Patienteninformationssystem
Diploma Thesis

Fröhlich Alexander, Kogler Paul, Weber Marina

Supervisor

DI Dr. Reinhold Mayer

in cooperation with

BOOM Software AG

Contact Person

Daniel Peinhopf

Secondary Technical College of Kaindorf

Department of Information Technology

April 2019

HTBLA Kaindorf an der Sulm

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Alexander Fröhlich

Paul Kogler

Marina Weber

Kaindorf an der Sulm, 05 April 2019

Acknowledgements

We would like to thank our teachers, family and classmates who supported us in various ways whenever necessary.

Furthermore, we would like to thank Dr. Wolfram Heidinger, our school's doctor, and Mrs. Renate Weber, who provided us with the necessary information about medical conditions and their symptoms. Also, we want to thank Daniel Peinhopf, who accompanied us during the development process and introduced us to various new technologies.

Moreover, we would like to thank Dr. DI Reinhold Mayer, our teacher and supervisor of this project, for introducing us to the topic and for his remarkable support during the entire project.

Abstract

Austria's hospitals face a problem: Countless patients with an easily treatable illness or injury are admitted into their rooms every day. ER rooms in particular often face a shortage of doctors in late hours, since quite a lot of people decide not to go to a hospital when feeling unwell, however, during the night, their symptoms only worsen. On the other hand, there are doctor's offices that are scarcely visited.

Since many times, the patients are clueless about how severe their illnesses are, they go to a hospital to be sure. For such situations, we decided to implement a program that can predict which kind of doctor is most suitable for them.

The result of this diploma thesis is a prototype of an application that a potential patient can use to make a rough diagnosis themselves. It enables patients to diagnose themselves by answering questions based on their symptoms. Based on this diagnosis, the system recommends doctors that are able to treat the illness and displays the route to the nearest one.

This prototype enables people with little knowledge about medicine to correctly identify the urgency of their health problems and therefore reduces the number of people admitting themselves into the hospital.

German version of the abstract

In österreichischen Spitälern werden häufig Patienten mit leicht heilbaren Krankheiten oder Verletzungen aufgenommen. Besonders in den Notfallaufnahmen stellt dies ein Problem in der Nacht dar, da zu diesen Zeiten die Krankenhäuser spärlicher besetzt sind. Doch genau zu dieser späten Zeit suchen Leute ein Krankenhaus auf, da sich die Schmerzen in der Nacht oft verschlimmern und sie diese nun nicht mehr ertragen können.

In vielen Fällen ist den potentiellen Patienten die Schwere ihrer Krankheit nicht bewusst, was dazu führt, dass sie zur Sicherheit lieber ein Krankenhaus als eine Arztpraxis aufsuchen. Genau aus diesen Gründen haben wir uns dazu entschieden, ein Programm zu entwickeln, welches einschätzen kann, welche medizinische Institution für bestimmte Patienten am besten geeignet ist.

Das Ergebnis dieser Diplomarbeit ist ein Prototyp einer Applikation, welche potentielle Patienten dazu verwenden können, sich selbst zu diagnostizieren. Der Patient kann Fragen zu seinen Symptomen beantworten und erhält eine vorläufige Diagnose. Mithilfe dieser Diagnose werden dem Patienten passende medizinische Einrichtungen empfohlen und die schnellste Route zu ihnen in einer Karte angezeigt.

Der Prototyp ermöglicht es Leuten mit nur geringen medizinischen Kenntnissen, die Schwere ihrer Krankheit korrekt einschätzen zu können. Dies führt dazu, dass weniger Menschen mit leicht zu behandelnden medizinischen Problemen ein Krankenhaus aufsuchen.

Contents

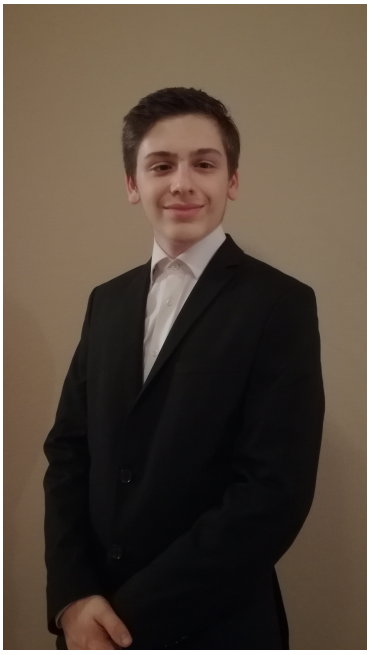
1 Project

1.1 Project team

The team consists of three students of the class 10DHIF of the HTBLA Kaindorf, which belongs to the Department of Information Technology.

Paul Kogler

Marina Weber



Alexander Fröhlich

The supervisor of this project, who is also our teacher, is

Dr. DI Reinhold Mayer

Our contact person of the company Boom Software AG is

Daniel Peinhopf

1.2 Customer

Figure 1.1: Official logo of Boom Software AG

BOOM Software AG was founded in 1995 by three software developers. They have been expanding ever since and are now counting around 70 employees. BOOM has two different locations, the main one being in Leibnitz, Austria. The second one is of a smaller size and is located in Lower Saxony. Working in various industries, BOOM has an international audience.

BOOM describe their concept as ”innovative and individualised software solutions for maintenance and production processes.” They belong to the leading companies in that field. BOOM uses the bald eagle and the owl as their mascots for their software solutions, the ”BOOM Maintenance Manager” and the ”BOOM Production Manager.” The birds are supposed to represent their formula to success: adaptability and individualisation. Sharp eyesight and excellent hunting skills were the main reason for choosing them (*Boom Inside* 2019).

The BMM (BOOM Maintenance Manager) allows a continuous optimisation and standardisation of all maintenance processes, while the BPM (BOOM Production Manager) is responsible for the quality assurance, documentation, and comprehensibility of production processes.

In order to complete their tasks more efficiently, they developed a framework called ”BORA” which stands for ”Business Oriented Rapid Adaption.” It consists of various modules that allow a total adaptation to the customer. It is independent of technologies and business requirements (*Boom Software AG* 2019).

1.3 Actual State Analysis

A shortage of doctors is a huge problem in Austria's hospitals. At night, this becomes an even bigger issue as there are not as many employees working in the hospital. Often, many patients admit themselves into the hospital, although their illness or injury could simply be treated by a general practitioner or another kind of medical institution. This problem is a consequence of the lack of awareness of how severe ones medical condition is. As a consequence, they choose the hospital over a more suitable medical institution.

On the other hand, some doctors' offices are scarcely visited and could treat more patients if required. Specialised doctors have great expertise in their field, which means they can treat patients as professionally as doctors working in a hospital.

1.4 Idea

The idea of this diploma thesis is to create a prototype of a desktop application that will help people with little knowledge of medicine estimate the severity of their health problems. In order to do this, our system asks the user which symptoms they have experienced so far and analyse them to create an approximate diagnosis.

With this diagnosis, the system searches for doctors that are specialised in the field the diagnosis can be categorised in. Moreover, the distance and time needed to reach the doctors are calculated and displayed along with a map that shows the route to the doctor. If the user chooses to see one of these doctors, they will be informed about a new arriving patient.

A doctor that is informed about a new patient can access the patient's data and the symptoms they stated to suffer from when answering the questions. Since the doctor already knows about the patient's problem before their arrival, the time needed for the examination can be reduced.

1.5 Project Management

1.5.1 Responsibilities

	Marina Weber	Paul Kogler	Alexander Fröhlich
Projectmanagement	V	I	I
Creating & maintaining the questionnaire about illnesses	V	I	I
Calculating & Displaying route	I	I	V
Matching suitable doctors with patients	V	M	I
Managing doctors' data	M	V	I
GUI Design	I	V	M
User Registration & Login	V	M	I
Displayment of diagnoses for doctors	I	V	M

Figure 1.2: IMV Matrix of the project

As everyone is involved in the developing process, our project team decided that we will divide the project into different parts and apply the IMV Matrix.

Marina Weber

Marina Weber is the project leader of the group and therefore responsible for the project management. Furthermore, she is responsible for gathering data about illnesses and implementing an algorithm that allows one to diagnose themselves.

Paul Kogler

Paul Kogler is mainly responsible for user management, regarding the administration of the patient's information. Additionally, he is in charge of filling and maintaining the database.

Alexander Fröhlich

Alexander Fröhlich's assignment is the development of the physical routing and navigation between a patient and doctor, taking into account driving conditions, means of transport, potential traffic jams and more.

Strategy

In order to organise and analyse our project, the method of ”Rational Unified Processes” is used.

The project is divided into seven use cases. These use cases are analysed entirely individually. The goal of analysing them is to identify every object that is described in the use cases. Once all the objects are correctly identified, all the attributes of the objects to store the necessary information for the use cases are defined. The last step consists of determining the behaviour and associations of the objects. After analysing them, each use case is implemented one by one by using the analysis as a template for the object-oriented design.

Milestones

Milestone	**Completed by**
Patient registration	September 2018
Creation of diagnoses	September 2018
Searching for suitable doctors	October 2018
Display route to doctors	December 2018
Adding new doctors	January 2019
Informing doctors	February 2019
Editing of doctors	March 2019

1.6 GDPR Note

Since this project is solely a prototype that does not use real people's data, it was not a priority to implement all the requirements set by the GDPR. However, since this project could be used and further developed in the future, a way to inform the user about our use of their data was implemented. When someone decides to sign up, they will be informed that we need to process data concerning their private information such as their name and their health-related issues as well as their current location.

Furthermore, in order to use a doctor's information, they need to fill out a printed form in which they declare that they allow us to use their data in our system. Only then they will be added into the system by an administrator.

2 Object Oriented Analysis

2.1 Use Case 1 - Register patient

Description

In order to use the system, it needs to be able to identify the user. For that matter, the system asks the user to type in their email address. Then, the user can choose a password and type that in as well. When the user submits the data, the system saves it. It then associates the user with the role “patient”, so when the user logs in, they can access the necessary services of the system.

Use Case Flow

Primary Actor	Patient
Precondition	The patient has started the application
Condition Success	The patient is successfully registered
Condition Failed	The patient cannot be registered
Initial Event	The system does not know the patient's data, and displays the form
Course	1. The patient opens the registration form 2. The patient enters an email address 3. The patient enters a password 4. The patient submits the data 5. The system encrypts the password so that it is not in clear text 6. The system saves the data 7. The system associates the role "patient" to the created user
Alternatives	2a. The email address is not valid, the patient is asked to enter a correct address 4a. The email address is not valid, the patient is asked to enter a correct address

OOA Class Diagram

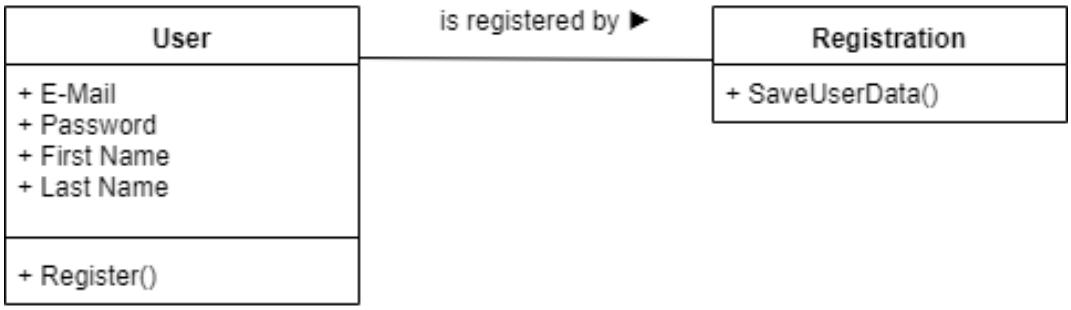


Figure 2.1: OOA class diagram of Use Case 1

2.2 Use Case 2 - Create a diagnosis

Description

The patient uses the system in order to get a provisional diagnosis. The system displays a questionnaire with multiple choice questions that the patient is asked to answer. After the patient answered all the questions, the system creates a diagnosis and associates it with a medical field, a so-called “speciality” of a doctor. Finally, the result is displayed for the patient.

Use Case Flow

Primary Actor	Patient
Precondition	The patient is logged in
Condition Success	The patient knows how severe their illness is
Condition Failed	The system fails to diagnose the illness based on the described symptoms
Initial Event	The patient has a medical problem
Course	1. The patient opens the questionnaire 2. The system displays a specific set of questions 3. The patient answers the questions based on their symptoms 4. The system searches for possible illnesses that are known to cause said symptoms 5. The system associates the illness with a speciality 6. The system displays the information
Alternatives	3a. The user aborts the questionnaire

OOA Class Diagram

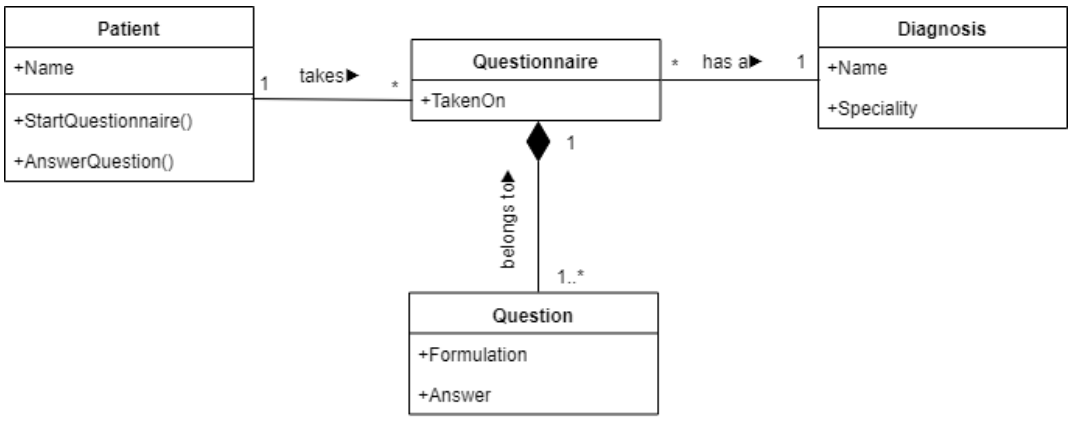


Figure 2.2: OOA class diagram for Use Case 2

2.3 Use Case 3 - Find suitable doctors

Description

After a diagnosis is created, the patient needs to know about which doctors can help them. Thus, the system searches for all doctors that can treat the illness based on their "speciality". For every doctor, the system calculates the time needed to reach them and the distance to the patient by using the patient's current address and the address of the doctor's office. The system then displays the available doctors that are suitable for the patient's illness. The patient then selects a doctor and confirm that they want to see them.

Use Case Flow

Primary Actor	Patient
Precondition	The patient has been diagnosed with an illness
Condition Success	The patient knows which doctor they need to go to
Condition Failed	The system cannot find a suitable doctor
Initial Event	The patient has submitted a questionnaire
Course	1. The system searches for the required speciality that is needed for the illness described by the patient 2. The system searches for doctors that have the required speciality 3. The time needed to reach the doctor is being calculated 4. The distance to the doctor is being calculated 5. The system sorts the doctors ascending by the time needed to reach them 6. The patient selects the doctor they want to go to 7. The patient confirms their selection
Alternatives	2a. The system cannot find a suitable doctor, an error message will be displayed

OOA Class Diagram

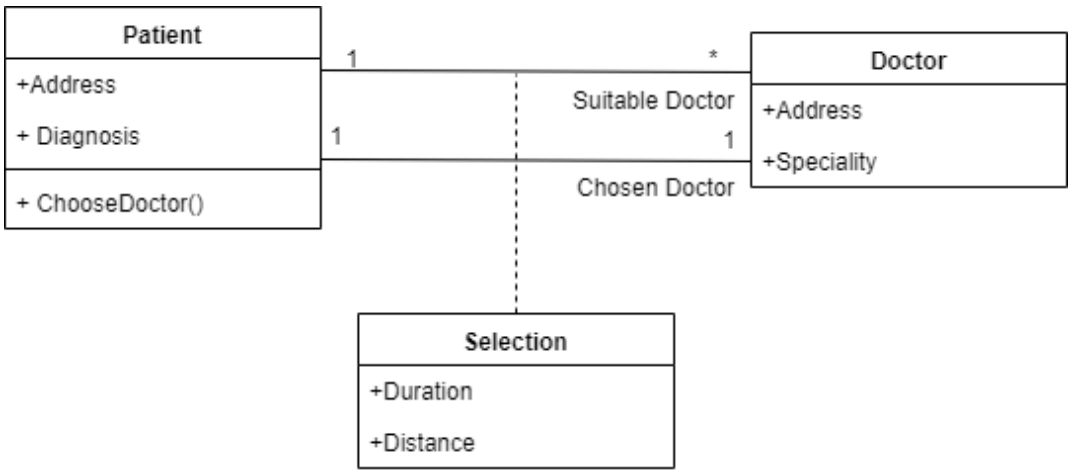


Figure 2.3: OOA class diagram for Use Case 3

2.4 Use Case 4 - Display the route

Description

To assist the patient in reaching a doctor, the system displays a route. The patient first needs to select their doctor of choice and their favoured means of transport. The system then draws a route from the current address of the patient to the doctor. Based on the means of transport and the traffic, the suggested route is adjusted.

Use Case Flow

Primary Actor	Patient
Precondition	The system is allowed to use the patient's location related data
Condition Success	The patient knows the physical location of the doctor, the route required to reach them, the duration of the route and the assumed arrival time
Condition Failed	The patient does not know the route to the suitable doctor
Initial Event	The patient has selected a doctor
Course	1. The system determines the current address of the patient 2. The system determines the address of the doctor 3. The user selects the means of transport of their choice 4. The system calculates the best possible route between the two locations 5. The system displays the best possible route, along with the time required to use the route and then the assumed arrival time
Alternatives	-

OOA Class Diagram

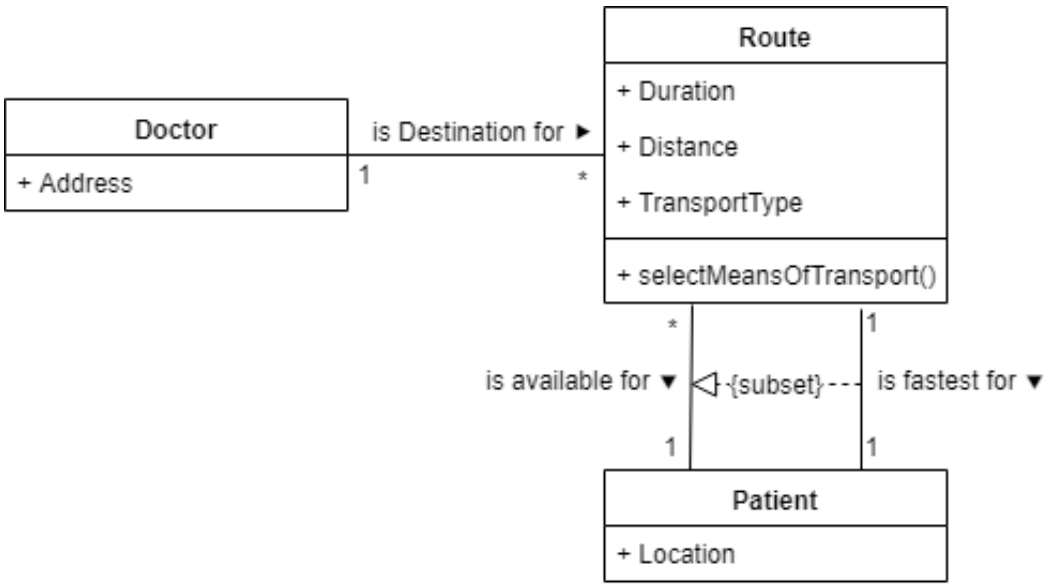


Figure 2.4: OOA class diagram for Use Case 4

2.5 Use Case 5 - Add a new doctor

Description

In order to make it feasible for the system to find a suitable doctor for the patient's problem, it is mandatory that the system is up-to-date with doctors of various specialities. The data of the doctors are managed by an administrator. In order to add a new doctor their speciality, the address of their office and their name are required.

Use Case Flow

Primary Actor	Administrator
Precondition	The administrator is logged in
Condition Success	A new doctor is inserted into the system
Condition Failed	The doctor cannot be inserted into the system
Initial Event	The patient has selected a doctor
Course	1. The admin types the doctor's name into the form 2. The admin types in the doctor's office's address 3. The admin associates the doctor with their speciality 4. The admin submits the data 5. The system stores the data
Alternatives	-

OOA Class Diagram

Figure 2.5: OOA class diagram for Use Case 5

2.6 Use Case 6 - Inform doctor

Description

As soon as the patient selects a doctor, the system informs the doctor that a new patient is on their way. When the doctor sees this information, they view the details about the patient, such as the diagnosis created by the system, their estimated arrival time and which symptoms the patient stated to suffer from in order to schedule their appointments better.

