia – a disorder known since the time of Hippocrates[1] that we still do not know enough about. Although much research has been done to better understand preeclampsia, its exact pathogenesis remains unknown. Occurring only in pregnancies, it can have severe consequences for both the mother and the fetus. Early detection of high-risk groups is then crucial so the patients can get suitable treatment [1].

Latest research has found that using statistical methods and contemporary machine learning algorithms can be helpful to achieve more precise risk calculation. The Fetal Medicine Foundation has developed a method for early screening for preeclampsia which can be performed in the first trimester of gestation (11-13 week) to calculate the patient-specific risk of preeclampsia [2].

The goal of this thesis was to create a web application for managing patients' pregnancy data, with the additional ability to calculate the risk of preeclampsia and to generate medical reports for patients. The risk estimation model used in this thesis was inspired by the method implemented by the Fetal Medicine Foundation. The Sisters of Charity Hospital in Zagreb, Croatia, for which the application was developed, had provided the data which was used to train and test the model. Some extra features for the application were requested, such as authentication, role-based authorization and visualization of statistical data related to biochemical measurements.

The application is developed in JavaScript and Python and it covers the first trimester of pregnancy with a room for expansion into the two other trimesters in the future.

[1] "The first known description of the condition was by Hippocrates in the 5th century BC." [1]

2. Overview

2.1. Preeclampsia

Preeclampsia (PE) is a disorder which occurs in pregnancy and the postpartum period (the period after delivery). It affects 2-8% of all pregnancies. It is characterized by high blood pressure and usually the presence of protein in the urine. Preeclampsia is rapidly progressive and can have consequences on both a mother and a child. Patients with severe cases of preeclampsia can experience swelling, kidney disfunction, impaired liver function, red blood cell breakdown, shortness of breath caused by fluid in the lungs, a low level of thrombocytes in the blood or visual disturbances. If left untreated, the patient can develop eclampsia which is a later stage characterized by seizures [1].

Preeclampsia usually occurs after 20 weeks of pregnancy and up to six weeks after delivery, but in some rare cases, it can occur even earlier. When preeclampsia is diagnosed, the patient can be treated with medication such as low-dose aspirin, which is found effective in reducing the complications [2]. Blood pressure medication can also be prescribed to improve the mother's condition before delivery. An abnormal placenta is thought to be the cause of preeclampsia and the removal of placenta ends the disease in most cases. However, this procedure also includes the delivery of the baby, therefore the baby's health has to be considered along with risks for the mother [1].

Preeclampsia is routinely screened for during prenatal care. The earlier it is diagnosed, the better prenatal care can be provided, therefore it is essential to identify preeclampsia as early as possible [3].

2.2. Traditional Approach to Screening

In the United Kingdom, according to the National Institute for Health and Care Excellence (NICE), some of the high-risk factors of preeclampsia include chronic hyper-

tension, autoimmune disease, chronic kidney disease, history of hypertensive disease in previous pregnancy and diabetes mellitus. Moderate-risk factors are first pregnancy (nulliparity), obesity (BMI over 35 kg/m^2), interpregnancy interval over 10 years, age over 40 years and family history of preeclampsia.

In the USA, according to the American College of Obstetricians and Gynecologists (ACOG), risk factors are chronic hypertension, chronic renal disease, diabetes mellitus, systemic lupus erythematosus, thrombophilia, history of previous pregnancy with preeclampsia, family history of preeclampsia, obesity (BMI over 30 kg/m^2), age over 40 years and conception by in-vitro fertilization [2].

2.3. New Approach to Screening

The Fetal Medicine Foundation had developed an alternative approach to early preeclampsia screening in the first trimester (which covers 11-13 weeks' gestation) which calculates a patient's risk of developing preeclampsia. It combines the prior risk from maternal factors with results of biochemical and biophysical measurements, using Bayes' theorem and regression models to compute the distribution of the gestational age at delivery with preeclampsia.

The new approach is based on a competing risk model and a survival time model for the time of delivery for preeclampsia, treated as a continuous variable. The survival time model assumes that, if the pregnancy was to continue indefinitely, all women would develop preeclampsia. The competing risk model is used to describe the competition between delivery before or after the development of preeclampsia. In pregnancies with low risk for preeclampsia, the gestational age distribution is shifted to the right which means that, in most pregnancies, delivery will occur before the development of preeclampsia. In pregnancies with high risk for preeclampsia, the gestational age distribution is shifted to the left with the implication that in many pregnancies, preeclampsia will develop before delivery. The risk of preeclampsia developing at or before a specified gestational age was given by the area under the distribution curve [4].

2.4. Statistical Methods

2.4.1. Linear Regression

The linear regression is a statistical model which assumes the linear relationship between the vector of observed values, known as the dependent variable, $\boldsymbol{y}$ and the vector of predictor variables $\boldsymbol{X}$. Its goal is to find the model values $\boldsymbol{\beta}$ which explain this relationship the best, assuming $\boldsymbol{\epsilon}$ to be the error term also known as the model's noise [5]:

$$\boldsymbol{y} = \boldsymbol{\beta}^T \boldsymbol{X} + \boldsymbol{\epsilon} \tag{2.1}$$

2.4.2. Bayesian Linear Regression

"Bayesian Linear Regression is an approach to linear regression in which the statistical analysis is undertaken within the context of Bayesian inference" [6] .

The main difference between Bayesian linear regression and standard linear regression is that the Bayesian linear regression tries to determine the posterior distribution for the model parameters as opposed to finding the single best value of the model parameters. Standard linear regression is formulated to use probability distributions instead of point estimates.

The output of a Bayesian Linear Regression model $\boldsymbol{y}$ is generated from a Gaussian (normal) distribution N which is characterized by mean and variance:

$$\boldsymbol{y} \sim N(\boldsymbol{\beta}^T \boldsymbol{X}, \sigma^2 \boldsymbol{I}) \tag{2.2}$$

The mean is calculated as $\boldsymbol{\beta}^T \boldsymbol{X}$ and the variance σ^2 is multiplied by the identity matrix $\boldsymbol{I}$.

Sampling methods are used to draw samples to approximate the posterior distributions for parameters. Monte Carlo sampling algorithms are often used for this purpose and the most common are variants of the Markov Chain Monte Carlo (MCMC) algorithm, which was used in this application [7].

The risk before a specified gestational age is calculated as the area under the distribution curve using the mean and the standard deviation of the curve. The formula is derived from the formula for the cumulative distribution function (CDF) of the standard normal distribution [8]:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt = \frac{1}{2}\left[1 + \frac{erf(x)}{\sqrt{2}}\right] \tag{2.3}$$

The derived risk probability formula is:

$$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right) = \frac{1}{2}\left[1 + \frac{erf(x-\mu)}{\sigma\sqrt{2}}\right] \tag{2.4}$$

where x is the specified gestational age.

3. Used Technologies

3.1. TypeScript

TypeScript is a programming language developed and maintained by Microsoft. It is a superset of JavaScript which introduces a static typing system into JavaScript and it is used both for frontend and backend development. It compiles to plain JavaScript using a TypeScript compiler. TypeScript supports ECMAScript 2015 features and syntax [9] [10].

TypeScript and JavaScript files can be mixed because every JavaScript code is also a valid TypeScript code. Therefore a slow transition from JavaScript to TypeScript can be made and TypeScript can be introduced where it is most needed. The backend of this project was written in TypeScript. Using types is very useful because many errors can be caught in development. What most people forget is that TypeScript won't catch the runtime errors better than JavaScript because, in the end, it is compiled to a plain JavaScript, so all extra features which it adds are lost after transpiling. Despite that, it is recommended to use it because it is very helpful.

3.2. Node

Node.js is a cross-platform, open source JavaScript runtime environment for executing JavaScript code outside of a browser. It is built on Chrome's V8 JavaScript engine, which is an open source high-performance engine written in C++ [11]. Node has an asynchronous event-driven architecture which contrasts with today's more common concurrency model where OS threads are employed. One of the plus sides of this is that there are no locks in Node because it operates on a single thread event loop and uses non-blocking I/O calls, so there isn't a chance of dead-locking the process. Because of that, it is very reasonable to develop scalable systems in Node. Also, Node is often chosen to unify web application development around a single programming

language which makes development easier because programmers do not have to learn multiple languages or shift between two or more languages every day if they are full-stack developers [12] [13]. Node also comes with a preinstalled npm package manager, which is currently the most popular JavaScript package manager and one of the fastest growing package managers with its huge registry. A project's dependencies are written in the package.json file and they are installed in the *node_modules* directory [14].

3.3. Express

Express is the most popular Node framework and is used in many Node applications. The most specific features of Express are routing and middleware. Routing is the way the endpoints respond to client requests. The application's endpoints are defined as routes and the application listens for requests that match the specified route. When the route is matched, its callback function is called [15]. Middlewares are functions which accept the request object (*req*), the response object (*res*), and the *next* function as parameters. The *next* function executes the middleware which comes after the current middleware [16].

3.4. Sequelize

Sequelize is a promise-based Node ORM (Object-relational mapping). It has support for Postgres, MySQL, MariaDB, SQLite, and Microsoft SQL Server [18]. The application uses a Postgres database so PostgreSQL client for Node must be installed along with Sequelize. For that purpose, the pg library is used in this application [19].

3.5. Passport and JTW

Passport is an authentication middleware for Node web applications. It contains many strategies which support different authentication methods such as using a username and password, Facebook, Twitter, etc. [20]. The *passport-local* strategy is used for authenticating with a username and password [21] for login and signup endpoints. The *passport-jwt* strategy is used for authenticating with a JSON Web Token [22].

JWT or JSON Web Tokens are "an open, industry standard RDC 7519 method for representing claims securely between two parties." [23]. When a user successfully logs in using their credentials with passport-local strategy, a JWT token will be signed using

user data and the JTW secret (which should be stored as the application's environment variable) and returned with the response. The library jsonwebtoken [24], which by default uses the HMAC SHA256 algorithm, was used for signing tokens. Whenever the user wants to access a protected route, the JTW token should be sent with the request, typically in the *Authorization* header using the Bearer schema [23].

