Delft University of Technology

An Online Patient Scoring System

Authors:

W.J. Vaandrager
O.G.T Maas

Supervisors:

Prof. G.J.P.M. Houben
K. van Golen MSc.

December, 2014

Preface

In this report the development of the application 'An Online Patient Scoring System' will be discussed. This application was built for Clinical Graphics. This project was executed as our Bachelor Project.

We would like to thank the Clinical Graphics team for giving us this opportunity and especially Korijn van Golen for supporting us during the project and providing us with advice where necessary. Furthermore, we would also like to thank Geert-Jan Houben from the Web Information Systems group at the TU Delft for supervising our project as our TU Coach.

Olaf Maas
Willem Vaandrager
26 November 2014, Delft

Summary

At this moment when a clinician needs to conduct a study he will send all his patients at various moments a questionnaire form. After retrieving these forms the clinician will manually process all the data by hand. This is a time consuming job and could be automatized. During this project we created an application that automates this process. When using the developed application the only thing a clinician has to do is create forms, add patients and tell the system when a patient has to receive a form.

In the orientation phase multiple off the shelf solutions have been considered, however most of these solutions had some flaws. So we decided to build our own application using C# in combination with ASP.NET MVC5. By using ASP.NET MVC 5, the application could easily be deployed in the Microsoft Azure cloud environment. This increases the scalability of the application.

During the process we had weekly meetings with the client. During these meetings we showed the progress made and the client could suggest changes to be made. By having these meetings we made sure the client was satisfied with the delivered product. In week 7 and in week 10 of the project the code was sent to the "Software Improvement Group" to assess the maintainability of the code. During the last feedback our code scored four out of five stars.

The delivered product is a web-based application which contains a formbuilder with a drag and drop interface. Furthermore, it contains a study overview where clinicians can manage their studies by adding patients and setting a schedule. The retrieved data can be exported to an Excel document at any time during the study. Therefore, the delivered product fulfills the requirements and will be an improvement on the current way studies are conducted.

Contents

1 Introduction **6**

I Orientation **7**

2 Problem Analysis **7**
2.1 Current situation 8
2.2 Problem Description 8
2.3 Prerequisites 8
2.4 Goals 8

3 Existing Technologies **10**
3.1 Survey System 10
3.1.1 Features 10
3.1.2 Solutions 10
3.1.3 Survey Project 11
3.1.4 SurveyMonkey 11
3.1.5 Building our own system 12
3.1.6 Conclusion 12
3.2 Mail system 12
3.2.1 Comparison 12
3.2.2 Conclusion 14

II Design **15**

4 Requirements **15**
4.1 Functional Requirements 15
4.2 Non-Functional Requirements 17

5 System Design **18**
5.1 High Level overview 18
5.2 Domain Model 19
5.3 Design Patterns 19
5.3.1 MVC 19
5.3.2 Dependency Injection 20

6 Database **21**

III Implementation **22**

7 Finished Sprints **22**
7.1 Sprint 1: Setting up the foundation 22

7.2 Sprint 2: Parsing & Rendering Forms 22
7.3 Sprint 3: Verifying Answers 23
7.4 Sprint 4: Refactoring & Mailing 23
7.5 Sprint 5: Layout Customization & Saving Progress 23
7.6 Sprint 6: Polishing & Testing 24

8 Quality Assurance 25
8.1 Testing 25
8.2 Software Improvement Group Feedback 25
8.2.1 Feedback 1 25
8.2.2 Improvements 25
8.2.3 Feedback 2 26

9 Used Libraries 27
9.1 Formbuilder 27
9.2 Bootstrap 27
9.3 .NET Libraries 28
9.3.1 JSON.NET 28
9.3.2 EPPlus 28
9.3.3 Imageprocessor 29

10 Product Overview 30
10.1 Study 30
10.2 Form 31
10.3 Mailing system 31
10.4 User system 32

IV Final Remarks 33

11 Evaluation 33
11.1 Requirement Evaluation 33
11.1.1 Must Have 33
11.1.2 Should Have 33
11.1.3 Could Have 34
11.2 Process Evaluation 34

12 Recommendations 35
12.1 Single Forms 35
12.2 Built in Statistics 35
12.3 Advanced Forms 35
12.4 Graphical Elements in Forms 35
12.5 More Customisation Options 36

13 Conclusion 37

References 38

V Appendix **39**

A Initial Project Description **39**

B Sig Feedback 1 **40**

C Sig Feedback 2 **41**

D Project Plan **42**

1 Introduction

In a constantly growing digital world the medical world also wants to digitalize most of its data and processes. Clinicians want an easier way to conduct studies and obtain useful medical data from patients. In order to solve this problem we had the assignment to build an online application that allows clinicians to create forms and conduct studies with these forms. The studies should contain the option to add a list of patients and set a schedule of what and when forms have to be send.

This report provides all information of how this application was build and what decisions were made during the process. The report consists of four parts: Orientation, Design, Implementation and Final Remarks.

In the Orientation we will provide an analysis of the current problem and what goals we want to achieve in order to solve this problem. This part also contains the research we did on tools, which could help us during the project. In the Design we will give an overview of the requirements. We also define what our application needs in order to accomplish the goals set in the orientation phase. This section will also elaborate on the architecture and the database of our product.

The Implementation provides an overview of what libraries we used and what our final product looks like. We will give a detailed insight in how our application is constructed and what functionality the application has. This part will also give a process overview of how the application was created during the project.

In the final part, Final Remarks, the project and its process are evaluated. This part will give an elaboration of which requirements are achieved. Furthermore, it gives overview of which requirements are changed or dropped during the process. We will also give some recommendations on how the project can be developed further.

In this report not much insight is given on the privacy aspect of our application. However in cooperation with the client we thought this aspect through. In the end this aspect of our application will be handled by our client Clinical Graphics.

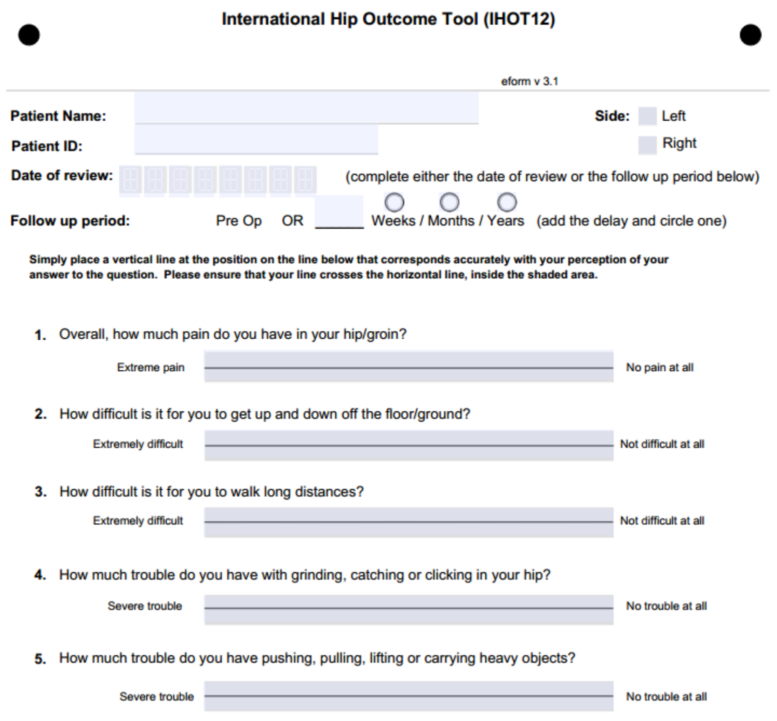
Part I

Orientation

In this part we will present the results of the orientation phase of the project. During this phase we tried to get a better understanding of the client's problem. We achieved this by interviewing the client and by conducting research on similar systems.