User Case Flow

Primary Actor	Doctor
Offstage Actor	Patient
Precondition	The patient has decided which doctor they are going to see
Condition Success	The doctor is informed about the arriving patient
Condition Failed	The doctor is not aware of the arriving patient
Initial Event	The patient selects a doctor
Course	1. The system retrieves the doctor's contact details 2. The system informs the doctor that a new patient is on their way 3. The doctor reads the information about the arrival time 4. The doctor opens the details about the patient 5. The system recalls the self-generated diagnosis and the symptoms of the patients as well as the estimated arrival time. 6. The system displays the gathered information about the patient to the doctor
Alternatives	-

OOA Class Diagram

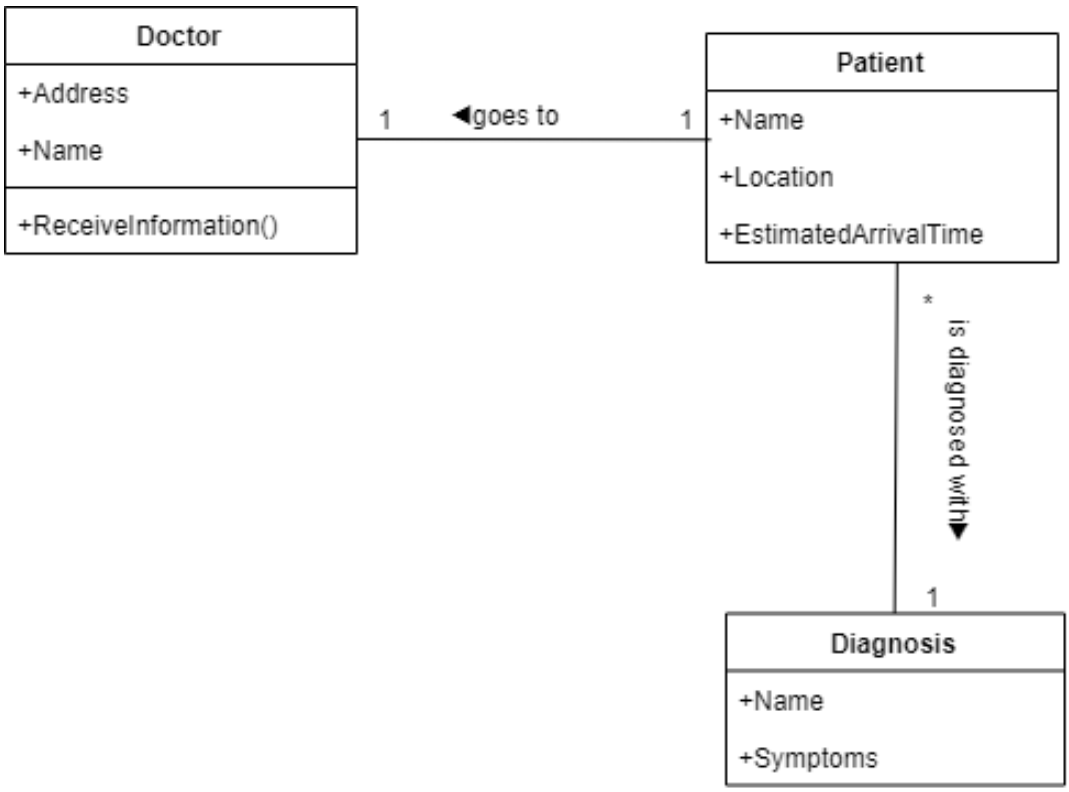


Figure 2.6: OOA class diagram for Use Case 6

2.7 Use Case 7 - Maintain the doctor's data

Description

The doctor's data, such as their address or specialities, may change. In this case, an admin has to alter their data so that it is up-to-date. The admin selects the doctor whose information changed. After that, the admin corrects the data to its current state and then submits it. After the submission, the system saves the new data.

Use Case Flow

Primary Actor	Admin
Precondition	The doctor whose information has changed already exists in the system
Condition Success	The data is updated
Condition Failed	The data cannot be updated
Initial Event	The admin has been asked to alter the doctor's data
Course	1. The admin opens the list of all doctors 2. The admin selects the doctor whose data needs to be altered 3. The system displays the doctor's current data 4. The admin alters the data 5. The admin submits the changes 6. The system saves the new data
Alternatives	-

OOA Class Diagram

Figure 2.7: OOA class diagram for Use Case 7

2.8 Project

Use Case Model

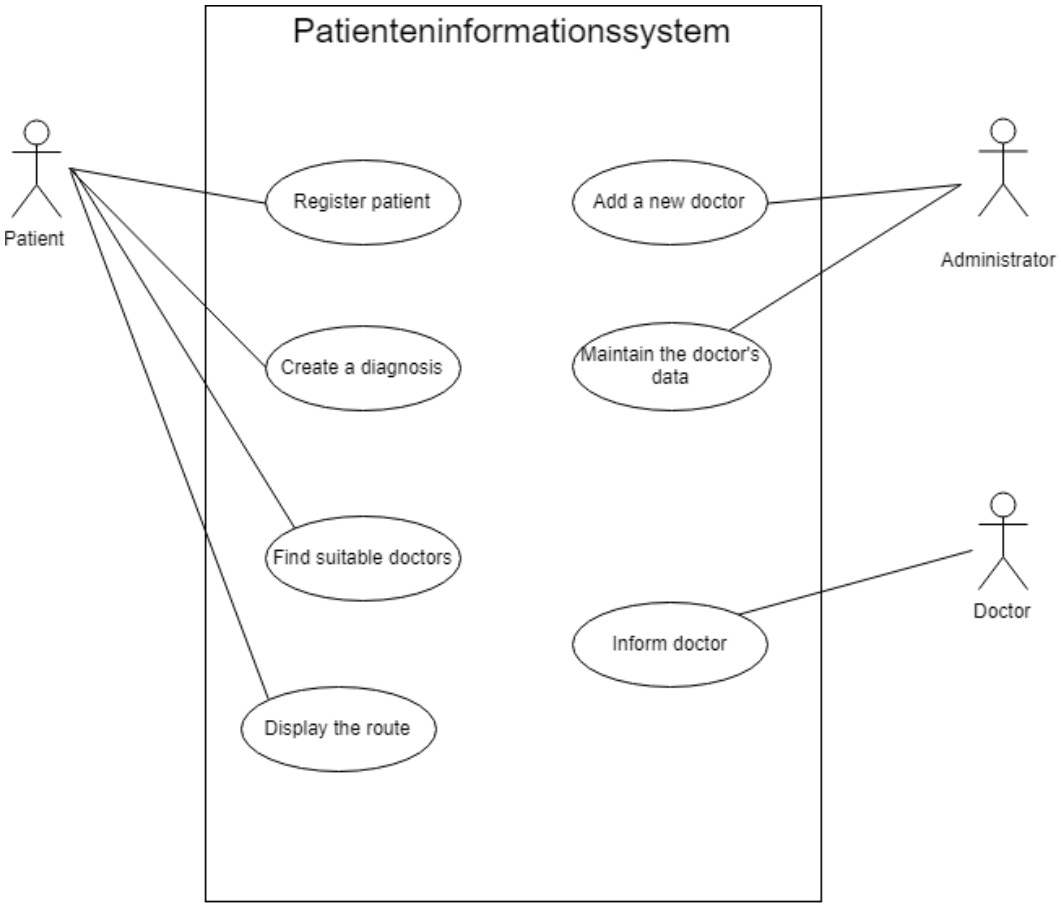


Figure 2.8: Use Case Model of our project

OOA Class Diagram

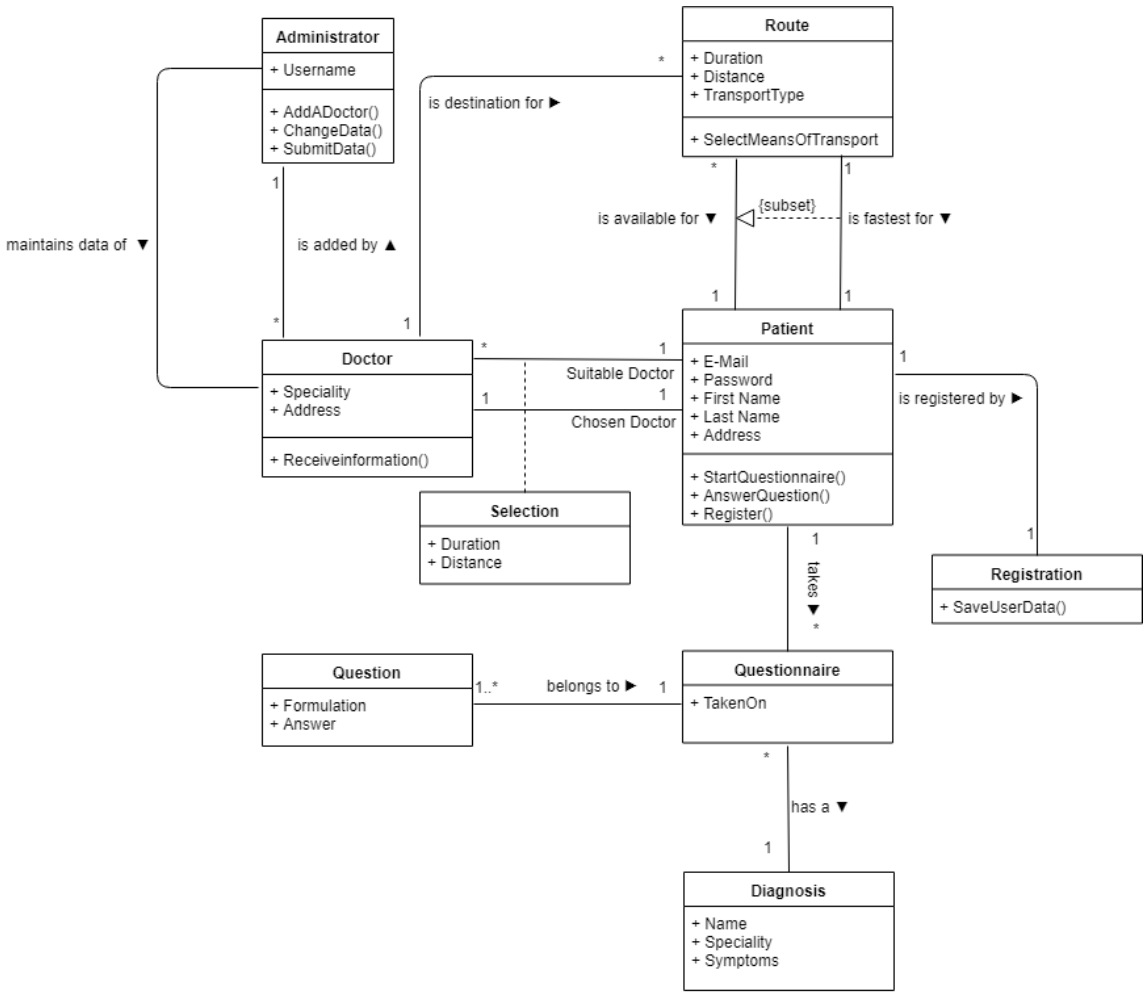


Figure 2.9: Use Case Model of our project

Activity Diagram

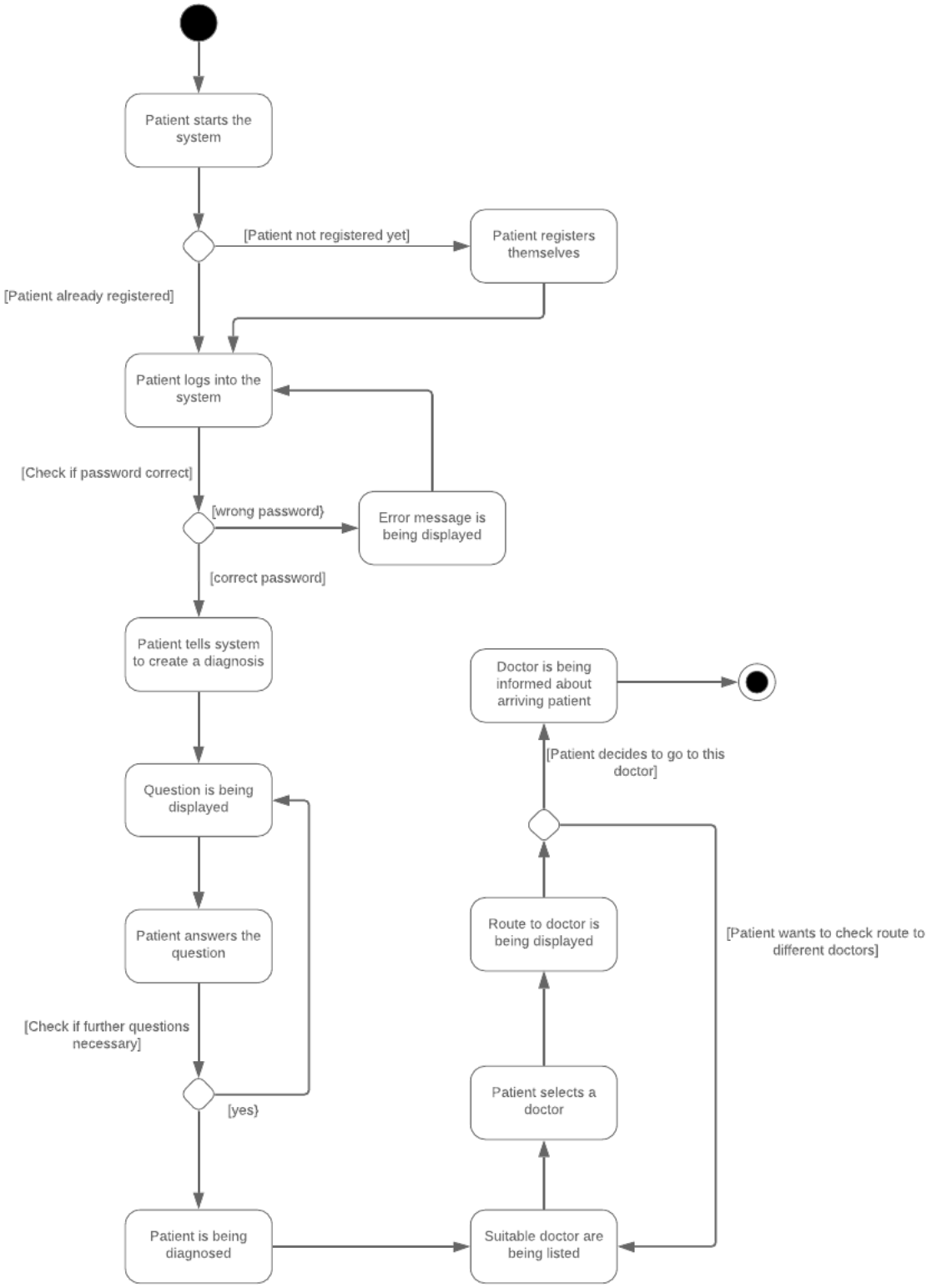


Figure 2.10: Activity diagram in point of view of the patient

These are all the activities of the system in the point of view of a patient. A patient first needs to be registered to use our system. If they are registered, they can log into the system. When they are successfully logged in, they can request the system to diagnose them. Once the diagnosis is created, doctors that can treat the problem are displayed. The user can select a doctor and view the route to it. If they decide to go to this doctor, the doctor is informed about the arriving patient.

Gantt Diagram

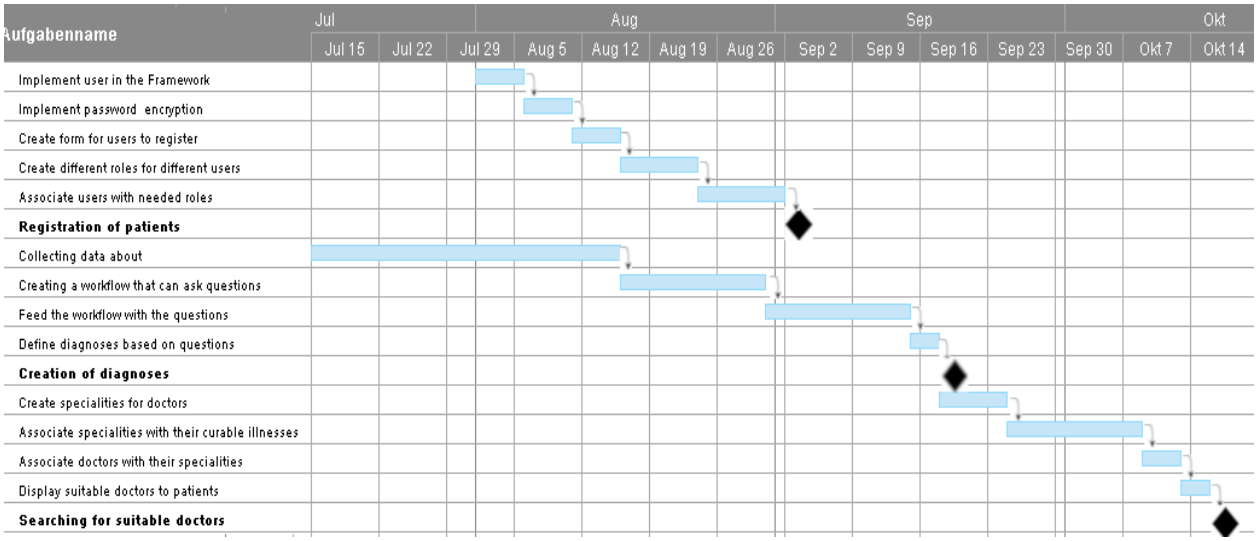


Figure 2.11: Part one of the Gantt diagram

This is the detailed Gantt diagram of the project. Since the time frame is relatively big, it has been divided into two parts.

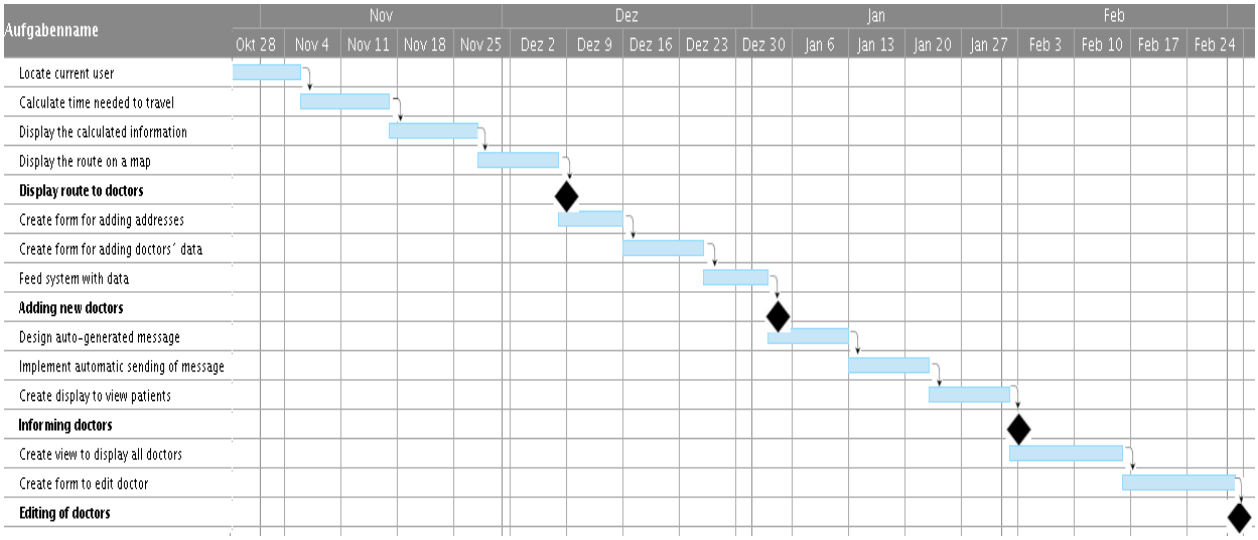


Figure 2.12: Part two of the Gantt diagram

3 Solution

3.1 Registering a patient

Before a patient can use any function provided by the application, they must register themselves. The patient's credentials are entered into a database so that the doctor can later be informed which patient is going to arrive soon. The patient's credentials consist of:

- e-mail address
- first name
- last name
- password

The BORA client already had a method of logging someone in, however, this was not sufficient as either the system automatically created an account or an administrator had to manually add the patient into the database before they could log in. Due to this reason, a custom-made login and registration form had to be created, in order to allow the patient to create an account themselves.

3.1.1 Database

The BORA framework already provides a predefined class for the patient. However, only the attribute for the e-mail (called Username) of the patient was already existent.

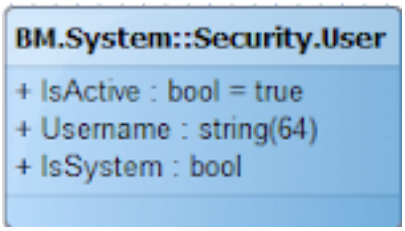


Figure 3.1: predefined class for the patient

In order to secure the patient's account with a password, two additional classes were implemented by the *User* class:

- IUserWithPassword
- IPasswordValidation

The BORA framework uses SHA1 for password encryption as standard.

If a new account is created, it is now mandatory that a password is entered. Due to the class *IPasswordValidation*, the password has to be entered a second time, to confirm that it was entered correctly.

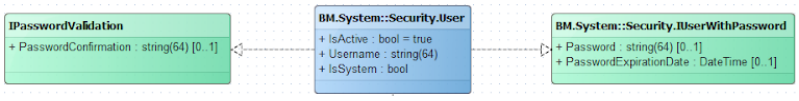


Figure 3.2: implemented classes for password encryption

Since a doctor should be alerted of the imminent arrival of a patient, it is beneficial to know the first- and surname of a patient. As a result, a new class (*IClientUserName*) was created in order to store the names.

