TU Delft

TI3806 Computer Science Bachelor Project (2016)

Final Report

Widget Dashboard

Fenêtre online solutions

Authors:
C.E.I. Langhout

M. Pasterkamp

Client Supervisor:
ir. R.J.F. Hendriks
Fenêtre online solutions

TU Delft Coach:
Dr. C. Lofi
Assistent Professor
Web Information Systems group

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

July 17, 2016

Preface

After 10 weeks of hard work, we are proud to present you the paper about the Widget Dashboard. In this paper, you will find out the work we have done to allow Fenêtre to view their Razor partial views in an organized matter.

We would like to thank our client Roger Hendriks for providing us with a nice and quiet working environment to work on the project and for always being available for questions. We would also like to thank our supervisor Christoph Lofi for his tremendous effort in keeping us on track with the project and for the feedback we received to improve our writing.

Finally, we would like to thank you, the reader, for showing interests in the project. If you have any questions, please feel free to contact us so that we can answer them for you.

Chris Langhout
Mark Pasterkamp
July 17, 2016

Summary

During the ten week project the implementation of a Widget Dashboard was made for the company Fenêtre. Fenêtre is a company that provides online solutions like web applications and consultancy. The dashboard displays different existing pages in the form of widgets which are freely movable and resizable by the user of the dashboard and give the user a high level of personalization.

The Dashboard is made using the Gridstack framework. This framework provides the functionality to move and resize the widgets. The choice made on using this framework took place in a three week research phase at the start of the project, where multiple different dashboard frameworks and examples were investigated. Gridstack was chosen because it gave the most freedom in implementing the dashboard around the existing features. Other frameworks gave too much of a restriction on implementation. To make sure that an unauthorized user cannot view confidential content the existing authentication of Fenêtre is used to only load widgets that meet the rights of the user.

During the development stage of the project, scrum was used to organise the project team. Using the scrum methodology made it easier to recognize the different tasks which needed to be done. However, since the product team consisted of 2 people scrum made things sometimes more tedious than it has to be.

During the development phase of the project we have learned a lot about working in an company environment, the importance of a good planning and asking questions early to get a complete overview of what needs to be done during the project.

Contents

1 Introduction **5**

2 Project description **6**
2.1 Fenêtre . . . 6
2.2 Project . . . 6
2.3 Widget Dashboard . . . 6
2.4 Requirements . . . 7
2.4.1 Move and resize widgets . . . 7
2.4.2 Add and remove widgets . . . 7
2.4.3 Saving the configuration . . . 7
2.4.4 Mobile view . . . 8
2.4.5 Authorisation . . . 8
2.4.6 Localisation . . . 8
2.5 Authorisation . . . 8
2.5.1 Authorisation at Fenêtre . . . 8
2.6 Widgets . . . 9
2.6.1 AngularJS . . . 9
2.6.2 Widget content . . . 9
2.7 Importance . . . 9
2.8 Challenges . . . 10

3 Process **11**
3.1 Scrum . . . 11
3.1.1 Meetings . . . 11
3.1.2 Prototype based design . . . 11
3.2 Global planning . . . 11
3.3 Research on frameworks . . . 12
3.3.1 Feature matrix . . . 12

4 Design **13**
4.1 Fenêtre's design . . . 13
4.2 Class diagram . . . 14
4.3 Database . . . 14
4.4 Saving the configuration . . . 14
4.4.1 Configuration scheme . . . 15
4.5 Choices . . . 15
4.5.1 Error in config . . . 15
4.5.2 Data factory for the model . . . 16
4.5.3 Settings for the user . . . 16
4.5.4 Database type . . . 16
4.6 Looks . . . 17

5 Implementation **18**
5.1 Loading the dashboard . . . 18
5.2 Saving the dashboard . . . 18
5.3 Edit mode . . . 19
5.4 Loading the Razor partial views . . . 20
5.5 Column modes . . . 20
5.6 Gridstack properties . . . 21
5.7 Inspiration from research . . . 21
5.8 Quality and testing . . . 21
5.8.1 Testing . . . 21

6 Final product **24**
6.1 One-column mode . . . 25

7 Reflection **26**
7.1 Communication . . . 26
7.1.1 Learning from the choices . . . 26
7.2 Planning . . . 27
7.3 SIG feedback 1 . . . 27
7.4 SIG feedback 2 . . . 27

8 Discussion and recommendations **28**
8.1 Recommendations . . . 28

9 Conclusion **29**
9.1 Learning points . . . 29

Appendices **31**

A Research results **31**
A.1 Malhar-angular-dashboard . . . 31
A.2 Angular-dashboard-framework . . . 32
A.3 Flatlogic Angular Dashboard . . . 32
A.4 Gridstack . . . 33
A.5 (Angular-)Gridster . . . 33
A.6 Angular drag & drop . . . 34

B Original project description **35**
B.1 translation . . . 35

C SIG feedback **35**
C.1 SIG feedback 1 . . . 35
C.2 SIG feedback 2 . . . 35

D Update Scripts **36**

E Final version backlog **37**

F Infosheet **38**

1 Introduction

Fenêtre, a company specialized in web development, wanted to create a personalised widget dashboard to create an organised overview of their different applications. This dashboard should support the usage of different rows and columns to which the widgets can snap to when drag and dropping. The widgets of the widget dashboard will display the different views from applications of Fenêtre.

The aim of this Bachelor Project is to create a widget dashboard which supports resizable and moveable widgets. The project started with a three week research phase which was used to discover the different solutions available for the implementation of the dashboard, as well as getting familiar with the different technologies used at Fenêtre and designing the architecture of the dashboard. the research phase was followed by seven weeks of development.

This paper will describe the design and development process of the widget dashboard. Firstly, the project description will be described followed by the process which went into the project. After the process, the design choices of the widget dashboard will be explained. In section 5 the implementation of the widget dashboard will be discussed. During the project, a few decision were made where, in hindsight, should have been put more thought into. This is done in the reflection. Finally, the conclusion will given.

2 Project description

This section will focus on the project description as well as the possible challenges associated with the project. Firstly, a short introduction of the client, Fenêtre, and the project is given. After that, the requirements will be listed. This will be followed by describing why this project is important. To conclude this section, a description of the potential challenges this project poses will be provided.

2.1 Fenêtre

Fenêtre is a supplier of online solutions and software. Fenêtre focuses on three topics: web & mobile, business solutions and consultancy. The web & mobile department of Fenêtre aims to provide their clients with a functional and mobile friendly websites. Fenêtre helps in the design of the websites in order to provide the best solution for their clients needs. As the name implies, the web & mobile department also provides supplementary mobile applications for their clients' business needs. The business solutions department of Fenêtre aims to provide software solutions and extended support to their client. And to conclude, the consultancy department provides consultancy for their clients (design, advice, concepts).

The back-end of Fenêtre makes use of Microsoft ASP.net [1]. In the database of Fenêtre the different applications and application modules are stored. This information is then retrieved with server-side C# using Linq [2]. To use the information from the database in the front-end, Fenêtre makes use of Razor[3]. For the front-end applications of Fenêtre, Fenêtre makes use of the AngularJS framework[4].

2.2 Project

With the amount of applications from Fenêtre increasing, Fenêtre wants a way to organize the views of the different applications. To provide this overview, Fenêtre wanted a widget dashboard to display the different views of the applications. Interests has been shown by the project group to conduct a bachelor project at Fenêtre and Fenêtre thinking that this project might be a great learning experience led to the original BEP-proposal on BepSys (see appendix B). This proposal described the design and implementation of a widget dashboard.

2.3 Widget Dashboard

Having accepted the proposal of Fenêtre, it is important to understand what a dashboard actually is. Taking a look at an example of a dashboard (see figure 1) a lot of different things can be noted. Firstly, we can see that the dashboard consists of different separated widgets. The user of the dashboard can add new widgets to this dashboard and is also able to remove widgets he does not like. The widgets themselves are like their own little application on the dashboard. In the example, when the user hovers over a widget, a move and resize icon appear on the widgets. These allow for the user to move the widgets to different parts of the screen as well as resize the widgets. While moving or resizing the widgets, it can be noted that the widgets snap to the different rows and columns on the page.

When a change happens on the dashboard (changing or resizing a widget) and page is refreshed the changes did not revert, and the dashboard loads in the same way the user left it. Saving these changes adds a lot to the personalisation of a widget dashboard.

Personalisation is a fundamental part of a dashboard. The user has to be able to configure the dashboard to his or her likings. Being able to move and resize the widgets is one part of this personalisation, but having the dashboard be in your own language would also help a great deal. Having localisation can be of great use in giving the user the feeling that he matters.

Sometimes, people want to access their dashboard while they are not behind their computer (like with their phone). The dashboard has to be accessible for those people. The example dashboard, Quarca, makes their dashboard more accessible for mobile users by lowering the amount of columns the widgets snap to up to one column. Another thing to be noted is that the ability to move and resize widgets has been disabled. This is to prevent users from accidentally moving or resizing the dashboard on touch devices.

What also can be very important, but not shown in the example, is the functionality that some users might be able to view widgets other people can not. For instance, a CEO might have a widget which provides him information about each of the employees of the company. Providing this authentication can be of great use.