2 Problem Analysis

This section will provide an analysis of the main problem we tried to solve during the project. In the first part an overview of the current situation is provided. Afterwards a description of the main problem is given. Finally the main goal of the project is defined.



International Hip Outcome Tool (IHOT12)

eform v 3.1

Patient Name:

Patient ID:

Side: Left / Right

Date of review: (complete either the date of review or the follow up period below)

Follow up period: Pre Op OR ______ Weeks / Months / Years (add the delay and circle one)

Simply place a vertical line at the position on the line below that corresponds accurately with your perception of your answer to the question. Please ensure that your line crosses the horizontal line, inside the shaded area.

1. Overall, how much pain do you have in your hip/groin?
   Extreme pain ———— No pain at all
2. How difficult is it for you to get up and down off the floor/ground?
   Extremely difficult ———— Not difficult at all
3. How difficult is it for you to walk long distances?
   Extremely difficult ———— Not difficult at all
4. How much trouble do you have with grinding, catching or clicking in your hip?
   Severe trouble ———— No trouble at all
5. How much trouble do you have pushing, pulling, lifting or carrying heavy objects?
   Severe trouble ———— No trouble at all

Figure 1: Example of a form [1]

2.1 Current situation

In the current situation, when a clinician wants to conduct a clinical study on the status of a group of patients, he will use questionnaires (like the one in Figure 1). Each patient will receive multiple questionnaires on a set time intervals. (e.g. 1 week after surgery, 1 month after surgery). After a patient has filled in a questionnaire, the clinician will process the form manually.

2.2 Problem Description

The process as described in the previous section is a labour intensive and time consuming job. If this process could be automated this will save the clinician valuable time. Time which could be spent on other activities. As a service for their clients the company Clinical Graphics wants to create a web service through which clinicians can conduct a study. A study is a research on a group of patients with similar illness or medical condition or who have undergone similar surgical procedures. By using the proposed method a clinician can easily collect and analyse the study data. All questionnaires are sent automatically to the patients whom are participating in the study. After the study has finished the clinician can view and export the data of the study.

2.3 Prerequisites

When the project is finished, employees from Clinical Graphics will maintain our code. To make maintenance easy, the system should be written in a language that they already work with. The languages suggested by the client were either Python or C#. We chose C#, since one of our team members already had experience with this language. C# also has frameworks built around it for this kind of web-based applications.

2.4 Goals

At the start of our project we set multiple goals we wanted to achieve during this project. These goals were set during the orientation phase and some goals may have changed during the development in agreement with the client. The main goal of the project was to create a system where clinicians can conduct a clinical study. To achieve this goal the system should have the following global components:

1. **Questionnaire System**
   Using the system a clinician should be able to create a questionnaire. The forms created using this system will be available only to the clinician who created the form.

2. **Study Administration**
   Using the system a clinician should be able to create a study. A study consists of patients and questionnaires. As long as the study has not started the clinician should be able to add patients to his study. A study further consists of a number of questionnaires. A clinician can choose to add some predefined questionnaires or he can choose to add his own questionnaires.

3. **Mail System**
   The system should contain a mailing system to automatically send mails, which contain a link to a form, to the patients on set time schedules. A clinician can set the days the system has to email the patients assigned to a study. Reminders will be sent to the patient if the patient does not fill in the questionnaire on time. If the patient fails to fill in the questionnaire before the final deadline, that particular questionnaire will be skipped and if required the next one will be sent on the next interval. The mailing system should also contain an editable template of the standard email for admins and an option to sign the email with a logo and information of the clinic.

4. **Data Export**
   During and after a study a clinician should be able to see the data that is obtained in the study. The clinician must be able to export the generated data to an excelsheet [2].

3 Existing Technologies

In order to speed up the process of developing the application, we choose the look into different services that can support our application. After doing initial research we discovered that two main modules of our application could be substituted with an existing solution. In this section we will discuss the options for substituting two big parts of our application, the survey system and mail system.

3.1 Survey System

The Survey System is the core of the final product, in this section we will answer the question: Is there an existing solution that can be used in our system or should we build something new? To answer this question we will first look at some existing solutions and then we will compare the best fitting existing solution with the option to build something ourselves.

3.1.1 Features

To answer the first question we will assess available survey systems on the following features:

- **User Interface:** It is important that creating a questionnaire is easy for clinicians. The clinician should be able to create a survey without having to use an extensive manual.
- **Features:** What kind of questions can the system handle? How hard is it to add more types of questions?
- **Integrability:** When creating a survey we do not want a clinician to be sent to different sites. To achieve this it should be possible to integrate the chosen software within our system.
- **Data Storage:** Where does the system store his data and in what format? How can we retrieve the data and use it for analysis?
- **Pricing & License:** The chosen software will be used as part of a commercial service. Therefore, it is important to check if the license of the survey software allows this. If this is not the case, we will also check if it is possible to buy the licence in order to use the software in our application.

3.1.2 Solutions

There are many solutions on the web that offer systems to create surveys. Based on our requirements we selected the two best fitting solutions. We will also assess the possibility to build this part of the system ourselves.

1. Survey Project [3]: An open source project written in .NET which when installed "can be used by anyone who is properly authorised to create and design surveys, webforms or data entry forms online".
2. SurveyMonkey [4]: A commercial web based service used to create and write surveys.
3. Build our own system

3.1.3 Survey Project

At first glance the user interface of the Survey Project for creating a survey does not looks up to HTML standards. Digging through the source of the front page confirms this, using tables to layout rather than representing data is considered a bad practice by the W3C [5]. Furthermore, most of the styling is done in the style property of each method. This is also considered a bad practice by the W3C [6]. After logging in and trying to create a survey it appears the program has a steep learning curve. The editor has many buttons and many options and some buttons do not work. Out of the box the Survey Project can handle all basic questions. Branching is supported however during our demo session we could not find out how.

The Survey Project offers some form of integration with other applications. However, the documentation on this topic consists of a short document. Creating and adding new surveys still takes place in the administration panel of the Survey Project.

Data is stored in a MSSQL database and can be accessed through the Survey Project's administration panel. The database model is well documented and should not be hard to access the data using a third party application.

The Survey Project is distributed free under "GNU General Public License version 2" (GPLv2). This license states that if the Client wants to sell and distribute our application to a third party they should make the code open source. However, there is still a debate going on about the question if a web service is distribution to a third party.

3.1.4 SurveyMonkey

In contrary to the Survey Project the web-based user interface provided by SurveyMonkey is a modern drag & drop interface. It provides an easy way to create a simple survey with just a few clicks. Out of the box SurveyMonkey offers many types of questions. It is not possible to add more types of questions. Integrating a SurveyMonkey survey into a website is as easy as copy pasting some html code [7]. However, the creation of a survey still takes place on the SurveyMonkey website. It is not possible to integrate this into a website.

SurveyMonkey stores the data in their own database. This data can be retrieved using an API. As this API is well documented it should not be a problem to retrieve the data. As a commercial application SurveyMonkey has multiple plans. Only the gold and platinum plans include unlimited questionnaires and unlimited responses. For every user a fee should be payed. This fee is €800 per user per year.

3.1.5 Building our own system

By building our own system we can use a form design tool such as Formbuilder.js [8] as the drag & drop user interface we want. Building our own system costs extra time, but the system can be extensible. There is no license and there will be no licensing cost.

3.1.6 Conclusion

We think that The Survey Project is too broad in our situation. Furthermore, we think the integration possibilities are too limited and that the user interface will be too complicated for the end users. SurveyMonkey would be a better option, however the client has two problems with it. Firstly SurveyMonkey is too expensive and secondly SurveyMonkey stores medical data in their database, while the client prefers to store this data in their own database. This leaves us with only one option left, which is building our own Survey System.

3.2 Mail system

Our application will automatically send patients invitations to fill in the forms. This will be done according to a schedule that is defined by a clinician. In order to send these emails we will use an email service to be able to easily generate and send the emails.

