Context Project: Cultural Heritage

Monumentzo

Test and Implementation

Group 2

Date
28 March 2012

Group members
Bojana Dumeljic (4103246)
Mick van Gelderen (4091566)
Kevin de Quillettes (4077482)
Salim Salmi (4089715)

Advisors
Martha Larson
Christoph Kofler (Teaching Assistant)

Table of Contents

1. Introduction

This document contains the test and implementation plans for Monumentzo. For the implementation part of this document the plans will describe how, in what order the system will be implemented and when the deadlines are for the specific features.

Chapter two has an ordered list of the functional requirements according to the MoSCoW method.

Chapter three also has a bit of information on the implementation plan. Section 3.1 lists the iterations and milestones. More specific, section 3.1.1 divides the requirements over the iterations and the milestones that are set for Monumentzo and in which time period this has to be done can be found in section 3.1.2.

The rest of chapter 3, section 3.2, discusses the test plan. In the test plan we describe the three ways of testing that will used in Monumentzo. The three different ways that will be discussed are unit testing, integration testing and acceptance testing.

Finally in chapter four we have the risk analysis which will go in depth in the risks of successfully implementing the complete system. Topics discussed here will among other things be incomplete information and availability of or changes to third party APIs.

2. MoSCoW

In this chapter the functional requirements have been ordered according to the MoSCoW method (Must, Should, Could and Won't), which means that the requirements have been ordered by their importance.

2.1. Must

- **Let user register.**
  A user can make an account for himself on the system
  Requirement 1.
- **Let user log in and out.**
  A registered user can log in.
  Requirement 2.
- **Provide user with information about each monument.**
  When a user views a monument they are provided with the following information:
  - Name
  - Images
  - Date of construction
  - Category
  - Architectural attributes
  - Related articles

  Requirement 3.
- **Option to browse.**
  Browsing can be done by different identifiers in a 3D environment.
  Requirement 4.
- **Option to search.**
  Searching for monuments can be done using the following criteria:
  - Name
  - Address

  Requirement 5.
- **Gather information from external sources.**
  Using the APIs of the external sources their data will be either stored in the database or shown to the user real time.
  Requirement 6.
- **Tag enrichment by descriptions.**
  Gather the monuments architectural attributes by looking for certain words in the description obtained from the Monument Registry and add these words as information to the monument in the form of tags.
  Requirement 7.

2.2. Should

- **Option to browse.**
  The items are ordered according to the selected browse option. Multiple browsing options can be selected up to the number of axis.
  Requirement 4.
- **Gather extra information from external sources.**
  Using the APIs of the external sources their data will be either stored in the database or shown to the user real time.
  Requirement 6.
- **Provide user with extra information about each monument.**
  When a user views a monument they are provided with the following information:
  - Location on a map
  - Related books
  - Related historical events
  - Related people
  - Related videos

  Requirement 3.

2.3. Could

- **Option to save a monument to list so the user can look them up later.**
  Make three different lists for each user. Option to show the monuments in the lists on a map with differentiation between the lists.
  Requirement 7.
- **Show graphically similar monuments.**
  When a user looks at a monument that has a graphically similar monument present in the database than the other monument is shown as a recommendation to the user.
  Requirement 9.
- **Read list.**
  A user can save or remove books to and from their personal read list. These books are either from Google Books or Wikibooks. The read lists consists of links to the actual books. This way the user can read them online.
  Requirement 10.
- **Comments.**
  Provide users with the option to leave comments about each monument.
  Requirement 11.

2.4. Won’t

- Displaying monuments on a historical map of their time.

3. Implementation and tests

First 3.1 discusses the implementation plan for Monumentzo. Then in 3.1.1 the plan that designates which requirements will be implemented in each iteration is shown. 3.1.2 holds the defined milestones that have been set and the time by which they should be finished.

Next in 3.2 describes the test plan. Unit, integration and acceptance testing are discussed. Basically this sub chapter explains how each part of the system will be tested and when.

