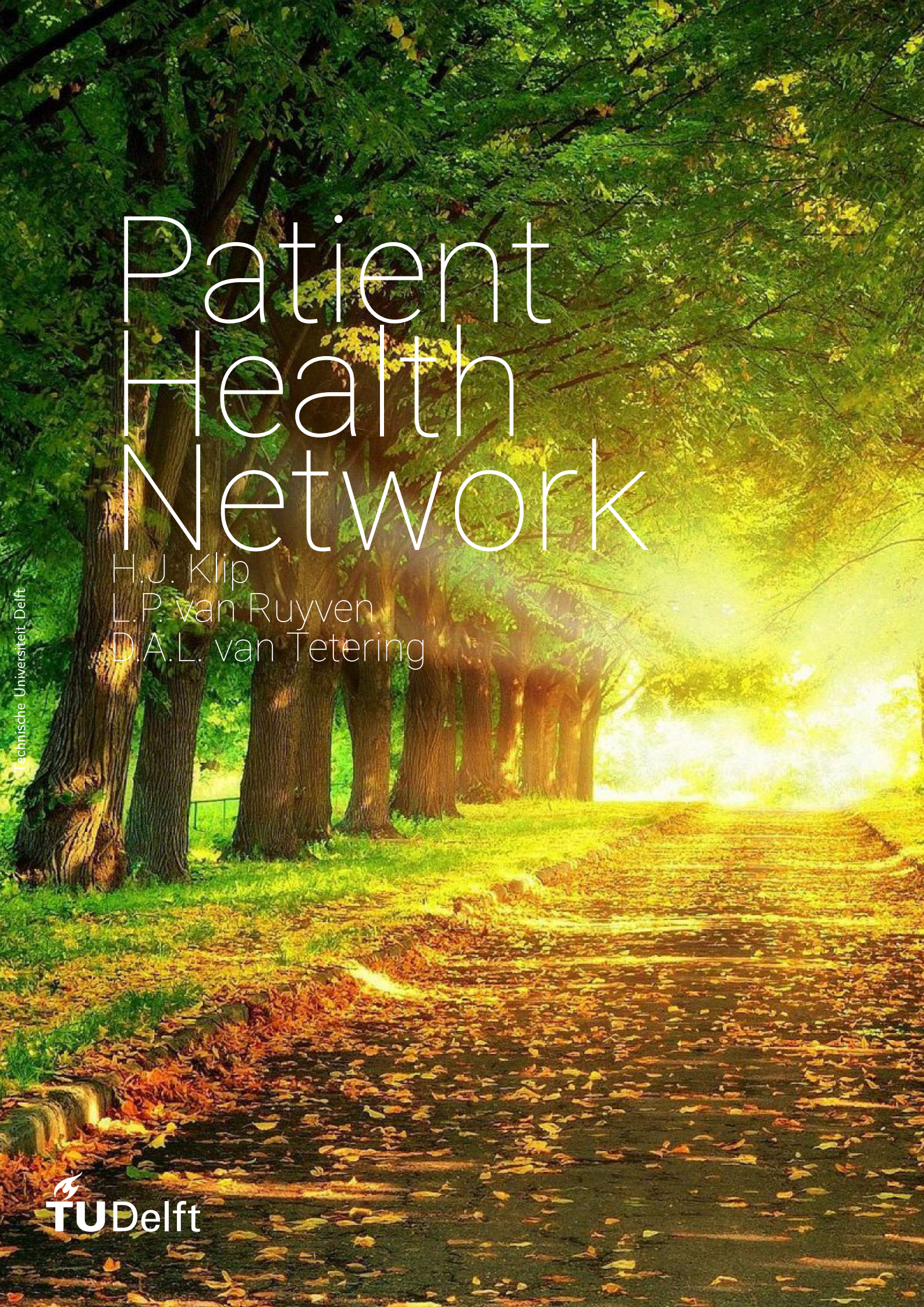

Patient
Health
Network
H.J. Klip
L.P. van Ruyven
D.A.L. van Tetering
Technische Universiteit Delft
TU Delft

Patient Health Network

by

H.J. Klip
L.P. van Ruyven
D.A.L. van Tetering

to obtain the degree of Bachelor of Science
at the Delft University of Technology.

Group members:	H.J. Klip
	L.P. van Ruyven
	D.A.L. van Tetering
Project duration:	April 24, 2017 – July 5, 2017

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft Delft University of Technology

IVIDO
mijn weg naar vitaliteit en gezondheid

Preface

This Bachelor Thesis is written by Roy Klip, Noor van Ruyven and Daphne van Tetering to conclude the Bachelor Project as part of the Bachelor Computer Science Program. Over the course of three months, we developed a Patient Health Network in assignment of Ivido B.V..

We would like to thank everyone at Ivido B.V. for their support during the project, especially Tristan Garssen and Hans Niendieker from Ivido B.V. for their coordination.

Furthermore, we would like to thank our TU Delft supervisor, Felienne Hermans from the Software Engineering Research Group at Delft University of Technology, for her enthusiasm and guidance during the project, as well as her clear feedback and expectations.

Roy Klip
Noor van Ruyven
Daphne van Tetering
Delft, July 2017

Contents

1 Summary **1**

2 Introduction **3**

3 Problem Definition **5**

4 Research **9**
4.1 Moodle . . . 9
4.2 Messaging system . . . 9
4.3 Structure . . . 10
4.3.1 Output classes . . . 10
4.3.2 Mustache files . . . 11
4.3.3 Asynchronous Module Definition . . . 11
4.3.4 AJAX and external functions . . . 11

5 User Interface Design **13**
5.1 Dashboard . . . 13
5.2 Contact section . . . 14
5.3 Invitation section . . . 15
5.4 Message section . . . 15
5.5 Advanced Search . . . 16
5.6 Profiles . . . 17

6 Software Architecture **19**
6.1 General . . . 19
6.2 Output Classes . . . 20
6.3 Mustache files . . . 21
6.4 Asynchronous Module Definition . . . 21
6.5 Database structure . . . 21
6.6 Version Control . . . 22

7 Ethics **23**
7.1 Privacy in messaging . . . 23
7.1.1 Admin capabilities . . . 23
7.1.2 Non-contacts . . . 23
7.2 Extension to other parts of the system . . . 23

8 User Testing **25**
8.1 Pilot . . . 25
8.2 Questionnaire . . . 25
8.3 Results . . . 25

9 Process **27**
9.1 SCRUM . . . 27
9.2 Development resources . . . 27
9.2.1 Google Drive . . . 27
9.2.2 GitHub . . . 27
9.2.3 Software Improvement Group . . . 28
9.3 Individual Reflection . . . 29

10 Conclusions **31**
11 Recommendations **33**
A Info sheet **35**
B Project Description **37**
C Research Report **39**
D Pilot **47**
E SIG feedback **57**
Bibliography **59**

1

Summary

One in every five people in the Netherlands is suffering from a chronic illness. To help these people to a healthier and more manageable lifestyle, Ivido has created an online platform on which patients can seek for advice or join health programs. On this platform, a range of professionals with specialties are available for help. A health network is required to not only find the person with the right knowledge but also to connect and communicate with them.

In this report, a newly developed Health Network and the research done to create it will be discussed. To create the Health Network Moodle's current message system was enhanced with several features to turn it into a fully functional network tool. One of these features is the ability to invite other users into the current user's network and the ability to see and respond to received invitations. Besides this, the network also has group conversations, file sharing, and a functionality that allows the user to search for users using filters and categories.

Besides the previously mentioned features, there are other features that have not been implemented within the timespan of this project, such as notifications, e-mail invitations and a quick-add possibility.

The testing of the network plugin was done during a set up pilot phase. During this pilot a group of people from multiple disciplines was asked to perform actions to test the different components of the tool. Their feedback was used to improve both functionality and usability.

2

Introduction

Over 2.2 million people in the Netherlands have a form of chronic pain. Not only does this have a huge impact on their on their daily functioning and quality of life, it also impacts the lives of the people around them. A chronic pain patient often visits over ten different specialists, sometimes even as much as twenty-five. This often leads to a widely spread out health path where different specialists often have different opinions, which does not lead to a durable recovery or a relief from the pain.

The group has been asked to develop a Patient Health Network (*Patient Gezondheids Omgeving*) for patients with chronic pain. The goal of this network is to increase the patient's self-reliance and to make the communication between patient and specialist and between specialists easier and more efficient.

The Ivido platform is build using a course management system, Moodle. This is an open-source online learning environment, written mainly in PHP, but JavaScript, CSS and Mustache as well. These languages are also used in the Patient Health Network.

In this report, documentation can be found of our progress as well as our findings during this project. Also, the pilot, conducted at the end of the project to test our network and to evaluate its functionalities, is included. Last we make some recommendations as to what can be improved and implemented to develop this the network further.

3

Problem Definition

The Ivido platform has been developed over the course of two years and is already being used by patients. Within the Ivido platform, there is a message area where users can send messages to each other, one-on-one. However, any user can send messages to any other user: there is no way to authenticate to which user messages can be send. Since the Ivido platform is used by patients and professionals, privacy is important. Because of this, it should not be possible to send messages to a user who hasn't given authorization for this. To realize this, Ivido has asked for a Patient Health Network in which two users can establish a connection between them, with two-sided authentication, after which they can send each other messages. Besides this, the Patient Health Network should also allow users to create group conversations and view profiles of other users.

Problem Analysis

After analyzing the problem, the following MoSCoW list was made, together with our client.
In this list, the following terms are used:

User (stakeholder): a user or stakeholder is someone in the target audience. The target audience consists of patients, professionals and organizations in the Netherlands, who are suffering from a chronic illness or are providing health care for people with chronic illnesses.
Health Network: the plugin that will be developed for the Ivido platform as described in the introduction of this chapter.
Plugin: an independent software-extension which can easily be installed to an existing system.
Dashboard: the Dashboard is the main page of the plugin, where all other components are shown.
Network: a collection of all the users the current user has established a connection with.

Must Haves

In the Must Haves-section, we have defined the core features our Health Network needs to function.

1. **View profiles**
   User story: As a user, I want to view a contact's profile.
   Analysis: To meet this requirement, profile pages and a function to display the profile of a user have to be created.
2. **Search contacts and messages**
   User story: As a user, I want to quickly find the contact or message I am looking for.
   Analysis: To meet this requirement, a search function that searches for contacts and messages has to be written, and the results have to be displayed on the Dashboard.
3. **Send connection requests**
   User story: As a user, I want to send other users connection requests to add them to my network.
   Analysis: To meet this requirement, a functionality allowing users to send each other connection requests has to be build.
4. **Approve connection requests**
   User story: As a user, I only want to connect to another user after I have given approval.

Analysis: To meet this requirement, a connection should only be established after both sides have given approval for this.

5. **Communicate with other users**
   User story: As a user, I want to communicate with other users in my network.
   Analysis: To meet this requirement, we have to extend the currently available message system to ensure messages can only be send to users that have established a connection.
6. **Visual representation of connections**
   User story: As a user, I want to have a clear overview of the users that are currently in my network.
   Analysis: To meet this requirement, a section where current connections are shown with links to their messages and profiles needs to be created.
7. **Dashboard**
   User story: As a user, I want all components of my network, such as contacts, invitations and messages, to be shown in a clear overview.
   Analysis: To meet this requirement, a dashboard that encapsulates all views of the contacts, invitations and messages has to be created.
8. **Share documents with other users**
   User story: As a user, I want to share documents with other users in my network.
   Analysis: To meet this requirement, we need to extend the new message system to ensure documents can be send.
9. **Group conversations**
   User story: As a user, I want to be able to discuss certain problems or cases with more than one user.
   Analysis: To meet this requirement, the current message systems has to be extended with group conversations.

Should Haves

In the Should Haves-section, we have defined features that are not essential for the Health Network to function, but would increase functionality and usability.