3.2.1 Comparison

In this section we will compare the most commonly used mailing services. The key aspects we compare these systems to are pricing and their documentation and integration support with our application. Since we are sending the emails personally and not in huge amounts to a lot of people we do not need to take into account most extra features the companies offer. We will focus on gaining statistics of the results of the forms and not of the emails, so extra information about the sending behaviour is unnecessary. The services that will be compared below are: Mandrill, SendGrid, Mailjet and Elastic Email.

3.2.1.1 Mandrill

Mandrill is an email infrastructure service by MailChimp. The pricing is fairly cheap and is easily scalable as you pay according to what you use. Mandrill offers 12,000 emails a month for free and starts billing $0.20 per thousand emails if you send more emails after that.

Mandrill offers a lot of mailing services and has a clear documentation. It also provides an API for the most commonly used languages used for mailing services and wrappers made by third parties for other languages.

Another advantage of Mandrill is that our client, Clinical Graphics, is already using Mandrill in some of its programs and is familiar with this service. Thus choosing Mandrill will ease the maintainability of our application for them.

3.2.1.2 SendGrid

SendGrid is the biggest provider of transactional email delivery. They provide 200 emails a day for free and offer packages afterwards for sending more emails. The cheapest of those packages is bronze for $9.95, which offers 40,000 emails a month.

SendGrid offers a lot of options for integration into code on many platforms. It also has the most extensive API documentation of the four services.

3.2.1.3 Mailjet

Mailjet asks about the same price as SendGrid for its services. Their free plan also includes 6000 emails a month with a limit of 200 emails a day. After that the cheapest package they provide is 30,000 emails a month for $7.49 a month. Their API integration and documentation is clear, but less extensive than that of other providers. They also lack an API package for C#. This is a major disadvantage as we want to implement our system in C# as discussed in section 2.3.

3.2.1.4 Elastic Email

Elastic Email offers emails for as low as $0.09 per thousand. Looking into their pricing it became apparent that the prices drop as you send more emails. In order to reach the $0.09/thousand emails, an amount of half a million emails should be sent with higher prices, starting with $0.99/thousand and slowly lowering to $0.09. Elastic email has decent API documentation and includes a lot of code samples to send emails via different platforms.

3.2.2 Conclusion

Mailjet is not an option for us since it does not have an option for C# integration and does not offer more services than other providers. Elastic email offers the lowest price when sending a lot of emails. However, since we will only be sending a limited amount of patients a few emails per month, we do not expect to reach the amounts needed for Elastic Email to become the cheapest solution.

SendGrid and Mandrill are both good viable options for our problem. They both offer a fair amount of free emails a month and clear documentation about integrating the software in your desired application. While SendGrid is more commonly used and its integration options are a bit more extensive than Mandrill's, we think Mandrill will suit the application better. Mandrill is easier to scale if you need to send more mails. Since our application will be used on a smaller scale, Mandrill is more practical. The fact that Clinical Graphics already uses Mandrill will help with maintainability and compatibility within the company as well.

Part II

Design

4 Requirements

During our Design phase, we agreed with the client upon the following requirements. We will evaluate whether or not these requirements were met in section 11.1.

4.1 Functional Requirements

Must Have

1. **User System**
   (a) A clinician must be able to create an account.
   (b) A clinician must be able to sign in using the account he created.
   (c) A clinician must be able to change their username/password/email.
   (d) When a clinician forgets his password he must be able to receive an email to change his/her password.
   (e) A clinician must have access to the menu when signed in.
2. **Study**
   (a) A clinician must be able to create a new study.
   (b) A clinician must be able to add patients to a study.
   (c) A clinician must be able to remove patients from a study.
   (d) A clinician must be able to add a starting date to a study.
   (e) A clinician must be able to add a form to a study that has not started yet.
   (f) A clinician must be able to remove a form from a study that has not started yet.
   (g) A clinician must be able to set a schedule for a study.
3. **Form**
   (a) A clinician must be able to create a new form.

   (b) A form created by a clinician must only be available to the clinician who created it.
   (c) A patient must be able to fill in a form.
   (d) A patient must be able save the progress he made.
4. **Mailing System**
   (a) The mailing system must automatically send invitations to fill in a form via email.
   (b) The email sent by the system must contain unique link to fill in the form.
5. **Data Export**
   (a) A clinician must be able to export a study's data to Excel.
   (b) If a patient did not fill in all forms this will be apparent in the exported data.

Should Have

1. **User System**
   (a) A clinician should be able to add the logo of their clinic to his profile.
   (b) A clinician should be able to manage the account of patients.
   (c) A clinician should be able to get an overview of their patients and their forms.
   (d) A clinician should be able to update a patient's email address.
2. **Study**
   (a) A clinician should be able to see the results of a study.
   (b) A clinician should be able to set when one or more reminders must be sent. (in days after the original invite was sent).
3. **Form**
   (a) The logo a clinician uploaded should be displayed above all their forms.
   (b) A form should open with the answers that the patient already filled in previously.
   (c) A clinician should be able to see the results of a form.
   (d) A clinician should be able to edit forms that are not in use.

   (e) A clinician should be able to create a new form based on an existing form.

4. **Mailing System**
   (a) The mailing system should be able to send reminders to patients, when forms are not filled in by the patient in time.
   (b) The logo of the clinic should be included in an email to a user.
   (c) A clinician must be able to change the colors of the email template.

Could Have

1. **Form**
   (a) A clinician should be able to create a branching form where questions will be skipped based on the answers.
   (b) When a patient is filling in a form it should save automatically after every change the patient makes to the document.

4.2 Non-Functional Requirements

1. A page should load within a second.
2. The site should work on mobile devices.
3. The site should work with IE8.
4. The application should be properly tested.

5 System Design

In this chapter the design of our system is discussed. In the first chapter a high level overview of the components of our system is given. After this an overview of the main design patterns used in our application will be given.

5.1 High Level overview

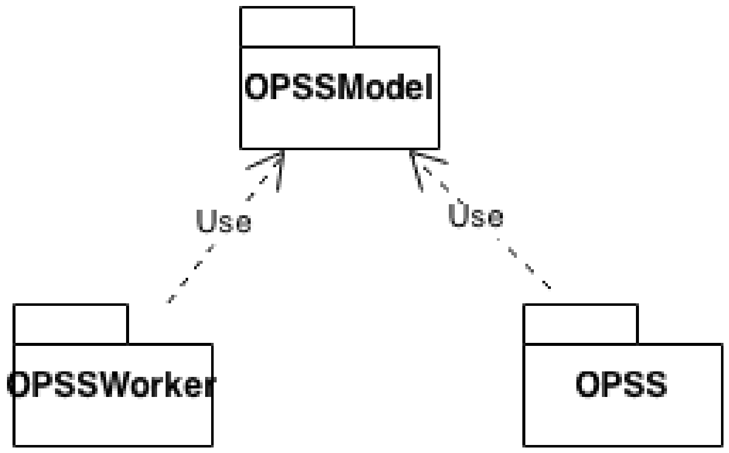


Figure 2: Example of a form

The delivered solution consists of four main components. OPSSModel is the basis which is used by all other projects as it contains all our domain classes. It also provides the service StudyContext which is the class that provides access to the database using Entity Framework [9].

OPSS is the project which houses the web application part of the solution. It is an ASP.NET MVC5 [10] project which uses OPSSModel as the data model. The Model folder in OPSS only houses viewmodels which are used by the controllers to pass data to the views. Furthermore, it contains all controllers and views necessary for the web application.

The third project is OPSSWorker, a Microsoft Azure WebJobs project. WebJobs are processes in Azure which can be run on certain time interval or on certain conditions. In the case of our solution we need a WebJob that runs once every day to check if invites or reminders must be sent to patients. Our last project is OPSSTest, which houses our Unit Tests.