3.6. Puppeteer and Handlebars

Puppeteer and Handlebars were used for generating PDF reports. Handlebars is a templating language and in this application, it was used to create a report template in an HTML-like format. The key feature of Handlebars is its embedded expressions which can be used to dynamically insert data into a template file. After evaluating a Handlebars template by executing the template with a context, the HTML is generated [25]. Puppeteer is a Node library which provides a high-level API to control Chrome or Chromium. It is used for generating screenshots and PDFs of pages, server-side rendering, UI testing, etc [26]. Puppeteer was used for generating a PDF file from the HTML generated by Handlebars.

3.7. React

React is a frontend JavaScript library for building user interfaces. It is component-based, which means the interface is a compound of multiple components, each of which encapsulates a part of the view and logic around it. Each component is a small and isolated part of the code which handles its own state and can also be reused across the application. That makes development easier due to less code repetition.

React is a declarative library, i.e. every component returns a description of what needs to be displayed on the screen. This is achieved using pure JavaScript functions for creating elements or using a special JSX (JavaScript XML) syntax which is similar to HTML but is actually just another way of writing JavaScript functions. JSX is also more powerful than pure HTML and it is great for structuring a component's display. Components are nested and together form a hierarchy of views. React's philosophy is centered around combining components and managing their data and events between them.

React follows the "data down, actions up" pattern which is used in many modern frontend frameworks. That means that all the data is passed down from a parent com-

ponent to its child components through so-called *props* (properties) and all events can be propagated up by calling actions (functions sent as *props*) from the child component to the parent component. This way of sending data via *props* also means that React has one-way data binding, unlike some other frameworks such as Angular or Vue which support two-way data binding [27] [28].

Components are rerendered only when it is needed, for example when their *props* have changed, and it is handled by the Virtual DOM (Document Object Model). The Virtual DOM is a virtual in-memory representation of the DOM. React uses it to compute the difference between the previous DOM and the new one to efficiently update the browser's DOM. Updating the browser's DOM is quite expensive and Virtual DOM is an efficient pattern for handling DOM updates [29].

React components come with a set of lifecycle methods. Lifecycle methods are hooks which can be overridden to have better control of the lifecycle of a component (mounting, updating, unmounting, error handling, etc.) [27] [28].

3.8. Redux

Redux is a JavaScript state management library. As frontend development became more complex and JavaScript single-page applications (SPA) took off, the state of frontend applications became hard to manage. The lack of state management is the main reason React is considered a library instead of a full framework. Because React itself doesn't come with pre-built state management, it is often paired with Redux.

Redux can be described in three fundamental principles: a single source of truth, read-only state and using pure functions for updating the state. Single source of truth means that the application's state is centralized in an object tree within a single store. The store can then be accessed from every component. The application's state cannot be changed directly from the components because the state is exposed as read-only. It can only be changed by emitting (dispatching) actions – objects which describe a state transition. After an action is dispatched, it is handled by reducers. Reducers are pure functions which are responsible for updating the state. They take the previous state and action and return the new state object without mutating the previous state [30].

3.9. Bootstrap

Bootstrap is a frontend framework used for styling the frontend web applications. It implements CSS classes and components that are ready to be used to design responsive web pages. Using a CSS framework like Bootstrap makes the development process easier because it can make a web page look aesthetically pleasing with minimal effort. However, the problem with using Bootstrap is that applications can look very similar to each other if they all use the same style sheet. This has a bad effect on the originality of a page's design. Therefore, it is suggested to enrich the Bootstrap's pre-built design with some custom stylings to make the page stand out. This can be achieved by adding custom CSS classes or directly overriding existing Bootstrap classes [31].

This project uses the react-bootstrap library which brings Bootstrap to React, encapsulating Bootstrap CSS classes into React components that can be used across the application. The CSS classes can still be added to React components in the JSX code in a way similar to how it is done in HTML. However, having special components for different styles can have some benefits for code readability [32].

3.10. Sass

Sass (Syntactically awesome style sheets) is a preprocessor style sheet language which compiles to plain CSS (Cascading Style Sheets) code. It has two syntaxes – the original syntax which is called "the indented syntax", and the newer SCSS syntax. The SCSS syntax was used in this application. When the application starts, every SCSS file (the *.scss* file extension) is compiled to a corresponding CSS file. Similarly to how every JavaScript code is a valid TypeScript code, every CSS code is a valid SCSS code because SCSS syntax only extends the CSS syntax. SCSS introduces variables, nesting, selector inheritance and mixins. Those functionalities make code more reusable and easier to maintain, which speeds up development [33].

```
1 $col-primary:           #1565c0;
2 $col-light-blue-01:     #bbdefb;
3 $col-light-blue-02:     #90caf9;
4 $col-light-blue-03:     #64b5f6;
5 $col-light-blue-04:     #42a5f5;
6 $col-error:             #cc0000;
```

Code snippet 3.1: Defining variables for colors using the SCSS syntax. These variables can be imported from any SCSS file and be used as CSS properties.

4. Implementation

4.1. Project Structure

The application is divided into two main directories: *preeclampsia-node-app* (the backend application written using Node and Express) and *preeclampsia-react-app* (the frontend application written using React).

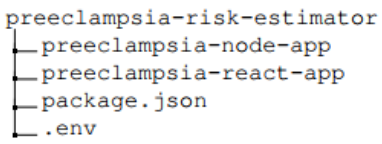

Figure 4.1: The application's folder structure

They share the same *package.json* file located in the project's root folder. That means their startup scripts are called from the root directory but in parallel, from different consoles, in development. In production, they are built together into a single application. Sharing the same *package.json* file also means that the frontend and backend applications share the same *node_modules* (the directory which contains installed libraries) so there are no duplicate dependencies installed. The only exception is the additional *package.json* file in the *preeclampsia-react-app* which contains scripts which can be called from the root *package.json* file and are responsible for building CSS and JavaScript files and starting the development server. These scripts can technically be moved to the root *package.json* file, but it was more elegant to separate them.

The ESLint linter is used for both backend and frontend development. The application uses absolute paths for its main directories. The absolute paths on the frontend side are configured in the *jsconfig.json* file. After they are listed in *jsconfig.json* file, they automatically work because *create-react-app* package (the package used for the React setup) is configured to read these paths. The absolute paths used on the backend are listed in the *tsconfig.json* file, but the extra step was needed for them to work. Node doesn't use these paths automatically, and therefore the *module_alias* library was used.

Additionally, all paths need to be listed in the *package.json* file and the *module_alias* package needs to be imported in the *server.ts* file, which is the entry file for the backend.

4.1.1. Backend

The backend application is written in TypeScript which is configured via a *tsconfig.json* file located in the project's root directory. TypeScript code is located in the src directory of the *preeclampsia-node-app* directory and it is transpiled to JavaScript code using the TypeScript compiler. The transpiled code is located in the *dist* directory. The src directory contains the following directories: *assets*, *configuration*, *constants*, *controllers*, *dataTransferObjects* (*requestModels* and *viewModels*), *enums*, *middlewares*, *models*, *routes*, *scripts*, *services*, *templates*, *typings*, *utils*, and *validators*.

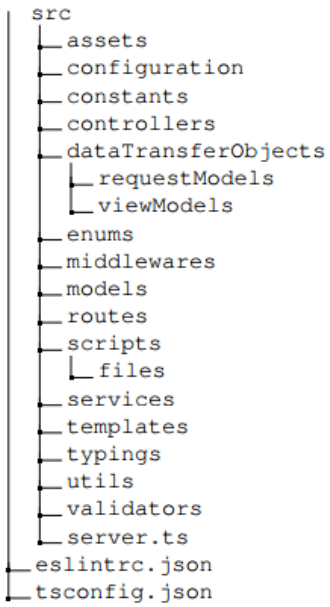

Figure 4.2: Backend folder structure

The *assets* directory contains images and other asset files.

The *configuration* directory contains configuration files, such as Express configuration, authentication middleware setup (Passport), routes and database initialization (Sequelize).

The *constants* and *enums* directories, as their names suggest, contain constants and enumerations. Controllers are defined in the *controllers* directory and are called from the *routes* directory where all the endpoints of the application are defined.

Middleware files are also imported into routes and added to request pipelines before controller actions are called (authentication and language middleware) or are imported directly into the *server.ts* file to include them in every request pipeline (error middleware). Controllers import *services* which handle database calls, *validators* which validate the request body, and *utils* (short for utilities) which contain helper functions used across the application.

The medical report template is located in the *templates* directory.

The *models* directory contains definitions of Sequelize models that are imported into services. Request models are interfaces that define the structure of a request's body and view models are classes that define the structure of a response.

The *scripts* directory contains scripts for populating the database with initial data and the Python scripts for training the risk estimation model and risk evaluation. The generated model is saved in the *files* subdirectory.

The *typings* directory is a special TypeScript directory whose purpose is defining global types or types for libraries which do not have any types (or have incorrect types that should be overridden).

4.1.2. Frontend

The frontend application is written in JavaScript and its build setup was generated using the *create-react-app* package, which configures everything needed for React application development. The *preeclampsia-react-app* directory contains two subdirectories: *public*, and *src*.

The *public* directory contains *index.html*, which is the root HTML file, *favicon.ico*, which is the favicon displayed on every page of the application, and *manifest.json*, which is a web application manifest that describes the application.

The *src* directory contains these subdirectories: *assets*, *components*, *constants*, *enums*, *redux*, *scenes*, *styles*, and *utils*.

The assets directory contains images, translations, and other application assets.

The *constants* and *enums* directories contain the application constants and enumerations.

The *components* and *scenes* directories contain React components. The reason why components are separated into these two directories is that scenes are components which represent whole pages and their content, whereas *components* are smaller components that are shared across the application. Every scene has a different route and the routing between scenes is handled by the *react-router-dom* library. Existing scenes

will be explained in more details later in this chapter.