3.1. Order of implementation of features

3.1.1. Iterations

3.1.1.1. First Iteration

- **Set up Kohana.**
  First set up a github repository for the project. Then install Kohana and add submodules.
- **Set up database.**
  Set up an MySQL database according to the entity relation diagram and description in the architectural design document.
- **Register / Log in / Log out.**
  Set up account and authorization system. Make use of the Kohana authorization module and the Fancybox.js library for the login and register boxes.
  Requirement: 1 and 2.
- **Gather information from external resources.**
  Download metadata (monument id, name, address, date of construction, longtitude coordionates, latitude coordionates, categories and description), related Wikipedia articles (id, title and description) and images from the Rijksmonumenten.info API and store them in the database.
  Requirement: 3 and 6.
- **Testing.**
  - Check that Git and Kohana are set up correctly.
  - Check that the database is set up correctly.
  - Test the that a an account can be registered.
  - Test that a registered user can log in and out successfully.
  - Test that the gathered information is saved in the database correctly.

3.1.1.2. Second Iteration

- **Lire.**
  The downloaded images have to be indexed with Lire. Then similar images should be find, again using Lire. Finally for each monument that is graphically similar this relationship should be saved in the database.
  Requirement: 9
- **Tags.**
  Using the description about the monuments architectural attributes will be extracted and saved in the database.
  Requirement: 8.
- **Gather external information.**
  Here the focus will be gathering information from external resources from which the information will be displayed in real time. These resources will be:
  - Wikipedia
  - Google Books
  - Wikibooks
  - YouTube
  - Vimeo

  The API will be studied and a way to communicate with them will be implemented.
  Requirement: 3 and 6.
- **Search function.**
  Implement a search function that uses full text search to find monuments that satisfy the query containing a name or place. Also implement the search results view.
  Requirement: 5.
- **View monument (part I)**
  Make a start on implementing the view monument template by displaying the following information for a monument:
  - Metadata
  - Images
  - Map with the monuments location
  - Architectural attributes

  Requirement 3.
- **Testing.**
  - Check Lire's results.
  - Check the tags.
  - Check that the information is retrieved and stored correctly.
  - Test the search function.
  - Test the search results page.
  - Test the view monument page so far.

3.1.1.3. Third Iteration

- **Browsing function (part I)**

  First set up the 3D environment with three.js. Then load the monuments as blocks into the environment. Make sure that an action is performed when a block is clicked.
  Requirement: 4.
- **Read list.**

  Make a read list for each user. Next add the add and delete functions. Also, implement a view for the read list where the user can perform the previously mentioned functions and from where the user can go to the actual book.
  Requirement: 10.
- **Lists.**

  Implement the following list classes:
  - Favorites
  - Visited
  - Wants to visit

  The class has to contain an add and a delete functions. Also implement showing the lists on a map by using the Google Maps API. The monuments on the lists should be displayed on the map using Google Maps Layers and the monuments longitude and latitude coordinates. Each list has to be represented differently.
  Requirement: 7.
- **View monument (part II)**

  Extend the view monument page with the following attributes:
  - Add buttons for the various lists.
  - Add the button for adding books to the user's read list.
  - Show graphically similar houses.
  - Show related people.
  - Show related historical events.
  - Show related articles.
  - Show videos.

  Requirement: 3.
- **Testing.**
  - Test the browse function.
  - Test the read list.
  - Test the lists.
  - Test the view monument page so far.

3.1.1.4. Fourth Iteration

- **Browsing function (part II)**
  Add ordering the monuments by browsing function and add the tab where these functions can be selected and deselected.
  Requirement: 4.
- **Comments.**
  Implement the comment model and controller.
  Requirement: 11.
- **View monument (part III)**
  Finish the template by adding the comments part. Finally, finetune the whole template.
  Requirement: 3 and 11.
- **Testing.**
  - Test the browse function.
  - Test the browse page.
  - Test the comments.
  - Test the view monument page.

3.1.2. Milestones

Each of the deadlines for the following milestones corresponds with the end of the iteration with the same number. For example milestone 1 has to be done when the first iteration is over. Below is a list of items that has to be done for each milestone.

3.1.2.1 Milestone 1: Setting Up

Date: 4 April 2012
In this milestone we focus on setting up the basics like the Kohana framework, the database and the user system.

The following have to be finished and tested for the end of this milestone:

- Database
- Register and Login/out system
- Information gatherer

If there is time over we can start working on the following:

- Lire
- Tag enrichment

3.1.2.2 Milestone 2: Lire and Tags

Date: 4 May 2012
Here we focus on enriching our data by using Lire to find similar monuments and architectural attributes from the descriptions to enrich the data and thus be able to gather more information.