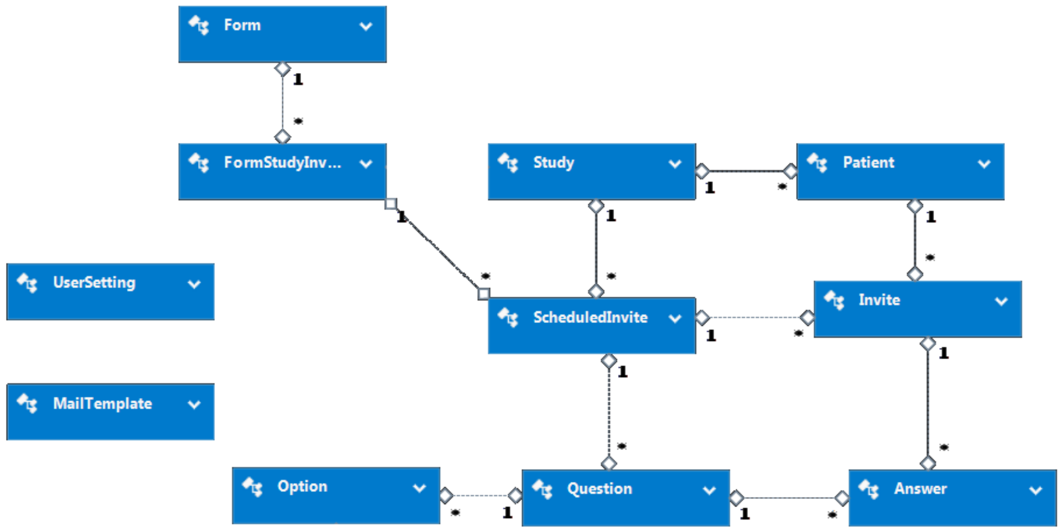


Figure 3: Model

5.2 Domain Model

The diagram in figure 3 shows our domain model. All option classes and children of the Question class are not in this overview due to the amount of space it would take (every Questiontype is a child of Question).

5.3 Design Patterns

In our application we use a few design patterns to help us build our application and keep our code more manageable. In this section we will discuss the design patterns we have used in the development.

5.3.1 MVC

Our frontend system mainly uses the Model View Controller [11] pattern. This pattern has three main components. The Model is the representation of the data, the Controllers are the glue between the Model and the View. The View is the presentation layer, which presents the data given by the controller to the user.

We use the ASP.NET MVC5 [10] framework to make it easier to work with the MVC pattern. MVC5 is a framework developed by Microsoft that implements MVC. Using MVC5 has a few advantages. The first advantage of MVC5 is that it is possible to generate views and controllers based on the model using a method called scaffolding [12]. Secondly it provides all functionality needed to use MVC. Furthermore, it comes with an HTML view engine called Razor, which reduces the amount of code needed to render a view. Finally, it comes with an automatic client-side and server-side code validation function which makes validation easy to implement.

In MVC5, when a user visits a link, he is actually calling a method of a controller. Using a POST or GET request, parameters can be passed to a method. MVC5 will create an instance of the controller and execute the method. An action will typically load some data from the model, do some operations on it and return a certain view, which will be rendered by Razor.

5.3.2 Dependency Injection

In our project we use Dependency Injection (DI), which is a design pattern that uses the "Inversion of Control" principle. The idea of this pattern is that a service is passed to a dependent class, instead of initializing a service in a class depending on that service.

The main advantage of using this principle is that it increases the testability of code. If a service is passed as a parameter it is easy to pass a mocked object, which simulates certain behaviour of that service. Furthermore, it reduces the knowledge needed by the dependent class about the service it is using.

In the project the Unity Application Block (UAB) [13] was used for DI. UAB is a part of the Microsoft Enterprise Library. The main reason for using Unity is its easy integrability with the ASP.NET MVC framework. Furthermore, Unity does not require much code or imports of libraries, which keeps the codebase clean.

6 Database

We used MSSQL [14] and Entity Framework 6.1 for the database connection in our application. Entity Framework is an Object Relational Mapping (ORM) framework developed by Microsoft. An ORM Framework is a framework that maps an Object Oriented Data model to a relational data model. The main advantage of Entity Framework is that it reduces the amount of code needed to access the database. Furthermore, it creates the database model for you, based on the given model which saves much time. Lastly it allows us to query the database using Linq. By using Linq it becomes easier to change the database type without having to refactor a lot of code.

The main reason for using MSSQL as our relational database management system is the fact the client wants to run our application in the cloud environment of Microsoft: Azure. Azure uses SQL Azure as its RDBMS which is a subset of MSSQL.

Part III

Implementation

7 Finished Sprints

We started the building process of the application in week 4. During the 6 weeks that followed we have done 6 sprints. In this section we will elaborate what was done during these weeks and what challenges we came across.

7.1 Sprint 1: Setting up the foundation

During our first week of the implementation phase our main concern was building the foundation of our application. During this week we were setting up our initial project and testproject. We both experimented with ASP.NET, with which both of us did not have any prior experience.

At the end of this week we had built our first models and scaffolded the corresponding views and controllers for editing these models. Furthermore, we extended Formbuilder with an extra button and integrated Formbuilder into our application. We added the ASP.NET Identity [15] library to make sure users could login and register.

7.2 Sprint 2: Parsing & Rendering Forms

One of the main challenges in the building process of the application was the ability to parse the forms built by Formbuilder.js. We addressed this challenge in the second week by building the formparser. When a user is building a form using Formbuilder.js a background process is sending JSON to the server. The server stores this JSON directly in the database. When a study starts, this JSON will be converted into Question objects. The parser that handles this conversion was built during this week.

The second challenge was the rendering of the forms. During the building process the rendering of a form is handled by Formbuilder.js, however after conversion Formbuilder.js is not able to render the Question objects. To render the forms we created a view that receives a list of Abstract Questions as input and loads the corresponding partial views of these questions.

7.3 Sprint 3: Verifying Answers

At the end of the previous sprint the application was able to parse and render forms. However, after rendering a form the application was not able to store the user input. During this sprint we built the functionality to handle the user input of the surveys.

This problem was harder to fix than we initially thought. This was caused by the dynamic nature of the created forms that made it unable to use the default verification mechanism of ASP.NET. The default verification system expects that verification constraints are known at compile time. However, in our system the user sets these constraints at runtime, so we implemented a verification system ourselves, which can handle constraints inserted at runtime.

7.4 Sprint 4: Refactoring & Mailing

The system that checks which invites should be sent should not have a dependency on our whole MVC system. Prior to sprint 4 we were not able to achieve this with the current structure. This was the main reason we decided to refactor the whole project. We extracted the domain model of our ASP.NET MVC project and placed it in a separate library. Both the .NET MVC project and the Mailing project have a dependency on this project.

We also implemented another key part of our application, the automatic mail system. The mail system sends invites and reminders to users. Furthermore, the export functionality of data was implemented this week. Up until now it was only possible to store the data in the database, at the end of the week we were able to export the study data to an excel document.

7.5 Sprint 5: Layout Customization & Saving Progress

When a user fills in a form it should be done in a familiar environment. To achieve this, clinicians should be able to customize the layout of the forms. So we created an option for a clinician to be able to set the theme of a form. This was the main requirement we implemented in week 5. We introduced a Settings panel on which clinicians were able to change the colors of the forms they send. We also made it possible for a clinician to upload a custom image, which is saved in the Azure Cloud.

The second requirement was the ability for a user to save their progress of filling in a questionnaire. We implemented that a patient can save their progress any time when answering a survey. This enables the patient to resume with their previous answers later on. However, only when a user has filled in all required questions their survey is marked as finished. This will become apparent in the exported data.