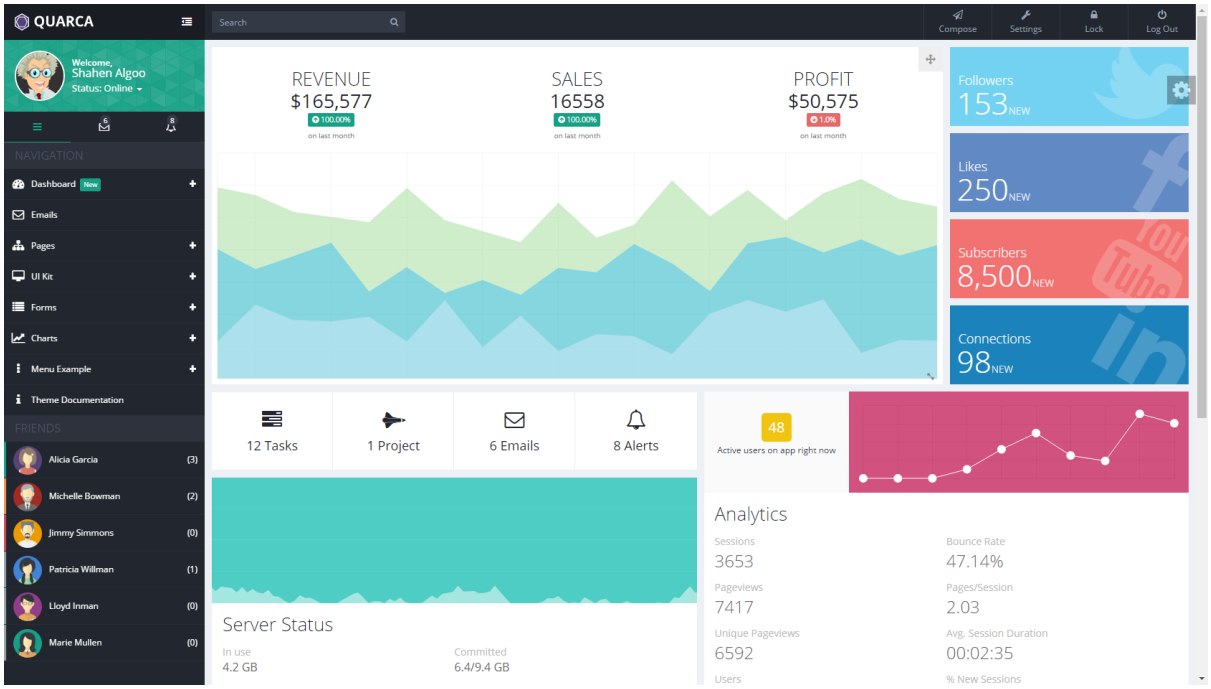


Figure 1: Dashboard example [5]

2.4 Requirements

From the previous subsection it can be seen that a lot of requirements have to be met in order to provide the best widget dashboard experience. These requirements range from moving and resizing to localisation and authentication. This subsection will go into more detail about the different requirements seen in the previous subsection in the form of user stories.

2.4.1 Move and resize widgets

- As a user, when I hover over a widget, I should see an option to move or resize the widgets.
- As a user, when I hover over a widget and move the widget to another location on the screen, the widget should snap to the closest row and column.
- As a user, when I hover over a widget and move the widget outside the grid, the widget should move back to its original position.
- As a user, when I resize a widget, then the widget will change size.
- As a user, when I hover over a widget and resize the widget, the widget should snap to the closest row and column.

2.4.2 Add and remove widgets

- As a user, I want to be able to add new widgets to my dashboard.
- As a user, I want to be able to remove widgets from my dashboard.

2.4.3 Saving the configuration

- As a user, when I save the dashboard configuration and close the dashboard, the dashboard configuration should be saved.
- As a user, when I come back to the dashboard after it was saved previously, my dashboard should look the same as when I saved the configuration.

2.4.4 Mobile view

- As a user, when my screen size is small and I open my dashboard, I will see a limited view of the dashboard in which I can't move widgets.
- As a user, when my screen size is small and I open my dashboard, I will see a limited view of the dashboard in which I can't resize the widgets.
- As a user, when my screen size is small and I open my dashboard, I will see a limited view of my dashboard. This limited dashboard should then be separate from the multi-column mode.

2.4.5 Authorisation

- As a user, when I want to add a widget to my dashboard, only the widgets to which I have rights should be able to be added to my dashboard.
- As a user, when a widget is present on my dashboard to which I have no rights (like when the user roles are changed) the widget will not be loaded but instead an error widget should load.

2.4.6 Localisation

- As a user, when I open my dashboard, I expect that the contents of the widgets are in a language I understand

2.5 Authorisation

Authorisation is an important aspect of the dashboard. It assures that people who have no right to view a specific widget do not get to see those widgets. Fenêtre already has en existing framework to provide authorisation to their applications. This means that we have to integrate the widgets from the dashboard into the existing authorisation system that will be explained in the next paragraph.

2.5.1 Authorisation at Fenêtre

Fenêtre already has a system for authorisation in place for their widgets.Their database design illustrated in figure 2 gives an overview of the tables that play a role in the authorization process. Users can have different roles which are mapped to their respective rights. Some application modules (the widgets) are mapped to these rights. In other words, For a user to view a specific widget, the user should have a role corresponding to the right of that widget.

The important thing now is to know which user is actually logged in right now. Luckily, the system of Fenêtre keeps track of which user is logged in and stores the user id of the active user. This means that getting the authorised widgets can be done quite easily with an SQL-query to the database.

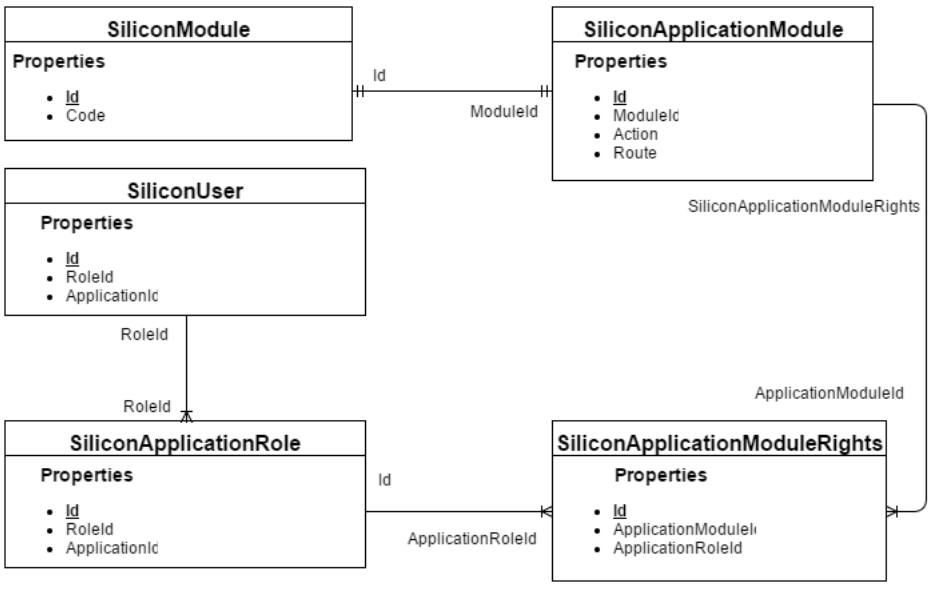


Figure 2: Database diagram

2.6 Widgets

The widgets themselves are a fundamental part of the widget dashboard. As mentioned in the requirements, they should be able to be moved, resized, added and deleted. To provide the most flexible widgets, we chose the implement the widgets as HTML-containers. This allows for Fenêtre to put anything in the widget containers as long as it is valid HTML.

2.6.1 AngularJS

One of the requirements from Fenêtre was to implement the widgets using the AngularJS framework. Fenêtre wants this since this is the most advantageous for them, namely because Fenêtre uses AngularJS themselves for their front-end applications. By creating the widgets in AngularJS, Fenêtre would have a much easier time maintaining and integrating the widgets.

2.6.2 Widget content

The widgets on the dashboard are implemented as HTML-containers. However, this leaves the question as of what needs to go into these containers. Fenêtre wants to be able to load the different Razor partial views from their applications into the widget containers. The problem arises in how the widgets can get the HTML-content since Razor partial views have to be parsed on the server first. To load the Razor partial views into the HTML-containers, the Razor partial views needs to be parsed on the server first and then the content should be retrieved and put into the widget containter.

2.7 Importance

As discussed earlier, this dashboard is important for Fenêtre as it will provide an organised and personalised overview of the applications from Fenêtre. Widgets are becoming more and more popular nowadays. Providing flexible and extendable widgets is key in creating the best widgets available. We provide this by making the widgets accept HTML as their content. This essentially makes them mini web pages inside the page. The advantage of this approach is that Fenêtre can make the widgets as complex as they wish, while still having the ability of re-using them whenever they please.

For Fenêtre, the dashboard can be of great impact as it provides a lot of efficiency for the users. Fenêtre is an ever moving company. Providing more efficiency to Fenêtre might help them become more efficient in their business practices as well.

2.8 Challenges

One of the most important challenges is how we are going to integrate the dashboard into the back-end of Fenêtre. This means that the project has to be up to the standards of Fenêtre. The front-end has to be written using the AngularJS framework, the code has to be well documented, readable and written according to AngularJS standards. The widgets also have to support authentication and Localisation for improved security and more personalisation.

A big challenge will be combining the server side Razor partial views with the client-side AngularJS directives. It would be ideal for Fenêtre if the razor partial views can be dynamically loaded in AngularJS directives because that would provide the most versatility. AngularJS has a function called "compile". The compile function compiles an HTML string or DOM into a template. This template can then be used as a template for a directive. This is what allows us to dynamically load the Razor partial views into the directives.

Authentication is another important aspect of assuring the security of the widgets and a big challenge. It was previously shown how authentication on the widget level can be performed in the database of Fenêtre. The implementation of the authentication can still prove to be quite a challenge which requires knowledge of the ASP.net database and using Linq to query to this database.

Localisation will help the increased feeling of personalisation. Luckily, Fenêtre already has an existing localisation system for their Razor partial views. This system can thus be easily used to load the localised widget content onto the dashboard.

Making the dashboard with rows and column support and making the widget containers snap on those rows and columns can be a difficult task and a time consuming process. This is why research was done to find a suitable framework which provides these functionalities. This research will be elaborated in subsection 3.3.

Additional challenges include getting familiar with the workflow of Fenêtre, learning how to use AngularJS and the different functionalities it provides, learning the setup of ASP.net and how to use C# linq to communicate with the ASP.net database.