1. **Forum discussion**
   User story: As a user, I want to discuss or share important information in a way that everyone can advise or comment on the posed problem.
   Analysis: To meet this requirement a forum functionality has to be developed. The difference between forums and group conversations is that forums are open to everyone in a network, whereas group conversations are only accessible to members added to the conversation.
2. **Notifications**
   User story: As a user, I want to be notified after my connection request has been accepted by another user or after I have received a message from another user.
   Analysis: To meet this requirement, the currently available notifications have to be extended to match our clients needs.
3. **Invite new users**
   User story: As a user, I want to be able to invite users that do not have an account yet to use the Ivido platform.
   Analysis: To meet this requirement, if the user cannot be found, a link must be send to a user to invite them to create an account on the Ivido platform.
4. **Quick-add feature**
   User story: As a user, I want to add other users in my program to my Health Network fast and easily.
   Analysis: To meet this requirement, a button has to be added in the program of a user, if there is another user participating in that program, to which the user is not connected yet.
5. **Advanced search**
   User story: As a user, I want to find users based on other criteria than their name, such as their profession.
   Analysis: To meet this requirement, an advanced search function that filters or searches on other fields than name has to be developed.

Could Haves

In the Could Haves-section, we have defined features that would be a good extension of the Health Network but aren't necessarily in the scope of the project.

1. **Budget management integration**
   User story: As an organization, I want to be able to see how much is left of my budget and how it was spend.
   Analysis: To meet this requirement, we need to make it possible to link the Health Network to the Budget System.
2. **Available programs**
   User story: As a user, I want to be able see which programs might be suitable for me to participate in.
   Analysis: To meet this requirement, we have to show which programs are available to participate in within the network.
3. **Budget assignment**
   User story: As an organization, I want to be able to assign the professional of the organization a budget, that can be used to purchase programs for their clients.
   Analysis: To meet this requirement, a system that distributes budget among professionals, based on the organizations preferences, has to be created.
4. **Invitational message**
   User story: As a user, I want to send a message when sending someone a connection request.
   Analysis: To meet this requirement, the connection function has to be extended with the ability to send someone a message.

Won't Haves

In the Won't Haves-section we have defined features that are not feasible within the duration of the project or are out of the scope of the project.

1. **Webshop**
   User story: As a user, I want to able to purchase new programs to participate.
   Analysis: To meet this requirement, a webshop in which available programs are shown and can be bought has to be build.
2. **Budget transactions**
   User story: As an organization, I want to be able to increase the balance of my account, so members of the organization are able to purchase new programs.
   Analysis: To meet this requirement, the Budget Management System has to be connected to a payment system and secure the transactions.

4

Research

In this chapter, the research done to find out the best way to execute this project is discussed. It mostly concerns the Moodle environment and the Moodle messaging system. Since we were unfamiliar with both of them, a lot of research went into their structure and inner-workings.

4.1. Moodle

Moodle stands for Modular Object-Oriented Dynamic Learning Environment and is an open-source online learning environment written mainly in PHP, supported by other languages such as Javascript, Mustache and CSS. It consists of two main parts: the core and the plugins. The core is maintained by developers from Moodle Pty Ltd and is updated regularly. The plugins are developed by developers all around the world and are uploaded to the Moodle community, where anyone can reuse or adapt a plugin to their needs. When creating a plugin, the functionality of a Moodle system can be extended while the core remains untouched, which means that the core can still be easily updated without disruption from other development. Since Ivido prefers to keep up with the latest Moodle version, we decided to develop our project as a plugin.

Moodle uses the naming convention Frankenstyle, which separates parts of names with an underscore. So our plugin will be called my_network plugin or my_network, but for convenience we will use '(my) network' onwards in this report.

4.2. Messaging system

One of the parts of our network plugin is the ability to send messages to a user the current user is connected with. Moodle already has an existing messaging system, but besides its database structure for messaging, most of its functionality can not be re-used. This is because of differences in how contacts can be found, added to your contact list and messaged.

In the current messaging system every user can be found and directly added to a contact list and messaged. However, for privacy reasons the network plugin will provide adding contacts on mutual agreement. In other words, a user can send a connection request to another user to be his or her contact and the other user has a choice to either accept or reject this request. This way a user can only send messages to people he is connected with.

Besides this, in the current messaging system it is not possible to have group conversations or share documents, which are Must Haves for our network. Therefore, the database used by the current message system also is not suitable to support group conversations. It stores two user ids, one for each user in a conversation. If we were to use this structure and change one user id into the group id, we would not be able to keep track of which user has read which message in a group. Therefore, we have to find a way to use most of the existing structure and extend this with the possibility to have group conversations and file sharing.

4.3. Structure

The Moodle core has recently been updated to version 3.2, which means that the messaging system has also been updated. This new version of Moodle makes use of PHP classes, Mustache files and AMD JavaScript with AJAX calls. From the message page a user's contact- and messages-data is retrieved using functions in the library file (*lib.php*). This data is used to create objects such as conversations and contacts to render and visualize the data. During the rendering phase, the objects are processed using their corresponding Mustache file, which generates HTML based on the object's data-properties. The generated HTML is then displayed as content on the page.

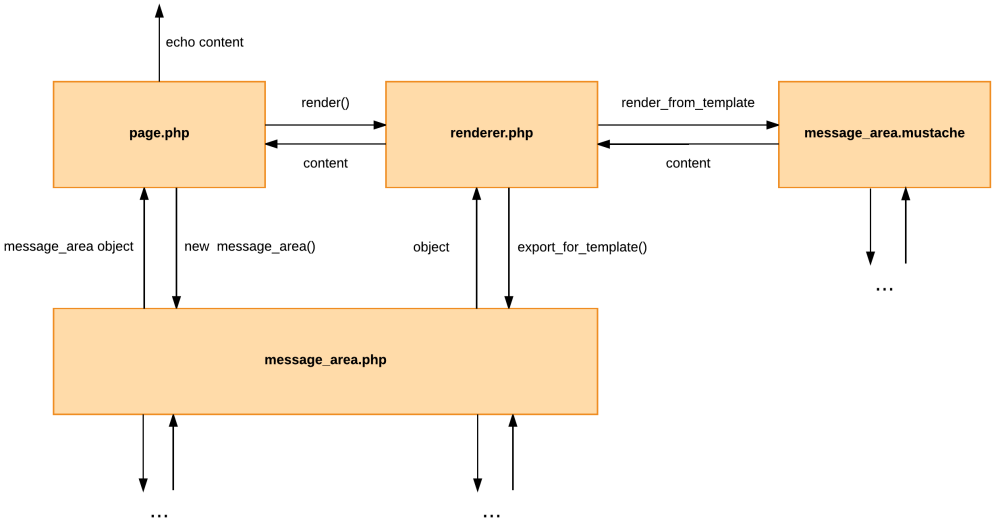


Figure 4.1

4.3.1. Output classes

After receiving data from the library functions, the page calls the constructor function from the *message_area* class (*message_area.php*). This class uses the data to construct a message_area object with the provided properties as variables. Since some of these variables are other output-class objects, the constructor function calls their constructor functions to instantiate those objects. This happens with contacts, conversations and messages.

When all objects are instantiated, the page passes the objects to the renderer which calls the *export_for_template* function of the *message_area* object, which is a function present in all output classes. This function returns an object that can be used to provide data for the Mustache files. This function follows the same principle as the constructor does, which means that each *export_for_template* function of the previously instantiated objects is called. When this process is finished, all data is ready to be given to a Mustache file [3]. This process is visualized in figure 4.2.

Figure 4.2

4.3.2. Mustache files

As mentioned in section 4.3.1, all data is exported to a template, which is a Mustache file. Mustache files generate HTML based on provided data and can also call JavaScript functions The files can also include JavaScript files and call JavaScript functions [4].
The Mustache files make use of a backtracking strategy as well. Only the *message_area* Mustache file is called from the page, after which this file calls the other necessary Mustache files to generate the correct HTML. This process is visualized in figure 4.3.

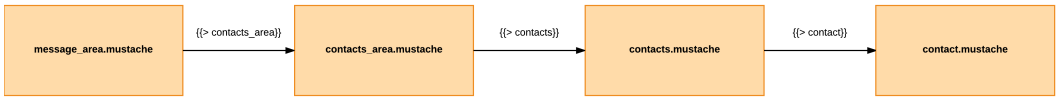


Figure 4.3

4.3.3. Asynchronous Module Definition

In the *message_area* Mustache file, the *message_area* JavaScript is called, which uses Asynchronous Module Definition (AMD). This means that JavaScript modules can be loaded asynchronously to boost performance and usability.

When calling the *message_area* JavaScript (*message_area.js*), functions that can be used throughout other modules and event handlers are defined. From this file, the other modules used in the messaging system are initialized as well. Every module handles a different part of the page, so the contact JavaScript (*contacts.js*) handles events affecting the contact list. These events can be local or global. The global events can be found in the Events file (*events.js*) and can be triggered by other modules, local events are module-specific. For example, if a contact was removed from the contact list by clicking the 'remove contact' button on the profile, a global event defined in the profiles JavaScript (*profiles,js*) is called, but it is handled in the contacts JavaScript, resulting in the contact being removed from the list.

4.3.4. AJAX and external functions

Some events in the JavaScript modules require a modification in the database. To take care of this, AJAX calls are used to call PHP functions in the external library. These functions will process the modification and are able to return variables as well, such as a success boolean. An external function consists of three mutually dependent functions. One for defining the structure of the parameters, one for defining the structure of the result and one to carry out the modification. The first two functions will make sure the wanted type of the parameters is given and the wanted type of result is returned. This process is visualized in Figure 4.4.

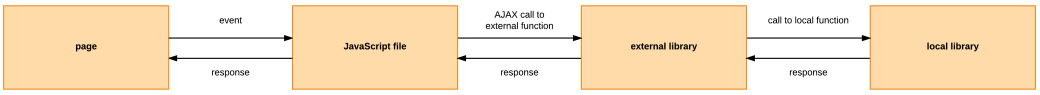


Figure 4.4

5

User Interface Design

An important part of our plugin is the user interface. The network is designed for every day use by different kinds of people, such as patients with chronic pain and the professionals helping them. Therefore the user interface has to be nice on the eyes and easy to use. In this chapter we will discuss the user interface of our network plugin and the placement of the functionalities on the page.

5.1. Dashboard

Must Have 7: Dashboard states that we have to create a dashboard with a clear overview of all the components.
The dashboard is the main page when 'My Network' is opened. The page is divided into three sections: the contact section and the invitation section in the left column and the message section in the right column, as can be seen in figure 5.1.