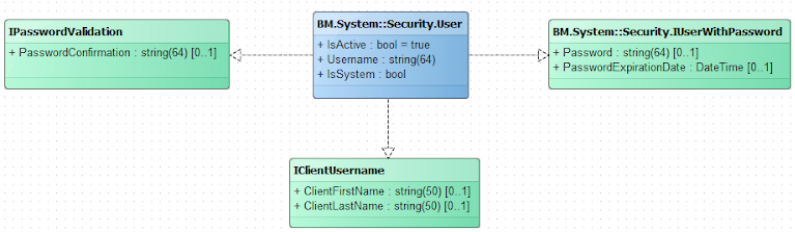


Figure 3.3: final diagram for storing patient information

With this final architecture each user is compelled to have a full name and a password next to his e-mail address. Since very personal and intimate information about the patient is managed by the application, password encryption is critical. As mentioned above, the patient's full name is used to help the doctor see which patients are en route to his facility.

3.1.2 Code

Login

After starting the application a dialog will appear where the patient can enter their login information and subsequently proceed with using the functions of the system. The login information consists of the e-mail address and password. The BORA framework provides the file *Gui.xml*. The entirety of custom made GUI-features are realized in this file. As it is in XML-format all commands are defined by tags.

In order to create the login dialog, an entity was created in Gui.xml and was given the name *LoginDialog*. Between the *entity*-tags, two features were defined, one for the e-mail address and one for the password.

```
<entity name="LoginDialog">
      <feature name="UserName">
            <style>
                  <title>E-Mail</title>
            </style>
      </feature>
      <feature name="UserPassword">
            <style>
                  <title>Passwort</title>
            </style>
      </feature>
</entity>
```

The feature-tags can be enhanced with a style, as shown in the figure above. Within the style the possibility is provided, to set a title for the feature. This title determines the content of the label which will be displayed in front of the input field where the information is entered.

To display the dialog, a form has to be created first. A form describes which features of an entity will be shown to the patient. In between the form, the type of window and the two features, are specified.

```
<form>
      <style><window-style>dialog</window-style></style>
      <feature name="UserName"/>
      <feature name="UserPassword"/>
</form>
```

When the *window-style* is set to *dialog* two buttons *OK* and *Cancel* are created automatically.
If the patient clicks the OK-button, the information entered into the textboxes will be compared with the data of the database. If the username is existent and the password is valid, the application will proceed to start. If the login is canceled, the window will terminate itself and the application stops to run.

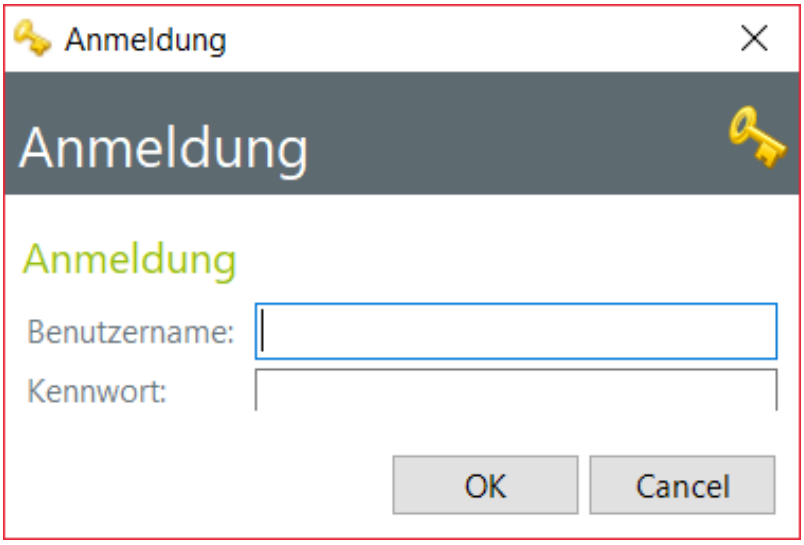


Figure 3.4: user interface for the login mask

Registration

However, if the patient has not got an account yet, the possibility to create a new one should be provided. In order to achieve this the following model was implemented:

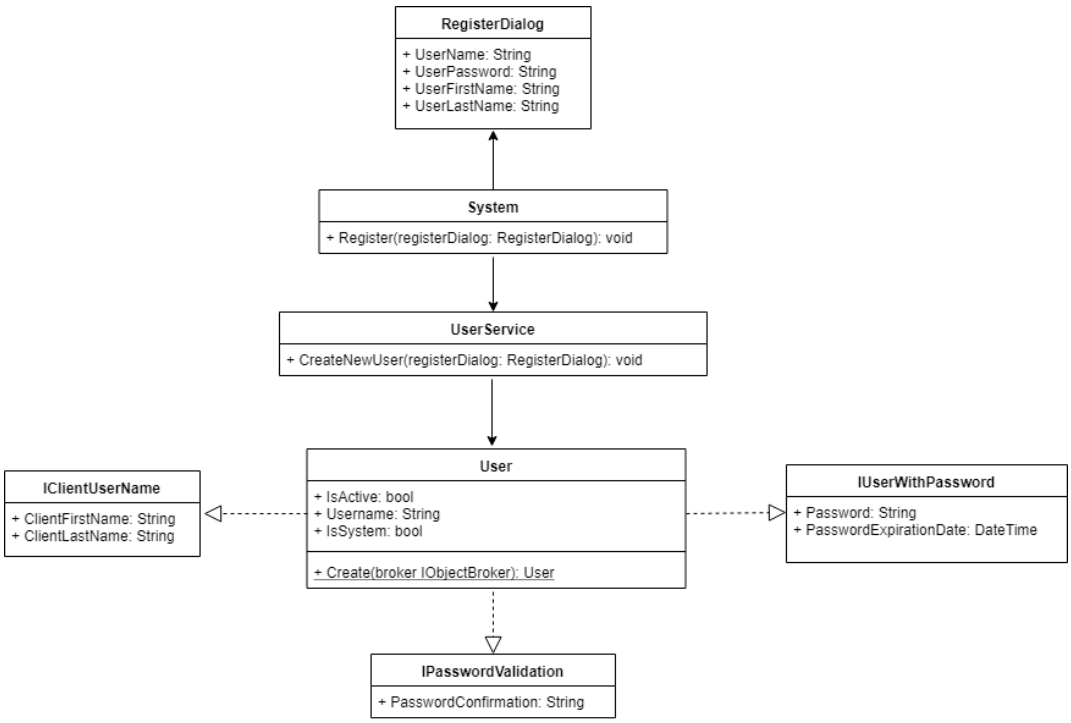


Figure 3.5: class diagram for registering a user

3.1.3 System

For this to work, a new button had to be added to the login dialog window. The idea is, that with this button an additional window is opened, where the patient can register as a new user. Buttons are called *MenuDefinitions* inside the BORA environment and are handled in the file *Menus.xaml*.

```
<MenuDefinition Entity="dotNetBF.Security.UPAuthInfo, bfsvclib"
            Name="Dialog">
  <WorkflowMenuCommand Title="Registrieren" WorkflowName="Register"
   AutoHide="false"/>
</MenuDefinition>
```

After a MenuDefinition has been defined, the button shows up in the dialog, but it has no function yet. To add a function to a button, a so-called workflow must be given. These workflows describe what action should be taken if the button is pushed. In the code above, the button points to a workflow named ”Register”.

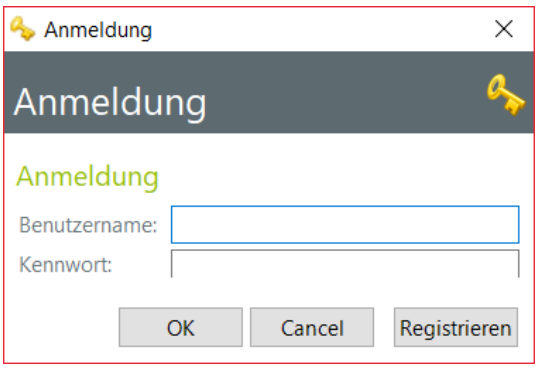


Figure 3.6: login window with register button

All workflows can be found in *Workflows.xaml*. The ”Register” workflow is supposed to open a new window in which the information is entered. However, before proceeding the patient has to be informed that their data, like their current location, is managed by the system, as it is critical for functions such as the distance calculation. Therefore, the workflow displays a warning, that personal data will be used by the application.

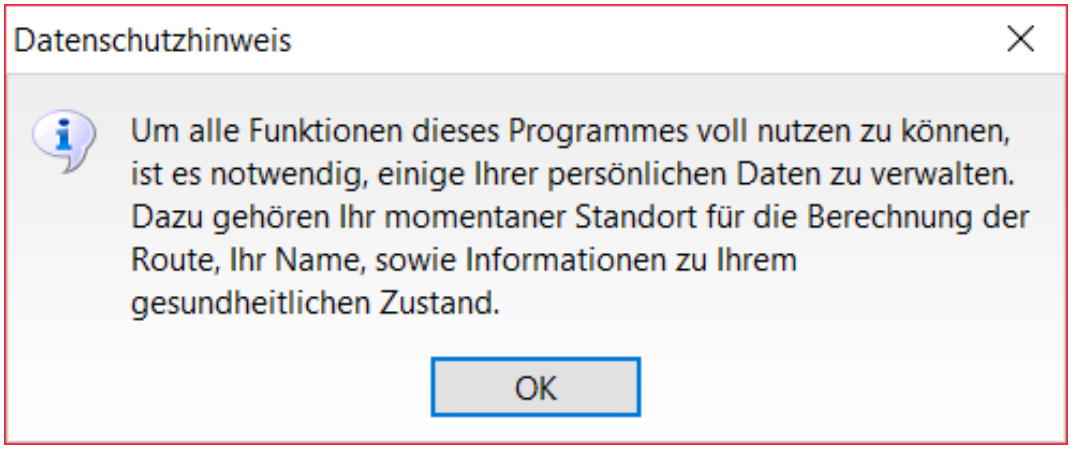


Figure 3.7: Confirmation notice

Now that the patient has allowed the system to interact with their data, the registration window can be shown. As a result, a new entity in Gui.xml had to be created, the same as with the login window. Only two additional fields, for the first- and surname of the patient, were added.

```
<entity name="RegisterDialog">
      <feature name="UserName">
            <style>
                  <title>E-Mail</title>
            </style>
      </feature>
      <feature name="UserFirstName">
            <style>
                  <title>Vorname</title>
            </style>
      </feature>
      <feature name="UserLastName">
            <style>
                  <title>Nachname</title>
            </style>
      </feature>
      <feature name="UserPassword">
            <style>
                  <title>Passwort</title>
            </style>
      </feature>
```

```
      <form>
            <style><window-style>dialog</window-style></style>
            <feature name="UserName"/>
            <feature name="UserFirstName"/>
            <feature name="UserLastName"/>
            <feature name="UserPassword"/>
      </form>
</entity>
```

To tell the workflow which entity is to be shown, a ”ShowForm”-tag with the name of the entity has to be specified. Now the workflow will open the registration window every time the button is clicked. After the information is entered and confirmed, the workflow reads the input and sets it on a variable. The variable is an object of the class RegisterDialog.

```
<Workflow Name="Register">
  <Transaction
   Behavior="NewRootTransaction" AuthInfo="session...GetServiceAuthInfo()">
    <Set Variable="register">
        new Patienteninformationssystem.UI.RegisterDialog()/>
  </Transaction>
</Workflow>
```

With the set-tag the information, which comes from the dialog, is placed on the object ”register”. The class of the object is specified between the tags.

3.1.4 RegisterDialog (Class)

This class was created in order to hold the same properties as the register dialog in the UI. An object of this class is initialized with the values of the input and is later handed over to the class ”UserService”.

Although, before the object can be sent to *UserService* the workflow has to know what to do after the dialog is finished. This is achieved with the ”Script”-tag.

```
<ShowForm Context="register">
      <Script>
      Patienteninformationssystem...CreateNewUser(register)
      </Script>
</ShowForm>
```

The object ”register” is now set as a parameter for the function ”CreateNewUser” in the class *UserService*. This function is called when the patient has entered their information and clicked ”OK” to finish the registration process.

3.1.5 UserService

UserService is a class that has only one function, which creates a new user in the database with the information that has been inserted by the patient. First, it uses the static function ”Create” of the BORA-class ”User” to generate a new entry in the database and then fills it with the value of each property of the register dialog.

```
public static void CreateNewUser(RegisterDialog registerDialog)
{
      var user = User.Create(Session.Current.Broker);
      user.Username = registerDialog.UserName;
      user.Password = registerDialog.UserPassword;
      user.ClientFirstName = registerDialog.UserFirstName;
      user.ClientLastName = registerDialog.UserLastName;
}
```

3.2 Creating a diagnosis

3.2.1 Collecting data

In order to create a diagnosis, the first step is to collect data about illnesses. The project's focus is to identify the severeness of illnesses that occur in the upper body area. For that matter, we interviewed people working in the medical field about the subject. The most important information about each illness are their symptoms, the reason people are affected by it and how to differentiate it from other illnesses with similar symptoms.

After having gathered enough data about each illness, we categorised them into different medical fields and severities so that our system can later decide in which kind of medical clinic the patient needs to be admitted.

3.2.2 Inserting the data

Knowing that people express their discomfort differently, the best way to create a diagnosis is by asking predefined questions that have predefined answers. Asking such questions allows the system to correctly identify the problem, rather than having to interpret unclear answers. Thus, Microsoft Workflows Foundation, a software that allows us to create our own questionnaire, has been decided to be used to depict the illnesses.

How to create a questionnaire

Inserting the questions into the system works as follows: A system administrator first needs to be logged into the system, since only they are allowed to manipulate the data of the system. If an administrator is logged in, an otherwise hidden section of the window is displayed: The entity section that allows an admin to alter all of the data that is stored in the database.

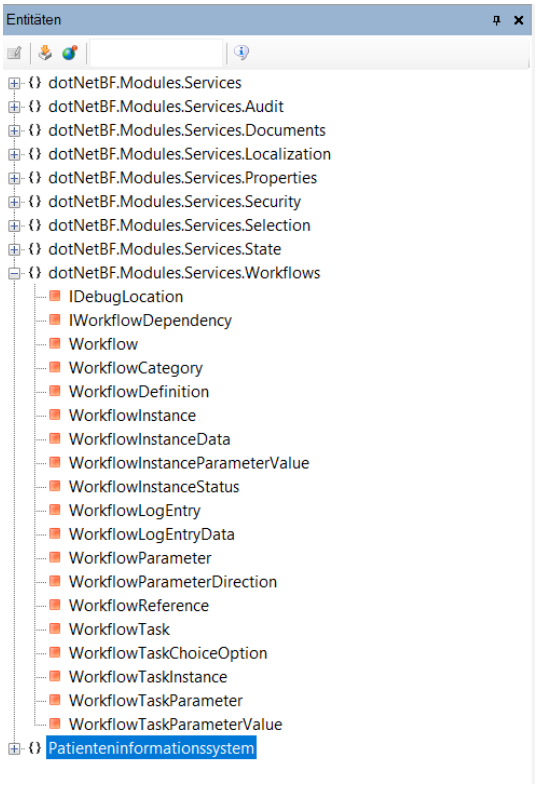


Figure 3.8: The entity section

The module `Workflows` has several different entities, the most important being called `Workflow` itself, since it has a customised form that is displayed if the administrator clicks on the entity name.

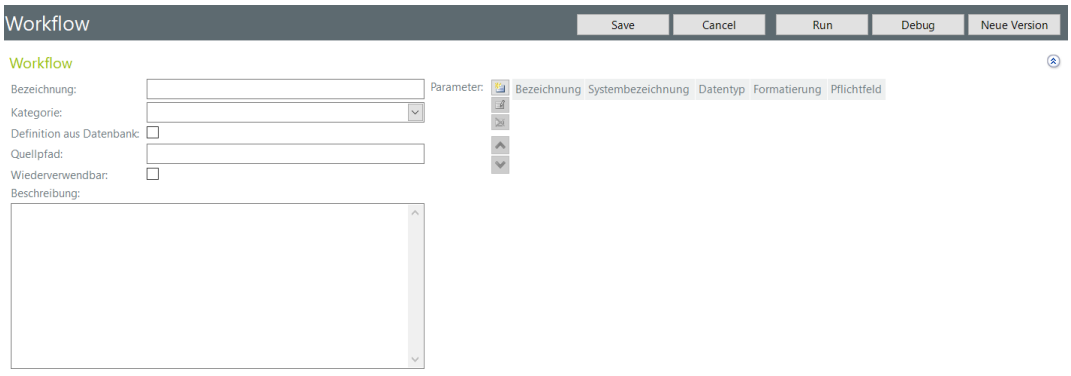


Figure 3.9: Microsoft Workflows integrated into the system

With this form, all properties of a Microsoft Workflow are adjustable. However, for

this project, only the properties ”Bezeichnung” and ”Definition aus der Datenbank” are required. ”Definition aus der Datenbank” means that the questions and answers are stored in the database; thus every user is immediately up-to-date once the changes are committed. The other option is to store the Microsoft Workflow locally on the device. For this project, however, it is necessary to store the questions in the database since every user should at all times be able to access the newest version of the questionnaire.

Once the administrator saves the basic information of the Microsoft Workflow, a designer is displayed. This designer is a part of the Microsoft Workflow Foundation that enables the administrator to enter the questions and answers with drag-and-drop.

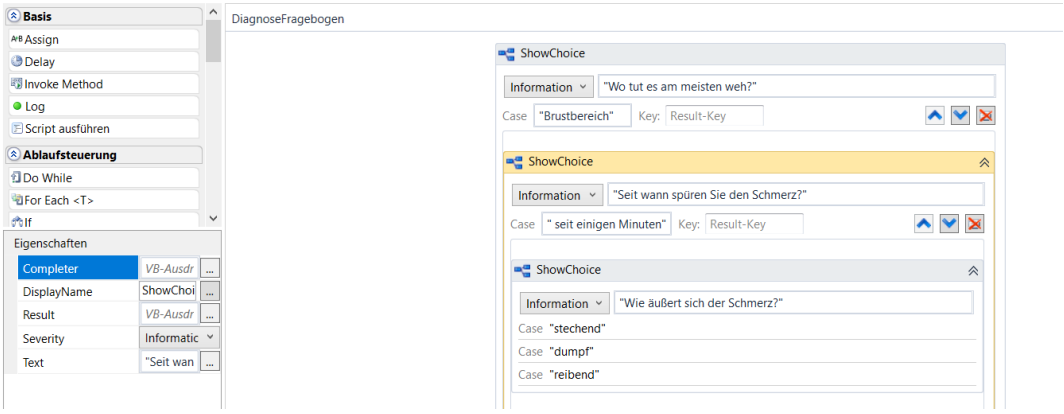


Figure 3.10: Microsoft Workflows Designer View

To add a new question, the administrator needs to select a so-called ”Choice” from the Microsoft Workflow tool catalogue on the very left. There, a ”text expression” can be entered. This expression is the first question that of the new questionnaire. To add a new predefined answer that the patient should be able to select from, a new ”case” is added. The case’s ”text expression” is the statement of the answer.

Once all possible answers are added, the administrator needs to specify which question follows next depending on the previous answer by dragging another ”Choice” from the tool catalogue into the white space of the case. These steps are repeated until it is clear what the diagnosis is.