We also changed the way a user can create a study. Prior to this sprint clinicians could add one scheduled invite at a time, which contained a form and a date when the questionnaire should be send. After consulting with the client we decided it was better that a user should just add forms to a study and then add dates to that form. This removed duplicate form entries when a form should be sent multiple times and created a better overview for the users.

7.6 Sprint 6: Polishing & Testing

This was our final sprint during the implementation phase. In this sprint we were mainly occupied with polishing the application. We enhanced the user interface by adding some fields, which made the interaction with the application easier. For example we added a colorpicker to a field where an HTML color code is expected, so a clinician does not need prior experience with HTML to fill in this field. We also implemented an image resizer using an external library [16]. This was necessary due to the fact that when a large logo is picked by the user it will slow down the mailing process. The large logo has to be uploaded to Mandrill for every form the clinician has set for that day. This unnecessary increases the running time of the mailing service.

We also wrote more test code where necessary and deleted some dead code. This dead code was mainly in the view. Furthermore, we deleted some buttons that were no longer used in the application. During the remainder of the week we were busy with fixing multiple tiny bugs such as a word count and a date bug. Fixing these bugs increased the stability of our application.

8 Quality Assurance

8.1 Testing

In order to guarantee a more stable application, testing assures that the program works as it should and does not crash on handling input. For our application we used unit tests. The unit tests were done using the built in C# unit testing framework. As a mocking framework we used MOQ4 [17].

At the start of our project we had the ambition to write many tests. However due to time constraint we were not able to write as many tests as we wanted. The best tested part in our application is the parser, which parses JSON to Question objects. Our main reason for testing this specific part of the application extensively is that an error could bias a study. This would be very undesirable behaviour.

8.2 Software Improvement Group Feedback

During our project we sent our code to the Software Improvement Group (SIG) [18] for feedback. The first time our code was sent to SIG for feedback was in week 7. The last time our code was sent to SIG was in week 10.

8.2.1 Feedback 1

The overall feedback we received was positive. Our code scored a 4 out of 5 stars on SIG's maintainability index. The reason our code did not get the highest score was due to a low score on Duplication and Unit Size.

When SIG assesses the unit size they search the code for methods that are longer than average. To increase maintainability these methods should be refactored to multiple smaller methods. The main reason for the low score on this part was the seed method of our initializer class, this method is only used by us to fill our database with test data.

Duplication is the amount of duplicate code that can be found throughout the codebase. Our code scored low on this due to the duplication found in our views. This was due to scaffolding option in the MVC Framework. When creating a new model we automatically generated the views of that model by scaffolding. This caused the views to contain a lot of duplicate code.

8.2.2 Improvements

During the remaining time we had in the development process we decided to take a critical look at the existing classes and search for long methods. During this process we made some changes to the code in order to reduce the unit size. We also paid more attention to the unit size and the amount of duplication when implementing new features in our project. This sometimes resulted in refactoring parts of the code.

8.2.3 Feedback 2

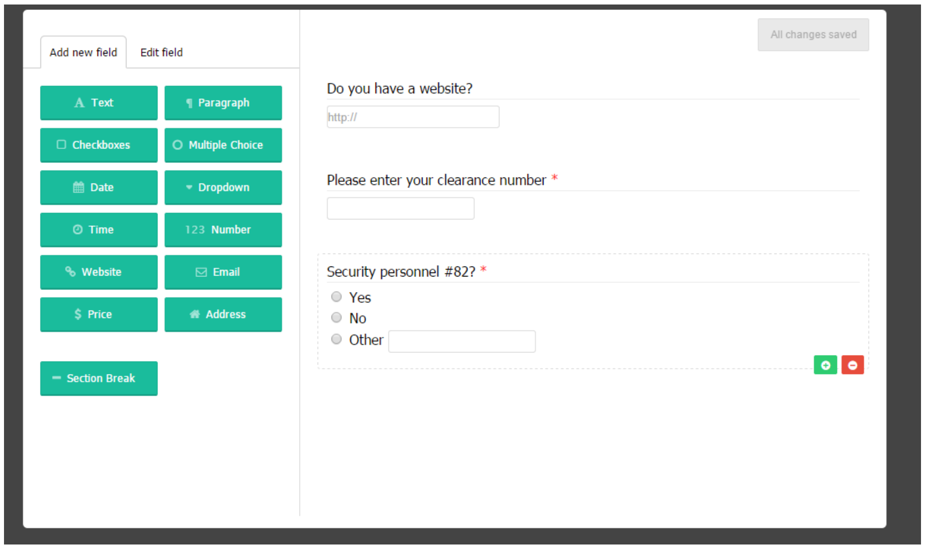
In the second review of SIG the overall rating our code scored for maintainability was the same. We did improve on code duplication and unit size. In general SIG was glad to see that we had taken their previous feedback into consideration and implemented most of their recommendations.

The biggest concern from SIG on our code was the code in the Razor templates, which is used in the views. It is hard to improve here, since these are standard components that MVC scaffolds when creating a new model. A recommendation for improving this could be to put parts of the general code of the views in partial views to decrease duplicate code.

9 Used Libraries

During the implementation phase we were on a tight schedule, therefore it was important to use some existing libraries to speed up the development. In this section we will discuss the libraries we used, explain what they do and explain why we used them.

9.1 Formbuilder



Do you have a website?

http://

Please enter your clearance number *

Security personnel #82? *

Yes

No

Other

Figure 4: Formbuilder in action

Formbuilder.js [8] is a front-end Javascript library we used in our application so clinicians are able to build and edit their surveys. It is written in CoffeeScript, which can be compiled into Javascript.

Using Formbuilder.js clinicians can create a whole questionnaire using a drag and drop interface. It supports all standard questions and can easily be extended if needed. Furthermore, it is possible to set constraints on the input users can provide (Integer only, max length, etc.). This can also be extended if necessary. Formbuilder is only a front-end library and outputs a structured JSON representation of the created form.

9.2 Bootstrap

Bootstrap [19] is a frontend HTML5 framework, it uses the mobile first approach for developing websites. The big advantage of Bootstrap is that it has multiple reusable components. Furthermore, it makes it easy to build a site that can run on multiple screensizes (Computer, Tablet, Phone). It is widely supported and is used by default in MVC5 that uses Razor as its view engine.

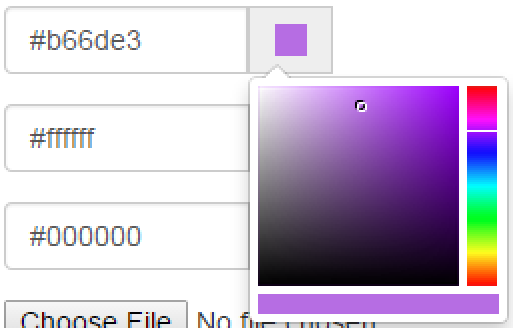


Figure 5: Bootstrap Colorpicker

To make the system easier to use we used some extensions to Bootstrap. All these extensions are custom input fields. The first extension we used is Bootstrap Datepicker, this extension spawns a calendar when a user is filling in a datefield. This makes it easy for a user to select a date.

We also used Bootstrap Colorpicker when a user is editing the color settings for his form so a user does not have to fill in a hexadecimal color code, which is too complicated for the average clinician.

The third extension we used is Bootstrap slider. Many medical surveys use a bar for a patient to indicate for instance levels of pain, concern or difficulty with tasks. A slider is a good alternative for these type of questions. A slider is not a default HTML element so we decided to add Bootstrap Slider.

9.3 .NET Libraries

9.3.1 JSON.NET

The JSON as generated by Formbuilder.js must be parsed to Question objects that can be stored in the database. We use JSON.NET to parse the JSON and then iterate through the JSON. JSON.NET is one of the most popular C# JSON parsers, furthermore it is open source and has LINQ support.

9.3.2 EPPlus

EPPlus is an open source library for generating Excel, which is used to fill the Excel sheets the clinicians download when they export the data. The main advantage of EPPlus is that it also supports the styling of the worksheet and it supports multiple Excel formats.

