Technical Description of the Web Application ImmediAID

Techchallenge Summer Semester 2019

Lightweight Service-oriented Web Application
Improving Situation in Emergency Rooms
A Product for the Rechts der Isar Hospital

Coached by	Lysander Homm Master Student Management & Technology
Team Improvemed	Anika Hylla Master Student Management & Technology
	Fiona Burckhardt Master Student Management & Technology
	David Mildenberger Master Student Computer Science
Repository Link	https://github.com/dave4422/immediaid
Date	Munich, the 10th of July

Contents

1. Introduction

1.1. Idea

Emergency rooms (ERs) are overcrowded. 70% of ER patients are non-emergencies and could be treated by any other doctor. Moreover, the majority of ER patients does not know about coexisting primary care options. Our product ImmediAID solves these problems.

Our main idea is to provide realistic information to non-emergency patients for how long they most likely have to wait when coming to the emergency room with a non emergency problem. We claim that confronting patients directly with their waiting time in the emergency room will discourage a large share of the 70% to go there in first place. This in result will lead to higher capacities and shorter waiting times for real emergencies and will save the hospitals money (since emergency room patients do have a negative profitability).
To be successful in convincing people no to come to the ER, during research and with surveys, we have identified three major requirements for our product.

1. The calculated waiting time needs to be as realistic and accurate as possible so that people put trust into the calculated number
2. The product must provide alternatives which are more attractive than going to the ER
3. The product must be placed so that as many potential ER patients as possible will see and consequently use it

1.2. Features

The following enumeration provides an overview over the implemented features of the product:

1. Non-Functional:
    - The application is a lightweight full-stack web application that runs on any mobile or desktop device
    - The application uses a microservice architecture to representation business capabilities and application logic
    - The application has easy maintainability
    - The application is flexible in using technologies and makes use of major web technologies: Angular with HTML and Javascript for frontend and Flask, SciPy, SQL light and Python for backend.

- Representational State Transfer (Restful) API for communication between services

2. Functional:

   - Categorizing patients into priority classes following the exact same procedure implemented and used by our customer, the hospital Klinikum rechts der Isar
     - Easy and fast diagnosis to diagnose patient into one of 50 different diagnosises from the Manchester-Triage-System
     - Determination of the priority of patients' treatments based on the severity of their condition by applying the Manchester-Triage-System
   - Calculation of waiting time estimates based on the factors: diagnosis, priority of patient, day, time of day, current number of patients in the ER
   - Presentation of recommendation for alternatives to the ER
   - Mechanism to transfer the collected data to medical professionals in a secure and privacy preserving way

1.3. Overview

The application is structured into four different services.

1.4. Technologies

Our product is a full stack application.

1.4.1. Front-end

The front-end was written in Typescript using Anular, Angular is a TypeScript-based open-source web application framework and Very common for modern web applications. Furthermore HTM, CSS and Javascript were used.

1.4.2. Back-end

The back-end services are independent web servers implemented in Python with the Flask framework. Their logic is based on SQLite and Scientific Python.

1.5. Repository

Our product repository can be found under: https://github.com/dave4422/immediaid

2. Instructions

2.1. Installation

We recommend to download and install the npm (short for Node Package Manager) packet manager first. This tool is a package manager for the JavaScript programming language and required later during installation. The npm packet manager can be installed with Node.js. The Node.js installer can be obtained from https://nodejs.org/en/download/
Furthermore we also require pip, the Python Package Installer. Pip can be obtained from https://pip.pypa.io/en/stable/reference/pip_install/
Please also download and install Pyhton 3 from https://www.python.org/downloads/

2.1.1. Angular

Check if npm was installed:

```
$ npm --version
```

Install Angluar CLI via npm:

```
$ npm install -g @angular/cli
```

Install Required Angluar packets:

```
$ npm install ngx-qrcode2 --save
$ npm install --save @angular/material @angular/cdk @angular/animations
```