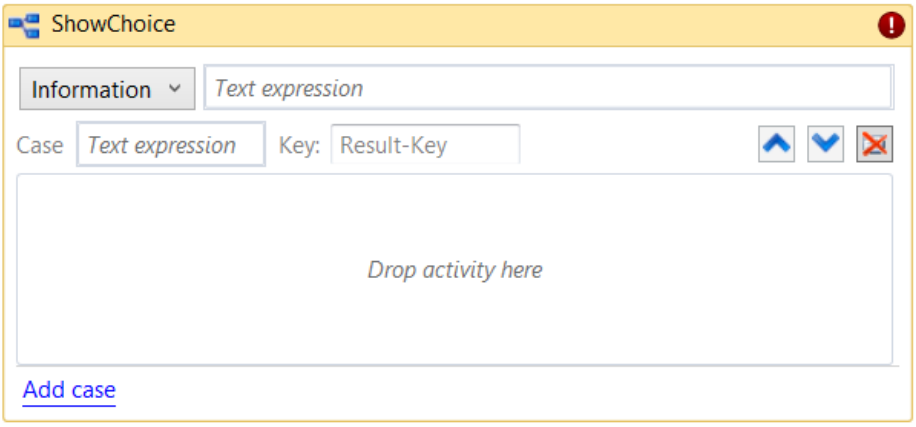


Figure 3.11: The input field of a choice

Defining a new log

Once the administrator has added all questions and answers needed to identify the illness, they need to set a log. A log is a piece of information about the Microsoft Workflow that can be accessed by a software developer in their code. However, the Microsoft Workflow regularly generates logs, so it is necessary to specify a customised log that is differentiable from logs the Microsoft Workflow generates automatically. The BORA framework defines all types of logs. These types are called ”severity”. To be precise, a `WorkflowLogEntry` is a kind of log that has a property called severity. All the severities that the framework currently recognises are stored in the database. So a new severity has been added by inserting a new row into the database.

```
INSERT INTO dbo.ServicesSeverity(ID, Level, ShortName, Name)
VALUES (6, 0, 'diagnosis', 'Diagnose');
```

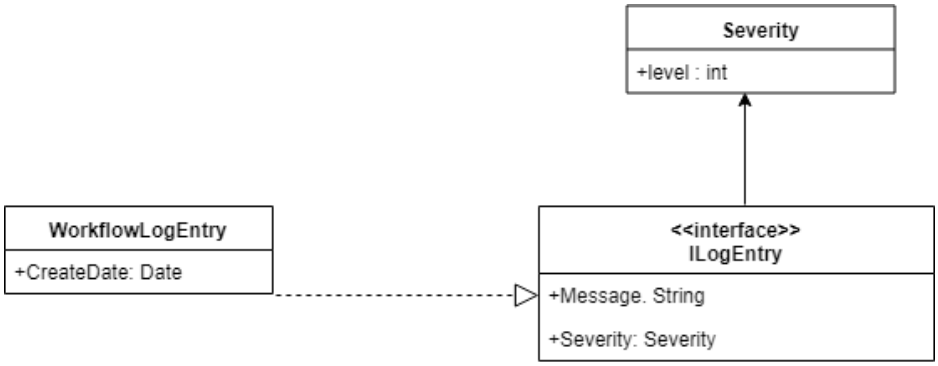


Figure 3.12: Relation between logs and severity

The `ShortName` of a severity is the universal name that is used to identify it in code, while the ”Name” of the severity is the text that is displayed to a user. The next step is to tell BORA that all the diagnoses of the user have to be accessible when coding. To achieve this, a so-called `KnownObject` with the name ”KnownSeverities” has to be created. A `KnownObject` allows you to access the severity ”diagnosis” globally by using KnownSeverities.<Diagnosis>.

```
<KnownObjects Repository="..\Patienteninformationssystem.Module
\Patienteninformationssystem.CR.xml">
  <Container Name="KnownSeverities">
    <Type Name="dotNetBF.Modules.Services.Severity"
       Identity="ShortName">
      <Instance ID="Diagnosis" Name="Diagnose" ShortName="Diagnosis"
         />
    </Type>
  </Container>
</KnownObjects>
```

The administrator can create a log of any severity that is stored in the database. Since ”Diagnosis” has been manually added, they can create a log that specifies the illness of this type. To do this, they need to drag a ”Log” from the toolbox of the Microsoft Workflow Designer into the empty white space of the last answer, set the type to ”Diagnose” and insert the name of the illness into the text field of the log.

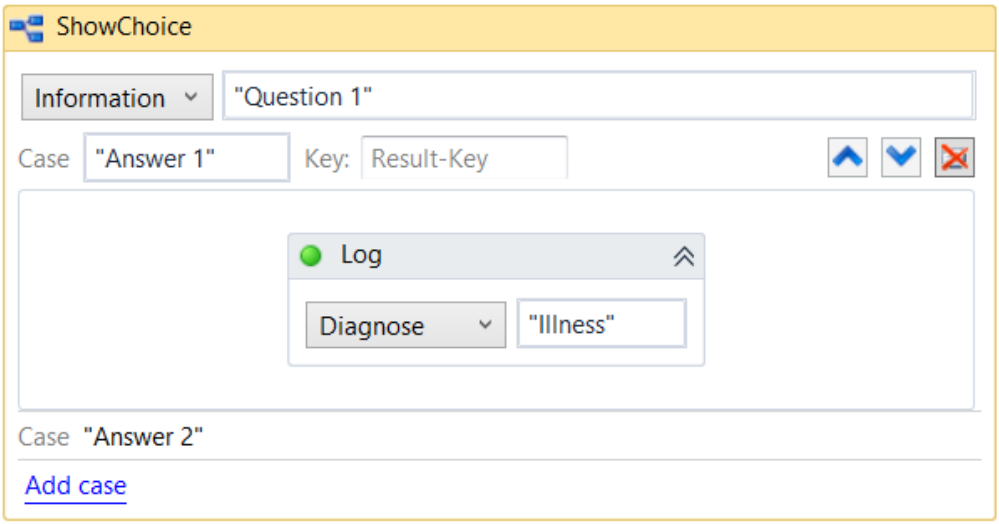


Figure 3.13: Inserting a log

Once the administrator saves all changes, any user can access the latest version of this with Microsoft Workflow created questionnaire by clicking on the button ”Vorfall melden” once they start the software, which is located in the top left of the main window.

Implementation

Microsoft Workflow is a module integrated into the BORA framework. Once the user clicks on ”Vorfall melden”, the BORA Workflow ”NewIncident” starts. BORA Workflow is a specific term used by BOOM to describe a method that is triggered by clicking a button.

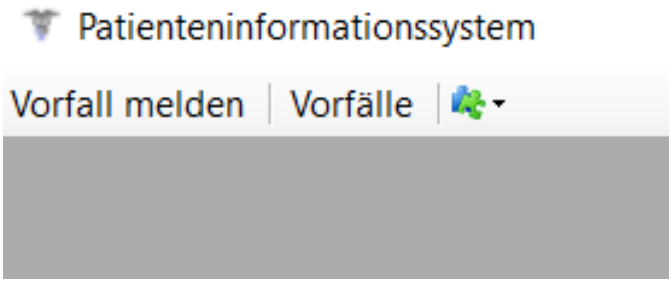


Figure 3.14: Button ”Vorfall melden”

```
<Workflow Name="NewIncident">

   <Set Variable="workflow" xml:space="preserve">
   dotNetBF.Modules.Services.Workflows.Workflow.Query(session.Broker)
      .Where(t => t.Name == "DiagnoseFragebogen")
   .OrderByDescending(t=>t.VersionMajor)
   .ThenByDescending(t=>t.VersionMinor)
      .FirstOrDefault()
                  </Set>
   <If Condition="workflow != null">

     <Create Variable="instance"
        Entity="dotNetBF.Modules.Services.Workflows.WorkflowInstance"/>
     <SetProperty Context="instance"
        Property="Definition">workflow</SetProperty>

     <Set Variable="newContext" Value="instance.DebugMode ? new
        dotNetBF.Modules.Services.Workflows.Debug.DebugContext(instance)
        : instance" />
     <ShowForm Context="newContext" Form="Input"/>
   </If>
  </Workflow>
```

When the method `NewIncident` is called, the system searches for all the Microsoft Workflows stored in the database called ”DiagnoseFragebogen”. Then, they are sorted by their versions so that the user is guaranteed to answer the newest one. Provided the system finds at least one Microsoft Workflow, it creates an object of the `WorkflowInstance`. While the Microsoft Workflow only consists of questions and answers, the `WorkflowInstance` is the object that contains the information about who started it, when they started it and what answers they selected. Then, the context, which in BORA language means the object that should be manipulated in the form and can be addressed by the keyword ”this”, is set to the instance, is displayed in a form called ”Input”. The default settings of the ”Input form” display the required input of the current context.

```
<section>
      <tile>
            <section>
          <style>
            <horizontal-fill>1</horizontal-fill>
          </style>
```

```
      </section>
      <section ref="parts.input"/>
      <section>
          <style>
            <horizontal-fill>1</horizontal-fill>
          </style>
      </section>
   </tile>
</section>
```

This form displays the question and buttons with all the currently possible answers that the system recognises. In the system, a question is defined as a so-called `WorkflowTask`. One task equals exactly one question. For each question that the user needs to answer, a `WorkflowTaskInstance` is created. This instance contains information such as when it was answered and by whom. Every time the user answers a question, an object of the class `WorkflowTaskChoiceOption` is created that contains the answer of the user. Both the `TaskInstance` and the `TaskChoiceOption` are automatically stored in the database by the BORA Framework once they are created.

Figure 3.15: Input form of the questions

For each answered question, the BORA Framework automatically saves the selected choice together with the id of the `WorkflowInstance` into the table `dbo.WorkflowsWorkflowTaskChoiceOption`. Once the user finishes answering the Microsoft Workflow, the property ”Status” of the `WorkflowInstance` is set to finished and a log with the severity ”Diagnose” will be added to the list of LogEntries of the WorkflowInstance.

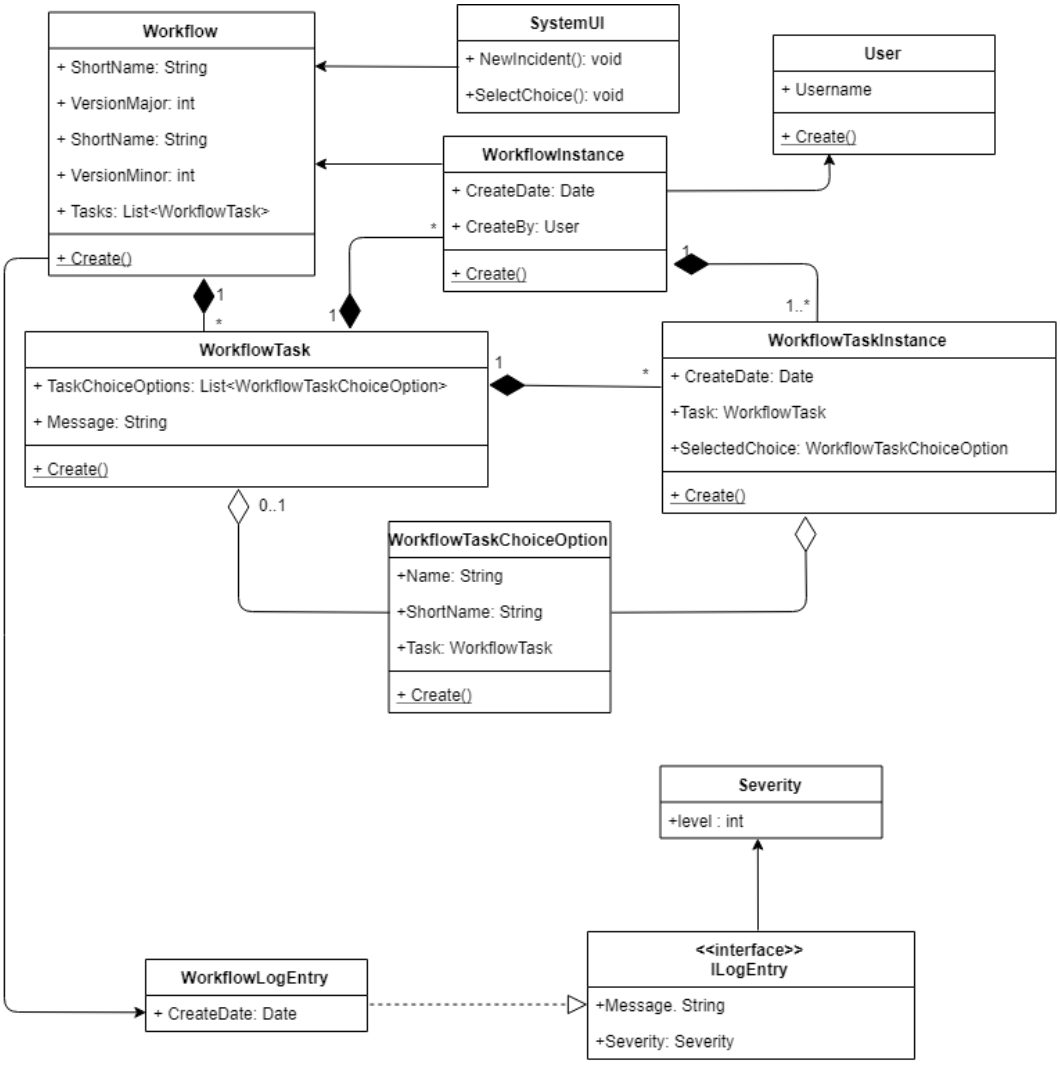


Figure 3.16: OOD diagram for creating a diagnosis

As modelled in the diagram, the `WorkflowLogEntries` themselves do not have any association to the class `Diagnosis`, since they only contain the name of it, which is a simple `String`. However, the class `Diagnosis` still has a property called ”Speciality” and association to the class `Speciality`, just as initially modelled in the OOA. The association between a diagnosis and a speciality of doctors is explained in detail in chapter 3.3.

3.3 Finding suitable doctors

3.3.1 Designing classes in BORA

Once a diagnosis is created, the system needs to find suitable doctors that are capable of treating this illness. Therefore it is necessary to store information about the doctors' specialities and their addresses. To accomplish this, different medical fields, the "specialities" of a doctor, have to be defined first.

With the BORA designer, a new diagram called "PersonManagement" has been created. Its purpose is to define any relations between the class `Doctor` and other classes.

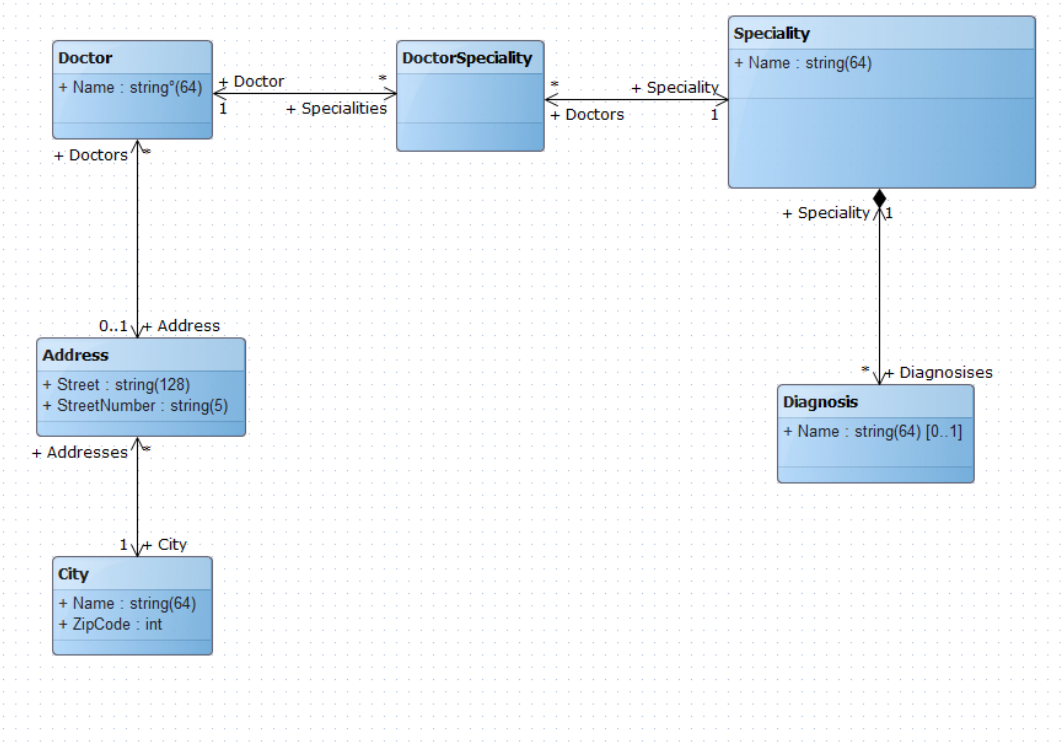


Figure 3.17: Diagram "PersonManagement"

BORA uses a diagram similar to UML. In the designer, new classes and associations are defined by dragging and dropping them. There are also various kinds of associations. There is a simple toolbox that one can use in order to create new classes, interfaces or associations. The important attributes of the doctor are their name, their address

and speciality. The entities in the database have the same name and columns as defined in this diagram, since the BORA framework automatically creates them if they are designed in the BORA Designer.

The relation between doctors and a diagnosis

The speciality has an attribute called ”name” since the system needs to identify it correctly. Due to the class being designed in the BORA designer, the framework stores information about the specialities automatically in the database. If the classed had been created in Visual Studio, the process of storing data to the database would have needed to be implemented manually. The classes `Diagnosis` and `Speciality` are associated with the ”composition” type since, in this project, a diagnosis that has no medical field assigned to it is irrelevant. In `DoctorSpeciality` all the information about the doctors and their specialities is stored. `Doctor` cannot have a direct association with `Speciality` because it is possible for a doctor to have multiple specialities.

3.3.2 Implementation

Implementation of the event

After a questionnaire, which is implemented as a Microsoft Workflow as described in chapter 3.2.2, is completed, the form, which displays the data of the current ”context” of the main window, needs to be adjusted. Otherwise, it would stay empty since it is still set to the required input of the already completed Microsoft Workflow. The context is defined as the object that is currently the main focus of the window. Once a patient starts a questionnaire, it is set to a new object of the class `WorkflowInstance` instance of the Microsoft Workflow, as described in chapter 3.2.3.

To tell a system that used the BORA framework what to do after such a Microsoft Workflow is completed, you need to implement an event. With BORA, you can take advantage of the fact that the system automatically stores and updates all the manipulated data of entities designed with the BORA designer. Since the Microsoft Workflow was

integrated into the framework using the BORA designer, it is possible to tell the system to do something each time the user answers a question. Whenever the database has to be updated, in this case, due to a new question being answered, the system calls the ”Saving” event of that object. There, you can define a C# Script. In this script, the static method `UpdateDoctorForDiagnosis` is called.

```
<ObjectScriptEventSubscription
   Entity="dotNetBF.Modules.Services.Workflows.WorkflowInstance"
   Events="Saving">
   if(Status.IsFinished)
   {
      Patienteninformationssystem.Services.DiagnosticService.
      UpdateDoctorForDiagnosis(this);
   }
  </ObjectScriptEventSubscription>
```

Searching suitable doctors

The class `DiagnosticService` has the purpose of assigning doctors to a diagnosis. Before searching for suitable doctors, the information about the diagnosis has to be retrieved. The name of the diagnosis can be accessed by using the information in the logs of the current `WorkflowInstance`. In case of a log being of the severity ”Diagnosis”, its message contains the name of the illness. Thus, the first thing that `UpdateDoctorForDiagnosis` does is calling the function `GetDiagnosisName`.

```
public static String GetDiagnosisName(WorkflowInstance instance)
{
      //Get the LogEntry with a severity of "Diagnosis"
   var diagnosisEntry = instance.LogEntries.Find(d =>
   d.Severity == KnownSeverities.Diagnosis);
   if (diagnosisEntry != null)
     {
       var diagnosisText = diagnosisEntry.Message;
       return diagnosisText;
      }

   return null;
}
```

Now that the name of the diagnosis is known, we can get all the diagnosis' properties, including its assigned speciality, by reading them from the database.

```
var diagnosis = Diagnosis.Query(Session.Current.Broker)
                .Where(x => x.Name == diagnosisText)
                .FirstOrDefault();
```

The next step is to look for the suitable doctors and save them into the database. Here is how the method `UpdateDoctorsForDiagnosis` searches for all doctors:

```
if (diagnosis != null)
{
  //get all doctors that can treat the illness
  var doctors = diagnosis.Speciality.Doctors;
  foreach (var doctor in doctors)
  {
    var dd = instance.DetectedDoctors.Find(x => x.Person ==
       doctor.Doctor);
    if (dd == null)
    {
    //create a "DetectedDoctor" and save it to the database
    dd = DetectedDoctor.Create(Session.Current.Broker);
    dd.Person = doctor.Doctor;
    dd.WorkflowInstance = instance;
    if(doctor.Doctor.Address != null)
    {
       //Calculate distance from device to the address of the doctor
       var result = DistanceService.CalculateDistance(GuiRepository.
       GetDisplayText(doctor.Doctor.Address));
      if (result != null)
      {
         dd.Distance = result.Distance;
         dd.Duration = (int)result.Duration.TotalSeconds;
      }
    }
  }
}
```

In this code snippet, all the doctors that can treat the illness are detected. Firstly, all doctors with a suitable speciality are selected, provided that the system could diagnose the illness yet. The diagnosis will be set once the Microsoft Workflow is completed. However, the method `UpdateDoctorForDiagnois` triggered whenever a question is answered, no matter whether there are still questions left to answer. Once the diagnosis does no longer equal null, every object of the class `DoctorSpeciality` that has

the required speciality is selected. For each `DoctorSpeciality`, a new object of the class `DetectedDoctor` is created and stored into the database. This detected doctor belongs to a specific `WorkflowInstance` and has the properties duration and distance. These properties describe how far they are away from our patient and the estimated time that the patient needs to reach the doctor. Furthermore, the property ”Person” is of the class `Doctor` and describes which person that doctor is.