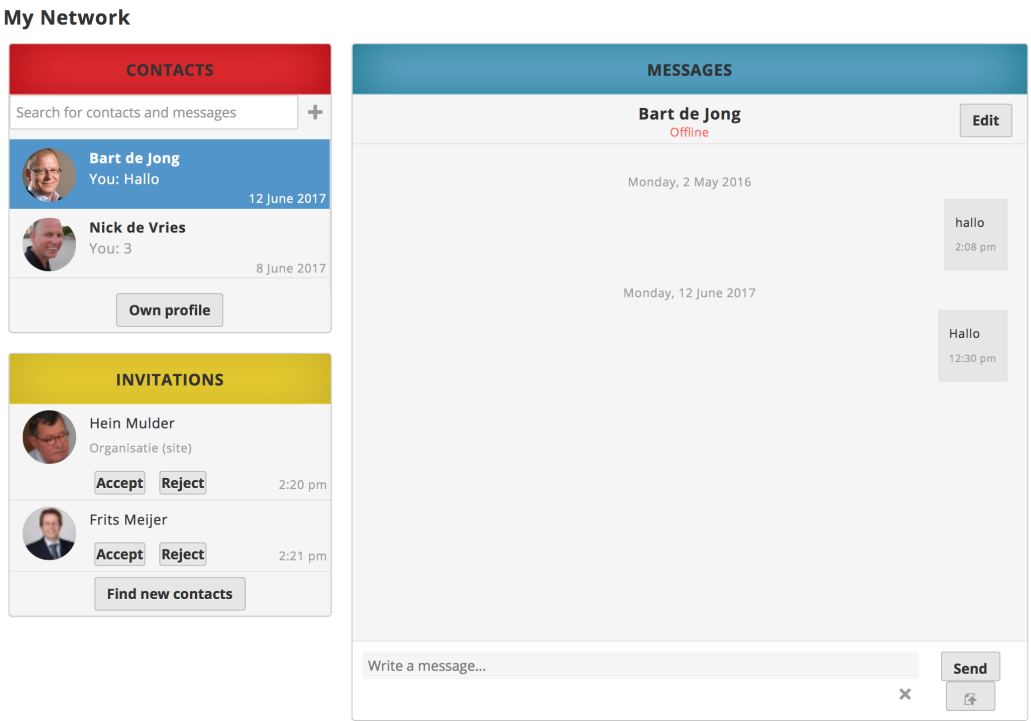
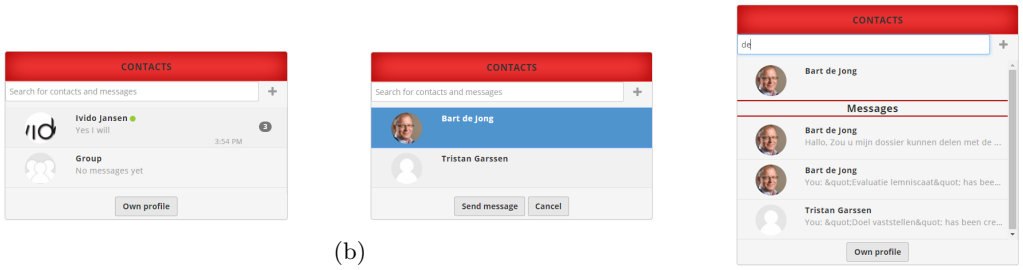


Figure 5.1

5.2. Contact section

The contact section is built up of a header, indicating that it is the contact section, a search bar, a plus icon, a list of current conversations and a button to go to the user's own profile.

To meet *Must Have 6: Visual representation of connections* we made a list of all conversations a user currently has. This section is shown when a user is not searching or starting a new conversation as shown in Figure 5.2.a.
If the user has no conversations yet, a message is displayed stating that there are no conversations. For each conversation the picture of the user or the group is displayed with his or her name beside it. A green dot is added next to the name if this person is online. Underneath the name, the last message send in the conversation is displayed. If this message was send by the current user, 'You:' is displayed before the message, if the message was send by someone else the name of the user who send it is displayed together with the message. In the right bottom corner the date and time when the message was send is shown. If there are unread messages in the conversation, they are indicated by a gray notification next to the last message with a number that shows how many unread messages there are, this is also shown in figure 5.2.a.
When a user clicks on a certain conversation, this conversation is shown in the message section next to the contacts and the user that was selected will be marked with a blue background, this is visible in figure 5.2.b.



(a) (b) (c)

Figure 5.2: The contacts section

To meet *Must Have 2: Search contacts and messages* we added a search bar where a user can type a (part of a) name or conversation. The results will be dynamically shown directly underneath the search bar and contain all contacts whose name contains the search string and all messages that contain the search string as shown in Figure 5.2.c. If the result is a contact, the user's picture and name are displayed and when clicked on, it opens either the existing conversation with this person or starts a new conversation with them. If the result is a part of a conversation, the picture and name of the contact are displayed as well as the message containing the searched string. When this result is clicked, the conversation with this contact is shown in the message area. When the search query has both contact and conversation as results, it first displays the contacts, then a header indicating that the results underneath it are messages and then the conversation results. If there are more results than fit in the box, a scroll bar is added on the right, this is also shown in figure 5.2.c.

Next to the search bar a plus icon is located. When clicking on this icon, the conversation list will be replaced by a contact list. The contacts that are displayed will contain their name and picture as shown in Figure 5.2.b. One or more contacts can be selected (again marked with a blue background) to form a new conversation or proceed with an existing one. At the bottom are two buttons, one for starting the conversation and one for canceling and returning to the conversation list. When the 'Send message' button is pressed while multiple contacts are selected, this means that a user wants to create a group conversation and a dialogue will show up. In this dialogue a group name and picture can be chosen. If the dialogue is confirmed the conversation list will be shown again along with the new group conversation. With this functionality *Must Have 9: Group conversations* is added to the dashboard.

At the bottom of the section a button to navigate to the user's own profile can be found. When pressed, the current user's profile is shown in the message area. Profiles will be discussed more thoroughly in section six of this chapter.

5.3. Invitation section

The invitation section contains a header, an area for the invitations and a button to search for new contacts as shown in Figure 5.3. When a user has no unanswered invitations, a message is shown saying so. An invitation consists of the picture and name of the user sending the invitation and two buttons: one to accept and one to reject the invitation. When the accept button is pressed, the invitation is removed from the section, the message area shows the profile of the new contact and the new contact can be found with the search bar in the contact area. With this, *Must Have 4: Approve connection requests* is met. When the reject button is pressed, the invitation is removed from the section. In the right bottom corner the date and time of when the invitation was received are displayed. An invitation can be selected by clicking on it without clicking on one of the buttons. When an invitation is selected the profile of the user sending the invitation is shown in the message section.

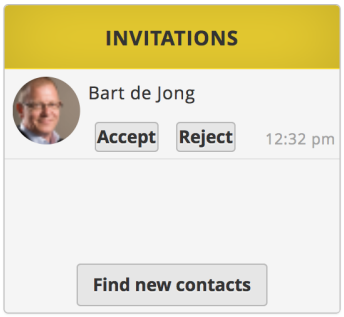
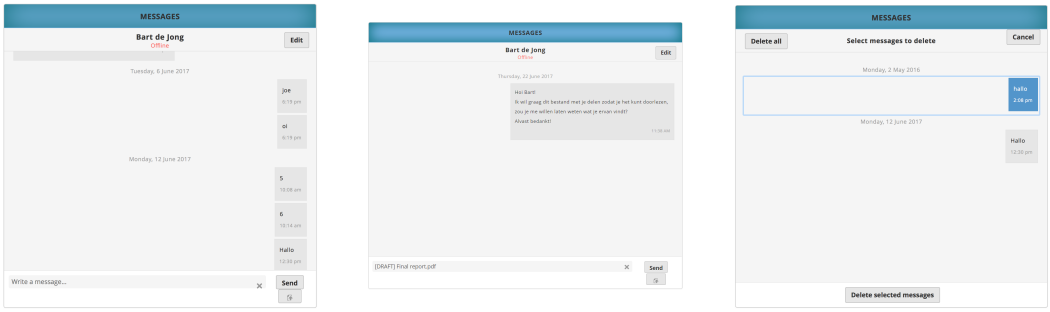


Figure 5.3

At the bottom of the section there is a button for searching for new contacts. When this is pressed, an advanced search option is shown in the message area, this will be explained in more detail in section 5.5.

5.4. Message section

When an existing or new conversation is opened in the contact list, it appears in the message section. The message section consists of a header, the name and status of the contact a user is contacting, an edit button, the area containing the actual messages, an input bar where a message can be typed, a send button and an add attachment button. These parts can all be seen in figure 5.4a.



(a) (b) (c)

Figure 5.4: The contacts section

The name of the contact has a link to their profile, where their personal information can be seen and where a user can block or remove them. Underneath the name and status is the area with the actual messages. The messages send by the current user are on the right side of the area and the messages from the other user are on the left. A message consists of its text and the time it was send. For each day messages have been sent there is a header with the date in it. The area also has a scrollbar to scroll to older messages. Underneath the message area there is an input bar where a message can be written, which will be send when either 'Enter' or 'the Send button' is pressed. Thereafter, the text in the bar will disappear and will appear in the message area in the right bottom corner. To achieve *Must Have 8: Share documents with other users* we added a file upload button next to the text input bar, which will activate a file browser pop-up where a file can be selected from the computer to share. The name of the selected file is then put in the input bar where it can still be removed with the 'x' button next to it or send it by pressing Enter or the Send button. This is shown in figure 5.4.b.

It is also possible to delete messages that have been sent, but these messages are only removed from the current user's screen, the other person can still see the messages unless he or she deletes them as well. Starting the process can be done by pressing the Edit button on the right upper corner. After

which the header with the name and status changes into a delete header with a button to delete all the messages in the conversation and a button to cancel the deleting operation. When deleting, every message in the message area can be selected,after which it is marked blue, and on the bottom of the area there is a button to remove the selected message(s). This is shown in figure 5.4.c.

5.5. Advanced Search

When a user wants to search for new contacts and the 'Find new contacts' button is pressed, the message area changes to the advanced search area as shown in figure 5.5.

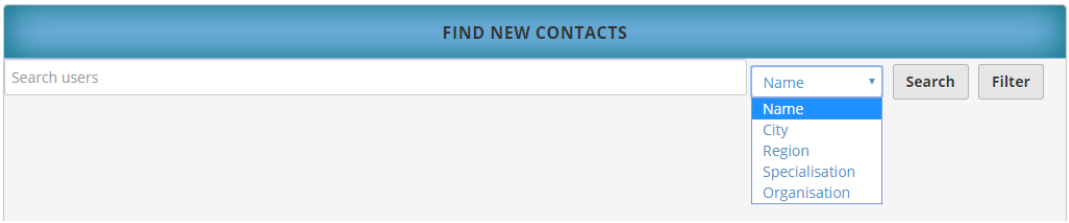


Figure 5.5

This area has a search bar with a drop down menu next to it where a user can decide in what category he or she would like to search or filter as defined in *Should Have 5: Advanced search.* There are currently five categories: name, city, region, specialization and organization. Furthermore, there are two buttons: Filter and Search. With Filter a filter can be added to the search query, that way a user can search on multiple categories at once. For example, when searching for a physiotherapist in the region Utrecht with the name 'Hein' as shown in Figure 5.6.

Figure 5.6

When a user presses the Filter button, a button with the category and the search term will appear underneath the search bar. A user can remove this filter again by clicking on the filter. When Search is pressed, the search query will be executed and the results will be shown. The results show the name, role and picture of the found users. When a result is clicked, the profile of that user is shown.

5.6. Profiles

For *Must Have 1: View profiles* we added the profile functionality. Upon opening 'My Network', a user's own profile is shown in the message area, as visible in figure 5.7.a. A profile consists of a header, a profile picture, a name and status, and the region, city and description if given by the user. If the contact has the role of professional at system level his or her profile can also contain their specialization, the organization they work at and their position at that organization. If the contact has the role of organization at system level their employees (professionals) are shown at the right side of their profile with links to the employees' profiles. These are the general components of a profile, but there are different situations where the profile is shown:

- Own profile: there is an Edit button in the right upper corner that directs to a page where a user's information, such as region and city, can be filled in (figure 5.7.a) or updated (figure 5.7.b).
- The profile of someone in a user's network: there are two buttons in the upper right corner: Block contact and Remove contact, as visible in in figure 5.7.c. When the remove button is pressed, a pop-up appears where the removal can be confirmed or canceled to ensure that the right decision is made. This is necessary because removing a contact can only be reversed by sending a new invitation, which has to be accepted again. On the bottom of the profile there is a Send message button that opens a (new) conversation with this person.
- The profile of someone who has been blocked: there are two buttons in the upper right corner: Unblock contact and Remove contact. Since this person is blocked, neither of the users can send messages to each other, so there is no Send message button on the bottom.
- The profile of someone who is not in a user's network: there is an Add contact button in the upper right corner to invite this person into the user's network. When this button is pressed, it changes into a Cancel connection button, where the invitation can be canceled.