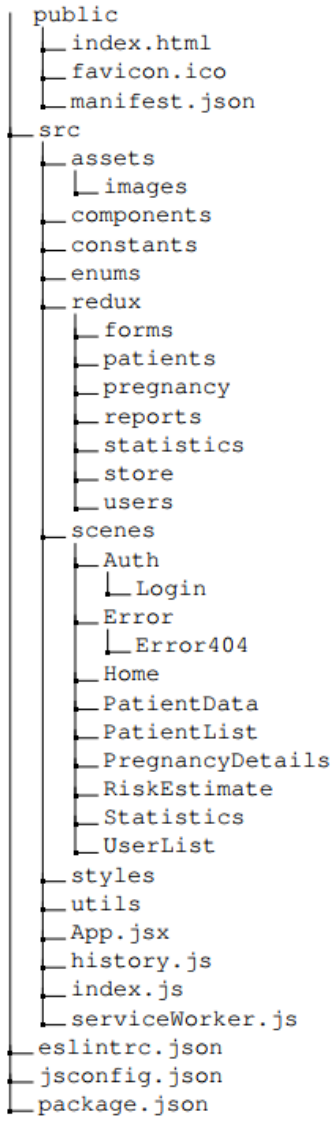

Figure 4.3: Frontend folder structure

The *redux* directory contains all redux logic. Subdirectories *patients*, *pregnancy*, *reports*, *statistics* and *users* each represent a part of the redux state and contain files with actions, action creators, action types and reducers for that part of the state. The *forms* subdirectory contains constants for all the application's forms and is used by the *redux-form* library. The *redux-form* library is used for handling forms and it provides a special reducer for forms. The *store* subdirectory configures the application's store, combining all of the existing reducers into a single root reducer. The store is imported in the *index.js* file, which is the root JavaScript file in the React application.

The *utils* (short for utilities) directory contains helper functions and the *styles* di-

rectory contains the SCSS files used for styling the application.

4.2. Database Model

The database model is generated by a code-first approach using Sequelize model definitions. Every Sequelize model is defined as a class which represents a database relation. The relations are *Users*, *Patients*, *Pregnancies*, *MedicalExaminations*, *Reports*, *Characteristics*, *BooleanMeasurements*, *EnumMeasurements*, and *NumericalMeasurements*. The last three represent measurements calculated during a medical examination and their purpose is to generalize these measurements by connecting them to a specific characteristic and containing a value for that characteristic. The value can be numerical (*NumericalMeasurements*), boolean (*BooleanMeasurements*) or it can be an enumeration value represented as a number (*EnumMeasurements*).

Characteristics are inserted into the database with a script for populating the database using a constants file which defines all available characteristics. The *Characteristics* table can be expanded in the future with other types of measurements.

```
1  import * as Sequelize from 'sequelize';
2  import UserRoles from 'constants/roles.constants';
3  import { encrypt, compareEncrypted } from 'utils/encryption.utils';
4
5  export default (sequelize) => {
6    const User = sequelize.define('User', {
7      id: {
8        type: Sequelize.INTEGER,
9        primaryKey: true,
10       autoIncrement: true,
11     },
12     firstName: {
13       type: Sequelize.STRING,
14       allowNull: false,
15     },
16     lastName: {
17       type: Sequelize.STRING,
18       allowNull: false,
19     },
20     email: {
21       type: Sequelize.STRING,
22       unique: true,
23       validate: {
```

```
24          isEmail: true,
25        },
26      },
27      role: {
28        type: Sequelize.ENUM(...Object.values(UserRoles)),
29        allowNull: false,
30      },
31      hashedPassword: {
32        type: Sequelize.STRING,
33        allowNull: false,
34      },
35    }, {
36      paranoid: true,
37    });
38
39    User.prototype.isValidPassword = async function(password) {
40      const user = this;
41      return await compareEncrypted(password, user.hashedPassword);
42    };
43
44    User.addHook('beforeCreate', async (user) => {
45      const hashedPassword = await encrypt(user.hashedPassword);
46      user.hashedPassword = hashedPassword;
47    });
48
49    User.associate = (models) => {
50      models.User.hasMany(models.Report, {
51        foreignKey: {
52          name: 'generatedById',
53          allowNull: false,
54        },
55        as: 'reports',
56      });
57    };
58
59    return User;
60  };
```

Code snippet 4.1: The User model definition using Sequelize. The *isValidPassword* is the instance method for checking if the provided password is the same as the user's password. The password is always hashed upon user creation because of the added *beforeCreate* hook which hashes the password using the *bcrypt* library. The User model has a one-to-many relationship with the Report model, which is defined using *hasMany* method.

4.3. Risk Estimation

The risk estimation logic is divided into generating the risk model and the risk calculation on a patient level. These two scripts were written in the Python 3 programming language. Python was chosen because it has the ecosystem for machine learning, consisting many different libraries such as *PyMC3*, *sklearn* and *numpy*, all of which were used in this application.

4.3.1. Generating Model

The *train_risk_model.py* Python script is used to generate a Bayesian linear regression model. The *PyMC3* library was used for initializing, training and testing the model. After the model is generated, it is serialized using the *pickle* library and stored on the backend as a binary file.

Training was done on 156 pregnancies, with only one pregnancy positive for preeclampsia. Risk parameters used for training the model are: biochemical measurements PLGF and PAPP-A (converted to MoM values), BMI (Body Mass Index), MAP (Mean Arterial Pressure), nulliparity, smoking during pregnancy, in-vitro fertilization, is the patient's age over 30, and whether or not the patient has diabetes. After filtering out all pregnancies that are missing any of these characteristics, 86 pregnancies remained for training the model. The results of that training are shown in figures 4.4 and 4.5. The calculated median for the model's intercept is 49.1, which is less than the value the Fetal Medicine Foundation's method calculated (which is 55). This is due to the limited amount of data available for training the model, especially the lack of positives. Additionally, the data provided for this thesis does not contain some high-risk factors, such as the information if patients have chronic hypertension and the history of preeclampsia in the patient's family and previous pregnancies, from which the model would have greatly benefited. In consequence, when taking week 34 of the gestational age as the upper bound for calculating the area under the patient's distribution curve (an example of the curve is shown in figure 4.6), which is the same upper bound that the Fetal Medicine Foundation uses, the model is going to predict a lot of false positives.

The example shown in figure 4.6 represents the result given by the model's observation of a pregnancy that has all of its parameters set to default, i.e. normal and expected values which represent a healthy pregnancy. The median calculated for the observation is 38.5527, which is lower than expected because healthy pregnancies shoud be shifted more to the right and therefore have a higher median value. The reason for such a low

median value is probably due the mean value of the model's intercept being lower than the one obtrained by the Fetal Medicine Foundation's model.

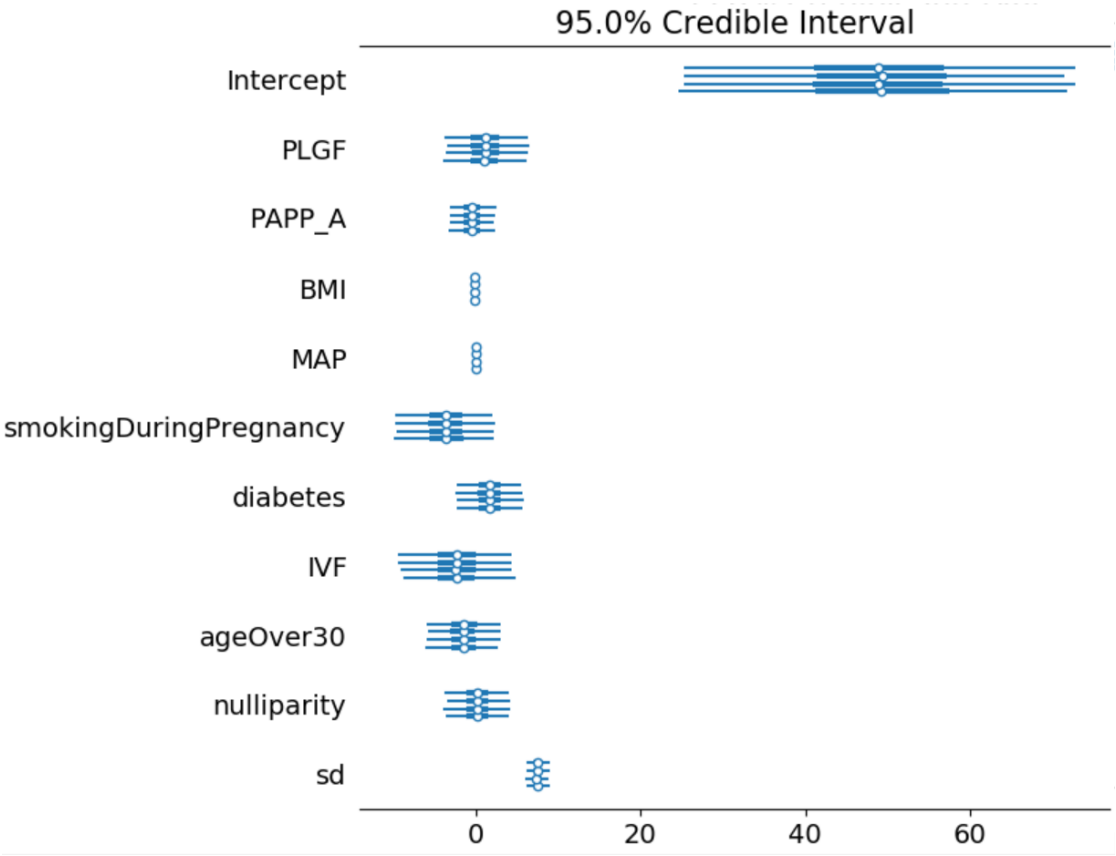

Figure 4.4: The forest plot displaying the most likely value for every parameter of the model with its 95% credible interval