3.3.3 Displaying the doctors

For displaying the doctors, a new section of the ”Input Form”, which is the currently active form, has to be activated. It is displayed once there are more than one currently detected doctors known in the `WorkflowInstance` that is currently the context of the main window.

```
<section visibility-condition="new
 dotNetBF.Gui.View(DetectedDoctors).Count().Value&gt;0">
    <style>
      <fill>true</fill>
    </style>
     <feature name="Diagnosis" gui="label" ></feature>
      <section >
        <style>
           <title ref="DetectedDoctors"></title>
           <fill>true</fill>
        </style>
        <feature name="DetectedDoctors" edit="none">
          <style>
            <title>Passende Aerzte</title>
            <title-placement>none</title-placement>
            <fill>true</fill>
            <control-source>
            system-win://ObjectListLong.xaml
            </control-source>
          </style>
        </feature>
     </section>
```

This form displays the diagnosis of the `WorkflowInstance` and its detected doctors. However, how the detected doctors need to be displayed is not specified here. It is specified in a ”feature”. Features are something similar to the well-known ”ToString”

methods in object-oriented programming languages, but in BORA, you can add multiple features for each class that needs to be displayed. The feature added to for the class `DetectedDoctor` defines what properties should be displayed when they are listed in a ”long list”. The long list, which is also called a complete list in BORA, is a list that contains more information about the attributes than the short list.

```
<listDefinition type="complete">
      <column name="Person"></column>
      <column name="personAddress"></column>
      <column name="DistanceCol"></column>
      <column name="DurationCol"></column>
      <orderBy sort="asc">Duration</orderBy>
    </listDefinition>
```

The list further refers to other feature definitions, which define how the property in question of the detected doctor should be displayed. In addition to that, the column header is also defined in the feature. The list is sorted ascending by the time needed to reach the doctor as this is the most relevant property.

```
<customFeature name="DurationCol">
      <style>
        <title>Dauer des Weges</title>
      </style>
      <expression>
      //Convert to minutes
        String.Format("{0} min", this.Duration/60)
      </expression>
```

In the UI, the list is displayed as a simple list that has selectable items.

Figure 3.18: List of detected doctors

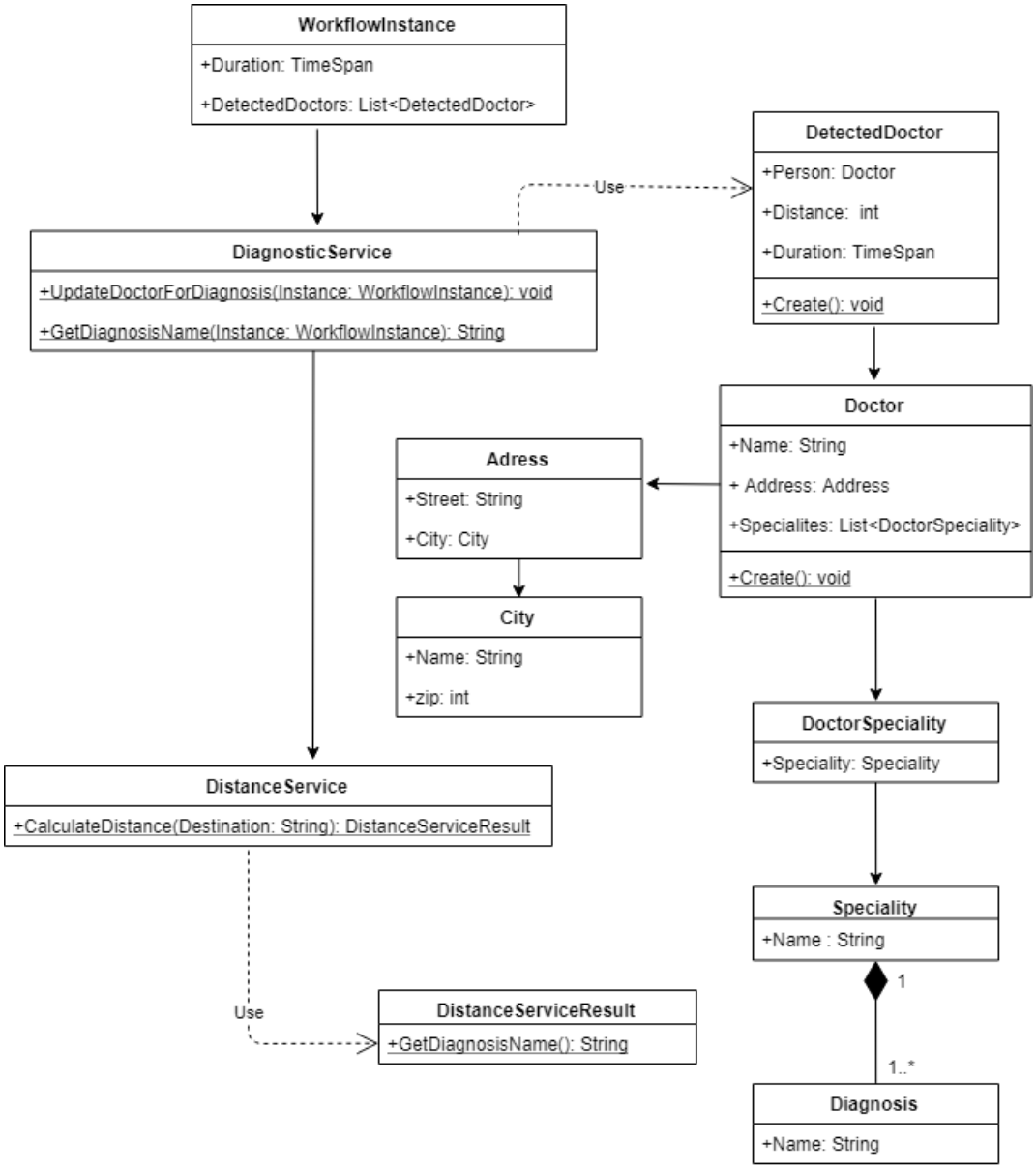


Figure 3.19: OOD diagram for Use Case 3

3.4 Displaying the Route

While our application is not particularly suited for usage in emergencies, in which accident notification time and emergency response time are of the highest priorities, the time taken for travel is nonetheless an important factor that should be taken into account when designing a medical application.

Although users of our system have the liberty to employ their preferred application for navigation, as one can simply input the doctor's address into their favourite route planner, providing the patient with a basic map that is able to display both the physical location of the patient, the doctor, and a route between them can significantly cut down on time spent.

There are multitudes of online services which are able to provide satellite maps of the earth, complete with points of interest, city and street names, and transportation directions between two or more custom set points.

A selection of these services are:

- MapBox
- MapFit
- Leaflet
- A custom-made one
- Google Maps

MapBox

MapBox is a live location platform, providing many different features like custom maps, navigation, the transformation of addresses into geospatial coordinates and more, for many different use-cases and platforms, ranging from the web over mobile and AR to even cars (*MapBox* 2019). Currently in use by many big companies such as Tinder, Bosch, IBM, CNN, Snapchat, National Geographic, Github, and more (*Mapbox Showcase* 2019), MapBox is a leader in the fields of physical map-based navigation. Customers of MapBox can communicate with its API over either HTTP or an SDK built

by MapBox. The service is free as long as no more than 50.000 map views, 50.000 Direction requests, and 50.000 Tilequery requests are used. After this limit has been reached, customers pay $0.50 per 1.000 requests.

MapFit

MapFit aims to provide ”Doorway-Accurate Maps,” where, if a route to an address is provided, the destination coordinate is not set to that address, but instead to an entrance of the building on the specified address (*MapFit* 2019). MapFit's goal is to correct existing geospatial datasets and to collect data from the street in order to provide the accuracy required. Instead of providing one geospatial coordinate for one address like conventional navigational systems, MapFit provides the customer with a range that represents different entrances, such as one or more pedestrian entrances, parking garages or loading docks. However, MapFit does not provide a navigation service, leaving the implementation to the customer, and additionally does not show pricing on its webpage, redirecting to a contact form instead.

Leaflet

Leaflet is not as much a service as it is a Javascript client library for displaying interactive maps (*Leaflet* 2019). Contrary to the other options, it is not built by a specific company, but instead open-sourced and contributed to many different people. Open-source often means no monetary expenses at all, but lacks the customer support and guides many companies offer. It does not natively have integrated navigation, but this specific feature is provided by a third-party library, the Leaflet Routing Machine (*Leaflet Routing Machine* 2019). Since it is written in javascript, embedding a browser into the application is the only supported use case, and unlike the previous options, no API is provided.

Custom-made

From a technical standpoint, developing a custom solution that handles displaying maps, providing navigation and directions, and taking points such as traffic information into account would be the preferable method, as it would allow the most customizability and

adaptability of all options. However, the massive time cost and big development process involved in rolling out a self-made routing application would be too much to handle.

Google Maps

For many use cases, Google Maps is the best fit, as it provides an interface known to and used by many, and has multiple features that fit the needs of a navigation system, such as turn-by-turn navigation, real-time traffic information, schedules of transit systems, and more (Rouse, 2019).

There are three main possibilities in order to utilize Google Maps in an application:

- Google Maps API
- Google Maps Embed
- Google Maps Link

Google Maps API

The API of Google Maps provides all features Google Maps has to offer, in an easy-to-use web interface which can be interacted with over HTTP. However, it does not provide a UI, making it the tool of choice for enterprise routing and navigating systems that bring their own, custom-made interface. The API also requires an API key, gated behind a monthly fee based upon how heavily it is used within the consumer application.

Google Maps Embed

If the infrastructure of the application allows for browser content to be displayed, Google Maps Embed may also be used. While not officially supported, simply displaying the Google Maps web page within the context of the application is the easiest way of displaying a navigational system without building a fully-featured UI yourself.

Google Maps Link

Merely providing a Google Maps link that displays the content in the user's browser of choice may by far be the easiest option, but not the prettiest, as it redirects the user away from the application.

Since a user might not only use the routing navigation on the device PINFOS is running on, but also on their mobile phone, and with both the API and the Embed method maps prove difficult to transfer to other devices, using simple links remains the best option in our case.

In order to utilize navigation on maps within our application, the following parameters must be known:

- Start point of the route
- End point of the route
- Mid points of the route (though optional)

Since the patient wants to get to their selected doctor in the fastest way possible, without any interruptions on the way, required information looks like this:

- Start point: Location of the patient
- End point: Location of the doctor

3.4.1 Retrieving the location of the patient

The location of the doctor is already known to our system as it is saved in the database. Therefore the only information left to retrieve is the patient location.

The information can be directly queried from the patient by letting them input an address over a UI. However, indirectly retrieving the location by instead determining the physical location of the device used by the patient works as well, is easier to integrate into the application and does not require interaction.

As the Bora Framework does not have a method for retrieving the device location, this feature needs to be implemented in C#.

The `System.Device.Location` Namespace provides the perfect tools for the job, as it defines a `GeoCoordinateWatcher` class that is able to retrieve the latitude and longitude of the current device.

Sample usage of this class would look like this:

```
GeoCoordinateWatcher watcher = new GeoCoordinateWatcher();
watcher.TryStart(false, TimeSpan.FromMilliseconds(1000));
GeoCoordinate coord = watcher.Position.Location;

if (coord.IsUnknown != true)
{
  double Latitude = coord.Latitude;
  double Longitude = coord.Longitude;
  // Use Latitude and Longitude further in the application
}
else
{
  Console.WriteLine("Unknown latitude and longitude.");
}
```

3.4.2 Defining the UI

The best possible solution for navigating to Google Maps is a simple button, displaying when an incident is selected within the application.

A .xaml file called `MapControl.xaml` is created, and filled with the following code:

```
<?xml version="1.0" encoding="utf-8" ?>
<?Mapping XmlNamespace="urn:bfwin" ... ?>
<?Mapping XmlNamespace="urn:system.ui.win" ... ?>
<?Mapping XmlNamespace="urn:patwin" ... ?>
<bf:SingleValueObjectFeatureControl ... >
  <Button Content="Link zu Google Maps"
     Click="OnNavButtonClick"></Button>
</bf:SingleValueObjectFeatureControl>
```

This code defines a simple button with the content ”Link zu Google Maps,” and registers a click event listener function called `OnNavButtonClick`.

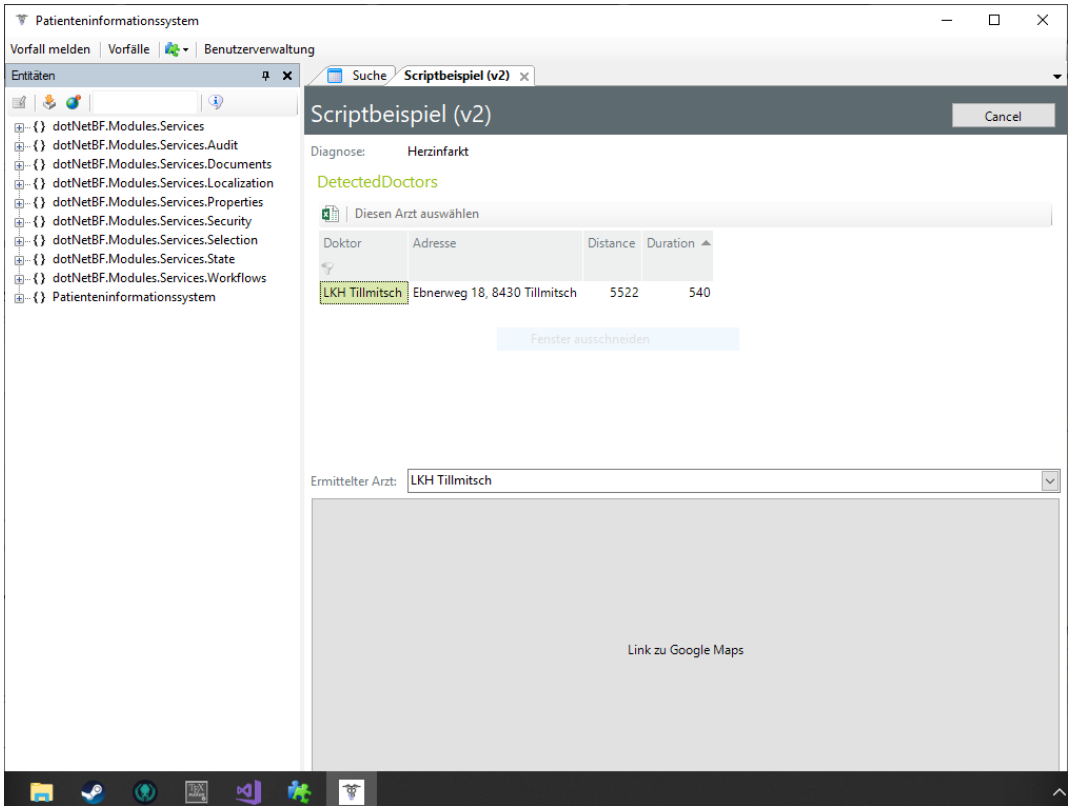


Figure 3.20: The ”Link to Google Maps” button as displayed in the UI

3.4.3 Redirecting the patient to Google Maps

Upon clicking on the previously defined button, it opens the patient's browser and navigate to Google Maps, automatically inputting the start and the destination location in the process.

Google Maps allows setting both the start and end point by modifying the URL, using a format of `https://www.google.com/maps/dir/{startPoint}/{endPoint}`. Since Google Maps is able to work with coordinate-based locations, the points do not need to be resolved to a civic address.

The function `OnNavButtonClick`, the click handler of the button, is defined in a new file called `MapControl.cs` and, using our previously defined method of retrieving the device location looks like this:

```
private void OnNavButtonClick(object sender, RoutedEventArgs e)
{
  base.OnValueChanged();
  if (!(Value is WorkflowInstance wi)) return;

  // Retrieve the device location
  GeoCoordinateWatcher watcher = new GeoCoordinateWatcher();
  watcher.TryStart(false, TimeSpan.FromMilliseconds(500));
  GeoCoordinate coord = watcher.Position.Location;
  var sourceLocation = $"{coord.Latitude},{coord.Longitude}";

  // Retrieve the doctor's location
  var doctors = wi.DetectedDoctors;
  doctors = doctors.OrderBy(x => x.Duration);
  var bestDetectedDoctor = doctors.FirstOrDefault();
  if(bestDetectedDoctor == null) return;
  if(bestDetectedDoctor.Person.Address == null) return;
  var address = bestDetectedDoctor.Person.Address
  var targetLocation = GuiRepository.GetDisplayText(address);

  // Find default browser of device
  String browserName = null;
  RegistryKey key = null;
  subkey = @"HTTP\shell\open\command";
  key = Registry.ClassesRoot.OpenSubKey(subkey);
  if (key != null)
  {
    browserName = key.GetValue(null).ToString().ToLower().Trim(new[]
       { '""' });
  }

  // Open the browser with the specified url
  if (browserName != null)
  {
    var url = $"https://www.google.com/maps/dir/" +
       "{sourceLocation}/{targetLocation}";
    Process.Start(browserName, url);
  }
}
```

As the function needs access to both the current `WorkflowInstance` and the `GuiRepository`, the class must inherit from `SingleValueObjectFeatureControl`, a BORA class, which will provide these values.

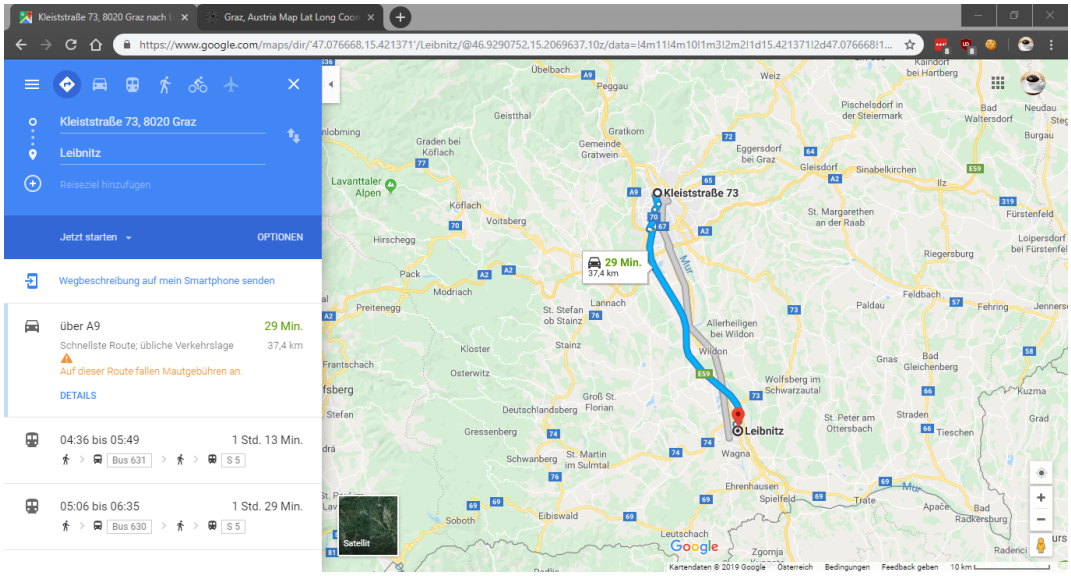


Figure 3.21: An example URL

3.5 Adding a new doctor

In order to ensure that the system can always provide the best-qualified doctors, it is of great importance that the database holds as many entries as possible. The more doctors are known, the better the recommendation for the patient. Therefore, it has to be possible to enter additional doctors in case new offices should be opened.

Since personal data about these doctors will be stored, it is important to interact with the doctors and inform them about what information will be stored. Only if the doctor agrees that they allow access to their data, they can be fed into the database.