(a) (b) (c)

Figure 5.7: Profiles

6

Software Architecture

This chapter describes the implementation architecture of the components of the system described in the previous chapter. In general, each component consists of one or more php output classes, one or more Mustache files and one or more AMD-modules. All the components are styled using one css-file, according to the Moodle guidelines. The relations between these files can be visualized with the class diagram shown in figure 6.1.

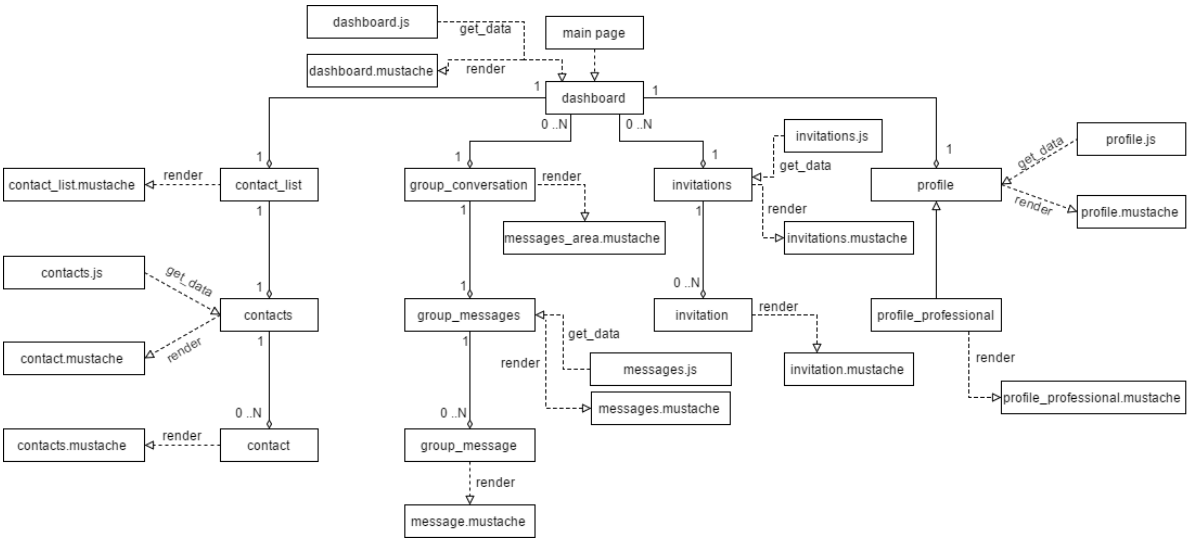


Figure 6.1

6.1. General

When opening the plugin in the Ivido platform, the Dashboard is shown with the Contact Section and the user's own profile on display. On the back-end this means that the created instance of the my_network block has gathered all the data needed to display. This can be described as the following time-line:

- The block instance calls the index file *(block_my_network_page.php)*, which is then executed from top to bottom. Within this file, methods in the library file *(lib.php)* are called to retrieve the data needed to fill the contact section, invitation section, message sections and profiles.
- When the data is retrieved, output objects are created by calling their constructors. These objects are used when the page is rendered and contain the data associated with the class. For example, the class *contact_list* contains only a list of contacts, which contains objects of the *contact* class, which contains all data associated with a single contact, such as username, profile picture and user id.

- After all output classes have been created, the renderer (*renderer.php*) is retrieved, and used to render the *Dashboard* class. When rendering an output class, the method *export_for_template* is called, which puts the data in the corresponding Mustache file, which is used to structure the data into an HTML-template. Since all classes and templates are linked, rendering the dashboard will invoke all the other sections to be rendered as well.

We can visualize the steps above in the sequence diagram shown in Figure 6.2.

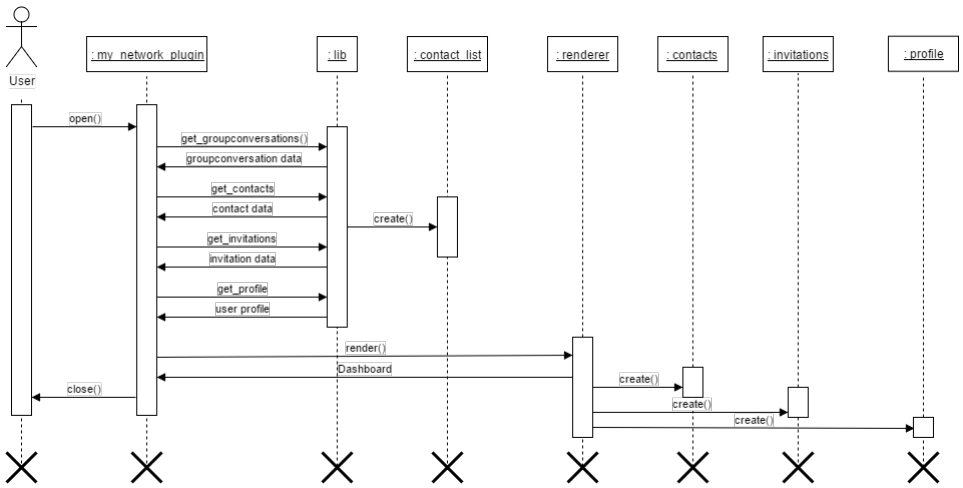


Figure 6.2

6.2. Output Classes

As mentioned in the introduction of this chapter, each section described in the Design chapter is represented as one or more output classes. In our plugin, the following classes are used:

- `Dashboard` The dashboard class is used to prepare the dashboard for display. It holds all contacts, invitations, conversations and profiles.
- `Contact_list` This class displays all contacts as a list and merges them with the group conversations a user can take part in.
- `Contacts` A list of the contacts a user has, it uses `Contact` to represent each contact.
- `Contact` A class holding all data needed to identify a contact, such as its id, name, role and profile picture.
- `Group_conversation` This class holds all data characteristics to represent a group, such as id, users in the group, groupname and the last message send in a group.
- `Groupmessages` This class holds a list of messages send in a particular group, these messages are represented by the `Groupmessage` class.
- `Groupmessage` A class holding all data to represent a message send in a group, such as the text, time it was sent and the sender id.
- `Invitations` This class holds all invitations a user still has to accept or reject, each invitation is an instance of the `Invitation` class.
- `Invitation` The class to represent an invitation, holding all data such as sender and receiver.
- `Profile` This class holds all data to display a profile, such as a user's name or role within the Ivido platform.
- `Profile_professional` A class to represent a professional displayed on the profile of the organization it is part of.
- `Edit_profile` A class to show all data fields that can be edited, when editing a profile.
- `Contacts_messages_search_results` A class to hold all search results when searching for contacts or messages.
- `Users_search_results` A class to hold all search results when searching for users.

6.3. Mustache files

As shown in figure 6.1, each output class has a corresponding Mustache file. Mustache files contain an HTML template for the corresponding data class, which is used to render the output to screen. Also, as mentioned in section 4.3.2, one Mustache file can call another Mustache file to render the template in that file. Because of this, in our plugin, calling the Mustache file for the dashboard is enough to render all other templates stored in other Mustache files. An advantage of Mustache files is that they are logicless templates, which means that the template itself does not contain any data. This allows for a better separation of design and data and logic, which makes it easier to change the design of the plugin in the future. Another advantage is that they are not tied to a particular language.
Below, a code snippet from one of the Mustache files in the my_network plugin is included, to give a better explanation of how these files work.

```
<div class="user" data-action="invitation-selected" data-senderid="{{senderid}}">
        <div class="picture">
            <img src="{{profileimageurl}}" alt="" />
        </div>
        <div class="information">
            <div class="name">
                {{fullname}}
            </div>
            <div class="role">
                {{senderrole}}
            </div>
        </div>
    </div>
```

As visible, some values are surrounded by brackets, in a Mustache file this indicates that the value of this attribute depends on the value of the attribute in the data object that is given to the template. For example, the *fullname* attribute in the template corresponds to the *data_object→fullname* attribute of the output class. So, if the invitation was send from 'Hein Mulder', the value of *data_object→fullname* is 'Hein Mulder', which will be found by the template and be rendered to the page.

6.4. Asynchronous Module Definition

In order to have sufficient behavior of the plugin, JavaScript was needed to perform actions dynamically. These actions are sending a message, accepting an invitation or searching for a user amongst others. Corresponding with the Moodle guidelines, we used Asynchronous Module Definition (AMD). Within AMD, files are stored twice, once as a build file and once a source file. The difference between the two is that the build file is a minified (compressed) version of the source file, these files allow the JavaScript to be loaded faster, which increases the performance of the plugin.
For the AMD, the same structure was used as with the output classes: from one JavaScript file, all other JavaScript files can be instantiated. Because of this, calling the Dashboard JavaScript file leads to all other JavaScript being instantiated.
When starting with AMD, no one was familiar with it. Fortunately, we were familiar with 'plain' JavaScript, so the change to AMD was easy. Two important advantages of AMD over 'plain' JavaScript we found are that with AMD it is easier to structure and separate the JavaScript code. Because of this, it is also easier to debug it.[1]

6.5. Database structure

As proposed in the Research Report (included in Appendix C), a connection between users would be stored in the database as a tuple with four fields: $(connection_{id}, user_1, user_2, status)$, where status can have the following values, 1 = accepted, 2 = pending and 3 = blocked. In total, our plugin uses six database tables, listed below:

- `my_network` This table stores the connections as mentioned above.
- `my_network_group_conv_msg` This table stores the messages that have been send in a group conversation and have also been read.

- `my_network_group_conv_unread` This table stores messages that have been send in a group conversation and have not been read yet. These messages are stored separately to keep track of which user has read which messages. The amount of unread messages for a user is shown in the `contact_list`.
- `my_network_profiles` This table stores the profile of each user, with the fields that are shown on the profiles, such as userid and the region the user is active in.
- `my_network_regions` This table stores all the predefined regions a user can show on its profile, currently they correspond with the twelve provinces the Netherlands has.
- `my_network_users_group` This table stores all users that are a member of a group conversation.

6.6. Version Control

Moodle uses a version file *(version.php)* to maintain the correct version of a plugin and to check if a plugin needs to be updated. Because of this, every time a new version needs to be installed on the acceptation platform, the version number has to be incremented. The version number has the following structure: YYYYMMDDXX, where Y stands for the current year, the M for the current month and D for the day, the X starts at zero and is incremented when a plugin is updated multiple times a day. Because of this, it is easy to check whether the correct version of the plugin is installed and if there needs to be an update.

7

Ethics

Privacy has become more and more important in the modern digital world. Personal data can be found on almost every website. Without knowing, people give away their data every day. Several companies even sell this data to the highest bidder. While most of these bidders are harmless advertisers, the data is personal and can have a great impact on someone's life. Especially medical data is sensitive and since the Ivido platform revolves around it, privacy is a top priority.

7.1. Privacy in messaging

Even though privacy does not seem to be a big issue in a network system initially, a conversation between two people is private and they most likely want to keep it that way. The current message system in the Moodle environment violates this privacy in some ways.

7.1.1. Admin capabilities

The admin user is the user that has the most capabilities within the platform. This user account is owned by Ivido and it is used to install or update plugins, update the language packs, create or modify courses etc. Also, different plugin settings can be configured by the admin user, depending on the type of plugin. In the message system the admin user has exclusive rights as well. One of these rights is viewing every existing conversation in the platform and the possibility to remove certain messages in that conversation. As mentioned above medical data is very sensitive. However, for medical reasons, now and then people need to share personal data with a professional from their network. Especially using the possibility of sharing documents. That is why we have decided to remove the exclusive rights for the admin user except for configuration settings. In the network plugin, the admin user is no longer able to see the conversations he or she is not participating in.

7.1.2. Non-contacts

Another issue concerning privacy in the current message system is the ability of communicating with every user in the platform. When using the search function every user in the platform can be found and messaged. Even though this can be a convenient feature, it is not always desired. Phishing and scamming are possible methods for criminals to retrieve medical data from ignorant users thanks to this feature. For this reason we have decided to restrict the message system to mutual approved connections only. This way solely people that the user has accepted can send him or her messages.