9.3.3 Imageprocessor

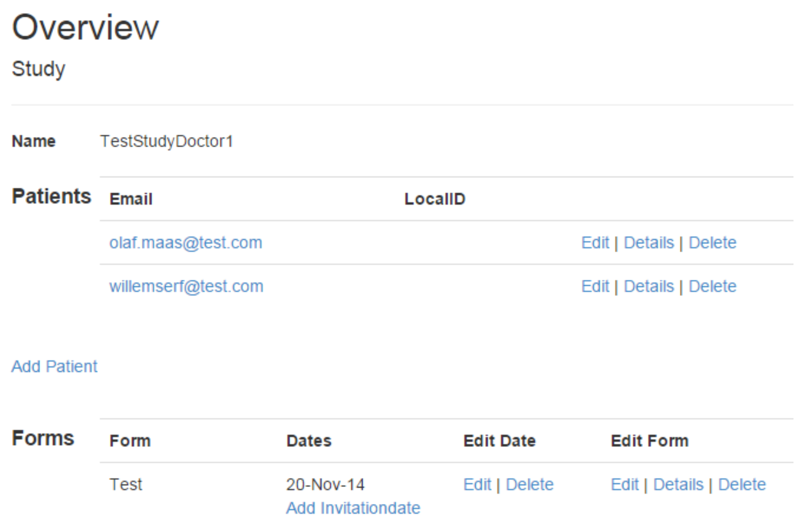
Imageprocessor is an open source library that is used in the application to resize the images uploaded by clinicians. The reason we used Imageprocessor is that it is lightweight and does not come with subscription services which competitors have. Furthermore, it is well documented and still actively developed.

10 Product Overview

This section will give a brief overview of the built application and what functionality it has. This overview will be given on the basis of the four components the application consists of: Study, Form, Mail system and User system.

10.1 Study

Under the studies tab a clinician finds all the studies he has created. When clicking overview the clinician gets an overview of a certain study. This overview contains a list of all patients and a list of all forms and for every form on which dates the form will be sent to all patients.



Overview

Study

Name TestStudyDoctor1

Patients

Email	LocalID	
olaf.maas@test.com		Edit | Details | Delete
willemserf@test.com		Edit | Details | Delete

Add Patient

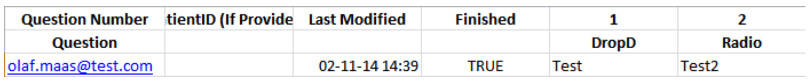
Forms

Form	Dates	Edit Date	Edit Form
Test	20-Nov-14 Add Invitationdate	Edit | Delete	Edit | Details | Delete

Figure 6: The overview

In this overview a clinician can easily manage the patients for the selected study. A patient can be added by clicking 'Add patient'. The only information required to add a patient is the email of a patient. After submitting the patient will be added to the list. Now a clinician will have the option to edit the patient's details or delete the patient.

In the same overview the clinician can set a schedule for the forms of the study. With 'Add Form' a form can be added to the study. After adding a form it is possible to add multiple dates on which the form will be sent to all the patients of the study. This can be done by clicking 'Add Invitationdate'. Each scheduled invite and form can be edited or deleted



Question Number	tientID (If Provide	Last Modified	Finished	1	2
Question				DropD	Radio
olaf.maas@test.com		02-11-14 14:39	TRUE	Test	Test2

Figure 7: Part of an exported excel sheet

The studies tab also contains an option to export the data. By clicking on this option an Excel File will be downloaded. This file contains a sheet for every form of the study. On this sheet all Questions and answers are rendered in a table form. When a patient has not finished the form this will be made apparent.

10.2 Form

Under the tab Forms a clinician can manage the forms he can use. By clicking on 'Create New' a clinician can create a new form. A form can also be based on another form and edited afterwards to save time. When adding a form, the new form will get options to edit, view or delete the form. When clicking 'Edit' on a form the formbuilder will be opened. As described in section 9.1, we used Formbuilder.js to create forms. With this library the clinician can easily build or edit forms. The created forms can also be previewed by the clinician to get a better impression of how the form will look when opened by a patient.

10.3 Mailing system

The Mailing System is in the back-end, this system contains templates for the mails, as well as services to send the invites and reminders on time and manage the unique keys of the surveys. The application for the admins contains an extra tab for editing mailing templates.

The mailservice will look through the database and check for scheduled invites that have to be sent today. It will then generate an invitation for each of the patients of the corresponding scheduled invite. This invite contains a unique key and will create an answer object for the patient's answers in the database. The unique key will be coupled to a website url so that patient can open the survey via the email. After that it will disable all previous invites of that form in the study, so only one link is valid per form in a study at a time.

The system also contains a reminder system that will check surveys that were sent three days ago. It will check if a patient has filled in a questionnaire and will send a reminder when this is not the case.

10.4 User system

In current system we have two types of users: clinicians and admins. Clinicians can view all forms and studies that they have created, while admins have access to all the data. A clinician can create an account by registering. This can be done by clicking 'Register' on the main screen or click on 'Register as a new user' under the login tab. A user can create an account with his/her email as username, password, and a name that will be used when sending the mails to the patients.

A user has several settings he/she can change. A user is able to change their password when clicking on the tab with their own name. Every user also has editable color settings. The color settings allow a clinician to set the theme and logo of their clinic. The logo and theme will be used in the forms and emails to assure more reliability to the patients.

Part IV

Final Remarks

11 Evaluation

In this section we will evaluate multiple aspects of the conducted process. In the first section we will evaluate the requirements set at the beginning of the project. We will do this based on the different levels of importance given to the requirements. After that, we will evaluate the process.

11.1 Requirement Evaluation

In chapter 4 the requirements we made at the start of this project are listed. This chapter will give an overview of which requirements are implemented during the development and which are changed or are not implemented. Some requirements have been altered or dropped, because they required too much work for their benefit in the application or were not applicable as they were.

11.1.1 Must Have

In the current system almost all the must haves are implemented. There is one requirement that has been altered. This is requirement 1C. This requirement states that a clinician must be able to change their username, password and email. Since username and email are coupled in our system for logging in, changing either of them could cause problems with the account. In accordance with the client we determined that we do not want a user to change their username and/or email in their account. The new requirement states that a clinician must be able to change their password.

11.1.2 Should Have

The should-haves have had more alterations than the must-haves. We have implemented most of the should haves, but some have been dropped. We noticed during development that a few of the requirements were not suitable in our application.

In the user system two requirements are dropped. Requirements 1B and 1C are dropped, because patients do not have an account in our system. Since the system is handling medical data we want to use as little data about the patient as possible. Patient emails are now coupled to a study and all the data of an individual patient can not be seen in our application. The focus is on gaining the general data of a study and so accounts for patients are not necessary.

Requirement 2B in studies is also not implemented. This requirement states that a clinician should be able to set reminders for surveys. We did not implement this since it requires unnecessary work for the clinicians to set a reminder date besides the original invite date for a form. Reminders will be sent to the patients when they have not filled in a survey in time though. This will instead be done by the server three days after the original invitation was sent.

The last requirement that has been dropped is requirement 4C, which states that a clinician must be able to change the colors of the email's template. In accordance with the client we have dropped this requirement, since this would cost too much time compared to the extra value of this requirement. Clinicians are able to set a logo of their hospital though, which will be sent with the email when a clinician has set a logo.

11.1.3 Could Have

Both could haves will not be implemented. The first one requires too much work and the second can not easily be implemented, because we want the patient to fill in all the required fields before saving in order to assure better results. This requirement would try to save the form a lot when this is not possible and will therefore not be implemented.

11.2 Process Evaluation

We are quite happy with how our collaboration played out. At the start of our project we made some agreements on what we would expect from each other and we both kept our promises. The communication between us was good, this was mostly due to the fact we worked next to each other. This had the advantage that it was easy to communicate and act when challenges arose.