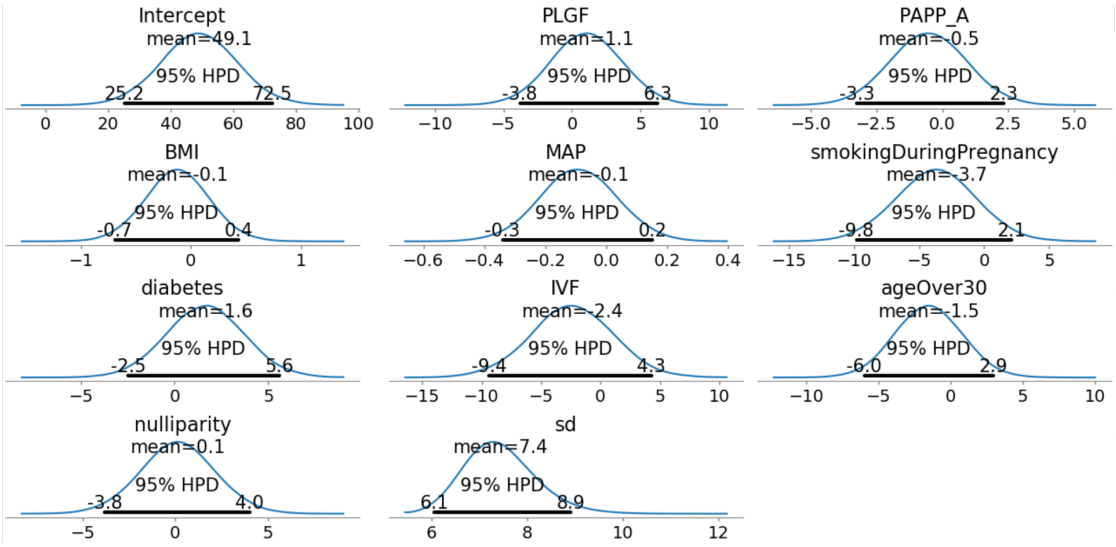

Figure 4.5: The posterior distribution plots for every parameter used in the model

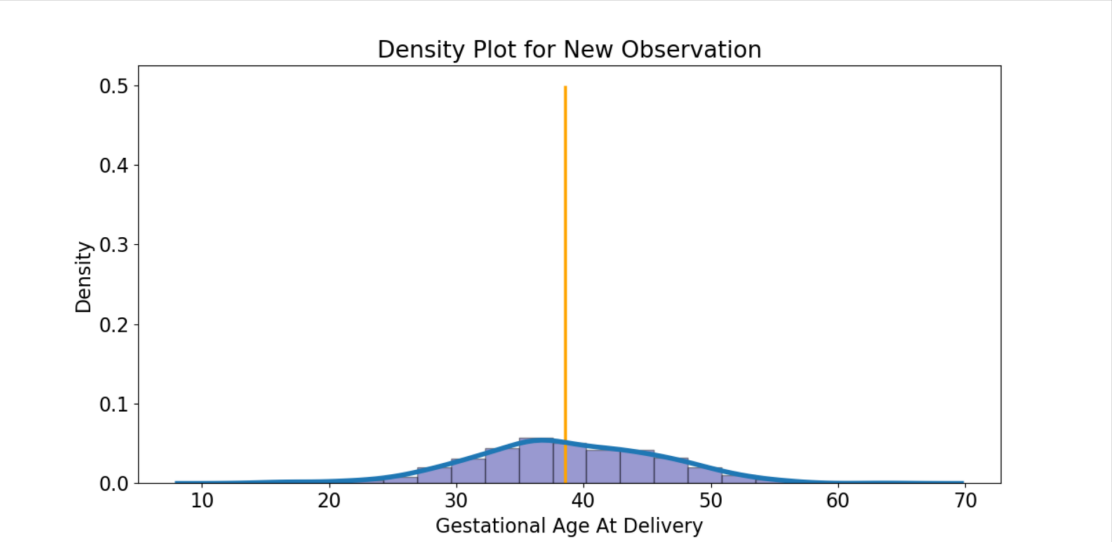

Figure 4.6: Healthy pregnancy risk observation

4.3.2. Calculating Risk

The risk controller contains the action for calculating the risk of preeclampsia for a given medical examination and generating a PDF report for the patient. The risk estimation is done by the *calculate_risk.py* script written in Python. The Python script for risk estimation is called from the Node application using Node's built-in *child_process* module, which provides the ability to spawn child processes. The *child_process.spawn()* method is used for spawning the child process asynchronously, without blocking Node's event loop [17].

The *child_process.spawn()* method accepts a command to run (for Python scripts it is *python*) and an array of arguments (the first one being the name of the script). The arguments passed to the *calculate_risk.py* script are the patient's data and those arguments are used as input parameters for the model inside the script. The script parses the aguments and deserializes the generated model from the file containing it using the *pickle* library. The distribution of gestational age at delivery is then calculated by querying the model with the patient's data. The mean and standard deviation of the distribution is then used to calculate the patient's risk of preeclampsia.

Meanwhile, the Node application listens for an output from the spawned process. If the output comes from the *stderr*, it means an error occured in the spawned process and the action returns the error response. If the script finishes successfully, it prints and flushes the response object containing the calculated risk. The Node application's *stdout* listener then receives the output from the script, parses it to a JavaScript object

and continues generating the report.

```
const params = await ReportService.parseRiskEstimationData(
    medicalExamination
  );

  console.info('Starting the python script...');
  console.info(`Params: ${params}`);

  const pythonProcess = spawn('python', [
    path.join(__dirname, '..', 'scripts', 'calculate_risk.py'),
    ...params
  ]);

  pythonProcess.stdout.on('data', async (data) => {
    console.info('The python script finished successfully');

    const response = JSON.parse(data.toString().replace(/'/g, '"'));
    const { risk } = response;

    /* code which handles generating PDF report */

    pythonProcess.kill();
  });

  pythonProcess.stderr.on('data', (err) => {
    console.info('The python script finished unsuccessfully');
    console.info(`Error: ${err.toString()}`);
    pythonProcess.kill();
    throw new Errors.InternalServerError();
});
```

Code snippet 4.2: Calling the *calculate_risk.py* script from the risk controller in Node. The script is started using the *child_process.spawn()* method which accepts a command to run and an array of arguments.

4.4. Authentication and Authorization

Authentication and authorization are achieved by using the Passport library and JWT Web Tokens.

The Passport middleware configuration for login, signup, and authentication are set

at startup of the Node application. The login middleware is used when a user tries to log into the application. The signup middleware is used for adding new user accounts to the application. In this application, that can be done only by an admin user and therefore there is no public route for user registration. The login middleware also generates a JWT token signed with a JWT secret if the action was successful. This token must be saved on the frontend and used in every request which needs authentication. Every other route uses the authentication middleware which extracts the JWT token from *Authorization* header with the Bearer scheme and validates it. If the token is valid, the pipeline of the request handling will continue.

```
1  // Login
2  passport.use('login', new LocalStrategy({
3    usernameField : 'email',
4    passwordField : 'password',
5  }, async (email: string, password: string, done) => {
6    try {
7      // Find the user by E-mail
8      const user = await UserService.getByEmail(email);
9      if (!user) {
10       // User not found
11       return done(null, false);
12     }
13
14     const isValidPassword: boolean =
15       await user.isValidPassword(password);
16     if (!isValidPassword) {
17       // Incorrect password
18       return done(null, false);
19     }
20
21     // Call the next middleware
22     return done(null, user);
23   } catch (error) {
24     return done(error);
25   }
26 }));
```

Code snippet 4.3: The Passport login middleware. This middleware uses the *LocalStrategy* to extract the user's E-mail and password from the request. The User service is used to find the user in the database. The User model's custom instance method *isValidPassword* is used to check if the password provided matches the user's password.

```
const login = async (req, res, next) => {
  const { translations } = res.locals;

  passport.authenticate('login', async (err, user, info) => {
    try {
      if (err) {
        throw new Errors.InternalError(
          translations.response.error.login
        );
      }

      if (!user) {
        throw new Errors.UnauthorizedError(
          translations.response.error.login
        );
      }

      req.login(user, { session : false }, async (error) => {
        if (error) {
          throw new Errors.InternalError(
            translations.response.error.login
          );
        }

        const body = { id: user.id, email: user.email };
        const token = jwt.sign(
          { user: body },
          process.env.JWT_SECRET,
        );

        return res.json(new UserLoginViewModel(user, token));
      });
    } catch (error) {
      return next(error);
    }
  })(req, res, next);
};
```

Code snippet 4.4: The login action located in the user controller. The action calls the login middleware. If the user is returned and no error is thrown, the Passport's *req.login* method is invoked. If it finishes successfully, a JWT token is signed using the user's data and the JWT secret, and the *UserLoginViewModel* is returned.

The user's password is not saved directly into the database but is first hashed using the *bcrypt* library [34] which uses the password hashing function of the same name.

There are three roles in the application: admin, supervisor, and standard. Authorization is handled after authentication with a special authorization middleware which checks if allowed roles contain the user's role.

4.5. Localization

The application has support for two languages – English and Croatian. This is achieved by using two files with constants for translations, one on the backend side for all text constants which are being returned as response messages and one on the frontend side which contains all the text that can be displayed. These constants can then be accessed from the code where needed.

On the frontend side, the Translator singleton class contains methods for fetching and changing the current language and it can be accessed from every part of the React application. It also contains methods for fetching translations, therefore all the logic regarding languages and translations is handled by the Translator singleton. The rest of the application doesn't have to know which language is currently used because there is a clear separation of concerns.

The Node application has a language middleware which extracts the *Accept-Language* request HTTP header and sets the translations depending on the language sent in the request header. If no language is provided or if there are no translations for the requested language, the default language is used. This middleware is called on every request before calling a controller's action. This way the rest of the application doesn't have to know which language is used because the translations are already set and ready for use.

```
1  import values from 'constants/values.constants';
2  import { languages } from 'constants/language.constants';
3  import { translations } from 'constants/translations.constants';
4
5  export const setLanguage = (req, res, next) => {
6    let language = req.headers['accept-language'];
7
8    if (!language || !Object.values(languages).includes(language)) {
9      language = values.DEFAULT_LANGUAGE;
10   }
11
```

```
12    res.locals.translations = translations[language];
13    res.locals.language = language;
14
15    next();
16  };
17
18  export default {
19    setLanguage
20  };
```

Code snippet 4.5: The language middleware on the backend side of the application. It extracts the *Accept-Language* HTTP header from the request (*req*), and if the language is provided and is valid, the translations for that language will be used further in the request's pipeline. In other cases, translations for the default language will be used. Selected translations and language are stored into *res.locals*.