3 Process

In this section the process of the development is explained. This includes the explanation of the weekly development cycle, the description of the contact with the people associated with the project and the description of the research on what framework to use.

3.1 Scrum

A modified version of the scrum methodology is used to bring structure to the process of this project. The reason why a modified version of scrum instead of just normal scrum is because the development team exists of just two people who work together a lot. This creates a weird situation when distributing the roles belonging to Scrum. With one person being the Scrum Master and one being the Product Owner the process would be too separated. Instead the team used these roles as guidelines for the members, where the final responsibility lays at one person, but the tasks are mainly executed together. The sprint plans and reflection are stripped down to a just list of tasks for a week and an estimation of time needed for each task. The product backlog is constructed with the initial requirements given by the client. During the project the backlog will be adjusted based on the progress and inputs from the project team and associated people. The final version of the backlog can be found in appendix E.

3.1.1 Meetings

During the weekdays the development team works together every day. Every week on Thursdays a sprint review meeting is held with the client at the office of Fenêtre. In this meeting the current process is discussed and reflected by explaining what is done in the past week and showing the current status of the product. This meetings ensures that the execution of the project follows the expectations of the client. If the implementation drifts away from the vision of the client, this meeting will ensure that the project will be put back on the right track. Furthermore the global planning and the planning for the next week are discussed and adapted if necessary.

To make sure the project is in line with the guidelines of the TU Delft, the project team meets about once every two weeks with the TU Delft Coach. In this meeting the progress of the project is clarified. This is also an opportunity to receive feedback on the process of the project and on the state of reports.

On Tuesday and Thursday the project team will work at the office of Fenêtre located in The Hague. This provides the opportunity to ask questions directly to the employees of Fenêtre, which provides a quicker and more detailed answer to questions than when asked via chats in Skype for Business or via e-mail. The direct contact with the other employees also helps to improve code quality since the employees are used to the coding standards of Fenêtre and can spot deviations of these standards easily. The usability of the final product will also be better for the employees because they will already be introduced to the implementation during the development of the project.

3.1.2 Prototype based design

To be able to give the client a quick representative view of the status of the project each week, the product is build using prototype based design. This means that a minimal version of the product is constructed as soon as possible and with each iteration of scrum new features will be added to this prototype.

3.2 Global planning

The original planning of the project was to spend the first three weeks on research. The remaining seven weeks are planned for development. Things listed in the planning will be explained in the later sections of the report. In the first week of development the plan is to set up the project to work in, after this week the created files should represent the designed class diagram. The next week the first prototype should work. This means that widgets need to have basic functionality. The third week of development is planned to be used for the implementation of the design of the widgets and saving the configuration of the dashboard. The connection to the database to save this configuration is planned for the fourth week of development. After this week four weeks are left. In these weeks the model of the dashboard will be finished, the edit mode of the dashboard will be implemented the remaining must-have features need to be implemented and the code should be checked if it is of good quality.

3.3 Research on frameworks

In order to provide the best basis of the prototype, the research phase contained a detailed research for the best framework to use. Together with the client a list of requirements for a suitable framework is formed and for each framework a score is given to each requirement. The individual tables of the research on the frameworks can be found in appendix A.

3.3.1 Feature matrix

In order to make the decision on what framework to use the research tables are combined in the matrix 1. With the results in this matrix, three candidates were chosen to be the framework that provides the main dashboard functionality. The three candidates; "Malhar", "Angular-dashboard-framework" and "Gridstack" are discussed with the client. The final decision on what main framework to use can be found in section 5.

	Malhar	Angular-dashboard-framework	Flatlogic angular dashboard	Gridstack	Gridster	Angular Drag & Drop
Must have						
Drag & drop	+	+	-	+	+	+
Resize	-	+/-	-	+	+	-
Dynamic content in widget	+	+	-	+	+/-	-
Widget configuration	-	+	-	+	+	+/-
Add / remove widget	+	+	-	+	+/-	+
Bootstrap support	+	+	+	+	+/-	-
Angular 1.x compatibility (1.4 or higher)	1.4	1.4.8	1.3.8	-	1.2.0	-
Active	+/-	+/-	-	+	-	-
Comprehensive	+	+/-	-	+	+/-	+
Should have						
(Bootstrap) column snapping	-	+	-	+	+/-	-
Scrollbar when content doesn't fit	-	-	-	+/-	-	-
Animations	+/-	+	-	+/-	+	+/-
Minimise/Maximise widgets	-	-	-	+/-	+/-	-
Undo	-	+	-	+/-	+/-	-

Table 1: Research matrix

4 Design

This section focuses on design of the final product and the choices that are made with regards to the design. Firstly, the existing design of the back-end of Fenêtre is explained with the focus on the important parts of the product. Secondly, the design of the dashboard will be explained with reference to a class diagram followed by the design choices made to serialise the dashboard. After this, the considerations which lead to the design choices will be elaborated. Concluding, the look of the dashboard will be discussed.

4.1 Fenêtre's design

As seen in figure 2 the main applications of Fenêtre are stored in the SiliconApplicationModule table. In this table, the different applications per module are saved. As an example, Fenêtre has a module called "Order". This order can have a lot of different actions like "Edit", "Add" "Create" and "List". The "Edit" action will thus be stored in the SiliconApplicationModule and the "Order" module will be stored in the SiliconModule table.

To accomplish authorisation, the SiliconUser table has a one to many relationship with the SiliconApplicationRole table. This relationship represents the fact that users can fulfill different roles. These roles each have different rights to application modules. This is represented in the one to many relationship between the SiliconApplicationRole table and the SiliconApplicationModuleRights table. Finally, to represent the relationship that some application modules can only be access if a user has a certain right. This relationship is represented in the one to many relationship between the SiliconApplicationModule and the SiliconApplicationModuleRights table.

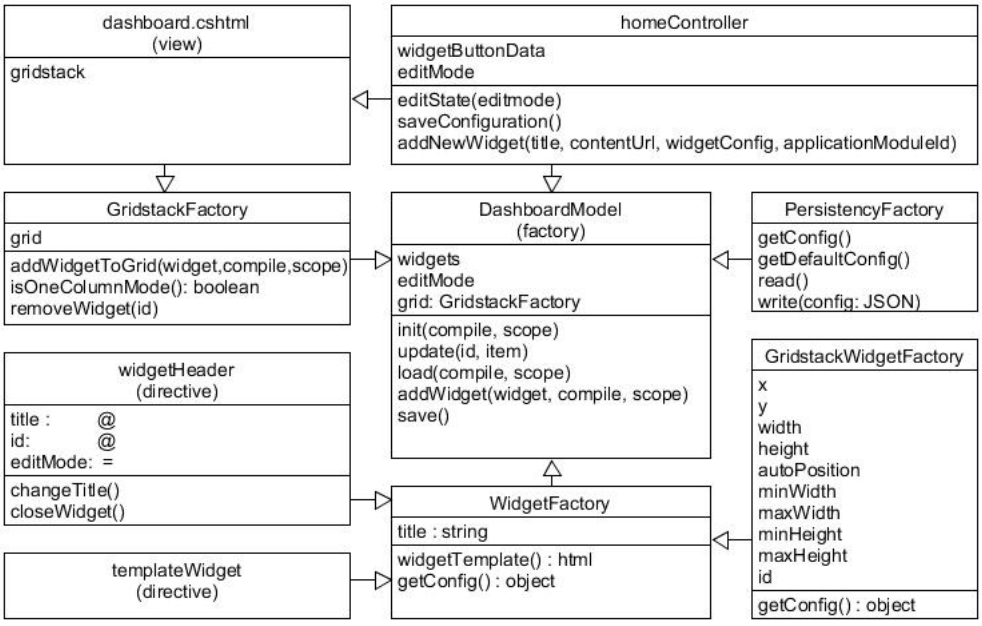


Figure 3: Class diagram

4.2 Class diagram

Before the start of the implementation of the project, a scheme was made that describes the classes. This scheme shows the classes that are needed for the widget dashboard. During the project this scheme was updated regarding changes in the implementation. The image in figure 3 shows the final version of the internal structure.

The early version of the dashboard was made using the Gridstack framework as primary model for the Dashboard. However, during a meeting with the client it was discovered that using Gridstack as a model was not very readable and comprehensive, which are important traits of source code. This led to the creation if the DashboardModel class. The DashboardModel represents the data model of the dashboard. It takes care of having the widgets in memory, changing the column mode (when called from the controller), loading and saving the dashboard and adding widgets to the dashboard.

Gridstack provides a lot of options to add widgets to the dashboard. This might have obscured the code in the WidgetFactory. To prevent this, the class GridstackWidgetFactory was created. GridstackWidgetFactory stored the values associated to Gridstack (like the x and y properties).

the WidgetFactory is extending from the GridstackFactory. The WidgetFactory creates the data type that represents the widgets with the help of the prototype from GridstackWidgetFactory. The WidgetFactory helps the DashboardModel with the serialisation by implementing a serialisation function on the widgets themselves.

For the communication with the database, the PersistencyFactory was created. This factory creates an object which takes care of the communication with the DashboardModel and the database. This abstracts storing and loading the dashboard from the DashboardModel.

The widget views are created using the widgetHeader directive and templateWidget directive in AngularJS. The widgetHeader stores the title of the widget. The widgetHeader also stores the ability to delete the widget by clicking on the "close" icon. The templateWidget shows the different views from the database as widgets.

The interaction between the user and the dashboard is done via the HomeController. The HomeController takes care of the edit mode, changing the column mode, deleting widgets and adding widgets. The HomeController functions as the main controller for the dashboard.

4.3 Database

The database of Fenêtre serves two main purposes for the widget dashboard. The first is saving the definitions of existing widgets. For each user a selection of these definitions is selected and shown to the user when trying to add a new widget granting that this user has rights to see this widgets. The second purpose of the database is saving the configuration of each user. Initially the idea was to create a new table in the database with two columns, the userId and the configuration data in JSON format. When the implementation of this connection to the database started, this choice was reconsidered in a discussion with Kai from Fenêtre. He pointed out that a table exists that holds user data per applicationModule. Adding the configuration data to this table means that no extra table needs to be added to the database and makes backwards compatibility easier to manage. The final update scripts for the database can be found in appendix D.