2.1.2. Flask

Install Flask:

```
$ pip install Flask
```

2.1.3. Scientific Python

Install Scientific Python (SciPy):

```
$ python -m pip install --user numpy scipy matplotlib
ipython jupyter pandas sympy nose
```

2.1.4. SQLite

First download sqlite https://www.sqlite.org/download.html and change to download directory. Version names may differ! Install SQLite Database:

```
$ tar xvfz sqlite-autoconf-3071502.tar.gz
$ cd sqlite-autoconf-307150b
```

```
  $  ./configure --prefix=/usr/local
  $  make
  $  make install
```

2.2. Setup

You find the two backend services that need to be separately deployed inside the sub directories: /immediaid/time_service and /immediaid/diagnosis_service

2.2.1. Time Service

Start the Time Service:

```
  $  cd time_service
  $  export FLASK_APP=webservice_time.py
  $  flask run --port 5001
   * Serving Flask app "webservice_time.py"
   * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
   * Debug mode: off
```

This service will first display all training data for the machine learning algorithm and then continues to display the http requests.

2.2.2. Diagnosis Service

Start the Diagnosis Service:

```
  $  cd diagnosis_service
  $  export FLASK_APP=diagnosis.py
  $  flask run --port 5002
```

It is important that the SQLite data base is inside the same directory as the webserver.

2.2.3. Main Service

Now that we have started the back end service, we can start the front end service. Start the Main Service:

```
  $  cd frontend_service
  $  ng serve
 10% building 3/3 modules 0 active wds: Project is
 running at http://localhost:4200/webpack-dev-server/
wds: webpack output is served from /
wds: 404s will fallback to //index.html
```

```
chunk \{main\} main.js, main.js.map (main) 223
kB [initial] [rendered]
chunk \{polyfills\} polyfills.js, polyfills.js.
map (polyfills) 251 kB [initial] [rendered]
chunk \{runtime\} runtime.js, runtime.js.map (runtime)
6.09 kB [entry] [rendered]
chunk \{styles\} styles.js, styles.js.map (styles)
348 kB [initial] [rendered]
chunk \{vendor\} vendor.js, vendor.js.map (vendor)
7.11 MB [initial] [rendered]
Date: 2019-07-10T19:11:52.167Z - Hash:
98d5b0f617ed0ff6113b - Time: 9509ms
** Angular Live Development Server is listening on
localhost:4200, open your browser on http://localhost:4200/ **
 wdm: Compiled successfully.
```

2.3. Usage

The web application is served on http://localhost:4200/
The use is, simply type http://localhost:4200/ into the address bar of your preferred web browser.

3. Architecture

The application is based on a microservice architecture. Microservices are a software development technique that structures an application as a collection of loosely coupled services. Each of the services perform one specific task. This architecture has several advantages of single service architectures: easy maintainability, fast deployment of single services, flexibility in using technologies, scalability, representation of business capabilities and application logic in the architecture

The description of this section can be found in the repository under /documentation/architecture.

4. Services and Components

4.1. The diagnosis service

This service is a backend service. This service is implemented in the file /immediaid/diagnosis_service/diagnosis_service.py The purpose of this service is to find the correct diagnosis from the 50 diagnosises of the Manchester-Triage-System. The input to this service is general information like gender, date of birth etc. and the location of problem in the body.

4.1.1. How does the algorithm of the service work?

The service uses a SQL database to find the correct diagnosis. This database contains a table of all 50 diagnosis and characteristics of these diagnosises. The process is as follows:

1. The service receives a request containing the information which is retrieved in the front end service from all patients: gender, date of birth, location etc.
2. The service queries the database using the retrieved information
3. If the request returns just one diagnosis, the service returns this diagnosis and signalizes that the process is finished
4. If the request returns more than one diagnosis, the service queries the database again for the characteristics that uniquely define the remaining diagnosises
5. The service then returns these characteristics, signaling that it needs more information for a definite diagnosis. The front end service then has to ask the user for these further information. Example: When given the location of the abdomen: there are three possible diagnosises: Abdominal pain for children, Abdominal pain for Adults and Diarrhea and Vomit. Therefore the service sends the question to the front end service if the patient has diarrhea or vomits.
6. Since the service follows the restful principle, it is stateless. If the front end service queries again with further information is does the exact process again.
7. If there are no diagnosis that match the data, the service responds with an error code for undefined diagnosis