With this configuration, more languages can be easily added.

4.6. Application Structure

4.6.1. Scenes

The application is separated into several scenes (pages). The routing between pages is handled by the *react-router-dom* library. The application has a navigation bar at the top of every scene to enable easier navigation. The navigation bar also contains the language selector (figure 4.7) and the logout action button (figure 4.8), which is displayed when a user is logged in.

All existing scenes with their routes will be described in the following text.

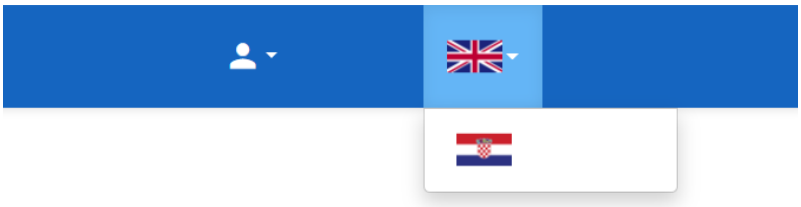

Figure 4.7: Language selector

Figure 4.8: Logout button

– **/login**

The login scene has the E-mail and password inputs where a user writes their login credentials. If the credentials are incorrect, a validation text will be displayed (figure 4.10). If the user tries to access the page which requires authorization without first logging in to the application, the application will redirect the user to the login scene.

After a successful login, the user is redirected to the home page of the application. The navigation bar also changes, and the available links are displayed. If the user's role doesn't have permission to view the scene, it won't be displayed in the navigation bar. For example, only the admin user role has access to the User List scene and therefore the link for that scene is shown only to users with the admin role. If a user tries to access a route he or she shouldn't have permission for, he or she will be redirected to the home page.

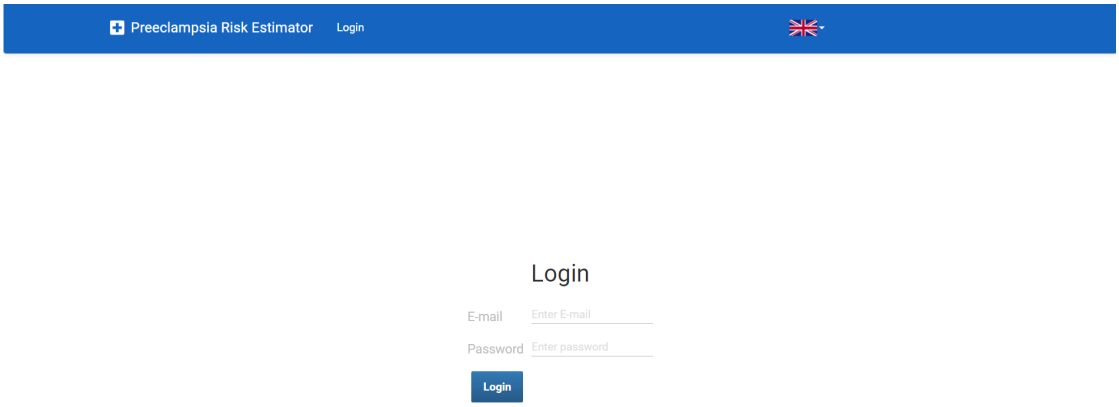

Figure 4.9: Login form

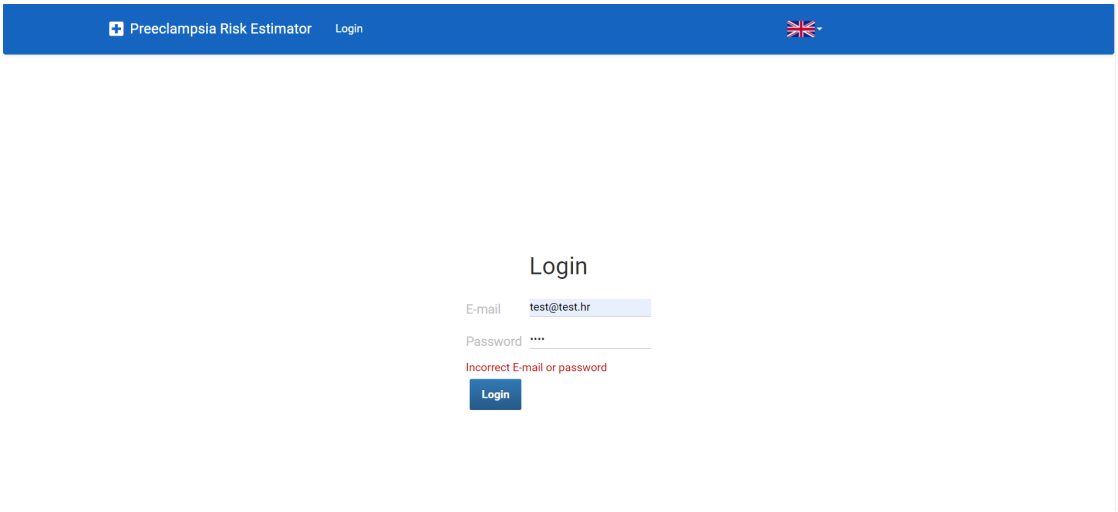

Figure 4.10: Login form with failed validation

– **/users**

The User List scene displays a table with all the application's users. Only users with the admin role have access to this scene. The table can be sorted by all columns and it has pagination, which is handled on the backend. The *react-table* library was used for all tables in this application.

In the top right corner there is an "Add User" button which opens a modal for adding a new user. The modal has inputs for all user data, including the password input and the extra input for repeating the user's password. The *redux-form* library was used to create forms. The form is validated on the backend and, if any field is incorrect, an error message will be shown under the input field (figure 4.12).

After a click on a user row, the sidebar with the user's details is shown (figure 4.11). The sidebar also contains three buttons: "Edit User Data", "Change Password" and "Delete User". Each button opens a modal for the corresponding action: the modal for editing the user, the modal for changing the user's password (figure 4.13) and the modal for deleting the user (figure 4.14).