The following have to be finished and tested for the end of this milestone:

- Lire
- Tag enrichment
- Search function

If there is time over we can start working on the following:

- Browsing function

3.1.2.3 Milestone 3: Browsing

Date: 16 May 2012
In this milestone we focus on starting up the browsing function and implementing the lists.

The following have to be finished and tested for the end of this milestone:

- Read list
- Lists
    - Favorite list
    - Visited list
    - Wish list

If there is time over we can start working on the following:

- Browsing function

- Comments

3.1.2.4 Milestone 4: Complete System

Date: 1 June 2012

After this milestone all the functionalities have to be implemented. The main focus here will be finishing browsing and view monument.

The following have to be finished and tested for the end of this milestone:

- Browsing function
- Comments
- View monument

3.1.2.5 Milestone 5: Testing and Final System

Date: 15 June 2012

During this milestone we will test the system as a whole and resolve final issues.

The following have to be finished for the end of this milestone:

- Final Testing
- Final system

3.2. Test plan

3.2.1. Unit testing

To discuss the unit testing of the individual units of source code it is first needed to determine each group of code. In the web server we have 3 subsystems, namely the view, controller and model. Except for the view, which has the 3d environment in javascript, all of these use classes and methods in PHP. The Kohana framework will be used for the model, view and controller. Kohana supports PHPUnit a unit testing software framework. This can be used for the testing of the php. For testing of the javascript 3d component of the system Testswarm can be used. Testswarm is an open source tool for javascript testing.

3.2.2. Integration testing

Monumentzo uses a bottom-up integration test strategy. This strategy is firstly applied to the information gatherer. During this test we first check the integration between the information downloader and the information extraction unit. We do this by generating correct and incorrect information that needs to be passed from the information downloading subsystem to the inforation extractor and feeding this to the information downloading unit, then we check if the data outputted by information extractor is correct by mocking the information writer or that the information extractor gives a signal that incorrect data is given. We also do this for the information writer component, but for this component we check if it correctly writes data to some sort of test database, so that we can also see if the database integration is correct.

The second part under test is the web server. For the web server we're only going to test the interfaces from the controllers to the models and from the controllers to the views, because we think that only these interfaces are testable with integration testing. The interface between the views and the models is tested in the acceptance testing.

To test the interface between the controllers and the models we're going to let the controllers create models and let those models search for data and then we check if the retrieved data is correct.

For the integration tests between the controllers and the views we're going to feed the controllers with correct and incorrect requests and then we check if the controllers return the correct view for the correct request and return error codes/messages for the false requests.

For the third component of Monumentzo, the database management system, we have opted not to do integration testing because we firmly believe that the database management system we have chosen is already thoroughly tested and proven technology. We will however do acceptance testing with the databases we have made, but more on that can be found in section 3.2.3.

The testing that was described above is going to be applied after each iteration to make it easier to detect faults and error early in the process, so that it costs less time to fix the faults. To aid in the integration testing process Monumentzo uses PHPUnit.

3.2.3. Acceptance testing

Acceptance testing will happen when a major milestone has been completed.. For the acceptance tests the use cases from the analysis and design document will be used as

scenarios. For each scenario a sequence of steps is going to be defined which have to be executed to establish if the required functionality is present in the final system. If the functionality is present the test has been passed successfully. Otherwise the test fails.

4. Risk analysis

There are a couple of risks involved in the implementation of Monuzmentzo. The most important of these risks is missing or incomplete information about monuments. This results in users that get a bad user experience and may not come back to help improve the information that already is incomplete.

Another big risk for Monumentzo is the dependency on systems from third parties. It can happen that the API to these external resources change. If the changes happen during the implementation phase of the system then the total time to implement the system may increase, because the system needs to be adapted to the new APIs.

It can also happen that the APIs change during the operational phase of the application lifecycle. When this problem happens this can't interfere with the technical workings of Monumentzo, but it may hinder the users that visit the website, because they see parts of the website that are empty or Monumentzo has to have a system that can fill these gaps implemented.

The third threat that Monumentzo may encounter during the implementation phase is that some of the features may not be technically possible to implement in the given timespan. If a problem is encountered then the feature has to be simplified or if that isn't possible the feature may even has to be scrapped, but the chance of this happening is very small.