3.5.1 Database

With the help of *BORA Designer*, an architecture was created so that all essential information could be accessible to the system. It consists of five entities:

- Doctor
- Speciality
- DoctorSpeciality
- Address
- City

The resulting model looks like this:

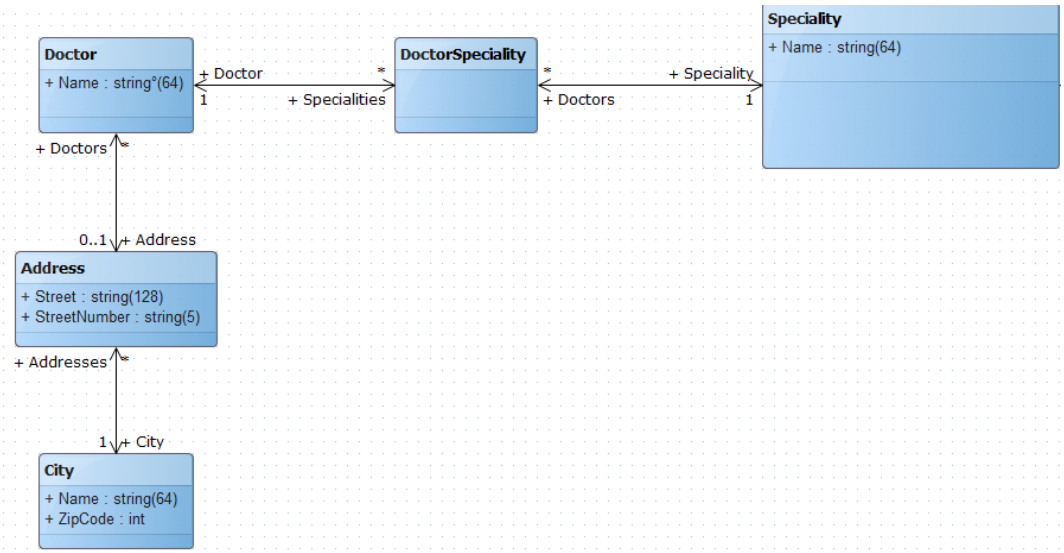


Figure 3.22: doctor entities

3.5.2 Procedure

For implementing a new doctor following model has been applied:

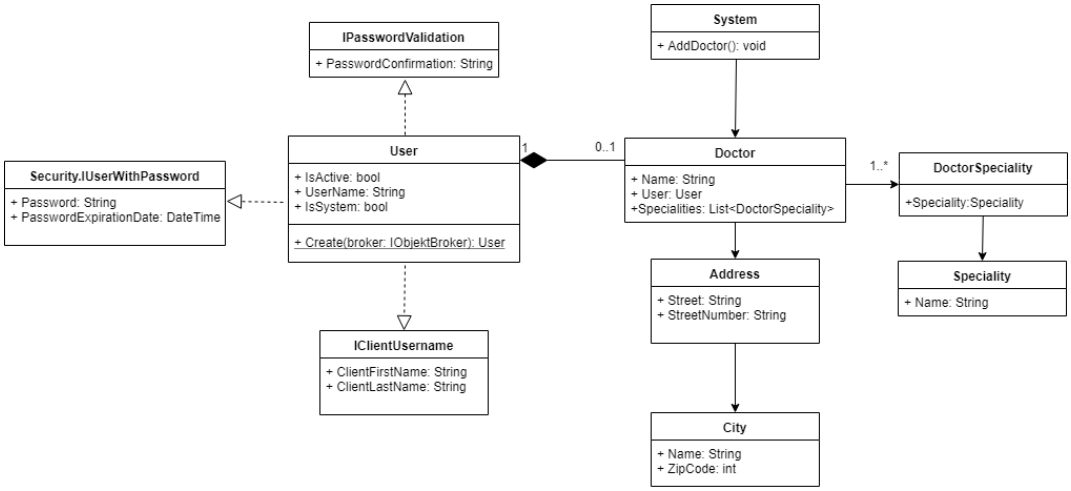


Figure 3.23: class diagramm for adding a new doctor

This architecture dictates, that each doctor also has to have an user account. It is necessary because, when the doctor is to be informed about arriving patients, an e-mail address is needed and the property for the e-mail address is stored in the class ”User”. To ensure that every doctor has automatically a user account upon creation, a new entry was inserted in *Events.xml*. In this file actions can be specified, which are activated when certain events happen.

```
<ObjectScriptEventSubscription
   Entity="Patienteninformationssystem.Doctor" Events="Created">
      User =
         dotNetBF.Modules.Services.Security.User.Create(session.Broker);
</ObjectScriptEventSubscription>
```

This code says that when a doctor has been created, a user is simultaneously inserted into the database. As a result, every doctor now has an e-mail address.

If an administrator now wants to insert a new doctor, the application has to be started first. Now that the application is running, a list of entities appears on the left. If the

”Patienteninformationssystem”-tab is expanded, all properties belonging to the doctor will be displayed

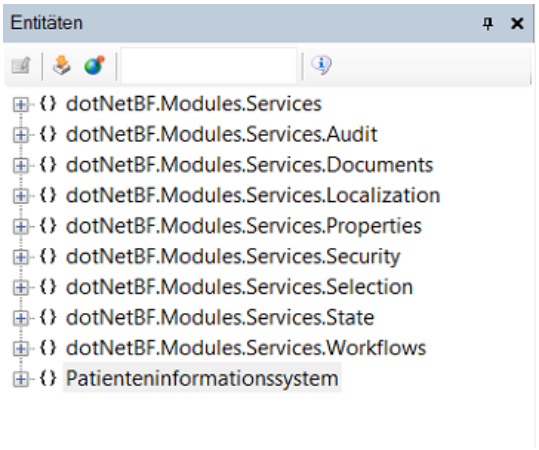


Figure 3.24: List of entities

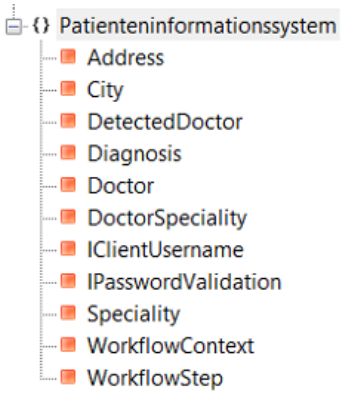


Figure 3.25: expanded subfolders

For all doctors, the distance and the duration from the patient have to be calculated. As a result, every doctor has to have the address of their office entered. First, the city or town is registered, if it is not existent already. If the *City* subfolder is clicked, all cities that are already added are displayed in a table. Each city consists of the name and the zip code. To add a new one, the *Add* button above the table has to be pushed.

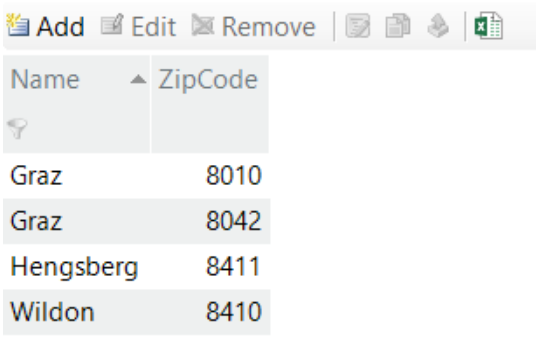


Figure 3.26: table with cities

A new window will appear and now the name and the zip code can be entered and after the input is saved, the information is entered into the database.

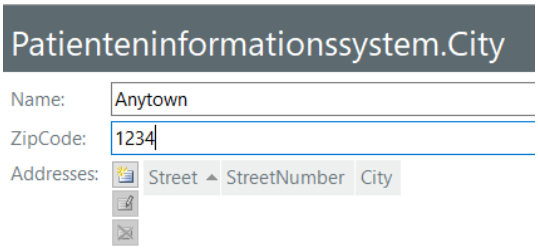


Figure 3.27: adding a new city

Now that the city exists, the address can get inserted too. After switching to the *Address* subfolder all addresses will be shown. And just like with the city, an add-button can be clicked to input the street and number. Additionally, the city is implemented, which is chosen from a list of all available cities.

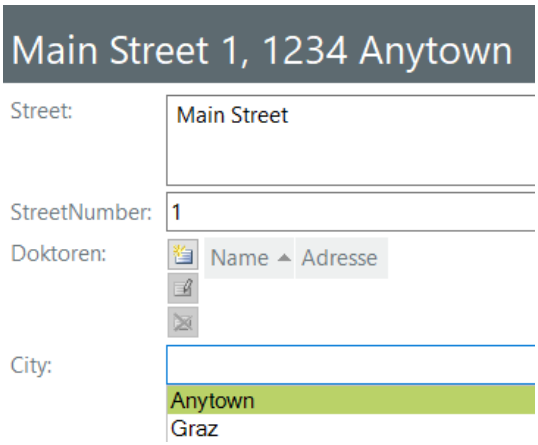


Figure 3.28: input for the address and selecting a city

It is very important that doctors can be filtered by their speciality in order to fit a certain diagnosis. So naturally different specialities have to exist in the database. Before entering them into the database, some research had to be conducted in order to find out for which problems they are appropriate. This is important for the reason, that it has to be known which specialities fit which diagnoses. A speciality is entered in the same manner as the address or city and only possesses its name as a property.

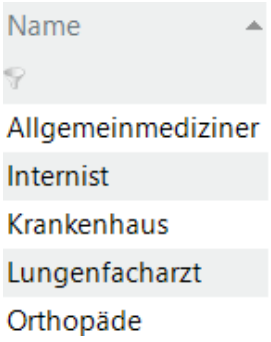


Figure 3.29: list of specialities

Lastly, the doctor themselves has to be entered. As mentioned before, a doctor also has to have a user. This means, that the name of the doctors and all properties of the user have to be inserted.

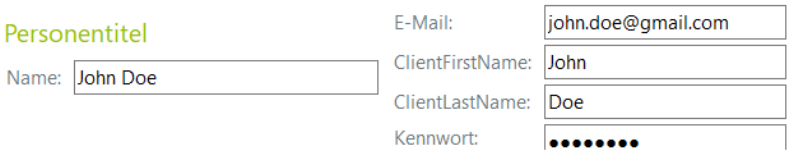


Figure 3.30: input fields for a doctor

However, an address has to be linked to a doctor. But this time this cannot be done by choosing an address from a list. Next to the add button, there is another option to edit the entity as XML-file.

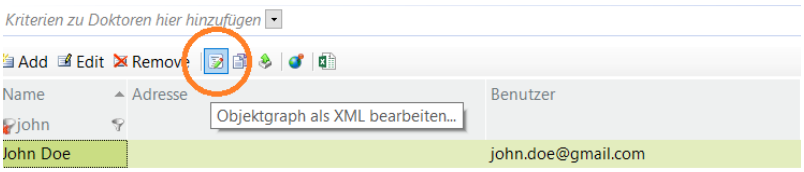


Figure 3.31: input fields for a doctor

When the file has been opened, the address has to be specified between the doctor tags.

```
<Doctor x:Role="root" OID=15" Name="John Doe"
   xmlns="model-package:Patienteninformationssystem">
      <Doctor.Address>
            <Address x:IsReference="true" OID="1"/>
      </Doctor.Address>
</Doctor>
```

The ID of the address refers to the address index in the database. After this XML has been updated, the doctor now has a location next to their name.

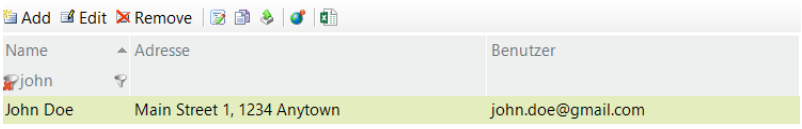


Figure 3.32: doctor with address

Now that everything is already fed into the database, the doctor just has to be linked to their speciality. Both the doctor and the speciality can be chosen from a drop-down list. Of course, more than one speciality can be linked to a doctor.

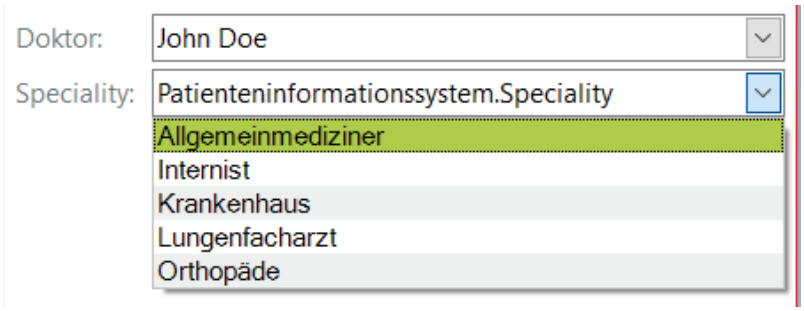


Figure 3.33: connecting doctor and speciality

With all the information inserted into the database, the new doctor entity is ready for use and will be taken into account in the next search run.

3.6 Informing a doctor

3.6.1 Basic functionality

Once a user decides which doctor they want to go to, the doctor has to be notified that a new patient is on the way. Once a doctor has received the notice, they should also be able to see the details about the patient and why they need medical assistance. A simple email was chosen to be the best choice to send messages since mobile notifications and other alternatives are probably too aggressive. Moreover, most doctors are working on their computers nowadays so they can easily access their inbox.

3.6.2 Sending an email

Basic configuration

Sending emails is a fundamental feature of the BORA framework. However, the framework first requires some basic information, such as which SMTP-Server to use, which port is required and which email account BORA needs to use when sending an email. Such configurations are saved in the `modules.config.xml` file. This file contains all kind of basic configurations that could be required anywhere when coding. The following lines had to be added to the file:

```
<system:EmailService SmtpServer="smtp-mail.outlook.com"
   SmtpPort="587" DefaultSender="patienten.system@outlook.com"
   PermittedRecipientPattern=".*" SmtpEnableSsl="true" >
            <system:EmailService.Credentials>
         <systemnet:NetworkCredential
            UserName="patienten.system@outlook.com"
            Password="password" />
            </system:EmailService.Credentials>
      </system:EmailService>
```

Implementation of the email sending process

To tell the BORA framework that it needs to send an email, the static method `SendToDoctorEmail` has been implemented. This method auto-generates a message and sends it to a doctor. The method looks as follows:

```
public static void SendToDoctorEmail(DetectedDoctor detectedDoctor,
    ISession session )
        {
            string subject = "Patient on the way";
            //auto-generate message
            string msg = GenerateMessage(detectedDoctor);
            var message = new EmailMessage(new
                EmailAddress(detectedDoctor.Person.User.Username),
            subject,msg,false);
            EmailService.Send(session, message);
        }
```

When the method `GenerateMessage` is called, it uses the last name of the doctor and inserts it into a predefined text. Once the message is generated, the email can be created. The user name of any account that uses the system is defined as their email address, which is why it can be converted into an object of the class `EmailAddress` easily. Only doctors that allow us to use their data and have an account for our system are recommended to patients. This guarantees that every doctor has provided an email address that is stored in the database.`EmailService` is a class provided by the BORA Framework that already implements the process of sending an email. It uses the information that is being provided in the modul.config.xml to send emails.

The patient can call this function by clicking on a button called ”Diesen Arzt über Anreise informieren”. It is only clickable once the patient has selected a doctor from the list. The doctors and the button are being displayed automatically after the user completes a Microsoft Workflow, as described in chapter 3.3.3.

Figure 3.34: Button ”Diesen Arzt über Anreise informieren”

This button triggers the BORA Workflow `SelectDoctor`. This Workflow first displays a message asking whether the patient is sure the email should be sent to avoid sending wrong emails. If the patient confirms that they want to inform the doctor about them wanting to see them, the email will be sent.

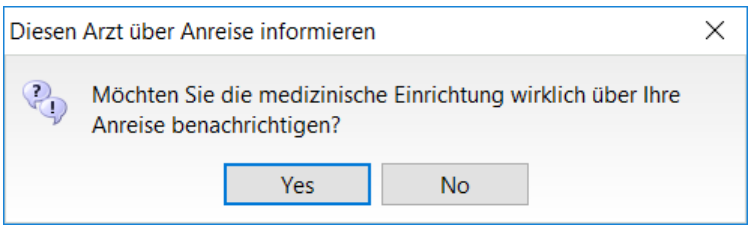


Figure 3.35: Confirm dialog for sending mails

The email that the doctor receives looks like this:

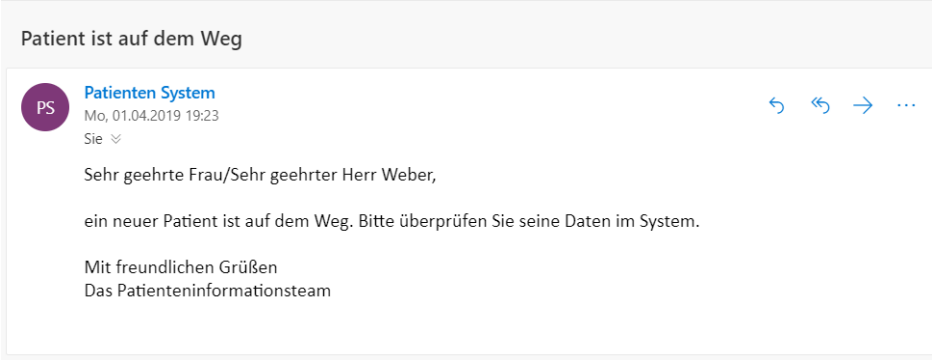


Figure 3.36: Email sent to a doctor

3.6.3 Displaying the diagnosis

Retrieving the data

Once the doctor receives the email, they can use the system to see what symptoms the patient has stated to have. This is done by clicking on the button ”Vorfälle” that is only visible for doctors and administrators. The button is located in the top left of the main window once the doctor is logged in.

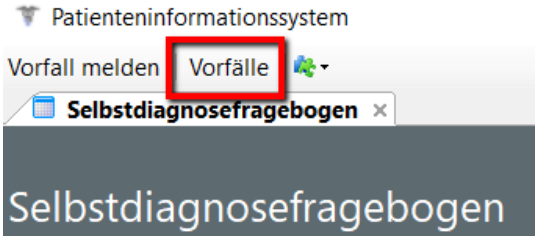


Figure 3.37: Button ”Vorfälle”

When clicking on this button, a BORA workflow called ”AllIncidents” is triggered. This workflow is responsible for selecting all the instances that of the Microsoft Workflow that have been completed. In other words, the method selects all rows of the table ”WorkflowInstance” in the database, transforms them into objects of the class `WorkflowInstance`. and returns them.

```
<Workflow Name="AllIncidents">
      <Transaction Behavior="NoTransaction">
         <Set Variable="List">new
            Patienteninformationssystem.Services.IncidentList()</Set>
         <ShowForm Context="List"></ShowForm>
      </Transaction>
   </Workflow>
```

The class `IncidentList` is a derived class of the class `SearchList`, which is specified in the BORA framework. `IncidentList` specifies that what the system is looking for is not any list, but a list of objects of the type `WorkflowInstances` that have the status ”complete”. What is crucial here is that ”complete” does not mean that the patient who started this `WorkflowInstance` has actually answered all the required questions, but rather that it is not ”running” anymore. Aborted `WorkflowInstances` are also marked as ”complete”.

```
public class IncidentList :SearchList
   {
      public IncidentList()
         :base(typeof(WorkflowInstance),"complete")
      {
         Lookup = FindLookup("search");
      }
   }
```

After the `IncidentList` has returned all found objects, the context, which is the content of the main window, is set to the list of `WorkflowInstances`.

However, there is a condition which filters these objects: Only objects of the class `WorkflowInstance` that are marked as ”finished” and belong to the doctor currently using the system are displayed. While any `WorkflowInstance` that is not currently running is marked as ”complete”, only the ones marked as ”finished” were answered until the end. The second condition filters the `WorkflowInstances` so that the doctor can only view a patient’s details and their symptoms if they are going to see them. All other patients that have selected another doctor or not specified any doctor to go to are not shown. Implemented in BORA, this filter looks like this:

```
<lookup>
      <source xml:space="preserve">
```

```
            let(currentUser=
            dotNetBF.Modules.Services.Security.
            SecurityService.CurrentUser)
          dotNetBF.Gui.View.FromExtent(typeof(
          dotNetBF.Modules.Services.Workflows.WorkflowInstance))
      .FilterInStore(x => x.Status ==
       dotNetBF.Modules.Services.Workflows.WorkflowService.
       FinishedWorkflowStatus)
         .FilterInStore(x => x.SelectedDoctor.?Person.?User ==
            currentUser)
 </source>
</lookup>
```

Design of the UI

The objects of the class `WorkflowInstance` are defined to display only the time passed since they have been created, by whom and what the diagnosis of the system was. What is notable here is that the time since the creation of the objects updates automatically, so if a doctor leaves the computer for a while, they do not have to tell to recalculate the time since its creation manually. Defining how to display the different objects is done in a feature, as described in chapter 3.3.3.

```
<listDefinition type="complete">
    <column name="TimePassed"></column>
     <column name="CreateBy">
       <style>
          <text>ClientFirstName + " " + ClientLastName</text>
       </style>
     </column>
     <column name="Diagnosis"></column>
     <orderBy sort="asc">TimePassed</orderBy>
</listDefinition>
    <feature name="CreateBy">
       <style>
         <title lang="de">Patient</title>
       </style>
     </feature>
<customFeature name="TimePassed">
   <style>
     <title>Verstrichene Minuten</title>
     <text>String.Format("Vor {0} Tagen, {1} Stunden, {2} Minuten",
        this.Days, this.Hours, this.Minutes)</text>
   </style>
   <expression>
```

```
        dotNetBF.Gui.ScalarView.FromNow(TimeSpan.FromSeconds(30)).Value
           - CreateDate
  </expression>
</customFeature>
```

The feature ”Diagnosis” displays the name of the diagnosis that has been generated by the system, while ”CreateBy” displays the user it was created by, but only the first and last name, since this is the most relevant information about the arriving patient. The expression defined in the feature ”TimePassed” uses the predefined function of BORA called `FromNow` that updates the time passed every 30 seconds. The time interval of 30 seconds has been chosen since only minutes are displayed. Therefore, it is not necessary to update the time every second. The ”CreateDate” is a property of a `WorkflowInstance`.

This is what the list looks like when a doctor opens it:

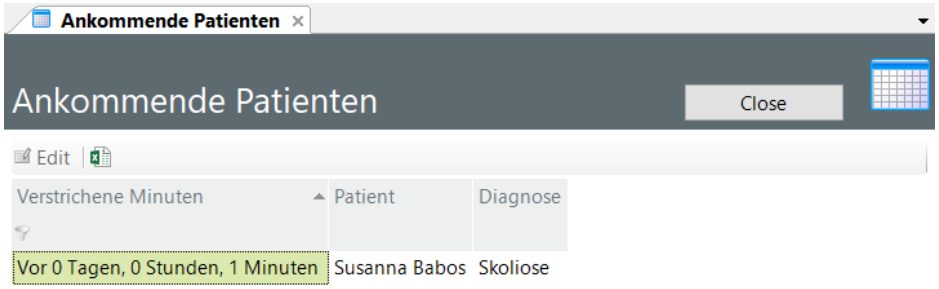


Figure 3.38: List of arriving patients

Once the doctor double clicks on a patient, the doctor can view the patient’s answered questions, which allows the doctor to inform themselves about the patient before they arrive.