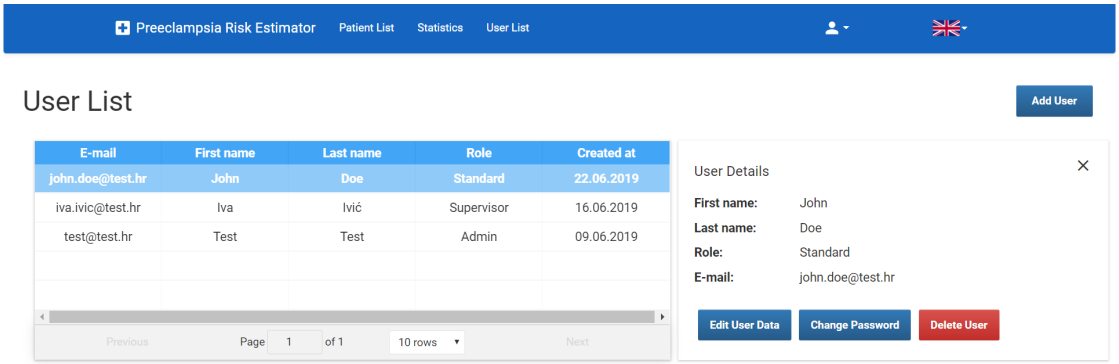

Figure 4.11: User list with an open sidebar displaying the user details

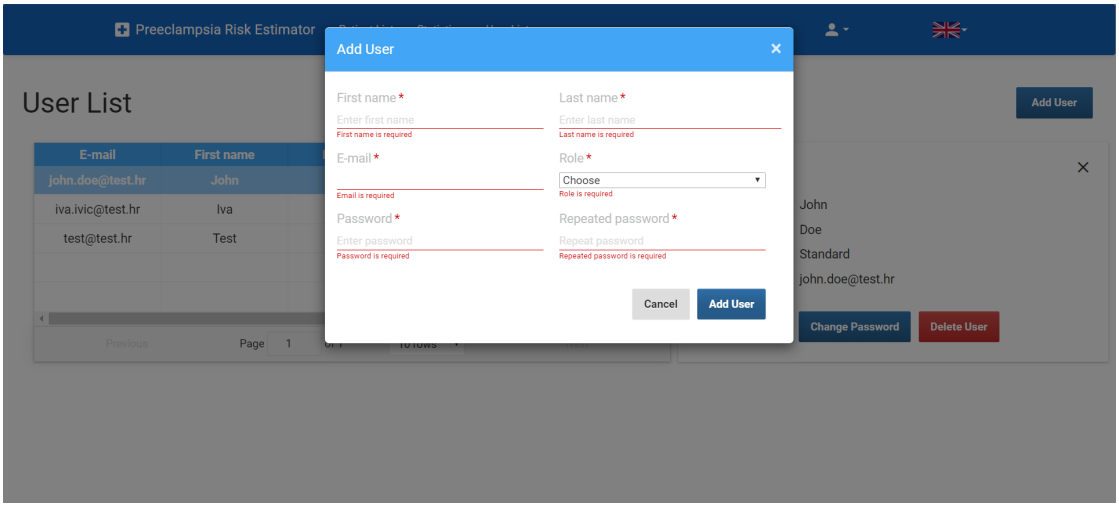

Figure 4.12: The modal for adding a new user with failed validation

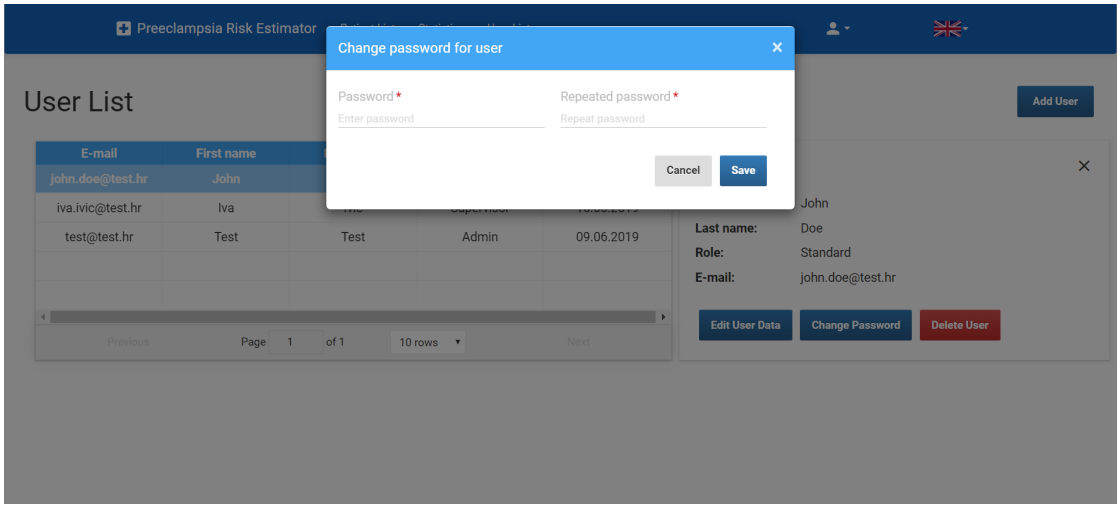

Figure 4.13: The modal for changing the user's password

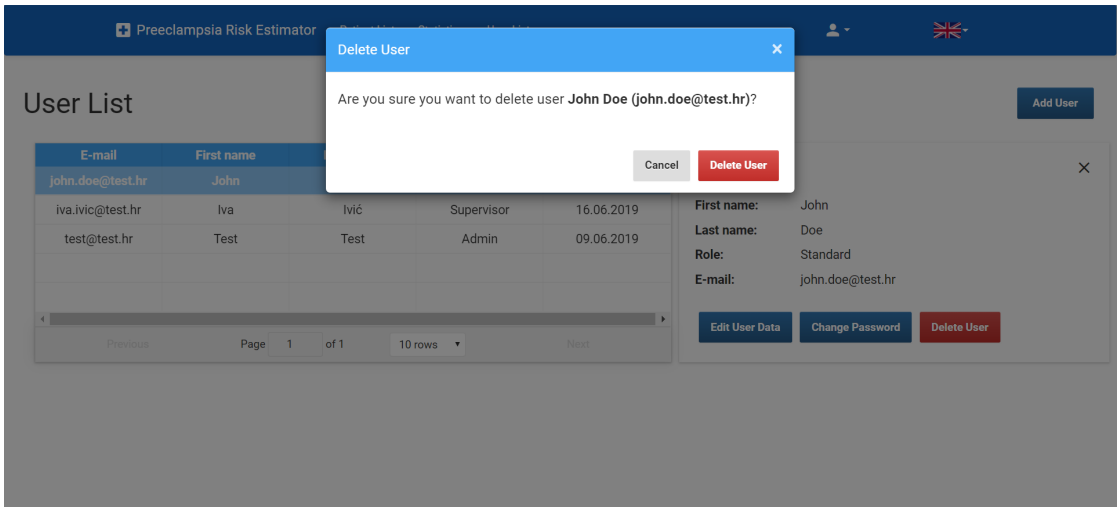

Figure 4.14: The modal for deleting user

– **/patients**

The Patient List scene displays a table containing all the patients. The table can be sorted by all columns and it has pagination, which is handled on the backend.

In the top right corner there is a search input and an "Add Patient" button. The search input is used to filter the table's rows by a patient's first name, last name or patient ID (which represents the special ID provided by the medical system of a country). The "Add Patient" button opens a modal for adding a new patient. The modal contains a form with patient data inputs such as first name, last name, birthday, etc. The form is validated on the backend and, if any field is incorrect, an error message will be shown under the corresponding input field.

After s click on a patient row, a sidebar with the patient's details is shown (figure 4.16). The sidebar also contains buttons for editing and deleting the patient, which opens special modals for those actions. A click on the patient's ID in the table opens the Patient Details scene.

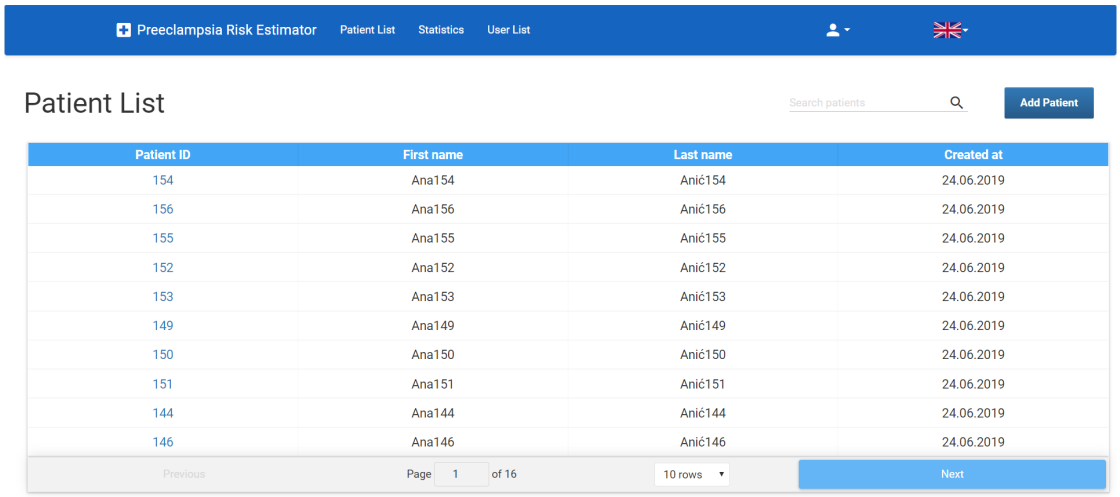

Figure 4.15: Patient list

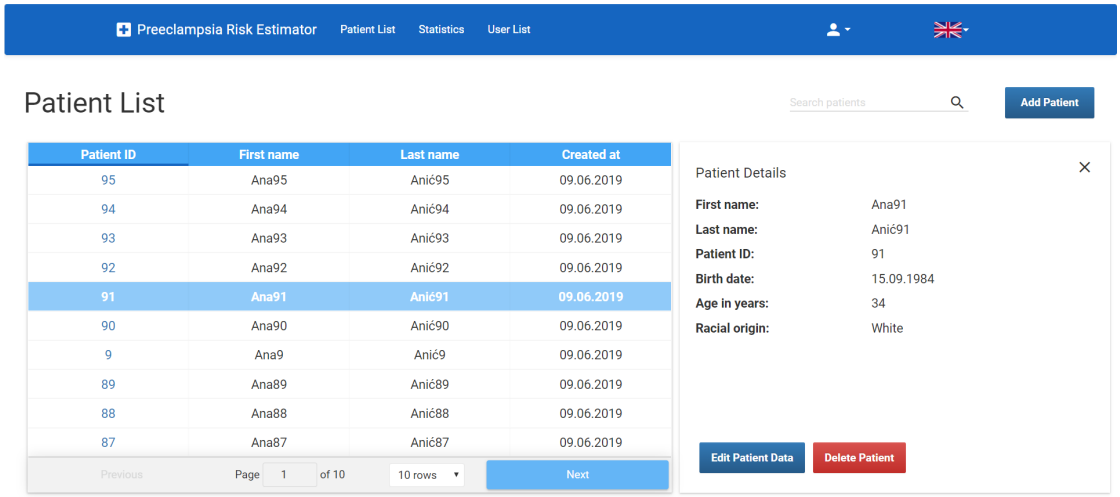

Figure 4.16: Patient list with an open sidebar displaying the patient details

– **/patients/{*patientId*}**

The Patient Details scene displays details about the patient and their pregnancy history. The pregnancy history is the list of all of the patient's pregnancies, each one linking to the Pregnancy Details scene. Next to the pregnancy history list, there is an "Add Pregnancy" button, which opens a modal for adding a new pregnancy for the patient.

The Patient Details scene also contains a button in the top-right corner for editing the patient's data.

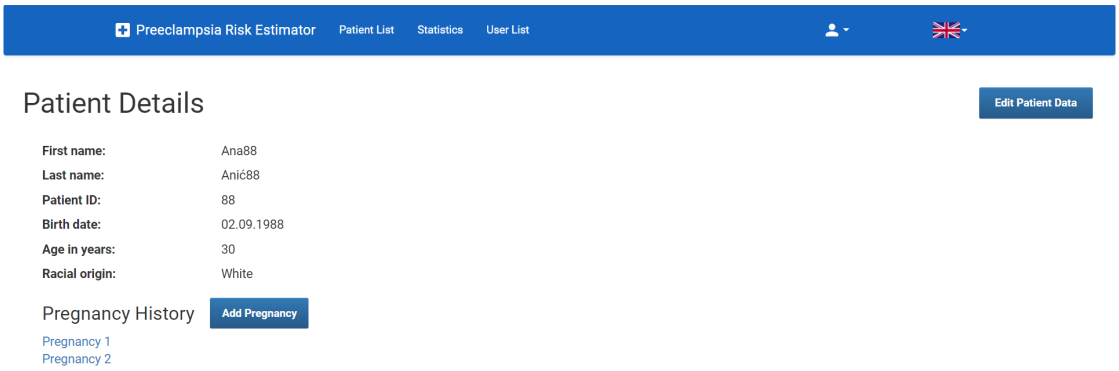

Figure 4.17: Patient details scene with the patient's data and pregnancy history

– **/patients/{*patientId*}/pregnancies/{*pregnancyNumber*}**

The Pregnancy Details scene displays details about a patient's specific pregnancy and medical examinations for that pregnancy (currently only for the first trimester but can be expanded to the second and the third trimester).

In the top-right corner there is a "Delete Pregnancy" button which opens the modal for deleting the pregnancy. The Medical Examination section contains medical examinations for pregnancy trimesters (figure 4.19). Next to the Medical Examinations section there is an "Add Medical Examination" button, which opens the modal for adding a new medical examination for the pregnancy. Each medical examination is divided into several data containers that can be edited separately (figure 4.20).

At the bottom of the medical examination there is a "Calculate risk" button which opens the Generate Report scene.