7.2. Extension to other parts of the system

Not only can the structure of connections be used in the network plugin but also in other parts of the system. The idea of connections is preventing the user from involving other users without their approval. In the network plugin it comes down to a restriction on messaging whoever the user wants. In the Ivido system are more parts where it is desired to have such restrictions. One of these parts is adding members to a course. A user should no be able to add every user in the system. This process can be restricted by making use of connections: a user can only add or maybe even just invite users

that he or she is connected to.

Another part where the idea of connections can play an important role is the budget system. In chapter three the Budget System was mentioned in the requirements and one of the Could Haves was integrating the budget system into the Health Network. However it could be integrated the other way around as well. At the moment the Budget System has its own small network section. This network consists of the professionals that work for the organization, since the plugin is only available to organizations. The Health Network could replace this small network section and use its own structure to relate professionals to an organization.

By extending the plugin to these other parts of the system privacy will overall be better conserved.

8

User Testing

When developing and testing the new plugin, we use a testing environment, which is a local platform on the developers working station. The changes made here are then stored in a branch on GitHub and once the change are done, a pull request is opened. The other developers check the pull request and if they approve, it is added to the master branch. Once the Must Haves have been met, the plugin is uploaded to acceptatie.ivido.nl and our employer starts the pilot.
This chapter describes how the prototype was tested to receive feedback from potential end-users in a pilot.

8.1. Pilot

During the pilot approximately 20 people dove into the network plugin to test not only the functionality but the usability as well. Since the network plugin will be provided to every user in the platform, including elderly and mentally disabled people, the plugin needs to be very easy to use. This means that buttons need to be in a logical place, the lay-out needs to be non-provoking, contacts and other users need to be easy to find etc. Furthermore, they were asked to write down functionalities that they might find desirable in the network plugin. This includes additional information on profiles. The group of people testing the plugin is diverse, it consists of managers, professionals, clients among others. This gives us useful feedback on all aspects because a professional has different preferences for a network tool than a patient. Also, they have different profiles, which all need to be tested. A professional can add the organization they work for, the position they have there and the specialization they are in. These fields are disabled for patient users.

8.2. Questionnaire

In order to collect the feedback, we created a Google Form. This form asked the user to perform certain actions and to rate the action on difficulty. Also, the user was asked to give feedback on how an aspect can be improved as explained before. The questionnaire is included in Appendix D.

8.3. Results

Unfortunately, not that many people were able to fill in the questionnaire in time, but we received plenty of feedback from the people who did. They found that the questionnaire was clear and the assignments were easy to understand and execute.
The most important points of feedback we got were:

- File sharing button: one thing that was unclear, was the ‘add attachment’ button. We changed it to a more commonly known button for adding files, namely the paper clip icon.
- Advanced search: another tip was to show a list of all contacts when the advanced search page is shown and no search query is run yet. Plus, it would be better if more information about a person is shown in the search results (it only showed the name and role). So we decided to add the city and date of birth beside the name, the user can then press the name to go to the profile for more information. As for styling, multiple people said that the way the filter were shown was

unclear. Since there are two buttons for the search (Search and Filter) is was not clear to them what the 'Filter' button did exactly. We decided to change the text on the button to 'Add Filter'. There was also feedback saying that it was illogical that pressing a filter button would remove it from the search query, they would rather see an 'x' button.

- Profile: for the profile section we received some suggestions for more fields, such as date of birth. These could be added as search filters as well in the future. Furthermore, the lay-out was not quite ready yet. The blocks for input when editing were all placed underneath each other, even though they did not need the entire row and did not expand sideways when needed. This turned out to be a small bug we missed and was easily fixed.
  One suggestion was to add a 'Make appointment' button to the profile of the professionals. This button should only be visible if the professional is within your network.

So there were mainly suggestions for the lay-out of the plugin, which is to be expected with a user pilot. The usability of the plugin and the difficulty of the task both had an average of 4 out of 5, 1 being difficult and 5 being easy, and the future users are overall satisfied with the system.

9 Process

In this chapter, the process of the project is discussed.

9.1. SCRUM

During earlier projects and courses within the Bachelor Computer Science program, we learned that an agile development method is very efficient when working with a small team on a relatively small project. The project is divided in smaller development cycles, during which the team is able to focus on parts of the project without interference. The result of this is efficient use of time and a clean workflow. The method is also used by other Ivido developers, which made it easy to integrate the method into our workflow.

SCRUM Reflection

Because of SCRUM, it was clear to all the members of the project as well as our client, what was happening and when certain features would be finished. Also, SCRUM allowed for a weekly reflection on certain events that happened the week before. Because of this, we were able to maintain a strict planning and everyone was aware of difficulties others were facing, which then allowed us to tackle these sooner rather than later.

9.2. Development resources

During this project we used a few development resources, we will discuss each of them in the sections below.

9.2.1. Google Drive

Google Drive was used in various ways to support the development method. For example, scrum plans were created in Google Sheets, bugs were written down in Google Docs, the pilot was conducted using a Google Form and all other documents were stored in the Google Drive folder.

9.2.2. GitHub

Version control is a useful and important aspect, since it allows for recovery and correction of parts of the code. To realize this, we used GitHub, which is a hosting facility for the open source Git software. Git is a revision control system created for software development.
The advantages of using a version control system are:

- Branching - each feature can be developed in a separate branch instead of on the master branch, which prevents merge conflicts and allows for better revision of newly written code.
- Merging - After a feature is completed and tested, it can be merged into the master branch by using a Pull Request. Because of this, each line of newly written code can be reviewed by other team members.

9.2.3. Software Improvement Group

As part of the project requirements, we submitted the code to the Software Improvement Group (SIG) for analysis. At SIG, a static code analysis was performed, which means that the code was reviewed without execution. The quality of the code is then rated based on a maintainability-scale from one to five. The rating depends on characteristics such as analyzability, modifiability, testability and re-usability, which are graded on the following criteria: volume, duplication, unit complexity, unit size, unit interfacing, module coupling, component balance and component independence.

Feedback

The feedback received from SIG on June 15th can be found in Appendix E. They gave the code 3 out of 5 stars, which means that the system has an average maintainability. Also, they gave the following points for improvement:

- **Unit Size**
  When assessing the Unit Size of the code, SIG determines the percentage of code that has a length above average. In our case, the library file *(lib.php)* contained some very long methods which could easily be split up into smaller methods. An advantage of a smaller Unit Size is that it is easier to reuse functionalities and to maintain functionalities.
- **Unit Complexity**
  When assessing the Unit Complexity of the code, SIG determines the percentage of code that has a complexity above average. This relates closely to the Unit Size, and could, in our case, also be improved by reducing the length of some methods. An advantage of a smaller Unit Complexity it that it becomes easier to understand code segments, test them and also debug them.
- **Automated Testing**
  At the time of the uploading to SIG, our code did not contain any automated tests. SIG strongly recommended us to include these to ensure certain functionalities would still be working after new functionalities were added.

Improvements

After receiving the SIG feedback, we have made a few improvements to our code:

- **Unit Size and Complexity**
  There were a few functions that were too long and too complex. We refactored these functions to multiple smaller functions with each just one task. SIG stated the function *my_network_get_recent_conversations* as example, which was a function with three different functions in it: collect the recent conversations, collect the unread messages and combine the two. We refactored it to the following functions:
  - *my_network_get_recent_conversation*, this function constructs the parameters for the other functions, combines them and sorts the result.
  - *my_network_get_recent_conversationssql*, this functions constructs the SQL used to retrieve the conversations from the database.
  - *my_network_get_recent_unread_conversations*, this function retrieves the unread messages. Since these messages are stored in a seperate table, we also created a seperate method to retrieve them.
  - *my_network_get_recent_unread_conversations_count*, this function counts the amount of unread messages in each conversation.
  - *my_network_get_all_recent_conversations*, this function combines the unread and read conversations.
- **Automated Testing**
  Since none of use had worked with automated tests Moodle or PHP before, we first did some research to find out how this could be doen. We found out that Moodle uses PHPUnit, an advanced unit testing framework for PHP, as an independent tool for running unit tests. Moodle divides test cases in two classes, the basic and the advanced unit tests. Basic tests are very simple and do not modify the database, dataroot or any PHP globals. Advanced tests are the ones that do test such modifications [2]. Since our project almost only consists of PHP functions that are called from AJAX calls and modify the database, we only created advanced tests. The Moodle message system already has a good amount of test cases to test the functionalities of messaging. That is why we decided to focus on the other parts of the network plugin: user searching, profile updating, connection updating, sending files and making group conversations.

Development Resources Reflection

Looking back on the project, it might have been more efficient if we had used a planning tool such as Trello to keep track of what everyone was doing, instead of Google Drive. An advantage of this could be that with Trello commits to GitHub can be linked to Trello cards in a very clear and efficient manner. However, no problems occurred while using Google Drive. If the duration of the project had been longer or the scope of the project had been wider, we feel that a Trello (or something similar) would have been of added value, but in our case using Google Drive was sufficient.
We feel that it is good to write code that is easily maintainable and therefore feel that the feedback given by the SIG was very helpful.

9.3. Individual Reflection

At the end of this project we had some time to look back at what each of us has done, how everything was divided over the group members and what role we would take in future projects:

Roy Klip

At the beginning of the project I worked on the basic search function for searching in contacts and messages. After that I switched to implementing and designing profiles which kept me busy quite a long time since there needed to be different kind of profiles each with different interactions possibilities. Whilst working on the profiles I also created database structure to store connections. When the profiles were considered done I did research together with Noor on document sharing. After Noor switched to bug fixing I remained busy on finishing the document sharing.
The division of tasks was fair and equal during the project. We all worked on the different aspects of the project. Everyone did some coding, styling, bug fixing, writing the report and so on.
I would like to keep the same dynamic in future projects. This way every member of the group takes part in every aspect. It motivates working on the project when I see that I am not the only one working on a particular part.

Noor van Ruyven

I started of with the design of the entire page and the placement of the blocks. After that I needed quite some time to find out and implement the advanced search function. Last I did some research as to how document sharing could be implemented and made a start with Roy, but then switched to start with bug fixing since there were quite a few.
The group dynamic was good, it was nice that every task we had could be developed separately, this way we could all work on our own project simultaneously. We divided all roles equally, each of us has done some styling, some back-end and we all wrote the report.
In future projects I would like to keep the same roles, since you see all aspects of the project and that way you really understand how it works from start to end.

Daphne van Tetering

When starting the project, Roy, Noor and I already knew each other, so I had the feeling that this cooperation would go very well. I turned out to be right, which is a good thing. When we needed to divide tasks for the new SCRUM-cycle we would divide them in such a way that everyone would have an equal amount of indicated hours and would be working on something they liked. When assigning a task to someone, this meant that they were responsible for everything: from developing the functionality, styling it to writing PHPDoc and connecting it to our back-end. Because of this, everyone got to work on front- and back-end as well as styling and testing.
My contributions are in the one-on-one message system, group conversations and the contact list. These were tasks that involved around database operations and connecting the front-end to the back-end to collect and send data a lot. Because there was data involved, I also needed to think of an efficient way to store contacts, groups and messages and create a suitable database design. I found these tasks very challenging and because of this I enjoyed them a lot. Partially because I have an interest in Data Science and partially because I didn't have a lot of experience in Web Services yet.
Something I would change in the future is that I developed by myself mostly. In the future I might want to cooperate with someone, since my contributions took up a lot of my time and were quite difficult. Also, developing something together offers a new perspective besides your own. Even though I don't

mind the fact that I did it alone and it wasn't always possible to work together on a feature (since there was only three of us), I would keep this in mind for a future project.
What I would do again in the future is working on everything from beginning to end (front-end, back-end, styling etc) and being the SCRUM master. I like to keep a close overview of how things are going and whether a project is on schedule to meet deadlines. Therefore, I took on the role as SCRUM master.