This is how the list of questions looks like:

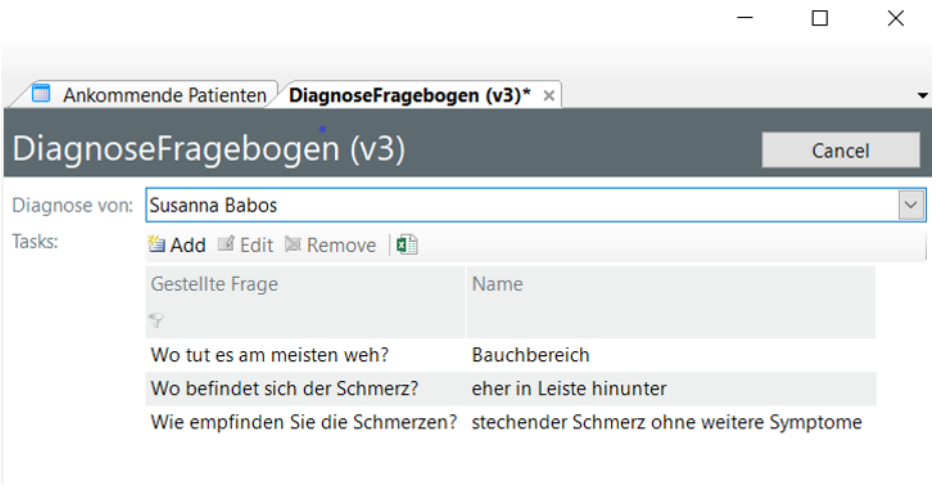


Figure 3.39: List of questions answered by a patient

The list is generated by a form that uses the class `WorkflowTaskInstance` and its property `SelectedChoice`.

```
    <form name="Doctor">
      <section>
        <style>
        </style>
      </section>
      <feature name="CreateBy">
        <style>
          <title>Diagnose von</title>
        </style>
      </feature>
      <feature name="Tasks">
      <!-retrieve all tasks that belong to this workflow by using
         the property "Tasks"-->
        <lookup >
          <source>new dotNetBF.Gui.View(Tasks)</source>
        </lookup>
        <style>
          <title ref="Tasks"></title>
          <fill>true</fill>
          <control-source>system-win://ObjectListLong.xaml
          </control-source>
        </style>
  </feature>
</form>
```

In the code snippet above, all the tasks that currently belong The `WorkflowTaskInstances`

have their own list definition. This list definition is necessary to display only the needed information. The list definition looks as follows:

```
<entity name="WorkflowTaskChoiceOption">
    <style><title>Antwort</title></style>
  </entity>
  <entity name="WorkflowTaskInstance">
    <style>
      <title>Antwort</title>
    </style>
    <listDefinition type="complete">
      <column name="Message"></column>
      <column name="option"></column>
    </listDefinition>
    <customFeature name="option">
      <style>
        <title ref="dotNetBF.Modules.Services.Workflows
        .WorkflowTaskChoiceOption/Name"></title>
      </style>
      <expression>SelectedChoice.Name</expression>
    </customFeature>
    <feature name="Message">
      <style>
        <title lang="de">Gestellte Frage</title>
      </style>
    </feature>
  </entity>
```

The first column which is called ”Message” displays the question of the `WorkflowTaskInstance`. The answer of the question is stored in the property ”Name” of the selected `WorkflowTaskChoiceOption`, which is the property ”SelectedChoice” of the `WorkflowTaskInstance`. The ”SelectedChoice” is automatically saved in the database every time a patient answers a question, as described in chapter 3.3.1.

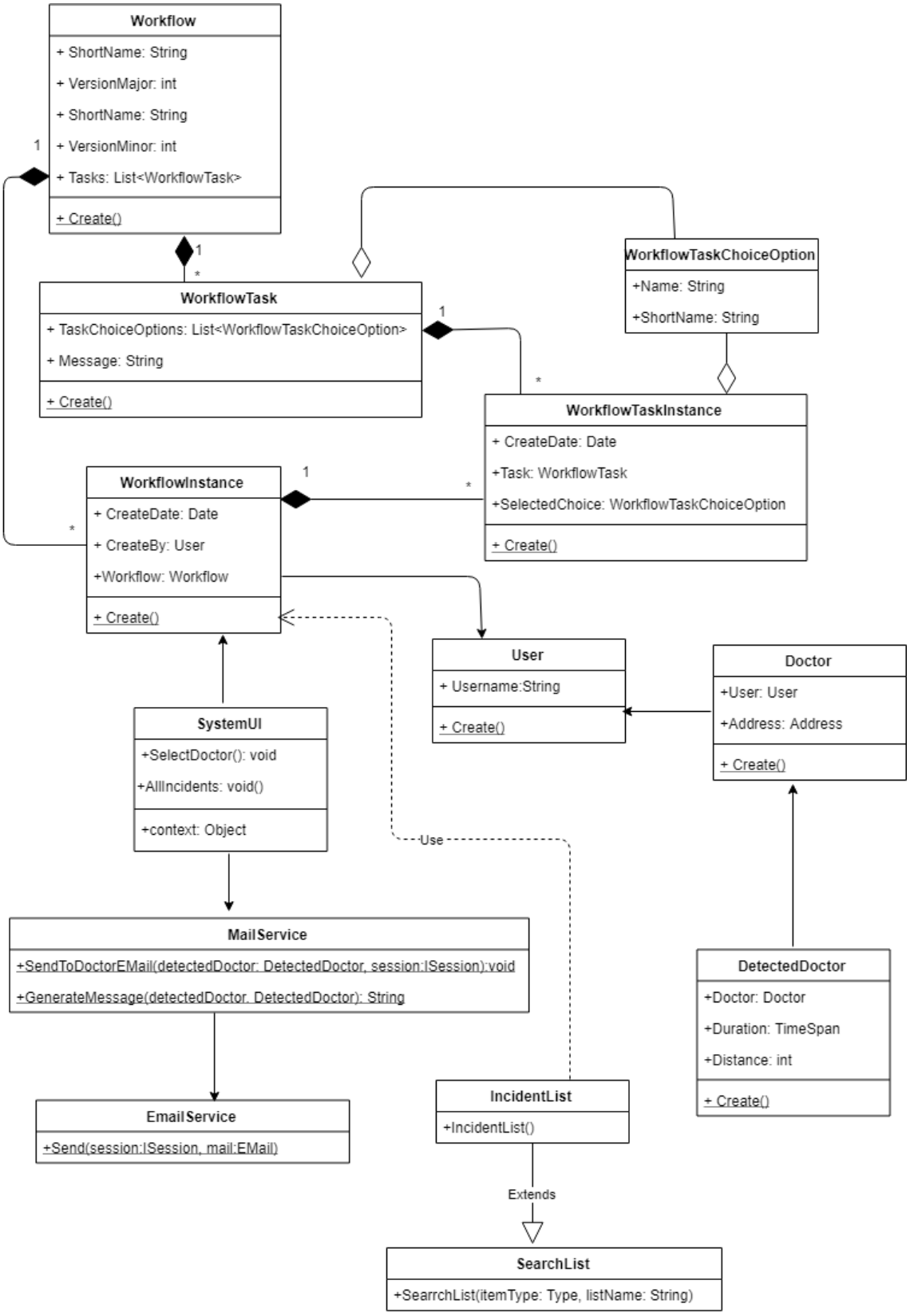


Figure 3.40: OOD diagram of Use Case 6

3.7 Maintaining the doctor's data

Since personal information changes quite frequently, information about doctors needs to be able to be adjusted in a simple manner.

3.7.1 Data Structure

Since the data structure necessary to store a doctor and his data has been designed in section 3.5, no new data objects have to be created through the *Bora Designer*.

3.7.2 Changing a doctor's information

Using the Bora Framework, altering data of a managed entity is straightforward and can be easily accomplished.

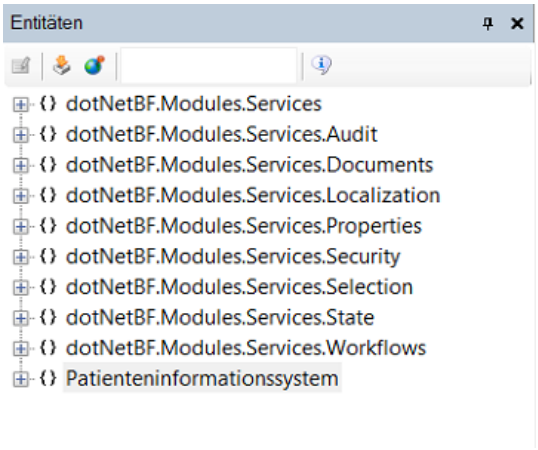


Figure 3.41: List of Entities

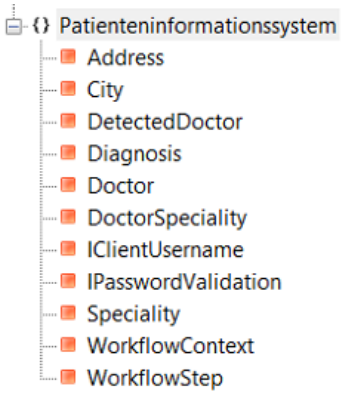


Figure 3.42: Expanding the "Patienteninformationsystem" subfolder

This is largely the same workflow used in 3.5; A list of entities appears to the left of the application upon being started and data regarding the doctor can be changed by expanding the "Patienteninformationsystem" subfolder and selecting either Doctor, DoctorSpecialty, or City. A specific item of the list displayed to the right can then be selected, and be edited using the "Edit" button.

The type of entity item to edit is based on what exactly one wants to change:

Doctor

Upon selecting a doctor item, one can change:

- The doctor's email
- The doctor's first and/or last name
- The doctor's password associated with their account
- The doctor's street name, street number, and city

DoctorSpeciality

This item needs to be edited if the following data changes:

- The doctor's assigned specialty

City

Adjustments can be made here regarding:

- The zip code of a city assigned to a doctor
- The name of a city assigned to a doctor

3.8 Challenges

In this section, we will describe a few challenges that have occurred within our development process, and other roadblocks that have prevented us from moving forward.

3.8.1 Maps and Navigation

Regarding navigation in our application, the original plan has been not to link to a Google Maps URL, but instead to integrate and embed a browser within our GUI that would display the actual Google Maps map, complete with navigation and direction-based routing.

This is theoretically possible: C# XAML exposes a browser element that is able to display websites directly in the program. This, in turn, makes it easy to use and control from within the BORA Framework, tying it together with the rest of the code.

As we have finished the implementation, we quickly noticed that the displayed Google Maps map was not working correctly, as some sections of the map were blank, and the browser didn't have proper panning support. This, as we have learned, has not been an issue with the browser itself, but with the underlying technology. Under the hood, the C# browser utilizes Internet Explorer with an old version of Javascript which has not been updated in a long time, and therefore newer web applications which rely on certain browser features will simply cease to work.

Some C# libraries aim to provide a browser that uses chromium, the rendering engine of chrome, under the hood, and CefSharp is the only one of the bunch without monetary costs. These chromium instances often utilize the latest in technology, supporting the latest HTML, CSS, and Javascript versions. Using CefSharp instead of the native C# browser element does not make a big difference in code, and therefore the changes that have been made when we have decided to implement CefSharp were trivial.

CefSharp works perfectly when integrating it into a test C# application, yet it imposes one interesting restriction: The project needs to be built separately for the target platforms

x64 and x86, which means that the compiled output and application will be different depending on the user's system architecture. Building for separate platforms does not work well with the BORA Framework, as it somehow requires the code to work for both target platforms, meaning that this method would not work.

In the end, we have decided to implement Google Maps functionality using the previously mentioned linking technique, as embedding CefSharp using the BORA Framework is not possible.

3.8.2 Web application

Initially, the idea of the project was to have both a desktop application as well as a web application. However, during our development we stumbled across one obstacle: The BORA framework does not support Microsoft Workflows or any other sort of tool that allows us to display and save questions. Since for that matter we would have to adjust and extend the BORA framework, which our project team is not even allowed to, since it is the framework that BOOM uses for all their projects. Also, adding such a huge feature into the framework is so time-intense and complex that there was no way that any of the employees would do it for only one simple project that is unlike their usual projects and therefore would not likely be needed again.

4 Technologies

4.1 Technologies

4.1.1 C#

C# is an object-oriented programming language which was developed by Microsoft. The IDE (integrated development environment) used to program with was Visual Studio, which is also a Microsoft Product. The main goal is to be as simple and easily accessible as possible and applications can also be written for embedded systems, which was of great importance for this project.

A simple code example looks like this:

```
using System;
namespace ExampleApp
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World!");
    }
  }
}
```

4.1.2 Visual Studio

Figure 4.1: Visual Studio logo

Visual Studio is a fully-fledged IDE primarily used for developing C# applications, although it also supports applications written in C++, Python, and web technologies (*Visual Studio* 2019).

In addition to basic text management and writing functionalities, Visual Studio supports a plethora of added features (*Visual Studio* 2019):

Debugging

Visual Studio sports an integrated debugger used to easily locally test applications. Together with diagnostics, computer resource monitoring and profiling tools, debugging makes finding and removing bugs a breeze.

Testing

Visual Studio supports a multitude of application testing techniques, ranging from simple unit tests that test decoupled parts of the system to IntelliTests that automatically test methods with dynamically generated input parameters.

Collaborating

Visual Studio provides support for many popular version management system right out of the box and displays changes and work items right alongside the code in the editor.

4.1.3 Google Maps

Figure 4.2: Google Maps logo

Google Maps is a web-based service which is able to display maps of geographic regions and sites all around the world (Rouse, 2019). Not only does Google Maps offer a conventional satellite view of the landscape, it also offers road maps and is able to even provide so-called street views, pictures that were taken from vehicles, for specific locations.

The application also provides the additional service of an integrated route planner that is able to efficiently plan routes taking means of transportation and current traffic status into account. Additionally, it provides an API for easy access to the underlying navigation logic if one were to construct a custom UI, making it the tool of the trade for anyone building an application that needs to plan routes between two locations.

4.1.4 Azure DevOps

Figure 4.3: Azure DevOps logo

Azure DevOps is a collaboration platform and version management system for coding. Having a tight integration with Visual Studio, Azure DevOps is the leading platform for managing applications based on C#, providing various features that strive to simplify the development and deployment process (*Azure DevOps* 2019).

Azure Boards

Boards provides a service similar to a Kanban board, where a developer is able to create tasks and assign different statuses to them, creating a simple and easy-to-use system that is able to effectively communicate to the developer what needs to be done. In addition, it provides statistics and tailored views of tasks and projects.

Azure Pipelines

Pipelines is an unobtrusive CI/CD solution, effectively reducing time spent on testing and deployment. By automatically running workflows on certain events, for example, testing the application if the new code is added, time spent not coding is kept to a minimum, and deploying the application on multiple servers is handled automatically.

Azure Repos

Repos, the feature we used the most, is a code management platform similar to Git, handling multiple versions of a codebase, and allowing each developer to work independently. Each developer working on the application publishes changes to the repository using a so-called changeset, containing information of what exactly has been added, removed or changed within the application. As knowledge about who changed what is more easily available, time spent on reviewing and approving code is greatly reduced.

Azure Test Plans

Test Plans provides the team with both planned manual and exploratory testing, making it far easier to detect and prevent breaking code changes.

Azure Artifacts

Artifacts provide an easy way to share and distribute uncoupled libraries among the team, supporting multiple package directories native to languages, such as Maven, npm, NuGet, and more.

4.1.5 WPF/WinForms

C# is a powerful language, and it is mainly used in GUI applications (Mitchell, 2018). Since many developers do not want to start from scratch when using graphical interfaces in their applications , WinForms has been created to simplify development by providing a robust GUI framework.

WinForms has been developed as an equivalent to Java's AWT Framework, and provides components that are easy to build a GUI with (*Introduction to Windows Forms* 2019).

For the release of NET 3.0, a new framework called WPF has been created in parallel, which is based on DirectX rendering and uses XAML for defining the UI, separating the view from the logic layer a bit further than WinForms.

4.1.6 SQL

SQL, or Structured Query Language, is a language used for communication with traditional relational databases. Contrary to other commonly used programming languages, SQL is largely declarative, meaning the developer specifies the result of the operation, not the control flow. SQL consists of multiple sublanguages, the DQL (Data Query Language), DDL (Data Definition Language), and DML (Data Manipulation Language).

Since SQL is largely used for retrieving and storing data, SQL only consists of a handful of operations, such as creating, modifying and deleting a schema, and creating, retrieving, modifying, and deleting data within a schema (*TimesTen In-Memory Database SQL Reference* 2019). The schemas of SQL are called tables, containing data sets called rows. Each table defines columns that restrict what data can be placed in the row, keeping the structure of data consistent.

Retrieving all rows from a table called `person`, where the `age` column of the entries is greater than 18, would look like this:

```
SELECT firstname, lastname
FROM person
WHERE age > 18
```

Applications that use SQL to communicate with the database often use client libraries that utilize special drivers in order to connect (*Connection modules for Microsoft SQL databases* 2018).

4.1.7 SQL Server Management Studio

Figure 4.4: SQL Server Management Studio

Since SQL is only the programming language used in order to communicate effectively with a relational database, a client is needed that handles the connection, sends the SQL and displays the result for manually testing. There are a few clients out there, but SSMS (SQL Server Management Studio) is among the best ones. Developed by Microsoft, SSMS offers various features necessary for managing a database, ranging from basic executions of SQL queries to complex deployment, management and upgrades of data-tier components (*SQL Server Management Studio (SSMS)* 2018).

4.1.8 Git

Figure 4.5: Git logo

Git is a version management system most often used for code (*Git - Distributed even if your workflow isn't* 2019) and solves the detrimental problem of sharing code between multiple team members over email or other file-sharing applications, which often results in faulty versions, people working with outdated files and more. Since Code managed with git is hosted on a server which is the single source of truth, keeping everyone up-to-date on recent changes is almost trivial, and chaos regarding code or file versions is kept to a minimum.

If a developer wants to work on the project, they first need to download the current state of the codebase to the client, which is called a `Pull` in Git terms. If they then make changes to the code and want to upload these changes, they first need to `add` the files to a so-called commit, which can then be finished using a `Commit` action. The developer can specify a certain message for a commit, and since one commit often contains exactly one feature addition or bug fixing, the commit history serves as a kind

of documentation of what has been implemented in the application. Now, the code only needs to be reflected on the server, which is accomplished by a `Push` action.

As more people work on a project at the same time, creating multiple `Branches` is suggested, which allow parallel development without one developer' actions influencing the others.

4.1.9 Github

Figure 4.6: Git logo

Since not everyone wants to work with Git is able to provide his own server, Github is perfect for all those craving the benefits of Git without the monetary cost of hosting a repository. Bought by Microsoft in 2018 (*Microsoft to acquire GitHub for $7.5 billion* 2018), Github is the world's leading git hosting provider, providing both free public and private repository hosting, as well as encapsulated git servers for enterprises (*GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz* 2012).

In addition to Git features, Github also supports so-called `Forks` and `Pull Requests`, which take the functionality of downloading and uploading code to repositories even if one themselves does not own them, making collaboration across the globe far easier. The site also provides features almost identical to social networking sites, such as user profiles, feeds, followers, and stars.

4.1.10 Sourcetree

Figure 4.7: Sourcetree logo

Sourcetree is a software that simplifies the managing of git repositories. The features of the GUI allow to quickly operate git operations and show which files have been changed, by displaying information on any branch or commit. Changes can be staged and disgarded by file, hunk, or line, to make sure that only the right code is being committed. (*Sourcetree* 2019)

4.1.11 GitKraken

Figure 4.8: GitKraken logo

GitKraken is a tool that aims to simplify the git workflow by providing a GUI, allowing the developer to interact with the project by using the mouse instead of writing in the terminal. While many Git users swear by their beloved command line, Git is fairly easy to mess up using the terminal, and there are no visualizations whatsoever, making a git client the perfect tool for beginners and people new to version management (Godinho, 2016).

Gitkraken displays the commit history in a beautiful interactive graph, allowing developers to quickly understand recent changes, and even provides undo/redo functionality in case something goes wrong. An easy-to-use merge tool and a simple code editor is also available within the application, making Gitkraken almost a complete solution for handling git projects.

Since last year, Gitkraken also provides an integrated task management feature called ”Glo Boards,” similar to solutions such as Trello or Microsoft Azure Boards, making it easy for users to manage assignments directly in Gitkraken (*Glo Boards* 2019).

4.1.12 Notepad++

Figure 4.9: Notepad++ logo

Notepad++ is a replacement for the normal Notepad editor and supports a vast number of different languages. The main goals while developing this editor were a high execution speed and a small program size, all while maintaining user friendliness, by optimizing as many processes as possible. (*Notepad++* 2019)

4.1.13 Sublime Text

Figure 4.10: Sublime Text logo

In the process of editing and developing code, the most often used technology is an IDE such as Visual Studio but for some use cases, for example simple configuration file editing, a full-fledged IDE is simply overkill. Such tasks can also be handled by using simple text editors, which do not nearly have as many features as IDEs, but score better in categories such as speed, ease of use, and little to no complexity.

One of the more popular text editors is Sublime Text, ”A sophisticated text editor for code, markup and prose” (*Sublime Text* 2019), a hybrid of simple and easy-to-learn text editors, and functional and feature-heavy IDEs, taking advantages from both sides.