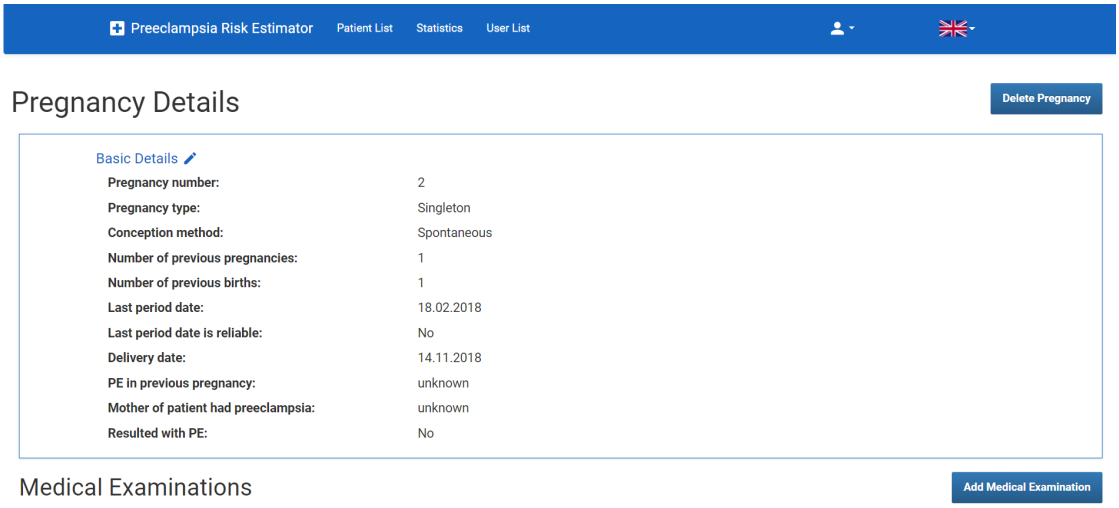

Figure 4.18: Pregnancy details

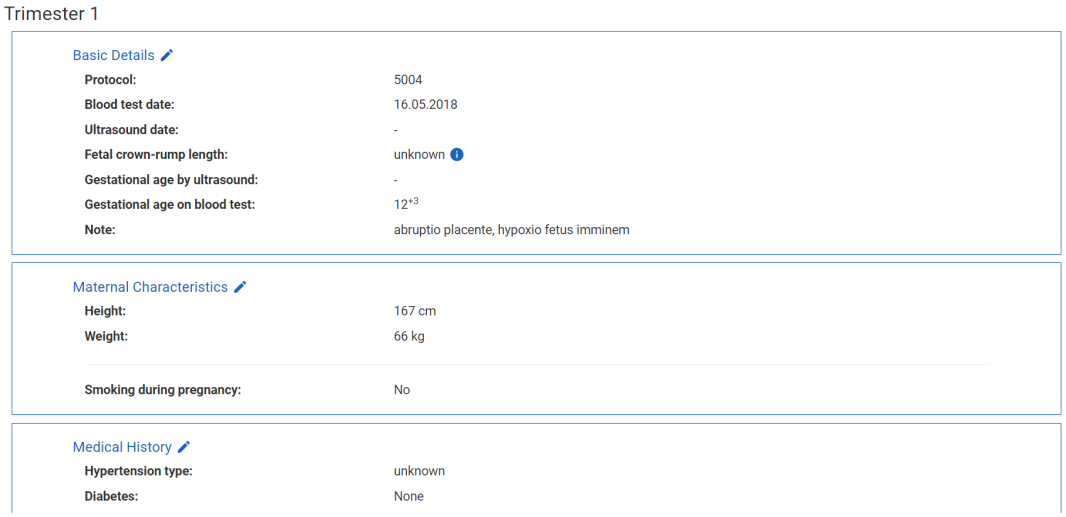

Figure 4.19: Medical examination details for the first trimester of pregnancy

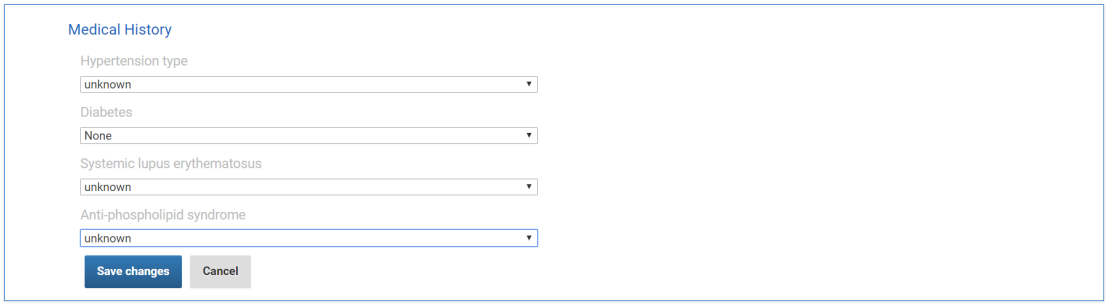

Figure 4.20: Editing measurements

– **/patients/{*patientId*}/pregnancies/{*pregnancyNumber*}/examinations/{*examinationId*}**

The Generate Report scene displays all known data for the medical examination (figure 4.21). Under the patient data there is a "Generate report" button which starts the risk calculation and report generation. A spinner will appear while the report is being generated (figure 4.22). After the report generation is finished and the data is received from the backend, the PDF report (figure 4.23) will be automatically downloaded by the browser.

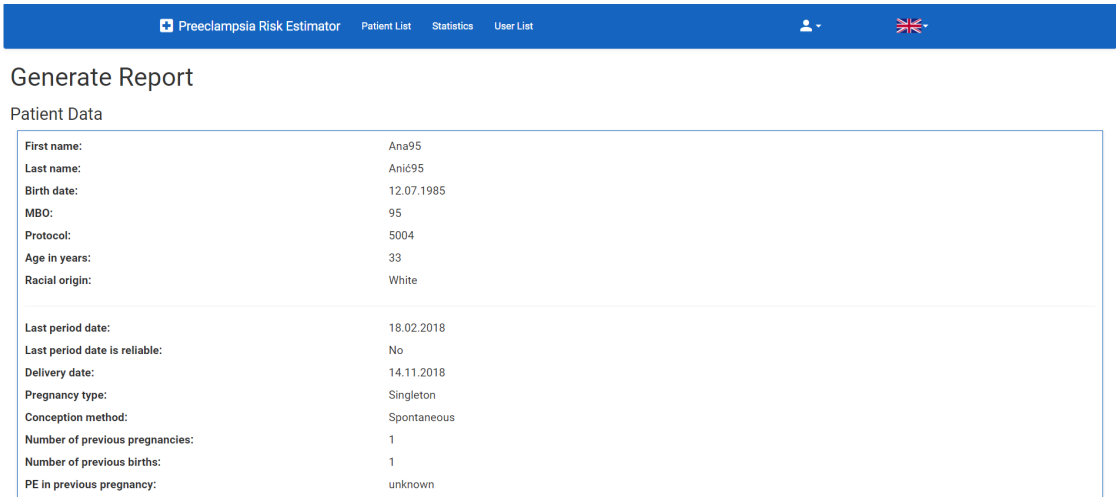

Figure 4.21: Generate report

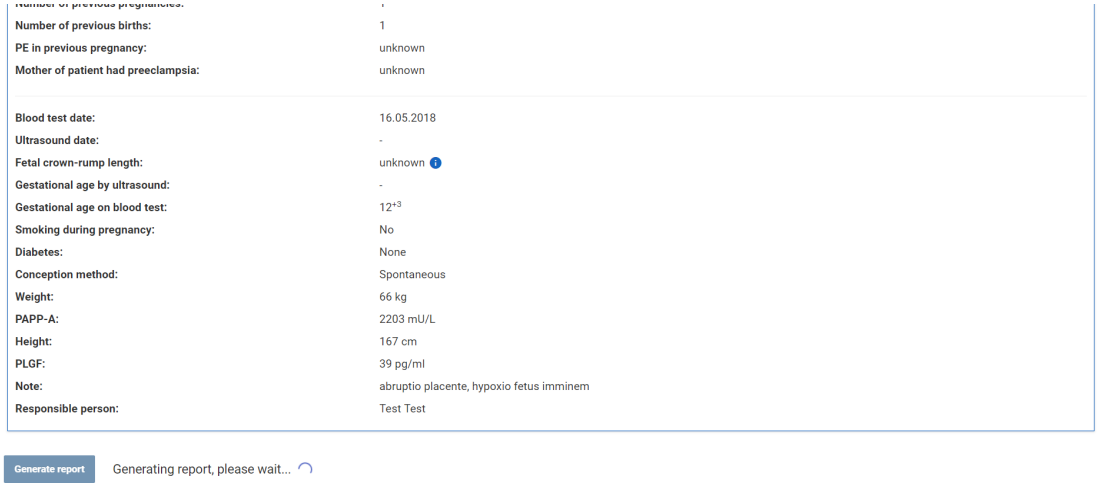

Figure 4.22: Generating the PDF report

Klinički bolnički centar Sestre milosrdnice/Klinika za onkologiju i nuklearnu medicinu
Endokrinološki laboratorij
Vinogradska cesta 29, Zagreb; 01 3787 163; www.endolab.kbcsm.hr

Preeclampsia Screening Results

PATIENT

Fullname: **Ana4 Anić4**
Birth date: 18.05.1982
Patient ID: 4
Protocol: 5001

Blood test date: 13.06.2018
Age on blood test: 36
Height: 161 cm
Weight: 69 kg
Racial origin: white

PREGNANCY

Gynecologist: -
Pregnancy type: singleton
Conception method: spontaneous

PE in previous pregnancy: unknown
Smoking during pregnancy: no
Diabetes: none

BIOCHEMYCAL MEASUREMENTS

PLGF: 60 pg/ml
PAPP-A: 2229 mU/L

Adjusted MoM: 1.40
Adjusted MoM: 0.78

ULTRASOUND DATA

Ultrasound date: 13.06.2018
Measured by: -

Fetal crown-rump length: 62 mm
Gestational age on ultrasound: 12^{+5}
Gestational age on blood test: 12^{+5}

RISKS ON THE BLOOD TEST DATE

<1:123
low risk of preeclampsia

COMMENT

salazoparyn, lupocet, folacin

Report date: 24.06.2019

Person responsible: Test Test

Figure 4.23: The PDF report example

– **/statistics**

The Statistics scene displays line charts for each biochemical measurement with medians by every week of pregnancy.