4.4 Saving the configuration

The user wants to see the same dashboard every time he logs in. Whenever a user has made a change he can press on the "Save Dashboard" button to send the configuration to the server. The next time this user loads the dashboard this configuration is used to load the widgets.

4.4.1 Configuration scheme

Saving the configuration of the dashboard with the different positions and sizes of the widget comes with the design of the configuration to save. To keep the process of saving the configuration in the database simple, the client advised to save the configuration of the dashboard in JSON format and save this in one column in the database. The initial design of the configuration of the database consisted of an array of widget objects. The widget object was defined as in the following example:

```
1  {
2    "directive": "color-widget",
3    "title": "widget",
4    "settings": {                        // settings for gridstack
5      "x": 0,
6      "y": 0, ...
7    },
8    "properties": {                      // properties specific for directive
9      "background-color": "blue"
10   }
11 }
```

Listing 1: Initial configuration design

However, as development continued, we found that this configuration of the dashboard is not sufficient. For instance, for mobile users we had to save a one column mode in the configuration. Since the widgets are loaded via the ASP.net views, the properties tag in the configuration became redundant. Another redudant aspect of the configuration was the autoposition. We found out that the autoposition enabled resulted in some unintended behaviour (dragging a widget outside of the grid should result in a reset to its original position, not being positioned at the end of the grid). What we did add to the configuration was the ApplicationModuleId. This Id will be used to map the dashboard widget to the list of widgets the user has rights to.

```
1  {
2      "MultiCol" : [
3          {
4              "title": "widget",                    // unchanged
5              "settings": { ... },                  // unchanged
6              "applicationModuleId" : "6"
7           },
8           { ... }
9      ],
10     "OneCol" : [ ... ]
11 }
```

Listing 2: Final configuration format

4.5 Choices

During the project a lot of choices needed to be made. For the more important discussion points the actual decision was made along with the client. For smaller decisions the project team discussed with each other what solution would fit the best in the project with the cleanest code solution.

4.5.1 Error in config

It is very unlikely that the JSON that holds the configuration of the dashboard contains an error. To ensure that when this configuration string gets corrupted the user can still use the dashboard, the choice needs to be made what to show to the user. Is the faulty configuration so important that it cannot be overwritten or is it okay to just overwrite the incorrect configuration with the default configuration or show a widget with an error message. Another option is to just return an empty dashboard where the user can re-add widgets to his liking.

4.5.2 Data factory for the model

The PersistencyFactory on the client side used to just return the data that it got from calling the database functions. The decision was made to change this read() method to the getConfig() method, which ensures the dashboard model that a valid dashboard configuration is returned that can be used to fill the dashboard with the correct widgets. To keep the functionality separated the getConfig() function calls the Read() function that still handles the calls to the database. The getConfig() function then checks if the read function actually returned a dashboard configuration and if not it calls the getDefaultConfig() function that gets the default widget configuration from a file.

4.5.3 Settings for the user

Initially the idea was that the user can edit a lot of settings per widget. Of course the placement and the size of the widget are configurable by the user, but apart from that the user can not configure a lot for a widget. The reason for this is that there are really not that much settings that the user needs control of.

Gridstack offers a bunch of settings for a widget. Included are minimum and maximum values for both width and height and an autoPosition. In the widget dashboard the user is not allowed to change these values. This decision is made along with the client because the designer of the widget has to have full control over the width and height settings because these could interfere with the actual content inside the widget. If a widget contains for example a table with more than 3 columns, the minimum width of the widget can be set to a minimum of 3 gridstack columns to ensure that the columns always fit the width of the widget.

The value of autoPosition is not configurable in the widget dashboard. Instead the value of autoPosition is initialized false when the widget is made and no option is given to change this. This decision is made by the development team because if the value of autoPosition is true, unexpected behaviour occurs when dragging and dropping widgets. If a widget is dragged outside of the widget dashboard and then dropped, this widget will auto position to the bottom of the grid. If the value of autoPosition is false, the widget will simply return to it's original place instead of moving to a place where the user does not expect it to move to.

4.5.4 Database type

One meeting occurred where both the Client and the TU Delft Coach were present. In this meeting a discussion arose about using the database of Fenêtre to save the configuration of the dashboard. The TU Delft Coach suggested that if the configuration of the dashboard is the only thing that needs to be saved, it may be better to save this in a separate database. And if it is JSON that is saved, a Document Database would fit, especially the use of a lightweight database without the need of a server is a nice thing to look into. Together with the Client research was done if this would be a viable option for the project. It turned out that using the database of Fenêtre is the better option. First of all this keeps things simple for Fenêtre since using an extra database brings more separation in data storage. Second not only the configuration of the dashboard but also the configuration of individual widgets needs to be saved and to be able to connect a widget to the rights of a user, it is far simpler to use the existing database since the security is already implemented. The final reason why the existing database is used is because the lack of knowledge about the structure of the database in the project team can be caught by the explanation of the system from employees of Fenêtre. And as pointed out in section 4.3 the existing tables of the database can be used.

4.6 Looks

During the development of the project not a lot of time is spend on the design and looks of the dashboard. During meetings with the client sometimes little things about the design stood out and those were fixed by the project team.

Fenêtre arranged that an employee dedicated to designs would take a look at the design of the dashboard. He asked to send screen-shots of the dashboard that he could use to modify the design in photoshop. Unfortunately this feedback was not ready in time to be used for the final product. This means that the design of the dashboard is not yet in its final form.

The initial general design of the dashboard is made by the team. This design is simple, but describes the core dashboard consisting of a grid with widgets placed on this grid. The widgets have a header which shows the title and button(s) to interact with the widget. At the top of the dashboard an extra header is added below the standard headers of Fenêtre where the buttons are placed that have functionality connected to the edit mode. When on the dashboard, this bar will show one button that switches the dashboard to edit mode. The client did not agree with this design because the header takes up too much space on the default page. Instead the button to take you to edit mode needs to be in the existing header and the buttons that show up when edit mode is enabled can appear on the extra header. This extra header contains three buttons, one for adding a new widget, one for saving the dashboard configuration and one for leaving the edit mode. Screenshots of the dashboard can be found in section 6.

When designing the header of the widget, originally 3 buttons were taken into account that show up during edit mode. Closing, opening the settings menu and minimising/maximising were features where buttons should exist for. To make the user able to change the title of the widget, the title of the widget disappears from the widget header and the space is instead filled with an input box where the user can change the title.

In the final product the widget header in edit mode has the input box to change the title and only one button that, when clicked, deletes the widget from the dashboard. The reason that only one button remains is because the settings button turned redundant since a widget does not have settings to configure that are not otherwise changeable in the dashboard. The minimize function is not implemented so this button is also left out.

5 Implementation

The description of the implementation of the product is given in this section. The encountered problems are listed along with the way these problems are handled and how this reflects on the final implementation. The delivered product forms a module that exists inside of an application of Fenêtre. The implementation started with making the Gridstack library work inside our project. After that, following the prototype based design, development of the widget dashboard started. The following subsections describe the most interesting part of the implementation.

5.1 Loading the dashboard

Upon loading the dashboard, the HomeController calls the init function of the DashboardModel. The aim of the init function is to retrieve the user dashboard configuration from the database with the help of the PersistencyFactory. In order to provide the user with the smoothest user experience, retrieving the configuration from the server happens asynchronously using AngularJS promises. (see listing 3). The read() sends a request to the server to retrieve the dashboard configuration. The read() method then returns this promise to be resolved later. Because it is important that the PersistencyFactory always returns a valid configuration when a connection to the server has been established, the getConfig method was created. This method allows for returning a default configuration when the user does not yet have a valid dashboard configuration. When the user, however, has a valid dashboard configuration, the promise from the read() method is chained back to the DashboardModel. If no connection was established, the error from the read() method is chained back to the DashboardModel to be resolved there.

```
1  {
2      // Always returns a config or an empty object
3      factory.getConfig = function() {
4          return factory.read().then(function(response) {
5              if (response.ActionResult == null || response.ActionResult == "") {
6                  return factory.getDefaultConfig().then(function(defaultResponse) {
7                      return defaultResponse;
8                  });
9              }
10             return JSON.parse(response.ActionResult);
11         });
12     }
```

Listing 3: Get dashboard configuration

5.2 Saving the dashboard

Beign able to save the dashboard to a json format is an important aspect of being able to retrieve your old configuration when you relog on the dashboard.

To save the dashboard, firstly the widgets on the dashboard are sorted on x and y positions. This is important because the widgets are not fixed on y position. Adding widgets in the wrong order means that some widgets might move upwards on the dashboard because the widgets which used to be there is not present yet. This issue is avoided if you start adding the widgets from top left to buttom right. After that, for each widget stored in the DashboardModel, their getConfiguration() method is called and concatenated. The resulting JSON is then passed to the Persistencyfactory which sends a request to the server to save the configuration in the database.

5.3 Edit mode

One of the features that caused quite a lot of problems is the edit mode of the dashboard. When the user opens the dashboard, the widgets are not customisable. The widgets should stay this way, even through changing column modes, until the "Enter edit mode" button is pressed.

When the user presses the "Edit Dashboard" the editState function in the HomeController is called. This method tells the DashboardModel that the edit mode should be enabled/disabled. This will enable/disable the widgets to be moved and/or resized. After that, the widgets on the dashboard are notified that they should be in the correct editmode.