4.1.2. How is the service implemented?

The service is an independent web service which listens on http://127.0.0.1:5002/diagnosis for incoming requests. The web server implementation uses the flask frame work. Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries Flask is ideal to implement lightweight webservices as it has a very low complexity. The database is a SQLite database.

4.1.3. What is the interface of the service?

The service communicates over a RESTful interface via Http messages. The payload of the messages follows the json format. The schema for incomming requests is as follows:

```
{
   "$schema":"http://json-schema.org/schema#",
   "title":"Diagnosis",
   "description":"Data to find correct diagnosis",
   "type":"object",
   "properties":{
      "id":{
         "type":"number",
         "description":"Product identifier"
      },
      "key":{
         "type":"string",
         "description":"To identify further"
      },
      "sex":{
         "type":"string",
         "enum":[
            "male",
            "female",
            "other"
         ]
      },
      "age":{
         "type":"string",
         "format":"date"
      },
      "location":{
         "type":"string",
         "enum":[
            "none",
            "all",
            "psychological",
            "sexorgans",
            "breast",
            "mouth",
            "eyes",
            "resthead",
            "neck",
            "armleft",
            "armright",
            "legleft",
            "legright",
            "footleft",
            "footright",
            "abdominal"
         ]
      },
      "questions":{
         "type":"array",
         "items":{
            "type":"string",
            "enum":[
               "question1",
               "question2",
               "question3",
               "question4",
               "question5",
               "question6",
               "question7",
               "question8",
               "question9",
            ]
         }
     }
   }
}
```

```
{
   "$schema":"http://json-schema.org/schema#",
   "title":"DiagnosisResults",
   "description":"correct diagnosis or further questions",
   "type":"object",
   "properties":{
      "id":{
         "type":"number",
         "description":"Product identifier"
      },
      "error_code":{
         "type":"number",
         "description":"0 ok 1 ambigiutiy 2 no match"
      },
      "result":{
            "type":"string",
            "description":"result of diagnosis, empty if not
                unambiguous"
        },
        "questions":{
            "type":"array",
            "items":{
                "type":"string",
                "enum":[
                    "question1",
                    "question2",
                    "question3",
                    "question4",
                    "question5",
                    "question6",
                    "question7",
                    "question8",
                    "question9",
                ]
            }
        }
    }
}
```

4.2. The time calculation service

This service is a backend service. It is implemented in the file /immediaid/time_service/time_-service.py The purpose of the time calculation service is to provide an accurate estimation of the waiting time in the ER given the following parameters:

- The diagnosis of the patient
- The day and time the patient wants to go to the ER (usually the current day and time)
- The priority of the patient in the Manchester Triage System
- The current load factor of the ER (How much capacities are used at the moment)

4.2.1. How does the algorithm of the service work?

The service uses a machine learning approach. As training data we use historical data from the hospital (this data is mocked at the moment because we did not receive it from the hospital, nevertheless we want to mention that it is available and can be easily obtained from their triage system ER path.)

The training nx5 matrix where n is the number of data points.
One single data point has the following format: [a,b,c,d,e,f] = [priority of the patient,diagnosis, day, time of day in seconds, load factor, waiting time].