10

Conclusions

For the past ten weeks we dove into the Ivido platform to develop a Patient Health Network. We have made a MoSCoW list to prioritize the wanted requirements in a way that we saw achievable within the limited time of the project. We have added a small overview of the Must and Should Haves we achieved at the end of this section in the form of a table (figure 10.1).

We created a Health Network where two parties can connect with each other based on mutual agreement and message each other if they have done so. Besides messaging each other, they can also create a group with other people they are connected to, to have a group conversation. To visualize this network we created a dashboard that contains the user's contact, messages and search functions. These search functions can be used to find new contacts or search in messages using filters and categories. The network also has profiles, which contain a user's information and a functionality that allows users to share files with each other or with a group.

There are still a few Should Haves we aimed to develop during this project, but did not fit in the timespan of the project. The reason we weren't able to meet these is that some features were harder to developed as estimated. For example, group conversations were more work since it turned out they needed a lot of different logic than the regular conversation, which wasn't anticipated in advance. The Should Have that we did implement, the advanced search, also turned out to be a big one. To realize this, we needed to write a lot of advanced SQL queries to search in multiple database tables. It was challenging to find a way to do this in an efficient way that would require the least overhead.

In these ten weeks we have built a Health Network which meets all of the Must Have requirements. Because of limited time, we did not meet all the Should Haves, we will discuss these in more detail in chapter eleven (see Figure 10.1).

At the end of the project we conducted a user pilot, where we received a lot of feedback about the difficulty of the plugin and whether we should change anything in the lay-out from future users of the network. This feedback turned out to be very useful and showed us some flaws we had not noticed ourselves.

Must Have 1: View profiles	✓
Must Have 2: Search contacts and messages	✓
Must Have 3: Send connection requests	✓
Must Have 4: Approve connection requests	✓
Must Have 5: Communicate with other users	✓
Must Have 6: Visual representation of connections	✓
Must Have 7: Dashboard	✓
Must Have 8: Share documents with other users	✓
Must Have 9: Group conversations	✓
Should Have 1: Forum discussion	x
Should Have 2: Notifications	x
Should Have 3: Invite new users	x
Should Have 4: Quick-add feature	x
Should Have 5: Advanced search	✓

Figure 10.1: An overview of the achieved requirements

11

Recommendations

When starting this project, a MoSCoW list was created to give a clear overview of what would be the features of the prototype that was to be created. Most of these requirements have been met, but not all of them. Besides this, even though a lot of thought has been put into the design and development of the plugin, there are still some improvements that can be made to increase the usability of the plugin. In this section, we will discuss these points along with points from the original project description that we feel would be a good addition in the future.

Notifications

Notifications are useful to alert the user that he or she has received a new message, has been added to a group conversation or has received an invitation. Currently, the user is only alerted of a new message by an unread-counter, which shows how much unread messages a user has in a conversation. However, this is only visible when the user is in the Health Network. A nice extension would be to also alert the user when he or she is somewhere else in the platform.

Invitational message

To give an invitation a personal touch, the user will be able to include a personal message when sending an invitation.

Forum discussion

A forum discussion is semantically very similar to a group conversation, the only difference is that a group conversation is private, and a forum discussion is public and has a certain topic. This extension should be fairly easy, since the group conversations are already there.

Invite new users

Some persons a user may want to connect with don't have an account on the Ivido platform, with this feature the user will be able to send these persons a request to create an account on the Ivido platform.

A

Info sheet

General information

Title of the project: Patient Health Network
Name of client organization: Ivido
Date of the final presentation: July 5, 2017

Description

Ivido is a start-up company that aims at improving the communication and efficiency in health care. To do so, Ivido has developed a platform where health care professionals can create health courses for their patients, to stimulate them to become more involved in their own health care program. Because of lack of means to communicate, the client asked for a Patient Health Network, that will allow professionals, caregivers and patients to communicate together. Important characteristics of this Health Network are that users must be able to share documents, to communicate in private and in groups and to build their own network of connections.
The most challenging part during the research was to decide on the design of the network, since a significant part of the users are struggling with illness, the network has to be self-explanatory, even if the user is struggling with certain conditions. Another challenge was to decide how to store the data: storing messages can take up a lot of space if it is not done efficiently.
At the beginning of the project, a MoSCoW list was made with the most important requirements. However, since the client was still unsure about some of the additional requirement, the team used the a SCRUM approach to respond to changes in the requirements, should there be any.
During the project, these changes did occur: some of the should and could have from the MoSCoW list depended on the development of a budget management system. This system wasn't available to us in time to meet these requirements, so they were assigned the lowest priority.
The final product is a plugin on the Ivido platform, that can be used on multiple devices. It gives the user the ability to create conversations with one or more other users, to connect with other proffessionals, patients or caregivers and to view profiles of the people they are connected with or want to establish a connection with. Every SCRUM iteration acceptance, compatibility and usability tests were conducted to test the plugin. After the last SCRUM iteration, the User Interface design was tested by a group of stakeholders to gather feedback and points of improvement for future work.
The outlook of the product is one central Dashboard, from which the user can send messages, accept invitations, search for users on region, name, city, organization and specialisation and send invitations to other users. The design of the outlook of the product was made in agreement with the client, since they have a lot of experience in interface design for patients struggling with a certain illness.

Members of the project team

Name: Roy Klip
Interests: Usability, data science, entrepreneurship, software engineering
Contribution and role: developer (front- and back-end), environment setup, unit-testing

Name: Noor van Ruyven
Interests: Software engineering, program language design, front-end design
Contribution and role: developer (front- and back-end), front-end design

Name: Daphne van Tetering
Interests: data Science, project management, software engineering, algorithm design
Contribution and role: Scrum master, developer (front- and back-end), database design, messaging system

All members contributed to preparing the research and final reports and the final presentation.

Client

Name: Tristan Garssen
Affiliation: Ivido B.V.

Coach

Name: Felienne Hermans
Affiliation: Software Engineering Research Group, Delft University of Technology

Contact information

Roy Klip	royklip2@gmail.com
Noor van Ruyven	noor_van_ruyven@hotmail.com
Daphne van Tetering	daphnevantetering@gmail.com

The final report for this report can be found at: http://repository.tudelft.nl.

B

Project Description

The following description is the original project description as found on BepSys. It was provided in Dutch.

Project beschrijving

De verwachting is dat de komende jaren de zorgvraag verder zal toenemen waardoor de zorguitgaven een steeds groter aandeel in het totaal van de uitgaven gaan innemen. Innovatieve oplossingen zijn nodig om bij te dragen aan duurzame zorg en om de kosten beheersbaar te houden. In de praktijk blijkt echter dat zorginnovaties vaak stranden omdat ze zich moeilijk laten opschalen of verbreden.
Het pijnnetwerk is een landelijk netwerk van geselecteerde zorgaanbieders in verschillende regio's binnen Nederland. Zij hebben zich gespecialiseerd in het behandelen van mensen met chronische pijn. Het Ministerie van Volksgezondheid(VWS) heeft een health deal status toegekend aan het project van het pijnnetwerk.
Om samen te werken in regionale netwerken en de patiënt kwalitatieve en doelmatige zorg te bieden is het noodzakelijk dat een Patient Gezondheids Omgeving (PGO) wordt ontwikkelt. Ivido is gevraagd om voor het pijnnetwerk en deelnemers van de health deal chronische pijn een PGO te ontwikkelen. Hetgeen betekent dat er in de komende tijd een aantal functionaliteiten ontwikkelt dienen te worden.

Zorgnetwerk voor patiënt en professional

Voor de professional en patiënt is het belangrijk dat zij een persoonlijk zorgnetwerk kunnen opbouwen. Hier moeten zij professionals of andere patiënten kunnen toevoegen tot het zorgnetwerk. Het doel is om hierdoor makkelijk andere personen bij het persoonlijk zorgpad te betrekken en direct te communiceren met het zorgnetwerk.

Probleemschets chronische pijn

In Nederland hebben meer dan 2,2 miljoen mensen chronisch pijn. Voor deze patiënten heeft chronische pijn een forse impact op hun dagelijks functioneren, welzijn en hun directe omgeving (gezin/partners). De chronische pijnpatiënt ziet vaak meer dan 10 tot soms wel 25 (zorg)specialisten en ervaart de organisatie van de pijnzorg vaak als een flipperkast. De diagnostiek en triage zijn niet eenduidig, de behandeltrajecten zijn versnipperd en leiden (te) vaak niet tot duurzaam herstel of verlichting van de pijn.

Ivido

Ivido is een organisatie die is ontstaan uit het gebrek van ICT/E-health oplossing om het zorgproces voor de chronische pijn zorg te ondersteunen. Ivido is opgericht door Hans Niendieker en Tristan Garssen. De organisatie is een jaar geleden gestart met de ontwikkeling van het platform. Er zijn in de loop van de afgelopen tijd een aantal functionaliteiten ontwikkeld. Voor de facilitatie van de regionale

netwerken is het belangrijk dat Ivido het platform in de komende maanden doorontwikkeld en echt pilots gaat uitvoeren met patiënten.

Wat doet Ivido?

Ivido is een patiënt gezondheids omgeving (PGO) waarmee cliënten gefaciliteerd worden om zelf gezond en vitaal te blijven en effectief in de werkomgeving. Cliënten krijgen met behulp van Ivido de regie over de zorg (terug) en kunnen met behulp van Ivido optimaal communiceren met hun netwerk van zorgprofessionals. Zorgprofessionals worden optimaal ondersteund in het leveren van kwalitatieve en doelmatige zorg en de registratielast wordt geminimaliseerd.

Doel Ivido

Wij willen met Ivido de relatie tussen de cliënt en de professional versterken. Wij brengen de cliënt en professional in onze omgeving dichter bij elkaar door individuele zorgplannen online te ondersteunen en zelfredzaamheid van de cliënt te stimuleren.

Wij willen met Ivido cliënten stimuleren en ondersteunen in de weg naar een verbeterde vitaliteit en gezondheid. Door middel van verschillende soorten gezondheid programma's en kwalitatieve en doelmatige zorg willen wij dit realiseren.
Wij willen zorgprofessionals en organisaties veel makkelijker en doelmatiger laten samenwerken in regionale zorgnetwerken. Zinnige, zuinige en excellente (keten) zorg is daarom binnen handbereik en het onderscheidend vermogen van de organisatie willen wij sterk vergroten.

C

Research Report

Ivido

Ivido is a start-up company that aims at improving the communication and efficiency in health care. They do this by developing a health platform where patients, professionals and organizations come together. On this platform the professionals can create health courses for their patients. The development of this platform is based on the worlds most used course management system, Moodle.

Moodle

Moodle is an open-source online learning environment consisting of mostly PHP code. Moodle stands for Modular Object-Oriented Dynamic Learning Environment.

Structure

The Moodle platform consists of two main components: the core and the plugins. The core of the system is continuously maintained and updated by Moodle developers. The plugins are the custom parts of the system. There are many different plugin types and they can be found anywhere in the system.

The type of the plugin does not only define what kind of functionality it supports but it defines the location of the code in the system as well. This is because Moodle has a different folder for each plugin type in its source code. Most of these folders can be found in the main folder, some have a more specific location. For example, a block plugin can be found in the */blocks* folder. Another way of finding the type of the plugin is in the url. Provided that the block plugin from the example has internal pages, the urls of these pages are */blocks/pluginname/**.

The naming convention that Moodle uses is Frankenstyle. Frankenstyle separates parts of names with underscores. This style of naming is not only used in the code (in function and object names) but has its most important role in component naming. Frankenstyle component names consist of a plugin type as prefix, an underscore and the plugin name. So the component name of the module 'quiz' is *mod_quiz*. Component names are used throughout the system, mainly for identifying the plugin.

Every plugin, independent of its type, consists of a few standard folders and files. The standard folders are 'db' and 'lang', where db stands for database and contains the files interacting with the database and lang stands for language and contains the language files. Disregarding the language folder, Moodle makes use of string identifiers. These identifiers can be evaluated through the *get_string* function, which looks up the corresponding string for the users chosen language. For every language used supported by system there is a unique language file in each plugin.