The planning we made at the start of our project was good enough. We had some delays during our project which was mainly due to the fact that one of the project members was ill. To compensate for this we decided in agreement with the client to skip the design report and only create some UML diagrams.

In retrospect we think the project went pretty good. If we had to point out our biggest flaw we would change if we had to do it again is the testing of the application. During the development we mostly user tested our code and at the end we lacked unit tests and such to guarantee the stability of our application. We have written several unit tests for the main components of the application, but the overall testing of the system could have been improved.

12 Recommendations

The application we delivered is, due to the tight schedule, not entirely finished. There are still improvements to be made and new features which could be added. In this section we will discuss these features.

12.1 Single Forms

At this moment in the system it is necessary to add a form to a study in order to use it. It could be a new feature when a clinician will be able to send a single form to a number of patients for a simple research. Another idea in the same context is the idea of an open form. At the moment a key is unique for a patient. With an open form the system does not require emails of the patients. This guarantees anonymity for the patients.

12.2 Built in Statistics

At this moment clinicians can only export data to Excel. In order to do some statistics on the data they require a program such as Excel, Matlab or SPSS. Some basic built-in statistics could be useful for clinicians who do not want to use or are not familiar with these programs. In the same context supporting export to other fileformats than Excel (such as CSV) could be useful.

12.3 Advanced Forms

At this moment the forms are long lists of questions. When a form contains 100 questions, it might be a good idea to add some paging to the form. This increases the usability for the patients as getting a page with 100 questions in front of you might be overwhelming.

Another advanced feature that might be considered is branching in a form. Branching in a form is the concept that if a patient has answered option A at question 7, he may for instance continue to question 10, since question 8 and 9 are not relevant to him. This could be done automatically by the system.

12.4 Graphical Elements in Forms

In a form a clinician might want to ask a question such as: Which picture resembles the scar you have the strongest? In the delivered application it is not possible to add images or other graphical elements in a form. This could be added as a new feature.

12.5 More Customisation Options

At the moment the only options of customisation are adding a custom color scheme and uploading a logo. However, it might be an idea to add more options for clinicians to customize items in the application. An idea that might be considered is a custom thanks message when a form is submitted and more customisable emails. Another idea might be to customise the number of reminders that will be sent to the patients and the interval on which these reminders will be sent.

13 Conclusion

The primary goal of the project was to build a system that clinicians can use to conduct studies. This main goal was split into subgoals for the four components our projects. The goals we set at the beginning of the project have been met in our delivered application. The application also meets the requirements set at the beginning of the project and can thus be marked as a success.

With the delivered application it is possible to create your own forms and add these forms to a study. When a study starts the mailing of surveys and reminders is done automatically. The data generated by a study can be exported to Excel.

At this moment the delivered application could be used in production. However it might be a good idea to give the product a bit more development. Using this development time, improvements to the stability of the delivered application and the user interface could be made. As the end-users of the application are non-technical and these improvements could enhance the user's experience working with the application.

In general we are content with what we have created during this project. The building of the application was under supervision of the client and can be taken into use shortly after this project. The application is a big improvement on the current way studies are conducted and can be of real value to medical world.

References

[1] Griffin et Al. “A Short Version of the International Hip Outcome Tool (iHOT-12) for Use in Routine Clinical Practice”. In: *Arthroscopy: The Journal of Arthroscopic and Related Surgery* (2012).

[2] URL: `http://office.microsoft.com/nl-nl/excel/`.

[3] URL: `http://surveyproject.org/`.

[4] URL: `http://www.surveymonkey.com/`.

[5] URL: `http://www.w3.org/TR/html5/tabular-data.html/`.

[6] URL: `http://www.w3.org/community/webed/wiki/Web_Education_community_group_style_guide`.

[7] URL: `https://www.surveymonkey.com/blog/en/blog/2013/04/23/how-to-embed-your-survey-on-a-website/`.

[8] URL: `https://github.com/dobtco/formbuilder`.

[9] URL: `https://entityframework.codeplex.com/`.

[10] URL: `http://www.asp.net/mvc/mvc5/`.

[11] Glenn E. Krassner and Stephen T. Pope. “A Cookbook for Using Model-View-Controller User Interface Paradigm in Smalltalk-80”. In: *Journal of Object-Oriented Programming* (1988).

[12] URL: `http://www.asp.net/visual-studio/overview/2013/aspnet-scaffolding-overview`.

[13] URL: `http://msdn.microsoft.com/en-us/library/ff648512.aspx`.

[14] URL: `http://www.microsoft.com/en-us/server-cloud/products/sql-server/`.

[15] URL: `http://www.asp.net/identity`.

[16] URL: `http://imageprocessor.org/`.

[17] URL: `https://github.com/Moq/moq4`.

[18] URL: `http://www.sig.eu/en/`.

[19] URL: `http://getbootstrap.com/`.

Part V
Appendix

A Initial Project Description

Below the initial project description as described by Clinical Graphics in BepSys.

> *An Online Patient Scoring System*
> In hospitals, many patient treatment protocols are experimental, with the efficacy of treatment still being unclear. To determine whether treatment is successful, it is necessary to score a patient's pain and progress on regular intervals, for example every 3 months. Generally, these scoring forms are on paper and they are labor-intensive to enter into a database system. The result is that many clinics do not use scoring forms and refrain from evaluating the efficacy of treatment. To solve this problem we would like you to build an online patient scoring system. There will be several types of interactive scoring forms that clinics can send to patient email addresses. These scoring forms should be accessible with a browser, preferably in HTML5 or comparable. The links to these forms should be sent periodically and with reminders so patients do not forget to enter their data. The data of the scoring form should be immediately entered into an online database. We will need an administrative panel and basic statistics overviews (e.g. a plot of average progress) to evaluate the scoring data.

B Sig Feedback 1

De code van het systeem scoort net 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Duplicatie en Unit Size.

Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er met name duplicatie te vinden binnen de Razor bestanden voor het tonen van data voor verschillende acties, bijvoorbeeld tussen 'Delete.cshtml' en 'Details.cshtml' en tussen 'Create.cshtml' en 'Edit.cshtml'. Het is aan te raden om dit soort duplicaten op te sporen en te verwijderen.

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 de 'Seed'-methode binnen StudyInitializer, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. In dit geval zouden de stukken voor het genereren van studie-data of de vragen naar een eigen methode kunnen gaan. Daarnaast valt op dat Razor bestanden relatief lang zijn. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Daarnaast nog een opmerking over de code-organisatie van de CSS. Op dit moment staat de code van de externe libraries (zoals Bootstrap) en de eigen code naast elkaar in 1 directory. Naar alle waarschijnlijkheid moet de externe code niet aangepast worden. Om dit duidelijk te maken is het aan te raden de code van libraries en de eigen code goed te scheiden, in dit geval bijvoorbeeld door de externe libraries in een 'lib' of 'ext' directory te plaatsen.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden of te verbeteren tijdens de rest van de ontwikkelfase. De aanwezigheid van een beetje test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code (en de test-coverage) ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

C Sig Feedback 2

In de tweede upload zien we dat de omvang van het systeem is gestegen (10%), de score voor onderhoudbaarheid is hierbij vrijwel gelijk gebleven.

Binnen de duplicatie zien wij een daling van 13% ten opzichte van de eerste aanlevering, dit is met name toe te schrijven aan veranderingen binnen de C#. Wat betreft de Razor zien we geen al te grote vooruitgang op dit gebied. Op het gebied van Unit Size zien we meer vooruitgang, de toegevoegde units zijn over het algemeen korter dan in de originele aanlevering. Als laatste zien we dat de verschillende externe resources nu verplaatst zijn naar een andere locatie.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie grotendeels zijn meegenomen in het ontwikkeltraject. Het is goed om te zien de score voor onderhoudbaarheid niet is gedaald en dat er ook nog een stijging in het volume van de test-code te zien is.