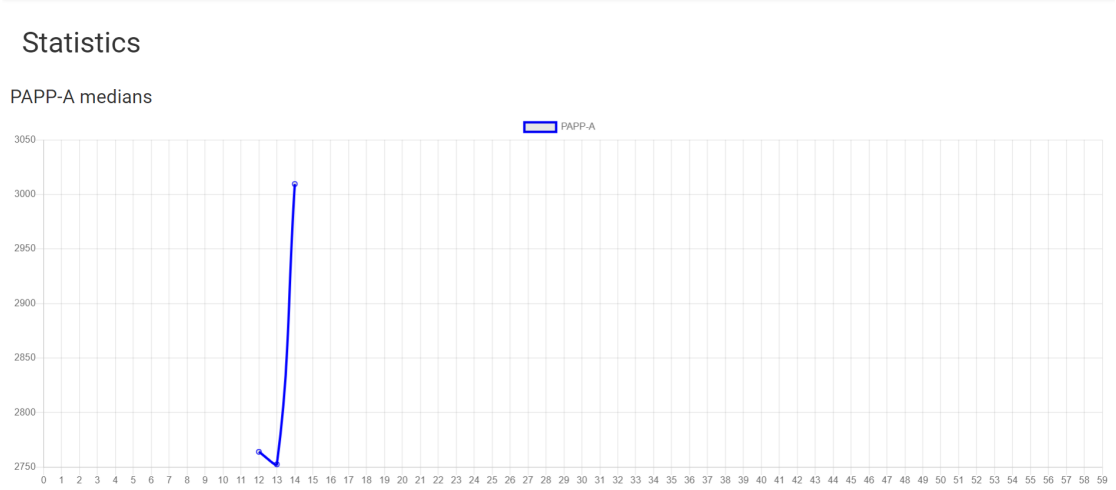

Figure 4.24: Line chart for PLGF medians (consists data for the first trimester)

5. Conclusion

Preeclampsia is a serious condition. Much research has been done to find out the causes of preeclampsia, but its pathogenesis still remains unknown.

Even though we lack knowledge of its causes, what can be done is to improve the detection of preeclampsia, so it can be treated as early as possible. Preeclampsia is a rapidly progressive disease and, if left untreat, can have fatal consequences for both the mother and child. Early identification of high-risk groups is crucial for reducing the severity of preeclampsia.

Therefore, preeclampsia is routinely screened for during prenatal care. The traditional approach to screening uses the maternal demographic characteristics and medical history to identify high-risk groups. The research done by the Fetal Medicine Foundation suggests an alternative approach to early preeclampia screening. The developed method is used to calculate a patient's risk of developing preeclampsia in the first trimester (11-13 week). This method combines the prior risk from maternal factors with results of biochemical and biophysical measurement using modern statistical methods, and provides a more precise risk calculation.

Within this thesis, a web application for managing patients' pregnancy data has been developed. The application consists of a risk estimation model used to calculate the risk of preeclampsia for the first trimester of pregnancy. The model was inspired by the method implemented by the Fetal Medicine Foundation. Some extra features for the application were implemented, such as generating medical reports for patients, authentication, role-based authorization, and visualization of statistical data related to biochemical measurements.

The machine learning algorithms, one of which was used in the risk calculation model, often require a large amount of data to provide a precise estimation. This application enables the collection of data needed for training the algorithm, so it will be more precise with time. The application also enables easy improvement and change of the model in the future.

BIBLIOGRAPHY

[1] Wikipedia – The Free Encyclopedia: Pre-eclampsia. https://en.wikipedia.org/wiki/Pre-eclampsia, . Date of access: 18 Jun 2019.

[2] O'Gorman, N., Wright, D., Poon, L.C., Rolnik, D.L., Syngelaki, A., de Alvarado, M., Carbone I.F., Dutemeyer, V., Fiolna, M., Frick, A., Karagiotis, N., Mastrodima, S., de Paco Matallana, C., Papaioannou, G., Pazos, A., Plasencia, W., and Nicolaides, K.H. Multicenter screening for pre-eclampsia by maternal factors and biomarkers at 11-13 weeks' gestation: comparison with NICE guidelines and ACOG recommendations. *Ultrasound Obstet Gynecol*, 32(3):171–178, 2012.

[3] The Preeclampsia Foundation – About Preeclampsia. https://www.preeclampsia.org/health-information/about-preeclampsia, . Date of access: 18 Jun 2019.

[4] Wright, D., Akolekar, R., Syngelaki, A., Poon, L.C., Carbone I.F., and Nicolaides, K.H. A competing risks model in early screening for preeclampsia. *Fetal Diagn Ther*, 49(6):756–760, 2017.

[5] Wikipedia – The Free Encyclopedia: Linear regression. https://en.wikipedia.org/wiki/Linear_regression. Date of access: 20 Jun 2019.

[6] Wikipedia – The Free Encyclopedia: Bayesian linear regression. https://en.wikipedia.org/wiki/Bayesian_linear_regression. Date of access: 20 Jun 2019.

[7] Koehrsen, W. Introduction to Bayesian Linear Regression. https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7. Date of creation: 13 Apr 2018. Date of access: 20 Jun 2019.

[8] Wikipedia – The Free Encyclopedia: Normal distribution. https://en.wikipedia.org/wiki/Normal_distribution. Date of access: 20 Jun 2019.

[9] Wikipedia – The Free Encyclopedia: Microsoft Typescript. https://en.wikipedia.org/wiki/Microsoft_TypeScript, . Date of access: 20 Jun 2019.

[10] TypeScript Documentation. https://www.typescriptlang.org/docs/home.html, . Date of access: 20 Jun 2019.

[11] V8 Documentation. https://v8.dev/docs. Date of access: 20 Jun 2019.

[12] Wikipedia – The Free Encyclopedia: Node.js. https://en.wikipedia.org/wiki/Node.js, . Date of access: 20 Jun 2019.

[13] Node.js – About Node.js. https://nodejs.org/en/about, . Date of access: 20 Jun 2019.

[14] Wikipedia – The Free Encyclopedia: npm (software). https://en.wikipedia.org/wiki/Npm_(software). Date of access: 20 Jun 2019.

[15] Express guide – Routing. https://expressjs.com/en/guide/routing.html, . Date of access: 20 Jun 2019.

[16] Express guide – Writing middleware for use in Express apps. https://expressjs.com/en/guide/writing-middleware.html, . Date of access: 20 Jun 2019.

[17] Node.js documentation – Child Process. https://nodejs.org/api/child_process.html, . Date of access: 20 Jun 2019.

[18] Sequelize Documentation. http://docs.sequelizejs.com. Date of access: 20 Jun 2019.

[19] npm – pg. https://www.npmjs.com/package/pg. Date of access: 20 Jun 2019.

[20] Passport Documentation. http://www.passportjs.org/docs, . Date of access: 20 Jun 2019.

[21] Passport Packages – passport-local. http://www.passportjs.org/packages/passport-local, . Date of access: 20 Jun 2019.

[22] Passport Packages – passport-jwt. http://www.passportjs.org/packages/passport-jwt, . Date of access: 20 Jun 2019.

[23] JSON Web Tokens. https://jwt.io. Date of access: 20 Jun 2019.

[24] npm – jsonwebtoken. https://www.npmjs.com/package/jsonwebtoken. Date of access: 21 Jun 2019.

[25] Handlebars Documentation. https://handlebarsjs.com. Date of access: 20 Jun 2019.

[26] npm – puppeteer. https://www.npmjs.com/package/puppeteer. Date of access: 20 Jun 2019.

[27] Wikipedia – The Free Encyclopedia: React (Javascript library). https://en.wikipedia.org/wiki/React_(JavaScript_library), . Date of access: 20 Jun 2019.

[28] React Documentation. https://reactjs.org, . Date of access: 20 Jun 2019.

[29] React Documentation – Virtual DOM and Internals. https://reactjs.org/docs/faq-internals.html, . Date of access: 20 Jun 2019.

[30] Redux Documentation –Three Principles. https://redux.js.org/introduction/three-principles. Date of access: 20 Jun 2019.

[31] Bootstrap – Documentation. https://getbootstrap.com/docs/4.3/getting-started/introduction. Date of access: 20 Jun 2019.

[32] React Bootstrap – Introduction. https://react-bootstrap.github.io/getting-started/introduction, . Date of access: 23 Jun 2019.

[33] Wikipedia – The Free Encyclopedia: Sass (stylesheet language). https://en.wikipedia.org/wiki/Sass_(stylesheet_language). Date of access: 23 Jun 2019.

[34] npm – bcrypt. https://www.npmjs.com/package/bcrypt. Date of access: 21 Jun 2019.

Rani probir za otkrivanje preeklampsije

Sažetak

Preeklampsija je poremećaj koji zahvaća 2-8% trudnoća i može imati fatalne posljedice za majku i dijete. Ranim probirom za preeklampsiju može se identificirati visokorizične skupine i započeti tretman lijekovima. Najnovija istraživanja pokazuju kako se korištenjem postojećih statističkih metoda i algoritama strojnog učenja može poboljšati procjena rizika za preeklampsiju. Korištenjem modernih metoda, rizik se može procjeniti već u prvom tromjesečju trudnoće. Napravljena je web aplikacija za upravljanje podacima o trudnoćama pacijenata koja omogućuje procjenu rizika za preeklampsiju i generiranje nalaza.

Ključne riječi: preeklampsija, probir, trudnoća, procjena rizika, web aplikacija

Early Screening for Preeclampsia

Abstract

Preeclampsia is a disorder which affects 2-8% of pregnancies and can have fatal consequences for both the mother and child. Early screening can identify high-risk groups so that medication can be started. Latest research has shown that using known statistical methods and machine learning algorithms can help with preeclampsia risk estimation. Using modern methods, the risk can be estimated as early as the first trimester of pregnancy. A web application for managing patient pregnancy data has been developed, which enables preeclampsia risk estimation and generating medical reports for that estimation.

Keywords: preeclampsia, screening, pregnancy, risk estimation, web application