Besides these folders Moodle plugins always have a version file, with this file an update can be indicated. Some plugin types require more than the previous stated standard folders and files. Fortunately, the Moodle documentation includes the requirements per type and the way to set up a plugin.

Database
The structure of the database connected to the platform is similar to that of the code. Furthermore, the naming of the database tables is also in Frankenstyle. In this case the plugin type is mostly not stated in the database table name, but the prefix of every table is 'mdl_', which stands for Moodle. Plugins can have multiple database tables which have an additional prefix: the plugin name. A database table for the quiz module would be named *mdl_quiz_**. Database tables can be created, modified or deleted via the PHP and XML files in the db folder of the plugin.

Moodledata
Besides the code and the database, Moodle also has a data folder for cache files, back-up files, temporary files and more. This data folder is used locally, so every developers can have a different session of the system and is called Moodledata. Back-ups of courses to copy or restore them in the system are also found in this folder. Temporary files are stored here as well. These files are used for easy retrieval of recently uploaded files.

Config
Moodle uses a configuration file, called *config.php*, which is stored in the Moodle directory. The file is used for the following things:

- Connection to the database: in the config-file, the developer's username and password to access the database are stored.
- Location of the moodledata: as mentioned above, Moodle stores its moodledata locally. The path to the moodledata is written in the config file.

PHP
The most commonly used programming-language in the Moodle platform is PHP, a language we are not familiar with. During the first two and a half years of our Computer Science Bachelor, most projects were written in Java. One of the differences between Java and PHP is the way objects are typed: Java uses static typing, which means that variables must have a declared type, while PHP uses dynamic typing, which means that variables assume the type of the value currently contained in them and are allowed to change their type to satisfy casts and conversions. Another difference between the two is that Java is a client side language, while PHP is server side language. This means that Java code will be executed on the client's computer, while PHP code will be executed in the actual server. However, both languages also have some similarities, an important one being that they both allow for object-oriented programming. When familiar with one of the two, it is fairly easy to learn the other, which is in our advantage.

System
Moodle Core
The currently used version of the Moodle Core is Moodle 3.2. Key characteristics of this version of Moodle are:

- Messaging: Moodle 3.2 includes a basic messaging systems that allows users to communicate with each other.
- Notifications: Moodle 3.2 enables users to get notifications when they receive a message or changes are made to a course they participate in.
- Improved user interactions: Moodle 3.2 also includes a new theme, based on Bootstrap 4, that is cleaner and gives a more structured view of a course.
- New graph library: Moodle 3.2 uses a new graph library, which improves scalability of graphs and also enables them to be shown in table view.
- Tours: with Moodle 3.2 it is possible to design custom-made tours of a course. This means that each course can have its own tour to explain core features of the course.

Plugins
The Ivido platform consists of many plugins that are build upon the Moodle Core. Some of the important plugins that form the base of the platform are listed below.

- Authentication: this plugin ensures that different types of accounts can be created. Because of this, we have different account types for health care professionals, patients and health care organizations, which optimizes someones experience with the Ivido platform.
- Enrollment: the enrollment plugin determines whether someone can enroll for a course or program based on their account type. Because of this, we can create programs specifically for patients with a certain disease or for health care professionals that have the same specialty.
- Course format: this plugin determines in what format a course or program is displayed, this includes which content is shown, the layout of the content and the activities or modules that need to be completed in a program.
- Modules: the module plugins enable us to create different activities within a course, such as quizzes, assignments and questionnaires.

Testing environment
When developing and testing a new plugin, we use a testing environment. This testing environment consists of three 'parts'. The first one is the version of the platform that is hosted locally on a developers working station, it is used to develop the plugin and for the developer to test their own work. This work is stored in a branch on GitHub. This branch is also checked by other developers to ensure quality and stability. The second one is the master branch on GitHub, which contains all final versions of plugins, except for the ones in development. This branch is uploaded to acceptatie.ivido.nl, which is the third part of the testing environment. Here, our employer tests the new features on usability and performance.

Live environment
When a plugin is approved by our employer, is it transfered to the live environment (platform.ivido.nl), where it can be tested and used by clients. Based on their opinions, changes can be made to improve a plugin.

Pilot
When our product meets all the must haves of the requirements, a pilot will be started to evaluate our plugin. We will have 30 to 50 persons, both professionals and patients, who will test the different functionalities of the plugin. They will help us to evaluate and improve our product. The pilot will be done using User Testing, which means that each person will be granted access to our platform individually. We will then ask them to perform certain tasks via a Google Form. After performing these tasks, they will be asked to give feedback, for example, tell us what they are thinking and how self explanatory the system is. This will give us insight in the strengths and weaknesses of the system. Our client Tristan Garssen will gather people for this pilot and email them the information with links to the platform and the Google Form. The tasks can be performed at home and are more convenient to perform in pairs.

Project
Description
Ivido aims to strengthen the relation between the client and professional by bringing them closer with an online individual health map. This stimulates the self-reliance of the client and makes the co-operations between health instances easier and more efficient. To achieve this goals, a Patient Health Platform is being built.
For our Bachelor Project, we have the choice between the following features:

- My Health: A personal dossier for the patient with all patient information. The patient should have full control of this, which means that they can decide for him or herself with whom and when this information is shared.
- Health network: Both professional and patient build their own health network by adding patients and professionals to their network. This way they can easily give assignments and schedule meetings, as well as communicate with each other.
- Budget System: Organizations should be able to buy and then manage their budget. They can spend this budget on different types of subscriptions.
- Work flow: Within the health paths of patients, professionals have multiple tasks as well. To

keep the work flow balanced, a signal should be sent to a different professional when the original professional was unable to complete the task before a certain moment, so this new professional can complete the task.
- Gamification: Patients get a reward for participating in their health program. This can either be in the form of a point system or in the form of a discount.
- Connect Ivido to EPD: The system of Ivido should be able to connect to the systems of the health facilities (EPD) so they can exchange data safely. To achieve this, a number of API connections should be added.

Since each of the features is a project on its own, we decided to focus on one and continue developing from there, should we still have time. We have chosen to develop the Health Network, since we find this the most interesting topic. Also, it can easily be expanded if necessary.

Plugin

For this project we have chosen to develop the desired functionality as a plugin. As explained before Moodle is developed in a matter that the core is maintained by Moodle developers. The Moodle core has approximately two major updates a year. The plugins are meant to add custom parts to the system. Ivido aims to keep the core updated and develops all its functionalities via plugins.
The type of plugin we are going to develop is a block plugin. The main reason for this is that the plugin needs to be accessed from the dashboard of a user and the dashboard consist of blocks. Furthermore a block plugin is easily set up and extended to match the requirements.

Requirements

To get a good grasp of the Health Network, we made a MoSCoW list with the desired components.

Must haves
In the Must Haves-section, we have defined the core features our Health Network needs to function.

1. **View profiles**
   User story: As a user, I want to view a contact's their profile.
   Feature: Create profile pages and a function to display the profile of a user.
2. **Search contacts and messages**
   User story: As a user, I want to quickly find the contact or message I am looking for.
   Feature: Build a search function that searches for contacts and messages.
3. **Send connection requests**
   User story: As a user, I want to send other users connection requests to add them to my network.
   Feature: Build a functionality so that users can send each other connection requests.
4. **Approve connection requests**
   User story: As a user, I only want to connect to another user, after I have given approval.
   Feature: Only establish a connection between users after both sides have given approval for this.
5. **Communicate with other users**
   User story: As a user, I want to communicate with other users in my network.
   Feature: Extend the currently available message system to ensure messages can only be send to users that have established a connection.
6. **Visual representation of connections**
   User story: As a user, I want to have a clear overview of the users that are currently in my network.
   Feature: Create a section* where current connections are shown with links to their messages and profiles.
7. **Dashboard**
   User story: As a user, I want all components of my network, contacts, invitations and messages, to be shown in a clear overview.
   Feature: Create a dashboard that encapsulates all other views of the contacts, invitations and messages.
8. **Share documents with other users**
   User story: As a user, I want to share documents with other users in my network.
   Feature: Extend the currently available message system to ensure documents can also be send.

9. **Group conversations**
   User story: As a user, I want to be able to discuss certain problems or cases with more than one user.
   Feature: Extend the current message system with group conversations.

Should haves

1. **Forum discussion**
   User story: As a user, I want to discuss or share important information in a way that everyone can advise or comment on the posed problem.
   Feature: Develop a forum functionality. The difference between forums and group conversations is that forums are open to everyone in a network, whereas group conversations are only accessible to members added to the conversation.
2. **Notifications**
   User story: As a user, I want to be notified after my request to connect has been accepted by another user or after I received a message from another user.
   Feature: Extend the currently available notifications to match our clients needs.
3. **Invite new users**
   User story: As a user, I want to be able to invite users that do not have an account yet to use the Ivido platform.
   Feature: If a user cannot be found, a link must be send to that user to invite them to create an account on the Ivido platform.
4. **Quick-add feature**
   User story: As a user, I want to add other users in my program to my Health Network fast and easily.
5. **Advanced search**
   User story: As a user, I want to find users based on other criteria than their username, such as their profession.
   Feature: Build an advanced search function that filters or searches on other fields than username.

Could haves

1. **Budget management integration**
   User story: As an organization, I want to be able to see how much is left of my budget and how it is spend.
   Feature: Make it possible to link the Health Network to the Budget System. op.
2. **Available programs**
   User story: As a user, I want to be able see which programs might be suitable for me to participate in.
   Feature: In the Health Network, show which programs are available to participate in within the network.
3. **Budget assignment**
   User story: As an organization, I want to be able to assign the professional of the organization a budget, that can be used to purchase programs for their clients.
   Feature: Create a system that distributes budget among professionals, based on the organizations preferences.
4. **Invitational message**
   User story: As a user, I want to send a message when sending someone a connection request.
   Feature: Extend the connection-function with the ability to send someone a message.

Won't haves

1. **Webshop**
   User story: As a user, I want to able to purchase new programs to participate.
   Feature: A webshop in which available programs are shown and can be bought.
2. **Budget transactions**
   User story: As an organization, I want to be able to increase the balance of my account, so members of the organization are able to purchase new programs.

Feature: Connect the Budget Management System to a payment system and secure the transactions.

Analysis

For each Must-Have, we have made a small design of how it will be implemented in the platform. We will now explain these designs and the choices that were made during the design process.

1. **View profiles**
   To support this feature, we will have to build profiles that can be generated when clicking on a user's username. A profile will be shown in instead of the messages and will contain someone's name, photo and role in the platform. If a user has the role of professional in the platform, his specialization and organization he or she belongs to will be shown as well. On an organization profile a list of employees will be shown with a link to their profile. We will also add a button to send this person an invitation to connect with them.
2. **Search contacts and messages**
   To support this feature, we will add a search field above the contact list and a search function that will compare the search string with all the contacts and messages of the current user. When the search field is empty, the contact list will be shown again. When the user clicks on a search result the message area will show the (selected) message(s) between the contact and the user.
3. **Send connection requests**
   To support this feature, first, we need to establish a data structure that holds all connections between users. We will realize this by creating a database table in which relations can be stored, the database will contain tuples which have four fields: $(relation_{id}, user_1, user_2, status)$.

   For example: Alice wants to connect with Bob, so Alice has sent Bob a connection request. The tuple would be as follows: $(1, Alice_{id}, Bob_{id}, pending)$. After Bob has accepted the request, the tuple will be: $(1, Alice_{id}, Bob_{id}, accepted)$.

   Then, we need to show each user who is in their network and let them know when a request is accepted.