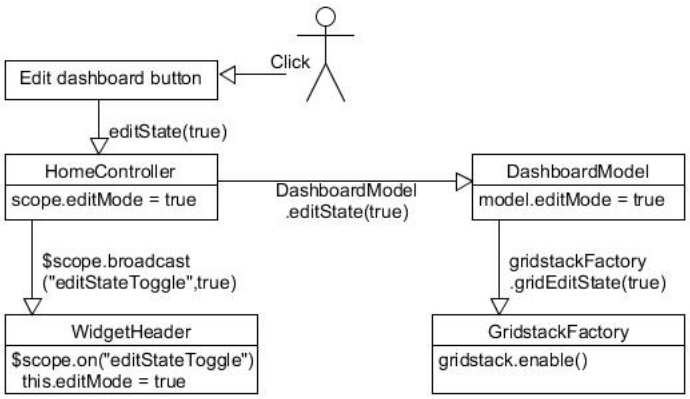


Figure 4: Edit mode call hierarchy

When the user presses the "Enter edit mode" button, the user can drag widgets around and resize widgets to his liking. Widgets are now also deletable and new widgets can now be added to the dashboard. The DashboardModel controls the current state of the dashboard, this value is sent to the HomeController scope to make it visible for the dashboard view and the widgetHeaders. The buttons in the view send an update to the model and the controller. That triggers a change in the widgetHeaders and the gridstackFactory. Which makes the gridstackFactory enable or disable the resize and move behaviour of widgets.

The widget headers need to know when the edit mode is enabled because when the dashboard is in edit mode, the user needs to be able to change the title of a widget and delete a widget from the dashboard. This is done by creating an event handler on the "editStateToggle" broadcast from the HomeController. A change in edit mode triggers the visibility of an input field in the header. When this input field is shown, the user can edit the title that is displayed in this field. When the title changes, the header sends an update to the controller and the controller makes sure the configuration in the model is updated. Enabling edit mode also makes a cross button appear in every widget header. When this button is pressed the widget is deleted from the dashboard and when the dashboard is saved, it will also be removed from the configuration.

5.4 Loading the Razor partial views

One of the challenges of this project was to load the Razor partial views in AngularJS directives. The challenges comes from the fact that the Razor partial views have to be parsed server side and therefor can not be dynamically added to the template of the widgets. After some research, we found that AngularJS has a function called compile. The compile function resolves the url in the templateUrl and puts the content into the template of the directive.

To dynamically add the content to the widget, the widgets have an attribute called contentUrl in the HTML-tag. If this attribute is not used, the default Fenêtre widget will be used instead as the contentUrl (see listing 4). Together with the compile function described before, this will allow us to load the Razor partial views dynamically.

```
{
    app.directive("templateWidget",
    function() {
      return {
        restrict: "E",
        templateUrl: function(elem, attrs) {
          return attrs.contentUrl || "/FenetreWidget/WidgetView";
          }
        }
        });
}
```

Listing 4: Template widget

5.5 Column modes

In order to still provide a good view on smaller screens, Gridstack implemented a one-column mode in their framework. In the implementation of Gridstack all widgets will get an x-coordinate of 0 and a width of 12. Gridstack version 0.2.5 (the latest version as of 17-06-2016) introduced a bug that when the maximum width of a widget is set less than 12, the widget will not take the full width of the screen when in one-column mode. An issue is filed on the github page of Gridstack. In the project this bug is worked around by always applying a maxWidth of 12 on widgets that are placed in the one-column configuration.

In the HomeController an event handler on the screen size is added. At the end of every resize, the changeColumnMode is called in the DashboardModel. The changeColumnMode function in the DashboardModel checks if gridstack has indeed changed column mode. If Gridstack has changed its column mode, the current dashboard is saved in memory and the new dashboard will be prepared. The changeColumnMode returns a boolean whether it has changed its column mode or not. This is used as an indicator for the HomeController to know whether it should call the loadDashboard function of the in the DashboardModel.

5.6 Gridstack properties

As seen in the configuration of the dashboard described in section 4.4, the settings of a widget are saved. The available settings for gridstack are: [6]

x, y widget positions

width, height widget dimensions

autoPosition changes placement behaviour of widget

min-/ maxWidth minimum and maximum width

min-/ maxHeight minimum and maximum height

id value for data-gs-id

These settings are all of the available options gridstack provides for their gridstack-items. These settings are mostly used as provided with two exceptions. In section 5.5 a bug is mentioned that occurs in the one-column mode. The maxWidth is therefore forced at 12 for every widget in one-column mode. The other exception is the value of autoPosition. In order to implement desired behaviour for the placement of widgets autoPosition is manually set to true when adding a widget to the dashboard. This is done because then gridstack automatically checks for the first position the added widget will fit. After adding the widget the value is changed to false. This is done to improve the behaviour of moving widgets. When the value of autoPosition is true and a widget is accidentally dropped outside of the valid range, the grid will automatically places the widget at the first spot where it will fit, which is usually in the bottom right corner of the grid, independently of where the widget originated from. The decision was made to set the value of autoPosition to false because with this value on false, whenever a widget is dropped outside of the grid, it just returns to the position it was before dragging the widget. This behaviour is preferred because the placement of the widget is more controlled by the user.

5.7 Inspiration from research

During the research a lot of knowledge is acquired about the implementation of dashboards. For the functionalities of the widget header and the implementation of the edit mode Angular-Dashboard-Framework [7] was a great inspiration. In the demo projects of Angular-Dashboard-Framework when the edit mode is activated, the widget headers show a row of buttons that provide functionalities for the widget. The idea for the functionality of the edit mode (the implementation is different) and the placement of the close button came from this framework.

5.8 Quality and testing

During the development of the product the development team constantly manually tested the product. When the dashboard was turned on and some extra time was available, the functionality of the dashboard was tested and any unexpected behaviour that occurred was noted. To make the testing of the product more structured, user stories are defined to make sure no functionalities are missed in the manual testing. The user stories also make the tests executable by others in the future to check if the system still meets the requirements.

5.8.1 Testing

For testing, we chose to create clear and well defined stories based on the expectations of the functionalities of the dashboard. For instance, when the user enters the edit mode of the dashboard, and switches to the one-column mode of the dashboard, the dashboard should still be in edit mode (see table 2)

The advantage of testing this way was that potential test cases did not have to be rewritten when something in the dashboard changes (which happened often). This would also mean that some tasks will be manually tested more than needed.

Unit testing is still something we would have loved to do. A combination between manually testing complicated things and unit testing mundane things would have been ideal. However, unit testing kept on being pushed back to a later sprint due to other things with higher priority (see section 7: Reflection)

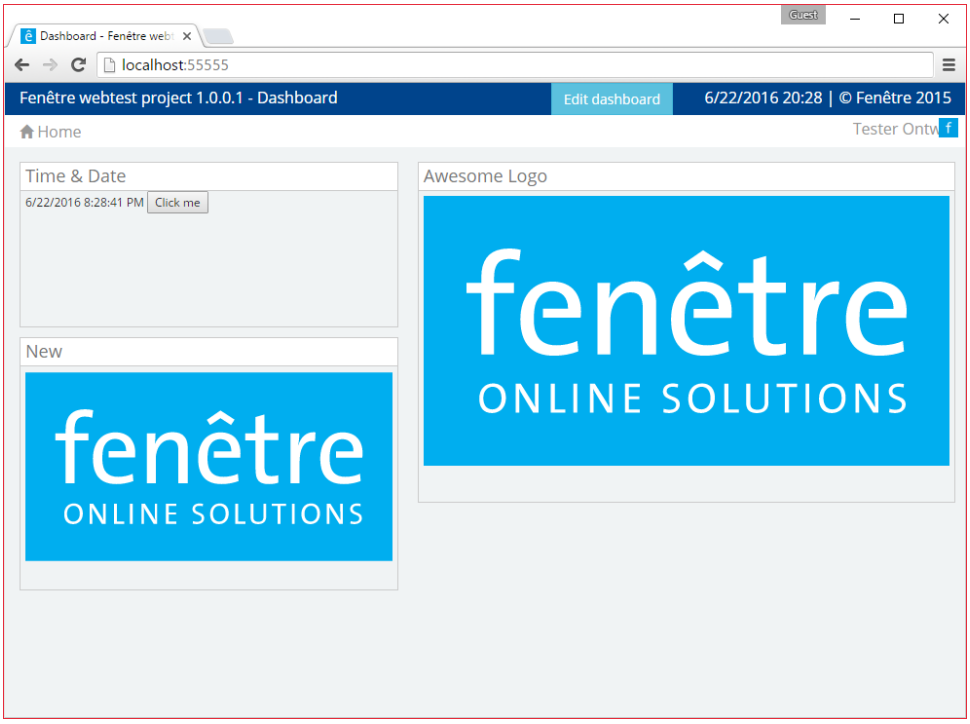
Table 2: User story tests

Category	User story	Status	Steps to reproduce
Open /Home/ Dashboard	As a user, when I open my dashboard, I can view my widgets.	As intended	Open the /Home/Dashboard page.
	As a user, when I open my dashboard and I don't have a previous dashboard configuration, I will see the default widget.	As intended	Open the /Home/Dashboard page with an empty dashboard configuration in the database.
	As a user, when I open my dashboard which previously had a widget from which I don't have access anymore, I will see the error widget.	As intended	Open the /Home/Dashboard page with a dashboard configuration which contains an Id of a widget from whcih you don't have access.
Edit Dashboard	As a user, when I press the "Edit Dashboard" button, the three buttons "Add a widget", "Save configuration" and "Exit edit mode".	As intended	Open the /Home/Dashboard page and then press the "Edit dashboard".
	As a user, when I press the "Edit dashboard" button, the title of the widgets should become editable and the close icon should appear next to the title.	As intended	Open the /Home/Dashboard page and press the "Edit dashboard".
	As a user, when I press the "Edit dashboard" button and then press the "Exit edit mode" button, the widget titles should not be editable anymore and the close icon should disappear.	As intended	Open the /Home/Dashboard page and press the "Edit dashboard" button followed by the "Exit edit mode" button.
	As a user, when I press the "Edit dashboard" button and I change the title of a widget and exit the edit mode, then the title of the widget is changed	As intended	Open the /Home/Dashboard page, press the "Edit dashboard" button, change the title of a widget and press the "Exit edit mode" button.
	As a user, when I press the "Edit dashboard" button and I click on the close icon on a widget, the widget will be removed from my dashboard	As intended	Open the /Home/Dashboard page, press the "Edit dashboard" button and press the close icon next to a widget title.
	As a user, when I press the "Edit dashboard" button and I drag a widget to another location, it should snap to the nearest column and row. When I then presses the "Exit edit mode" the widget should be at the position where the user left it.	As intended	Open the /Home/Dashboard page, press the "Edit dashboard" button, drag a widget to another position and then press the "Exit edit mode" button.
	As a user, when I press the "Edit dashboard" button and I resize a widget, then the widget will be resized and when I press the "Exit edit mode" button the widget should still be resized.	As intended	Open the /Home/Dashboard page, press the "Edit dashboard" button, resize a widget and press the "Exit edit mode" button
Save configuration	As a user, when I press the "Edit dashboard" button, and then press the "Save configuration" button, the dashboard should be saved and when I re-open the dashboard the dashboard saved previously should show up again.	As intended	Open the /Home/Dashboard page, press the "Edit dashboard" button, press the "Save configuration" and refresh the page.