D Project Plan

Preface

This report is our plan of the project we will execute for the TI3800 Bachelor Project. The goal of this document is to create a brief overview of the project, the plan and the deliverables needed for the project. This overview will function as a basis for the rest of the project.

Introduction

This document is the first document we will write during the course of the project. We will discuss all agreements made with the company. In the following sections we first discuss the problem the project will address and elaborate on what is expected from the project. After this a planning is given for the rest of the project. Then the organisation of our project is discussed. In the final section we discuss how we will assure the quality of the code and assess the risks involved in the project.

Accordance & Adjustments

This document will be discussed with the Client at the start of the project. When, during the course of the project, adjustments in this plan are needed, these adjustments can be made in agreement with the Client.

Project description

In this section we will discuss the actual project. In the first section some background information on the client is given. Afterwards we will give a brief description of the problem the final product should solve. Finally the goals and deliverable of the project are discussed.

Client

(See Report)

Current Situation

(See Report)

Problem Description

(See Report)

Goals

(See Report)

Deliverables

At the end of the project an web-application with the features described above will be delivered to the client. Besides this application a number of reports will be written during the project. These reports will be a Research Report, Design Document and the Final Report.

In the Research Report the results of the research phase are discussed. In the research report we will motivate the choices made for the aspects of the project. In our Design Document we will give a first architecture design. In this report we will also elaborate on the database design. Our Final Report will contain all information necessary for the client to maintain the application after the project is finished. Our design choices will be motivated in this document. This report will be written during the course of our project.

Prerequisites

When the project is finished, employees from Clinical Graphics will maintain our code. To make maintenance easy the system should be written in a language that they already work with. That's why the code should be written on Python or C#.

Project plan

This section contains our planning of the following weeks of the project. The project consists of multiple phases. The first two phases are mostly research and orientation on how to build the application and will provide a clear overview of how to implement the application. In the third phase we will be implementing the code. In the last phase we will complete the application and test it properly. We will also finish the final report and present the application to the client.

Planning

The time span for this project is about 11 weeks. In this time we have to research the best options for the program, implement and test the system and deliver a working application with a final report and presentation. In order to succeed these goals we have set up the following general planning in three phases:

Research Phase

The research phase will last two weeks. During this phase we will research similar systems and which software and technologies will suit our project best. This will lead to a better development in the implementation phase.

Design Phase

The design phase will last one week and starts after the research phase. In this week we will make the software architecture of the system and a database design. All the parts of the system will be defined in this phase. We will also create a test plan during this phase.

Implementation Phase

In the implementation phase we the actual implementation of the code will take place. This phase has a time span of 6 weeks. In week 4 and 6 the code is submitted to the Software Improvement Group (SIG) for evaluation. In the week after receiving the feedback of SIG, we will improve the code to increase code quality. During this phase parts of the code should already be properly tested.

Testing and Release Phase

In the last phase we will lay the finishing touches on the program and make sure it is working properly by exhaustive testing. In this phase the final report is finished as well. A presentation will be held in the last week, in which we will present the program and work we have done to the client and the TU coach.

Overview

A more detailed overview of the planning can be found below:

Week	To do	Deliverables
Week 1 (15 - 21 sep)	-Work on project plan -Start research	-Project Plan
Week 2 (22 - 28 sep)	-Work on database design -Finish research report	-Research Report
Week 3 (29 sep - 5 oct)	-Design	-Design document
Week 4 (6 - 12 oct)	-Implementation	
Week 5 (13 - 19 oct)	-Implementation	
Week 6 (20 - 26 oct)	-Implementation	-Code to SIG for evaluation
Week 7 (27 oct - 2 nov)	-Implementation -Review feedback SIG -Testing	
Week 8 (3 - 9 nov)	-Implementation -Testing	-Code to SIG for evaluation
Week 9 (10 - 16 nov)	-Implementation -Review feedback SIG -Testing	
Week 10 (17 - 23 nov)		-Final Report
Week 11 (24 - 30 nov)		-Presentation

Project Organisation

In this section we will elaborate on the project organisation. We will discuss the personnel of the project and administrative procedures. In the last section we will discuss how we will report the progress made in the project.

Personnel

We are both working on this project full time. TI3800 is a course worth 15 ECTS per person, as 1 ECTS equals 28 hours of work we are both required to put in 15*28 = 420 hours of work in the project. Both of us have some degree of experience with web development. Not everybody in the team has experience with C#.

Administration

To keep track of the progress of our project we will use a tool called Pivotal Tracker [1]. This is done using stories, a story can be labeled and assigned to someone. A story has a number of points, if a story has more points usually it will take more time. A story can be added to an epic, after all stories in an epic have been finished the epic is completed.

Reporting

During the course of the project we will have weekly meetings with our clients. These meetings will be used to discuss the progress made and get feedback on the intermediate results. We will meet our TU Coach on a need to meet basis. We will deliver the three reports as described in 2.4.

Quality Assurance

The project will be used by clinicians and patients and will handle medical data. It is therefore of great importance that the final product is properly build. In our case employees from Clinical Graphics will maintain our application so the code should be well maintainable. In this chapter we will describe how we will maintain the quality throughout the project.

Version Control

To keep track of the changes in our code we will use Git. The major advantage Git offers is the possibility to keep a working prototype by using Git's built in branching system. Furthermore we already used Git in previous projects. Since we are both familiar with Git it won't take extra time to learn it.

Risks

In this section the risks of our project will be addressed. In order to prevent failing the project due to unexpected problems we established the risks of our project and how to prevent them.

The first risk is the questionnaire creation system. This system should be able to handle quite a few different sorts of questions. This way a clinician is not limited in their clinical study, due to the fact the system can't handle a question. Creating a system like this can be a time consuming task. To reduce the size

[1] `https://www.pivotaltracker.com/`

of this task we will use already existing libraries where possible. During our research phase we will decide which libraries we will use.

After a study finished a clinician should be able to retrieve the data. Due to the dynamic nature of the questionnaire system it can be tricky to create a good representation of the data. In the design phase we will look into the representation of the results in the database to make it easy to export.

The system will handle medical data, collecting medical data and saving this data. Handling medical data in combination with a patients info is controversial. The created system should be secure and contain as minimal information as possible about a patient. The only information we will require from the patient is his/her email so an invitation can be sent and this will be linked to the patient's identification number (ID). Clinicians are thus able to connect the information of the surveys to the patients' medical file by ID of the patient. This will prevent using names or other personal information about the patient in the system.

During the course of the project we will need to make sure the quality of the final product will be maintained. When facing a deadline, it can be attractive to just 'make the program work' and don't care about the quality. As our software will be maintained by employees of Clinical Graphics it is important we won't take such decision. By creating a realistic planning and focusing on the core parts of the system we will try to reduce the risk of these kind of decisions.

The final risk we will assess is the risk of time shortage. If one part of the system will take more time than expected it could delay our whole project and we might have to drop some requirements. To achieve this a priority list of the requirements is very important. This will give a clear overview of the parts that are more important and gives us the option to reshape the list easier in case of a time shortage. We will present this list in our research report. It is important that if a task takes longer than expected, we discover this early on so we can handle accordingly.

Code Evaluation

Throughout the course of the project we will evaluate our code in multiple ways. The first is evaluation by Code Reviews done by ourselves and external parties. The second is evaluation by unit testing.

Code Review

During our project SIG will assess the quality and maintainability of our code. Furthermore during the development process people from the client will review our code. The goal of these reviews is to improve code quality and assure a more robust application.

Tests

To keep track whether we are writing working code we will need to test our code. We will use the built in .NET unit testing software. Furthermore we are still figuring out if a continuous integration (CI) tool will be used to keep track of backwards compatibility. In our research phase we will make a decision on whether we will use a CI tool.