   The ER schema is shown in Appendix A.
4. **Approve connection requests**
   To support this feature, an invitation section* is added to the dashboard. This block shows all invitations, each with an accept and reject button.
   When a connection is approved, the relation in the database is changed as shown above.
   When a connection is rejected, the relation will be removed from the database.
5. **Communicate with other users**
   The current system already supports a low-level form of communicating: it allows two users to send text-messages to each other. The problem we need to solve here is that everyone can send a message to anyone, even if there is no connection between them. To solve this, we need to rebuild the system in such a way that messages can only be send by users who have a connection together, that has been approved by both sides. We will do this by using the database table mentioned in point 2.
6. **Visual representation of connections**
   The connections made in the Health Network of a user will be shown as a list. When a user clicks on one of the connections the intermediate messages will be shown in the message section.
7. **Dashboard**
   Since Moodle is a modular platform working with plugins, we will create a plugin for this project as well. This plugin will be of the type block, an advantage of this type of plugin above other types is that block plugin can easily be integrate in webpages, amongst other blocks. Also, they can easily be extended with other webpages. Our *network plugin* will have a dashboard of its own, just as *block_my_health* has. This dashboard will be divided in different sections:
   - Contacts: this section contains all connections a user has made. The section will also have a search function to easily go through existing contacts and forums or to find new contacts, who can be invited.
   - Invitations: this section will contain the pending invitations sent to the user for contact

requests. Furthermore, a user can invite a person via email to sign up for Ivido to become a contact of the user.
- Messages: this section will occupy most space on the dashboard and will contain the message history between the user and the selected contact. New messages can be written and sent here as well. The section will be based on the current message section as defined in the Moodle core.

8. **Share documents with other users**
   To support this feature, a file needs to be uploaded to the platform from the users computer and be made available to download for the other user. Moodle already supports file uploading and downloading via PHP requests, but this is not present in the messaging system. To do this process in real time on one page AJAX will be used alongside PHP.
9. **Group conversations**
   To support this feature, we will have to further extend our message system. It is important to take into account that not everyone has to be connected with everyone in a group conversation. However, anyone who is added to a conversation, must have a connection with at least one of the member already in the conversation. It should also be possible to add someone who you are in a group conversation with directly.

   We find this to be the hardest requirement since it requires the current functionality to be rewritten almost completely. This functionality is build into the Moodle core, which means that it is not meant to be changed and that it is intertwined and connected with a lot of other functionalities of the Moodle core. Because of this, we will have to overwrite the whole plugin and rebuild a lot of functionalities ourselves, which will be a complicated task.

Organization

Code sharing

Everyone has been granted access to a GitHub repository, containing the source code of the Ivido platform. From this repository, branches will be created, following the Frankenstyle naming conventions, for example: *bug_plugintype_pluginname* or *dev_plugintype_pluginname*.
Besides GitHub, the team has also been granted access to Ivido's DropBox, which contains all important documentation on the platform and product specifications. The team itself will use Google Drive for the project only to store all important files, such as hour logs, scrum documents, draft versions of reports, minutes and agendas.

Timetable

To have a good idea of what team members are spending most of their time on and to approximate how long a task takes to complete, each team member will log their hours in a Google Drive sheet. Also, a specification of how the hours are spend will be given.

Terms of agreement
We have agreed the following terms with our client Tristan Garssen:

1. The team will work on the product 32 hours per week, preferably at the office (Radex Innovation Centre, room 19a), at the same time.
2. The team will work on the final report and all other documentation 8 hours per week.
3. The team will meet with Tristan Garssen once a week.
4. The team will meet with Felienne Hermans once every two weeks at TU Delft.
5. The team members will give notice when they cannot be present at the office or at meetings.
6. The team will prepare meetings using agendas, scrum documents and will take minutes during meetings.
7. All team members will be punctual and be on time for meetings.
8. All team members will perform their assigned tasks before the deadline is met. If it looks as though there are difficulties meeting the deadline, all team members will seek help from other member in time, to avoid a delay.
9. All team members agree to reflect on problems occurring during a scrum phase, should there be any.
10. Each team member has the right to point out whether any of these rules are being broken.
11. Each team member has the right to amend or extend these rules in agreement with the rest of the team.

In return, Tristan Garssen will provide a good working environment.

D

Pilot

Testen netwerk plugin Ivido

Beste Deelnemer,

Hartelijk dank dat u deel wilt nemen aan het testen van de functionaliteit en gebruiksvriendelijkheid van de netwerk plugin binnen het Ivido platform. Met behulp van deze vragenlijst loopt u door de verschillende onderdelen van het platform en maakt u kennis met de mogelijkheden. Nadat u een stap heeft ondernomen, kunt u feedback geven via dit formulier.

De opdrachten zijn het makkelijkst uit te voeren in tweetallen.

*Vereist

Contact toevoegen

Zoek een bekende en stuur deze persoon een uitnodiging om contact te worden.

1. **Was de opdracht duidelijk?** *
   Markeer slechts één ovaal.

	1	2	3	4	5	
Onduidelijk	◯	◯	◯	◯	◯	Duidelijk

2. **Was de opdracht moeilijk uit te voeren?** *
   Markeer slechts één ovaal.

	1	2	3	4	5	
Moeilijk	◯	◯	◯	◯	◯	Makkelijk

3. **Bent u nog ongewone dingen tegen gekomen?**

4. **Heeft u het gevoel dat er nog iets miste?**
   Is er nog iets waar u behoefte aan heeft bij het uitvoeren van deze opdracht?

5. **Heeft u nog opmerkingen?**

Pas uw profiel aan

U kunt één of meer van de volgende acties uitvoeren: voeg een beschrijving, uw woonplaats en/of uw beroep toe.

6. **Was de opdracht duidelijk?** *
   Markeer slechts één ovaal.

	1	2	3	4	5	
Onduidelijk	◯	◯	◯	◯	◯	Duidelijk

7. **Was de opdracht moeilijk uit te voeren?** *
   Markeer slechts één ovaal.

	1	2	3	4	5	
Moeilijk	◯	◯	◯	◯	◯	Makkelijk

8. **Zijn er profielvelden die u graag erbij wilt zien?**
   Naast de huidige profielvelden, is er nog informatie die u graag op iemands profiel (denk aan een professional) wilt zien?

9. **Bent u ongewone dingen tegen gekomen?**

10. **Heeft u het gevoel dat er nog iets miste?**
    Is er nog iets waar u behoefte aan heeft bij het uitvoeren van deze opdracht?

11. **Heeft u nog opmerkingen?**

Uitnodiging accepteren

Herlaadt de pagina mits u nog geen uitnodiging heeft.

12. **Was de opdracht duidelijk?** *
    Markeer slechts één ovaal.

	1	2	3	4	5	
Onduidelijk	◯	◯	◯	◯	◯	Duidelijk

13. **Was de opdracht moeilijk uit te voeren?** *
    Markeer slechts één ovaal.

	1	2	3	4	5	
Moeilijk	◯	◯	◯	◯	◯	Makkelijk

14. **Bent u ongewone dingen tegen gekomen?**

15. **Heeft u het gevoel dat er nog iets miste?**
    Is er nog iets waar u behoefte aan heeft bij het uitvoeren van deze opdracht?

16. **Heeft u nog opmerkingen?**

Bericht versturen

Stuur één van uw contacten een bericht.

17. **Was de opdracht duidelijk?** *
    Markeer slechts één ovaal.

	1	2	3	4	5	
Onduidelijk	◯	◯	◯	◯	◯	Duidelijk

18. **Was de opdracht moeilijk uit te voeren?** *
    Markeer slechts één ovaal.

	1	2	3	4	5	
Moeilijk	◯	◯	◯	◯	◯	Makkelijk

19. **Bent u ongewone dingen tegen gekomen?**

20. **Heeft u het gevoel dat er nog iets miste?**
    Is er nog iets waar u behoefte aan heeft bij het uitvoeren van deze opdracht?

21. **Heeft u nog opmerkingen?**

Bestand versturen

Stuur één van uw contacten een bestand.

22. **Was de opdracht duidelijk?** *
    Markeer slechts één ovaal.

	1	2	3	4	5	
Onduidelijk	◯	◯	◯	◯	◯	Duidelijk

23. **Was de opdracht moeilijk uit te voeren?** *
    Markeer slechts één ovaal.

	1	2	3	4	5	
Moeilijk	◯	◯	◯	◯	◯	Makkelijk

24. **Bent u ongewone dingen tegen gekomen?**

25. **Heeft u het gevoel dat er nog iets miste?**
    Is er nog iets waar u behoefte aan heeft bij het uitvoeren van deze opdracht?

26. **Heeft u nog opmerkingen?**

Contact blokkeren

Blokkeer één van uw contacten. Maakt u geen zorgen u kunt uw contact hierna weer deblokkeren.

27. **Was de opdracht duidelijk?** *
    Markeer slechts één ovaal.

	1	2	3	4	5	
Onduidelijk	◯	◯	◯	◯	◯	Duidelijk

28. **Was de opdracht moeilijk uit te voeren?** *
    Markeer slechts één ovaal.

	1	2	3	4	5	
Moeilijk	◯	◯	◯	◯	◯	Makkelijk

29. **Bent u ongewone dingen tegen gekomen?**

30. **Heeft u het gevoel dat er nog iets miste?**
    Is er nog iets waar u behoefte aan heeft bij het uitvoeren van deze opdracht?

31. **Heeft u nog opmerkingen?**

Personen vinden

Vind de persoon die voldoet aan de beschrijving.

Vind iemand uit Nijmegen

Mocht u niemand vinden, probeer dan een andere stad

32. **Wie heeft u gevonden?** *

Vind iemand met het beroep 'huisarts'

33. **Wie heeft u gevonden?** *

Vind iemand van de organisatie 'Ivido'

34. **Wie heeft u gevonden?** *

Vind iemand met het beroep 'fysiotherapeut' uit Utrecht

35. **Wie heeft u gevonden?** *

36. **Bent u ongewone dingen tegen gekomen?**

37. **Heeft u het gevoel dat er nog iets miste?**
Is er nog iets waar u behoefte aan heeft bij het uitvoeren van deze opdracht?

38. **Heeft u nog opmerkingen?**

Algemene opmerkingen

39. **Is de structuur van de pagina logisch?** *
Markeer slechts één ovaal.

	1	2	3	4	5	
Onlogisch	◯	◯	◯	◯	◯	Logisch

40. **Heeft u nog op- of aanmerkingen over het uiterlijk?**

41. **Bent u over het algemeen nog ongewone dingen tegen gekomen?**

42. **Heeft u het gevoel dat er over het algemeen nog iets miste?**
Is er nog iets waar u behoefte aan heeft binnen de plugin?

43. **Heeft u nog algemene op- of aanmerkingen**

E

SIG feedback

Analyse

De code van het systeem scoort 3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Size en Unit Complexity.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld *my_network_get_recent_conversations* in *lib.php*, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. In dit voorbeeld bestaat de methode eigenlijk uit drie functionele blokken die met commentaar van elkaar worden gescheiden: het eerste stuk is voor het ophalen van recente berichten, het tweede stuk voor het ophalen van ongelezen berichten, en het derde stuk voor het combineren van de resultaten van de eerste twee stappen. Het commentaar is overigens erg goed, maar desondanks wil je deze structuur ook in de code aangeven in plaats van alleen maar met commentaar. Op die manier wordt het makkelijker om later één van de functionele blokken te hergebruiken. Ook kun je de stappen op die manier onafhankelijk van elkaar testen.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

Over het algemeen scoort de code dus gemiddeld, hopelijk lukt het om nog wat verbeteringen aan te brengen tijdens de rest van de ontwikkelfase.

Bibliography

[1] Require JS. Why amd?, 2017. URL http://requirejs.org/docs/whyamd.html.

[2] Moodle. Writing phpunit tests, 2016. URL https://docs.moodle.org/dev/Writing_PHPUnit_tests.

[3] Moodle. Output api, 2016. URL https://docs.moodle.org/dev/Output_API.

[4] Mustache. Github repository. URL https://mustache.github.io/.