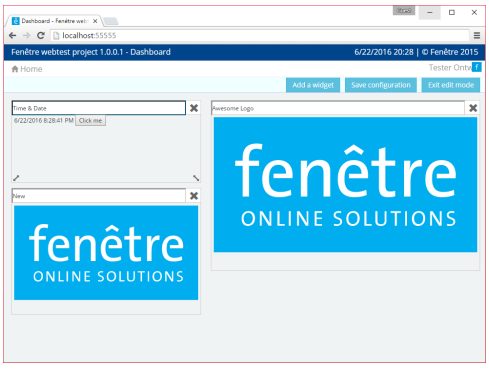
One-column mode	As a user, when my screen size is small and I open my dashboard, I will see a limited view of the dashboard in which I can add widgets	As intended	Open the /Home/Dashboard in a small screen (default: 768 pixels), press the "Edit dashboard" button, press the "Add widget" and add a widget
	As a user, when my screen size is small and I open my dashboard, I will see a limited view of the dashboard in which I can remove widgets	As intended	Open the /Home/Dashboard in a small screen (default: 768 pixels), press the "Edit dashboard" button and then press the close widget icon next to the widget
	As a user, when my screen size is small and I open my dashboard, I will see a limited view of the dashboard in which I can't move widgets	As intended	Open the /Home/Dashboard in a small screen (default: 768 pixels), press the "Edit dashboard" button and then try to drag the widget to another location
	As a user, when my screen size is small and I open my dashboard, I will see a limited view of the dashboard in which I can't resize the widgets	As intended	Open the /Home/Dashboard in a small screen (default: 768 pixels), press the "Edit dashboard" button and then try to resize the widgets.
	As a user, when my screen size is small and I open my dashboard, I will see a limited view of my dashboard. This limited dashboard should then be seperate from the multi-column mode.	As intended	Open the /Home/Dashboard in a small screen (default: 768 pixels), press the "Edit dashboard" button and then press the "Add widget" and add a widget and save the configuration. If you then open the dashboard on a larger screen the dashboard should still be the same as the last time before you left the multi column mode of the dashboard.

6 Final product

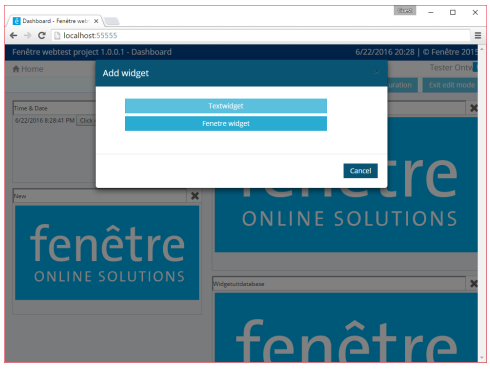
After ten weeks of working the final result of the project is a widget dashboard that meets the requirements as described in section 2. In figure 5a a screenshot can be seen of the dashboard in normal view mode. Figure 5b shows the dashboard in edit mode. In edit mode the titles are replaced with input fields and the widgets can be deleted. Furthermore at the top of the screen a new bar shows up with buttons to add widgets, save the dashboard configuration and exit the edit mode. When the button is pressed to add widgets the screen in figure 5c is shown to the user. In this screen a list is shown to the user for all the widgets that this user can add to the dashboard. If the user no longer has rights for a widget, this widget will not show up in this list. If the widget was previously added to the dashboard and the rights were pulled back, the user is shown an error widget with the message that he no longer has rights and that he should contact the company or delete the widget. If the user keeps this widget on the dashboard and the rights are re-enabled for this user, the error widget will be removed automatically and the correct widget is shown again as how the user left it.



(a) Dashboard view



(b) Edit mode

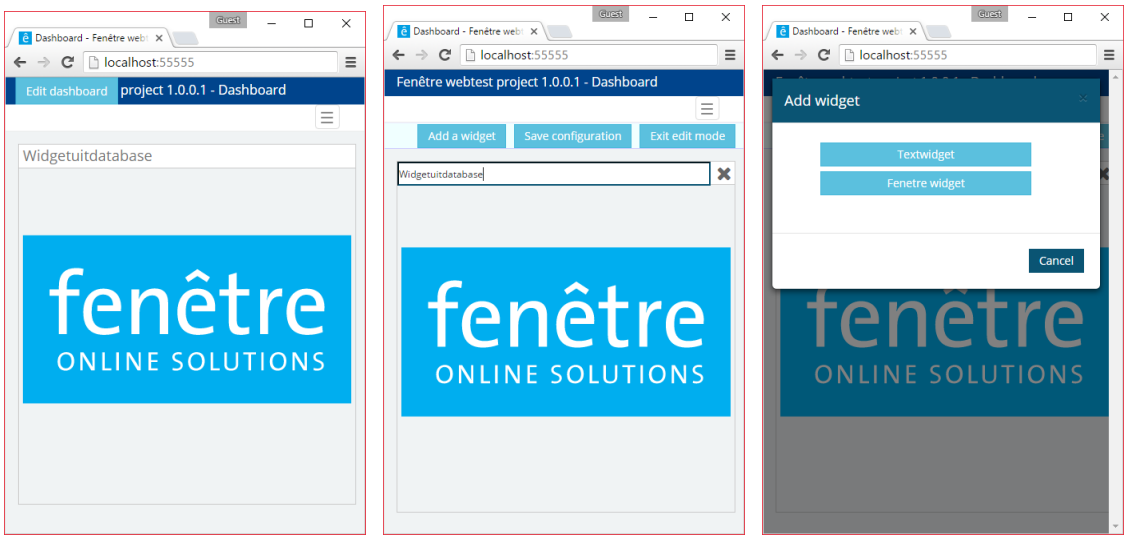


(c) Add widgets

Figure 5: Screenshots of final product on desktop

6.1 One-column mode

In the one-column mode, the user has a different configuration of the dashboard. In this mode, all the widgets have the same width and height (figure 6b). Outside from looks, the one-column mode also has a limited functionality than the multi-column mode. When in one-column mode, it is possible to add and delete widgets. Unfortunately, the option to move and resize the widgets is disabled. This is to prevent users from accidentally changing their dashboard.



(a) Small screen view (b) One column edit mode (c) One column add widgets

Figure 6: Screenshots of final product on desktop

7 Reflection

We enjoyed our time working on the project at Fenêtre. We gained a lot of experience and learned a lot on what and what not to do. We had a great workplace at the office. Most of the time it was a quiet spot with an ambiance that helped to keep us focussed.

7.1 Communication

The main learning experience of the project has to do with the communication with the client, and in our case more general, the company. At Fenêtre there is no time to stand still. Projects are always running and the people are busy. It takes some time to adapt to this day to day process of work. We had a hard time figuring out when the right time was to ask questions. Towards the end of the project we learned that if we have a question we just have to ask it and not wait for a "good" moment to ask a question to not disrupt someone. Since such a moment does not often occur and that time can always be made to answer our questions we learned to just go to someone and if they really are busy they will come to you later.

The importance of the definition of the final product at the start of the project is another thing we learned. We had a general idea of what was expected from us but along the development of the product some things turned out to differ from our expectation. Luckily the weekly meetings with the client caught this differences in view of the product, but some things could have been detected earlier with better communication from our part. An example of this is the realisation that the used way of defining widgets did not support the use of partial views of Razor. We did not know that our widgets have to support Razor partial views so we also could not have taken this into account when designing the widgets we used to. After the detection of this, the directives for widgets were replaced with just one generic directive supporting a templateUrl generated from the database, linking to a widget that is defined in the MVC structure of ASP.net, which made the dashboard much more usable for Fenêtre.

A different problem occurred with the implementation of the properties per widget. At the start of the project it seemed we needed to be able to have settings per widget. This can be seen in the old version of the dashboard configuration shown in listing 2. In our early examples this is where the colors of the widgets were saved and used as background color. However, as we learned more about the project and what we were supposed to do, it became clear that this properties part of the widget were unnecessary since the widgets were created from Razor views. This ties back to the issue of not being prepared well enough before we started the project.

In general, with better communication with Fenêtre time could have been saved and more of the non essential requirements could have been implemented. When help was asked for the code, we got some comments on why certain variables had a name other than standard at Fenêtre or the implementation differed from what Fenêtre is used to. These inconsistencies were then caught and rewritten but this took some time to correct which was otherwise spent on other development. Another part were some time was lost is with the database diagram. When the diagram of the database is viewed in the SQL Manager tool a table shows up called "SiliconUserRoles", an n-to-m table with two columns connecting users to their roles. In the connection of the code to the database this table is internally processed and cannot be used as a table to query on. This was not known by us and it took some time to figure out why the query did not do what it was supposed to do. After help from Fenêtre luckily this problem was solved but if we asked this question earlier, time could have been saved.