- The priority is in the format $a \in \mathbb{N}$ (1=red),...,(5=blue)
- The diagnosis is in the format $b \in \mathbb{N}^{+}_{50}$ (y being the yth diagnosis in the Manchester-Triage-System, see attachment)
- The day is in the format $c \in \mathbb{N}^{+}_{7}$ (1=Monday),...,(7=Sunday)
- The time is in the format $d \in \mathbb{N}^{+}_{86400}$ which are the number of seconds already passed in one day
- The current load factor is in the format $e \in \mathbb{N}_{100}$ (0=0% load),...,(100=100% load)
- The waiting time is in the format $f \in \mathbb{N}$, the number of seconds the patient waited in total in the ER until treatment

Our algorithm is a modified k-nearest neighbor algorithm. The process is as follows:

1. Setup: We cluster our training data into 5 data sets, one set for each priority. This is done because the priority of the patient is the most important factor for his waiting time. We only use historic waiting times in our calculations that have the same priority as the patient for which we do the calculation
2. Setup: We train a k-nearest neighbor algorithm with our training data and weight the features (the idea behind this is that some features like the current load factor are more important for the waiting time than others like for example the diagnosis)
3. Time Calculation: When receiving a new input vector to calculate the waiting time we search for the 4 closest data points using our nearest neighbor model. These data points are other patients which resemble our current patient the most and for which we have historic waiting times. We then calculate the waiting time of our current patient with the following formula: 0.75* waiting time of nearest data point+0.15* waiting time of second nearest data point+0.08 * waiting time of third nearest data point+0.02* waiting time of fourth nearest data point

4.2.2. How is the service implemented?

The service is an independent web service which listens on http://127.0.0.1:5001/time for incoming requests. The web server implementation uses the flask frame work. The algorithm itself uses the Scientific Python library to train our model and predict the waiting time as described above.

4.2.3. What is the interface of the service?

The service communicates over a RESTful interface via Http messages.

Get Request for a time estimate:

```
GET /time/?category=5&diagnosis=5&day=4&time=49920&auslastung=50
Host: http://127.0.0.1:5001/
Content−Length: 100
```

The server then responds with a single floating point value which is the estimated time in seconds.

4.2.4. Data Problem

We conjecture the algorithm to work very well but to actually test the algorithm, as for all machine learning algorithms, we need training data which we did not get from the hospital yet.

4.3. main service

4.3.1. Manchester Triage System

4.4. The data transfer service

The idea of the data deployment service is to encode the data in a QR code. Then if the user decides to visit the ER anyways or wants to go to a normal practice, the user can bring all the data which was already entered an therefore saves time and simplifies the processes. Using a QR code has the following advantages:

1. The data never leaves the local computer of the user, i.e. the data is never transported over the internet. Therefore it can not be stolen by criminals on the internet.
2. The user remains in full control over the data, the user has solely the decision-making power who he wants to show the collected data
3. An QR code is a very common wide-spread protocol, it is easy to connect this method to other system's interfaces like for example the systems used in the hospital

4.4.1. How is the service implemented?

The service is based on the library ngx-qrcode2. ngx-qrcode2 is an Angular 6 Component library for Generating QR (Quick Response) Codes. The format of the data follows the same json format as the diagnosis service but also includes all questions from the Manchester Triage. To make the codes smaller, all key values are ommited.

5. Outlook

5.1. Deployment of the product

Currently the product only runs locally as prototype. In a next step the product has to be deployed the the public internet. Therefore the rental of a sufficient hosting space is necessary.

5.2. What still needs to be implemented

1. The selection of location during diagnosis in the front-end service only works with specific screen sizes and resolutions, this was the reason why our product did not fully work during the demo (unsupported resolution of the beamer). This needs to be fixed urgently.
2. The product is highly depended on data, we need to train our models with real data that we still need to obtain from the hospital
3. An interface for the qr code format needs to be specified
4. For the entire product test cases need to be implemented
5. Not all 50 diagnosis of the Manchester-Triage-System have been fully digitized

5.3. Scope of work

To who it may concern: Please consider that this project and product was realized by a small team of three (two members with an economics background and one engineer) and the tremendous workload we put into the development of the project.

6. Contact and Troubleshoot

For technical inquiries, our team can be contacted at: david.mildenberger@tum.de