Sublime Text 3 is, due to it being written in C++ and using a custom GUI toolkit (*Sublime Text 2 Scrolling in OSX* 2019), faster than most text editors, handling even files that are multiple megabytes big gracefully, making it the ideal tool for not only editing but viewing large quantities of data.

4.1.14 UML

Figure 4.11: UML logo

Unified Modeling Language is used to create an overview of how a system is modeled. It is used as a guideline to how classes are implemented because it dictates the structural requirements. Diagrams which are created with UML show the entities of a system and what relationships they have with each other. Additionally, the cardinality further describes the relationship between two classes. For example, there can be a 1 to n-ary relationship, like with a group with many group members.

4.1.15 draw.io

Figure 4.12: Draw.io logo

A software like draw.io is very convenient and helpful when diagrams have to be created in a fast and efficient way. The editor is a service that is free of charge and provides a vast number of predefined components like classes or association links.

Whether it is a class-, object-, or activity diagram, all of them can be quickly drawn and edited. Diagrams can be exported in many different formats and can be opened again if there are some changes to be conducted.

4.1.16 MSSQL

MSSQL is a relational database from Microsoft that can also be used as a dataware house. It uses the SQL variant ”T-SQL”, that adds syntax for the use of stored procedures and functions.

We used this database for the project to store all the information needed to create a diagnosis. Also, the BORA framework requires a MSSQL Server to even open an application since a lot of information about the modules are stored in it.

4.2 BORA

BOOM Software AG specializes in tailor-made software solutions and projects. In order to provide absolutely individual products for customers, BOOM introduced the Business Oriented Rapid Adaption. With this technology it became possible to customize software specifically for one customer, irrespective of technology, requirements and time. BORA provides a base framework, which is simply extended in order to meet the demands of the client. (*Boom Software AG* 2019)

BORA Designer This is where everything starts. A new project is initiated with this application. With the designer, the architecture of the project is defined. Next to already existing entities, additional ones can be created which are automatically synchronized with the database. The designer allows the programmer to manage the entities and their properties and with a graphical user interface it is well illustrated what entities exist and what their relationships are.

BORA Client Uses Visual Studio as basis and has an own programming language, which is defined by tags, just like XML. The client provides a variety of classes which are written in this language and these classes can be enhanced with own entries, so that the application fits the requirements. With this client it is possible to access the entities of the designer and the values of their properties can either be set or read. Moreover, the look and the functionality of the application is managed by the client, so that it is unique for every intended purpose.

4.3 Rational Unified Processes

The method we used to organise our project is called ”Rational Unified Processes”. A Rational Unified Process is a software engineering process that provides a repeatable approach to assign tasks in a project team. The goal of this process is to create software of high quality while making it feasible to predict the necessary time and budget. The method of Rational Unified Processing (RUP) can also be seen as a process framework that can be adapted to suit the needs of any software organisation (Kruchten, 2000). There are six best practices that the Rational Unified process follows:

- Developing software iteratively
- Managing requirements
- Using component-based architectures
- Modelling software visually
- Continuously verifying software quality
- Control changes to the software

Iterative Development

Compared to using the waterfall model, iterative development has many advantages. The most notable is the fact that changing requirements can be taken into account. Other advantages include the fact that risks can be mitigated earlier since the system is integrated progressively instead of only one time in the end. Most of the risks and errors in the system are detected during the integration process, so integrating it slowly but steadily reduces the risks. Lastly, developing iteratively has the advantage of creating a relatively robust architecture because with each iteration errors are corrected.

In contrast to the waterfall model, use cases are described to implement into the system. Each use case has iterations that are entirely independent of the ones from the other use cases. In other words, each use case is designed, implemented and tested independently.

For each iteration, the classes that have to be revisited, which subsystems are affected or created and which interfaces need to be modified have to be defined.

Requirements Management

A systematic approach to eliciting, organising and managing requirements is crucial for every modern software development team. It provides better control of complex projects and results in improved team communication.

Architecture and Use of Components

While use cases are essential during the entire process of developing a project with RUP, however, the design activities are centred around the software architecture. The main focus of early developing is to create an executable prototype that gradually evolves into the final product in later iterations.

Modelling

One of the major parts of RUP is to model the system under development. The models are created to understand the problem and simplify the real world based on the needs of the software project. This enables developers to understand a complex system that is otherwise not comprehensible. For that matter, the UML is used to visualise and specify the system in question. These visualisations are used as a blueprint for the project.

Quality of Process and Product

With RUP, there is no specific person responsible for the quality of a product, but rather everyone working on the project. There are generally two kinds of quality:
Product quality is the principal quality of the produced software product. It does not only include the system itself but also all the necessary components that it uses, such as subsystems or architectures.

Process quality describes the degree to which a process is implemented. Furthermore, process quality describes the quality of all the parts of the project that are not the product itself, such as use-case realisations or the design model.

Configuration and Change Management

Iterative development means that there are many changes; it is vital to guarantee that everyone is up-to-date at all times. This does not only apply for the software itself: The project team has to communicate in order to avoid misunderstandings and to be able to work in sync.

Use-Case-Driven Process

Modelling is an essential part of RUP. When starting a new project, a model of the problem is designed. It will be the template used for the solution of the problem. Therefore creating a well-designed model is crucial. In our project, we used the method of use-case modelling. Use cases enable software developers to express the problem in a way that makes it possible for people not working in a technical field to understand their way of thinking.

A use case is defined as a sequence of actions which the system performs that will result in an observable result. This result is of value for a specified actor. An actor is someone outside the system that interacts with it. By splitting up the project into different use cases, an iteration's goals are easier to specify. One use case is treated as a single unit throughout the entire project (Kruchten, 2000).

Project Management

With RUP, there are three main aspects of project management.

- Planning the iterations
- Risk management

- Monitoring the progress

The most important aspect of the ones above is planning the iterations for our project.

Planning the iterations

When planning iterations, it is essential to plan the number of iterations required, how long they should be, what should be covered in each iteration and how to keep progress of an iteration.

The main objectives are to allocate tasks and responsibilities to every team member and to monitor progress so that potential problems can be identified early.

In order to do this, creating an iteration plan for each iteration is vital. Most of the times, there are two iteration plans used at a time. The first one is the one that the project team is currently working with and the second one is the plan for the next iteration. The plan for the next iteration is mainly being built during the second half of the current iteration (Kruchten, 2000).

List of Figures

Acronyms

API Application Programming Interface, a set of functions that allows using features of a system (*Oxford Dictionary* 2019).

AR Augmented Reality, an enhanced version of reality created by the use of technology to overlay digital information on an image of something being viewed through a device (*Merriam-Webster* 2019).

BORA Business Oriented Rapid Adaption, framework technology from BOOM Software AG which enables full customization.

CD Continuous Delivery, is the ability to get changes of all types into production in a quick, safe, and sustainable way (*Continuous Delivery* 2019).

CI Continuous Integration, is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early (*ThoughtWorks* 2019).

GUI Graphical User Interface, interface which is being represented by icons and graphics.

HTTP Hypertext Transfer Protocol, a way of formatting and transmitting messages on the internet (*Collins* 2019).

IDE Integrated Development Environment, is a software application that provides comprehensive facilities to computer programmers for software development (*Wikipedia* 2019).

PINFOS Patient Information System.

RUP Rational Unified Process, s an object-oriented approach used to ensure effective project management and high-quality software production (*Techopedia* 2019).

SDK Software Development Kit, a programming package that enables a programmer to develop applications for a specific platform (*Wepodia* 2019).

UI User Interface, the means by which user interacts with the software.

UML Unified Modeling Language, an object-oriented analysis and design language from the Object Management Group (*YourDictionary* 2019).

XML Extensive Markup Language, a way of marking text to be shown in computer systems (*Cambridge Dictionary* 2019).

Bibliography

Azure DevOps (2019). URL: https://azure.microsoft.com/en-us/services/devops/ (visited on 30/03/2019).

Boom Inside (2019). URL: https://www.incodewetrust.careers/inside/ (visited on 29/03/2019).

Boom Software AG (2019). URL: https://www.boomsoftware.com/de/loesungen/BORA-Technologie.php (visited on 01/04/2019).

Cambridge Dictionary (2019). URL: https://dictionary.cambridge.org/dictionary/english/xml (visited on 01/04/2019).

Collins (2019). URL: https://www.collinsdictionary.com/dictionary/english/http (visited on 01/04/2019).

Connection modules for Microsoft SQL databases (18th June 2018). URL: https://docs.microsoft.com/en-us/sql/connect/sql-connection-libraries?view=sql-server-2017 (visited on 30/03/2019).

Continuous Delivery (2019). URL: https://continuousdelivery.com/ (visited on 01/04/2019).

DocCheck Flexicon (2019). URL: https://flexikon.doccheck.com/de (visited on 28/03/2019).

Gesundheit - Akute Bronchitis (14th Feb. 2018). URL: http://www.gesundheit.gv.at/krankheiten/atemwege/akute-bronchitis (visited on 25/03/2019).

Gesundheit - Brustfellentzuendung (14th Feb. 2018). URL: https://www.gesundheit.gv.at/krankheiten/atemwege/bauchfellentzuendung-pleuritis (visited on 25/03/2019).

Git - Distributed even if your workflow isn't (2019). URL: https://git-scm.com/ (visited on 01/04/2019).

GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz (7th Sept. 2012). URL: https://techcrunch.com/2012/

07/09/github-pours-energies-into-enterprise-raises-100-million-from-power-vc-andreesen-horowitz/ (visited on 30/03/2019).

Glo Boards (2019). URL: https://www.gitkraken.com/glo (visited on 30/03/2019).

Godinho, Douglas (13th Aug. 2016). *Why should you use a GUI tool for Git*. URL: https://medium.com/thedev/why-should-you-use-a-gui-tool-for-git-9059bed72761 (visited on 30/03/2019).

Internisten im Netz (18th Aug. 2017). URL: https://www.internisten-im-netz.de/krankheiten/herzinfarkt/was-ist-ein-herzinfarkt.html (visited on 29/03/2019).

Internisten im Netz - Bluthochdruck (18th Aug. 2017). URL: https://www.internisten-im-netz.de/krankheiten/bluthochdruck/ursachen-risikofaktoren.html (visited on 29/03/2019).

Internisten im Netz - Nierensteine (18th Aug. 2017). URL: https://https://www.internisten-im-netz.de/krankheiten/nierensteine/ursachen-risikofaktoren.html (visited on 27/03/2019).

Introduction to Windows Forms (2019). URL: https://msdn.microsoft.com/en-us/ie/aa983655(v=vs.94) (visited on 30/03/2019).

Kraft, Ulrich (14th Dec. 2017). *Apotheken Umschau*. URL: http://www.apotheken-umschau.de/reizmagen#Welche-Symptome-treten-beim-Reizmagen-auf (visited on 25/03/2019).

Kruchten, Philippe (2000). *The Rational Unified Process, An Introduction. Second Edition*. Addison Wesley.

Leaflet (2019). URL: https://leafletjs.com/ (visited on 30/03/2019).

Leaflet Routing Machine (2019). URL: http://www.liedman.net/leaflet-routing-machine/ (visited on 30/03/2019).

MapBox (2019). URL: https://www.mapbox.com/ (visited on 29/03/2019).

Mapbox Showcase (2019). URL: https://www.mapbox.com/showcase/ (visited on 29/03/2019).

MapFit (2019). URL: https://www.mapfit.com/ (visited on 29/03/2019).

Meine Gesundheit (1st Apr. 2013). URL: https://www.meine-gesundheit.de/krankheit/krankheiten/zwoelffingerdarmgeschwuer (visited on 28/03/2019).

Merriam-Webster (2019). URL: https://www.merriam-webster.com/dictionary/augmented%20reality (visited on 01/04/2019).

Microsoft to acquire GitHub for $7.5 billion (6th Apr. 2018). URL: https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/ (visited on 30/03/2019).

Mitchell, Robin (20th July 2018). *Which programming language to choose for GUIs*. URL: https://maker.pro/custom/tutorial/which-programming-language-should-i-choose-graphics-and-guis (visited on 30/03/2019).

Netdoktor - Lungenentzuendung (13th Mar. 2019). URL: https://www.netdoktor.de/krankheiten/lungenentzuendung/ (visited on 28/03/2019).

Notepad++ (2019). URL: https://notepad-plus-plus.org/ (visited on 01/04/2019).

Onmeda - Pancreatitis (2019). URL: https://www.onmeda.de/krankheiten (visited on 25/03/2019).

Oxford Dictionary (2019). URL: https://en.oxforddictionaries.com/definition/api (visited on 01/04/2019).

Regionalmedien Austria AG (20th June 2018). URL: https://www.netdoktor.de/krankheiten/angina-pectoris/ (visited on 27/03/2019).

Regionalmedien Austria AG - Lungeninfrakt (13th Mar. 2019). URL: https://www.minimed.at/medizinische-themen/lunge/lungeninfarkt-notfall/ (visited on 26/03/2019).

Regionalmedien Austria AG - Magengeschwür (20th June 2018). URL: http://www.minimed.at/medizinische-themen/stoffwechsel-verdauung/magengeschwuer (visited on 27/03/2019).

Rouse, Margaret (2019). *What is Google Maps*. URL: https://whatis.techtarget.com/definition/Google-Maps (visited on 29/03/2019).

Sourcetree (2019). URL: https://www.sourcetreeapp.com/ (visited on 01/04/2019).

SQL Server Management Studio (SSMS) (16th Oct. 2018). URL: https://docs.microsoft.com/de-de/sql/ssms/sql-server-management-studio-ssms?view=sql-server-2017 (visited on 30/03/2019).

Sublime Text (2019). URL: https://www.sublimetext.com/ (visited on 29/03/2019).

Sublime Text 2 Scrolling in OSX (2019). URL: https://forum.sublimetext.com/t/sublime-text-2-scrolling-in-os-x/1205/4#p7238 (visited on 29/03/2019).

Symptome erklaert (May 2018). URL: https://www.symptome-erklaert.de/rippenfellentzuendung-symptome/ (visited on 27/03/2019).

Techopedia (2019). URL: https://www.techopedia.com/definition/3863/rational-unified-process-rup (visited on 01/04/2019).

ThoughtWorks (2019). URL: https://www.thoughtworks.com/continuous-integration (visited on 01/04/2019).

TimesTen In-Memory Database SQL Reference (2019). URL: https://docs.oracle.com/cd/E21901_01/timesten.1122/e21642/state.htm (visited on 30/03/2019).

Visual Studio (2019). URL: https://visualstudio.microsoft.com/?rr=https%3A%2F%2Fwww.google.com%2F (visited on 30/03/2019).

Visual Studio (2019). URL: https://visualstudio.microsoft.com/vs/ (visited on 30/03/2019).

Was ist COPD (2019). URL: http://www.lungenunion.at/index.php/erkrankungen/atemwege/copd/was-ist-copd (visited on 22/03/2019).

Wepodia (2019). URL: https://www.webopedia.com/TERM/S/SDK.html (visited on 01/04/2019).

Wikipedia (2019). URL: https://en.wikipedia.org/wiki/Integrated_development_environment (visited on 01/04/2019).

Wikipedia Perikarditis (8th Mar. 2019). URL: https://de.wikipedia.org/wiki/Perikarditis (visited on 22/03/2019).

YourDictionary (2019). URL: https://www.yourdictionary.com/uml (visited on 01/04/2019).

Illnesses

The following illnesses are the ones that our system recognises. The illnesses can be found in the upper body since the purpose of the project is to recognise whether pain in this body area is life-threatening or can simply be treated by a specialist in that field.

Myocardial infarction

Myocardial infarction, commonly called heart attack, occurs when the tissue of the heart muscle is damaged o to constipation of the coronary artery. moThis also results in an interruption of the blood flow. It is, in any case, life-threatening and therefore it is advised to seek a hospital immediately. Some factors that increase the probability of a heart attack include high blood pleasure, obesity, diabetes, high cholesterol and smoking (*Internisten im Netz* 2017). Common symptoms include:

- trepidation
- nausea
- dizziness
- shortness of breath
- dull pain that can occur either suddenly or slowly over time
- outbreak of sweat

Pulmonary embolism

A pulmonary embolism is a blockage of a pulmonary vascular in the lungs. If an artery is affected, congestion of the blood can occur. It is, in any case, life-threatening and therefore it is advised to seek a hospital immediately.

Some factors that increase the probability of a pulmonary embolism include prior surgery, cancer and constipation of the veins in the hip and leg area (*Regionalmedien Austria AG - Lungeninfrakt* 2019). Common symptoms include:

- dull pain that occurs suddenly
- shortness of breath
- a very intense outbreak of sweat
- shortness of breath
- Coughing up blood

Pleurisy

Pleurisy is the indignation of the pleura. Most of the time, bacteria is the cause. This is why antibiotics are being prescribed most of the time. (*Symptome erklaert* 2018). Common symptoms include:

- a grinding pain
- generally feeling unwell
- fever
- dry cough

Posterior wall infarction

A posterior wall infarction is a special kind of heart attack that occurs in the left heart ventricle. It is life-threatening. Most of the symptoms are similar to the ones of a regular heart attack (*DocCheck Flexicon* 2019). Common symptoms include:

- Fear of death
- shortness of breath
- outbreak of sweat
- dry cough

Angine pectoris

Angina pectoris is a seizure caused by an illness of the coronary vessels. During the seizure, the person is in life-threatening danger. Nitroglycerin sprays are commonly used to treat it (*Regionalmedien Austria AG* 2018). Common symptoms include:

- Stinging pain behind the sternum
- shortness of breath
- A feeling of narrowness, especially in the chest area

Cardiac arrhythmia

Cardiac arrhythmia is a condition during which the heartbeat is irregular. It often occurs in combination with high blood pressure (*Internisten im Netz - Bluthochdruck* 2017). Common symptoms include:

- cardiac pain
- nausea

Pericarditis

Pericarditis is an inflammation of the pericardium. It is a severe illness, so it should be treated in a hospital immediately. Common symptoms include:

- dyspnea
- fever
- stinging pain (*Wikipedia Perikarditis* 2019)

Pleurisy

Pleurisy is the inflammation of the pleura. Most of the times, there is an underlying disease (*Gesundheit - Brustfellentzuendung* 2018). Common symptoms include:

- chest pain that becomes more intense when breathing
- fever
- coughing

Pneumonia

Pneumonia is an infection of the lung. Most of the times, the cause are bacteria, but it can also be viral. Common symptoms include:

- coughing
- shortness of breath
- chest pain (*Netdoktor - Lungenentzuendung* 2019)

COPD

COPD stands for chronic obstructive pulmonary disease. It is more common among elderly people. There is currently no way to treat this illness. People affected by this illness generally do not suffer from any pain (*Was ist COPD* 2019).

Bronchitis

Bronchitis is an inflammation of the bronchial mucosa. It is a viral illness that often has various accompanying effects. Common symptoms include:

- coughing
- fever
- pain during coughing behind the sternum
- increased production of phlegm (*Gesundheit - Akute Bronchitis* 2018)

Irritated stomach

This is an umbrella term for symptoms of no organic cause. Common symptoms include:

- nausea
- heartburn
- lack of appetite
- vomiting (Kraft, 2017)

Flu

The flu is a viral infection that is most common during winter. It is also called influenza. It generally occurs quite suddenly. Common symptoms include:

- high fever
- limb pain
- dry cough
- sore throat
- stomach aches

Gastritis

Gastritis is an inflammation of the gastric mucosa. Various behaviours increase the risk of suffering from it, such as stress, an irregular diet or an unhealthy consumption of alcohol. Common symptoms include:

- nausea
- lack of appetite
- a swollen stomach

Gastric ulcer

Gastric ulcer is a defect of the gastric mucosa that is the result of an imbalance between the gastric acid and the protective functions of the gastric mucosa (*Regionalmedien Austria AG - Magengeschwür* 2018). Common symptoms include:

- pain in the central stomach area
- nausea
- vomiting

Duodenal ulcer

A duodenal ulcer is a benign tumour in the duodenum. It often occurs together with the gastric ulcer. Common symptoms include:

- acute stomach pain
- nausea
- vomiting
- black defecation (*Meine Gesundheit* 2013)

Appendicitis

Appendicitis is an inflammation of the appendix. Common symptoms include:

- fever
- nausea
- vomiting
- stomach aches in the lower right area

Gallstones

Gallstones typically cause spasmodic aches below the right costal arch. The pain randomly appears and disappears and radiates to the stomach area.

Pancreatitis

Pancreatitis is the inflammation of the pancreas(*Onmeda - Pancreatitis* 2019). Common symptoms include:

- Pain in the upper left stomach area that radiates to all directions
- fever
- vomiting

Splenomegaly

Splenomegaly describes a condition where the splenic is expanded. People suffering from it generally have a feeling of pressure in the upper left stomach area. Most of the times the cause is an infection, but it can also be the result of other blood-related illnesses.

Kidney stones

The pain caused by kidney stones is either wave-like or stinging. Often, the pain is combined with nausea and vomiting (*Internisten im Netz - Nierensteine* 2017).

Scoliosis

Scoliosis is a medical condition in which a person's spine is in a defective position. It causes a stinging pain in the area around the groin and loin. There are various reasons it occurs.