7.1.1 Learning from the choices

The choices made during the project form a great learning experience, for example the choice of using the Gridstack framework. We knew when choosing Gridstack as the framework to use for the behaviour of the widgets that this would bring challenges in the implementation but working with it brought a valuable new experience. We also believe that by using Gridstack the final result is formed more to the idea we had about a widget dashboard and we did not feel limited by the choice on framework.

7.2 Planning

We also learned a lot about planning during the project and how important it is to take notes during development and to document choices. During the project we kept track of the time spent on what tasks in a log. This log turned out to be really useful at the end of the project when recalling what tasks were executed during the project. The notes we have taken helped, but more could have been written down and we noticed that we need to work on making our notes future proof since sometimes it took a wile to understand what we meant in the notes.

Guessing how much time is needed for a task is another task we gained more experience with. At the start of the project when we were very unfamiliar with the working environment and we needed to calculate extra time for the learning process. Towards the end of the project our estimated time per task became better and better since we gained more knowledge about the system and could already make an impression on exactly where things need to change.

The implementation for the connection with the database is a thing we delayed towards the end of the project. This mainly came because we did not know really well what to do. We learned that delaying this task was not the smartest move of the project. We got a good explanation about the database from Fenêtre but making sure that the code we wrote fits in the code of Fenêtre remained a hard task and took quite some time.

7.3 SIG feedback 1

For the first SIG code delivery deadline the client suggested to make a demo project. This required some rewriting since the created demo project does not have a connection to the database. The razor code was translated to javascript and the methods that received info from the database were rewritten to just contain a JSON object.

The received feedback from the SIG group was mainly positive C. The major thing that kept the final score from being higher was the score on Unit Size. This lower score is mainly caused by the integration of JSON code in the demo project that was sent to SIG. In the final product this parts were replaced by function calls that received the needed information from the database specifically for the current user. We did not put very much effort in writing test code but instead went for an approach where the product was tested manually.

7.4 SIG feedback 2

The second feedback we received from SIG is not very extensive. The recommendations from last time were taken in consideration but it is a pity that no test code is added. Furthermore it is notable to SIG that all javascript files are wrapped inside Razor templates. This is common practice of Fenêtre because it allows the angular components to be added to a variable application.

8 Discussion and recommendations

In this section the process of the development team is evaluated. The quality of the code in the final implementation and the requirements of the client also receive an evaluation.

- We could have started earlier with testing. The priority was not on unit testing since the client did not forced us to have unit tests and manual testing will be sufficient. Towards the end of the project the realisation came that unit tests would have been a nice addition to ensure quality of the product but priority was more on making sure all requirements are met, so testing was done manually in order to check if modifications worked as intended and other features did not break.
- The naming in the database of the columns we had to add was not optimal. The naming has to follow a specific pattern which was not very clear since we have little experience and got different inputs for the naming. In the update scripts in appendix D the final names of the added columns can be seen.

Another thing was was sometimes working against us was the usage of SVN. Fenêtre has a test project available on their SVN server for testing new features in order to avoid breaking the main branch. This test branch from Fenêtre was also the branch that our project was made on, as well as the project of another bachelor project group. This caused a few accidents in which some things were committed that made the build of the other group fail. Luckily after contact this errors could be fixed quickly. We understand that the workflow of Fenêtre includes SVN so switching to other systems is not an option but the ability to use our own branch would have helped preventing the problem from becoming an issue and would have been our preferred way.

8.1 Recommendations

Since the project took place on a span of just 10 weeks, not everything could be implemented. During the project, the project team got a good view of the project and some tasks could not be executed during the project because of the available time but will add value to the product if implemented.

- During one of the explanation sessions about the database of Fenêtre, the localized tables were mentioned. In this project, the priority to look into the localized tables was not high enough to be executed, but using the localized tables for things as the default names for widgets that can be added will improve the product.
- The client suggested that if enough time was available, adding widgets could be changed from pressing a button to dragging new widgets from a side panel to the main dashboard. If it is the case that only one instance of a widget can exist, this functionality of adding widgets can be implemented. Gridstack is able to track multiple grids and dragging widgets along these multiple dashboards is also possible. This functionality is shown in this example provided by gridstack: `http://troolee.github.io/gridstack.js/demo/two.html`. If multiple instances of the same widget can exist in the dashboard, this feature will take some more time. The positions of individual widgets need to be tracked and if a widget leaves the column of available widgets to add, a new instance should appear in that grid and the added widget has to stay in the position the user placed it to.
- In the same example an extra grid exists with a trashcan image. This can also be used in the widget Dashboard for the feature to remove widgets instead of the current implementation with a close button in the header of a widget. If the drag to add functionality is implemented, the list of available widgets needs a reload whenever a widget is deleted because this widget may swap from being unavailable to available after deletion.
- Adding only widgets that are not yet present on dashboard. Not enough time to implement because the difference in present widgets in oneCol and multiCol mode.
- The functionality of the one column mode is not developed. The order of the widgets can only be changed by deleting and re-adding widgets as of now and the widgets can not be resized.
- Additional touch screen support can be added to support the gridstack functions on tablets.
- Some of the database manager classes need to be edited so that they use a custom view model.

9 Conclusion

For the final project of the Bachelor Computer Science on TU Delft the project of making a usable Widget Dashboard for the company Fenêtre has been executed. This project took place on a span of 10 weeks starting with a three week research phase. In the research phase different frameworks have been inspected and compared and the Gridstack framework has been chosen as used framework. Looking into the different frameworks also gave a lot of inspiration for features that have been implemented in the final widget dashboard.

The implementation took place in the remaining seven weeks of the project. At the start of the implementation phase the focus laid mainly on the client-side implementation. This includes the implementation of the looks and configurability of the widgets and the edit mode and column modes of the dashboard. In the last two to to three weeks the connection between the dashboard and database was implemented, allowing the saving of the configuration and the ability to get the list of widgets the user has rights for.

This project has been a great learning experience. We have gotten more experience in how a company works and operates as well as things which could be improved upon (see 7). In the future, more time will be spent on planning ahead and figuring out what is asked from us from the client as well as keeping the documents about the project up to date.

9.1 Learning points

The most convenient learning experiences we occurred in the project are:

- Experiencing process of working in a company and taking into account other people that need to work with the things you make.
- The importance of documenting and keeping notes during the project.
- The importance of a good planning as a solid foundation of a project.

References

[1] Microsoft, "Asp.net mvc," retrieved on May 10, 2016. [Online]. Available: http://www.asp.net/mvc

[2] ——, "Linq." [Online]. Available: https://msdn.microsoft.com/en-us/library/bb308959.aspx

[3] W3Schools, "Asp.net razor - markup," retrieved on May 3, 2016. [Online]. Available: http://www.w3schools.com/aspnet/razor_intro.asp

[4] Google, "Angularjs." [Online]. Available: https://angularjs.org/

[5] Bootpen, "Quarca," retrieved on June 22, 2016. [Online]. Available: http://www.bootpen.com/themes/quarca/

[6] P. Reznikov, "gridstack.js api addwidget." [Online]. Available: https://github.com/troolee/gridstack.js/blob/master/doc/README.md#addwidgetel-x-y-width-height-autoposition-minwidth-maxwidth-minheight-maxheight-id

[7] S. Sdorra, "Angular-dashboard-framework." [Online]. Available: https://github.com/angular-dashboard-framework/angular-dashboard-framework

Appendices

A Research results

A.1 Malhar-angular-dashboard

Link: `https://github.com/DataTorrent/malhar-angular-dashboard`

Must have		
Drag and drop/snapping	+	It supports basic drag and drop, it feels a bit clunky. There is no grid, therefore the placement of the widgets feels a bit random. Another problem is that, sometimes when there is enough space for the widget to fit, it just snaps back to far from this place and it is not possible to get the widget to position on this empty space.
Resize	+/-	Resize is possible but it does not work really well. Specifying a minimum and maximum width would probably make this better.
Dynamic content inside the widgets	+	In the advanced example especially the line graph works nice with mouseOver. The content itself, however, is not manipulable by the user.
Widget configuration	+/-	You can edit the title of the widget, the size. The demo indicates that you can add an edit template for each widget, but no documentation of such feature can be found in the documentation. Only onClose and onDismiss were found in the documentation.
Add / remove widget	+/-	Adding and removing widgets is available. In the demo adding widgets is done by clicking buttons in the header, which is not the nicest solution. For removal each widget has a close icon. There is no confirmation on deleting and the buttons are always active.
Bootstrap support	+	
Angular 1.x compatibility (1.4 or higher)	+	Version 1.4
Active	+/-	The project started Jan 5, 2014 and the latest commit is done a month ago.
Comprehensive	+	The readme provides an explanation of the basic features.
Should have		
Bootstrap column snapping	-	Not available
Scrollbar when content doesn't fit	-	Not available
Animations	+/-	Widgets follow cursor when dragging and keep their internal animations going.
Minimise/Maximise widgets	-	Not available
Undo button	-	Not available

A.2 Angular-dashboard-framework

Link: `https://github.com/angular-dashboard-framework/angular-dashboard-framework`

Must have		
Drag and drop/snapping	+	You need to press the edit button in order to move the widgets.
Resize	+/-	The widgets resize based on the columns. You can not give them a custom size.
Dynamic content inside the widgets	+	The hyperlink and markdown widget example showed that adding functionality for allowing dynamic content was possible
Widget configuration	+	You can change the location of the widgets and you can change the columns on the page. The widgets themselves, however, cannot be directly configured.
Add / remove widget	+	In the preview, adding and removing widgets can be done via an edit button in the header. This button, however, can be difficult to find at first.
Bootstrap support	+	Native bootstrap support
Angular 1.x compatibility (1.4 or higher)	+	Version 1.4.8
Active	+/-	The last action by the creator was done a month ago, in the last week 5 issues are submitted but none of them is answered.
Comprehensive	+/-	Small documentation, we still have to look at the code and the examples in order to understand the framework.
Should have		
Bootstrap column snapping	+	Native support for bootstrap column snapping
Scrollbar when content doesn't fit	-	Not available, widget increases in size. Some images can even go out of the widget
Animations	+	Clear and functional animations which help with figuring out what happens to the dashboard when you move something for example.
Minimise/Maximise widgets	-	Not available
Undo button	+	Available

A.3 Flatlogic Angular Dashboard

Link: `https://github.com/flatlogic/angular-dashboard-seed`

Must have		
Drag and drop/snapping	-	Not available
Resize	-	Not available
Dynamic content inside the widgets	-	Not available
Widget configuration	-	Not available
Add / remove widget	-	Not available
Bootstrap support	+	Native support
Angular 1.x compatibility (1.4 or higher)	+/-	Angular 1.3.8
Active	-	Last commit happened on 17-03-2015
Comprehensive	-	No documentation available and source code not documented
Should have		
Bootstrap column snapping	-	Widgets are not moveable
Scrollbar when content doesn't fit	-	Widgets scale with the information
Animations	-	Not available
Minimise/Maximise widgets	-	Not available
Undo button	-	Not available

A.4 Gridstack

Link: `http://troolee.github.io/gridstack.js/`

Must have		
Drag and drop/snapping	+	Supports drag and drop
Resize	+	The widgets are resizable
Dynamic content inside the widgets	+	Not in the demo, can be implemented
Widget configuration	+	Can be implemented
Add / remove widget	+	Demo allowed to add and remove widgets
Bootstrap support	+	Bootstrap support can be implemented
Angular 1.x compatibility (1.4 or higher)	+/-	Angular js was not used, but it can be combined with angular, see this *angular-gridstack* project (requires angular 1.3 or higher)
Active	+	recent activity
Comprehensive	+	Usage guide and documentation, source code not commented
Should have		
Bootstrap column snapping	+/-	Can be implemented
Scrollbar when content doesn't fit	+/-	Can be implemented
Animations	+/-	Can be implemented
Minimise/Maximise widgets	+/-	Can be implemented
Undo button	+/-	Not implemented by default, has to be implemented.

A.5 (Angular-)Gridster

Link: `https://github.com/ManifestWebDesign/angular-gridster`

Must have		
Drag and drop/snapping	+	Supports drag and drop, even has a a floating function
Resize	+	Supports the resizing of widgets
Dynamic content inside the widgets	+/-	Not seen in the demo, can be implemented
Widget configuration	+	Title and size can be adjusted by the user
Add / remove widget	+	You can add and remove widgets easily
Bootstrap support	+/-	Need to be implemented
Angular 1.x compatibility (1.4 or higher)	-	Angular 1.2.0
Active	-	Lots of unanswered issues, last commit happened a year ago
Comprehensive	+/-	No documentation, just a how to use guide, source code is commented
Should have		
Bootstrap column snapping	+/-	Has to be implemented
Scrollbar when content doesn't fit	-	Content goes out of the widget, need to be implemented ourselves
Animations	+	Nice and clear animations
Minimise/Maximise widgets	+/-	Has to be implemented
Undo button	+/-	Has to be implemented

A.6 Angular drag & drop

Link: `https://jasonturim.wordpress.com/2013/09/01/angularjs-drag-and-drop/`

Must have		
Drag and drop/snapping	+	Uses a basic grid to fit content
Resize	-	Not supported, implementing looks like a challenge
Dynamic content inside the widgets	-	It's just a basic grid
Widget configuration	+/-	You can only relocate the widgets, has to be implemented ourselves
Add / remove widget	+	You can add and remove widgets
Bootstrap support	-	Has to be implemented ourselves, framework is not easily adaptable to bootstrap
Angular 1.x compatibility (1.4 or higher)	-	Angular.js 1.2
Active	-	Hasn't shown activity in the last year
Comprehensive	+	The code itself does not have a lot of comments but the documentation gives a detailed explanation about anything that is happening. This is a small project which makes the readability great. Because this project does not depend on external frameworks it is very easy to follow the steps of the developer.
Should have		
Bootstrap column snapping	-	Bootstrap is not supported
Scrollbar when content doesn't fit	-	Not supported
Animations	+/-	Really basic animations
Minimise/Maximise widgets	-	Not supported
Undo button	-	Not supported

B Original project description

Ontwerpen en realiseren van een Widget dashboard waarbij widgets door de gebruiker aan te passen zijn via drag and drop. De indeling van de widgets wordt opgeslagen bij de betreffende gebruiker zodat deze de volgende keer zijn eigen voorkeur heeft.

Belangrijk onderdeel van dit project is een onderzoek naar de juiste frameworks/libraries voor de realisatie. Op basis van de gewenste functionaliteit en andere voorwaarden zoals soort techniek, aanwezigheid broncode, leesbaarheid, & onderhoudbaarheid en aantal gebruikers van framework kan er dan een solide keuze gemaakt worden uit de framework alternatieven. Er kan ook uitkomen dat het handiger is libraries of delen van verschillende frameworks te combineren.

B.1 translation

Designing and realization of a Widget dashboard where widgets can be adjusted by the user via drag and drop. The layout of the widgets is saved for each individual user which allows this user to choose his personal preference.

An important part of this project is the research of using the right frameworks/libraries for the realization. Based on the desirable functionality and other conditions like used technique, presence of source code, readability, maintainability and the amount of users of the framework a solid choice can be made from the different existing frameworks. Another result can be to use libraries or using a combination of different frameworks. (source: https://bepsys.herokuapp.com/projects/view/189)

C SIG feedback

C.1 SIG feedback 1

[Analyse]

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

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.

In jullie project zit het probleem in widgetButtonDataCreate() in HomeController.js. Jullie hebben daar configuratie in JavaScript-code staan. Ib het algemeen proberen mensen een zo hard mogelijke scheiding tussen data en logica te plaatsen. Het is daarom beter om dit soort constructies naar aparte bestanden (bijvoorbeeld JSON) te verplaatsen en die vervolgens in te laden. Het bestand DbCommunicationFactory.js heeft een soortgelijk probleem, maar daar geven we minder prioriteit aan omdat jullie dat bestand duidelijk in elkaar hebebn gehackt voor de upload naar ons.

Tot slot is het positief om te zien dat jullie ook testcode hebben geschreven, maar de verhouding tussen productiecode en testcode kan nog wel wat beter. Hopelijk lukt het nog om daar in het vervolg van het project nog aan te werken.

C.2 SIG feedback 2

[Hermeting]

In de tweede upload zien we dat de omvang van het systeem is gestegen. Het is opvallend alle JavaScript-code nu binnen Razor templates wordt geschreven.

Het is goed om te zien dat jullie de aanbevelingen van de vorige evaluatie hebben gevolgd. Wat betreft de testcode zien we helaas dat er sinds de eerste upload geen nieuwe testcode is toegevoegd.

Uit deze observaties kunnen we concluderen dat deel van de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.

D Update Scripts

```
--------------- SQL ---------------

ALTER TABLE [dbo].[SiliconApplicationModules]
ADD [Type] tinyint DEFAULT 1 NOT NULL
GO


ALTER TABLE [dbo].[SiliconUserApplicationModuleSettings]
ADD [Configuration] nvarchar(max) NULL
GO


--------------- SQL Make new column in siliconApplicationModules and rename to correct name "Setting" ---------------

ALTER TABLE [dbo].[SiliconApplicationModules]
ADD [WidgetDefinition] nvarchar(max) NULL
GO

EXEC sp_rename '[dbo].[SiliconApplicationModules].[WidgetDefinition]', 'Setting', 'COLUMN'
GO
```

E Final version backlog

MoSCoW	Requirement	Solution	implemented
Must have	Drag & drop	Gridstack	done
	Snapping	Gridstack	done
	Resize	Gridstack	done
	Dynamic content		
	Widget user configuration		done
	Add Widget	Bootstrap Modal	done
	Remove Widget	Gridstack	done
	Bootstrap support	Gridstack	default
	Angular support	Angular-Gridstack	done
	Active development of framework	Gridstack	-
	Comprehensive framework	Gridstack	-
	Persistent configuration	JSON	done
	Save 3 configurations	JSON	changed to 2 configurations
	Code is comprehensive for Fenetre employees		Yes
	Edit mode		done
Should have	Bootstrap column snap	Gridstack	native
	Scrollbar	CSS	yes
	Minimise/ maximise		no
Could have	Implement query widget		no
	Load widget dashboard via ng-repeat		not needed
	float widget mode		no
	unchangable widgets		no
	add widget via sidepanel and drag them to the grid		no
	add support for touch screens		no
	Why gridstach resize does not work on one column mode		no
	Undo button	Angular	no

F Infosheet

TI3806 Computer Science Bachelor Project (2016)

Infosheet

Widget Dashboard

Fenêtre online solutions

Description

The company Fenêtre is a supplier of online solutions and software. To provide users of one of their online applications with a clear view of screens that are available to them, a widget dashboard module is made. In this dashboard the user is free to move and resize widgets. The content of the widgets comes from screens that are available to the user that are adapted to fit inside.

In order to make the implementation of this dashboard as smooth as possible suitable frameworks were researched and the Gridstack framework was chosen for the implementation of most of the functionality of the dashboard. With the research done on all sorts of dashboard frameworks and examples, the idea for the implementation took a better shape.

In ten weeks the project was realised where the team worked two days of the week at the office of Fenêtre in The Hague. The rest of the week the team worked either at home, communicating via Skype or at the TU Delft at a project working space.

The final product is a working Dashboard which can be filled with widgets that show any existing or new Views from the ASP.NET server side MVC framework. Which allows Fenêtre to easily convert their existing pages to widgets. When the dashboard is put in edit mode by the user, the user is able to move and resize the widgets. After saving the configuration and leaving the edit mode, the widgets are stationary and the content inside the widgets can be accessed by the user. The current version of the dashboard is well usable but some functionality for, for example touchscreens, can still be added.

Authors:

C.E.I. Langhout — chris.langhout@gmail.com
Contribution: Focus on client side implementation

M. Pasterkamp — markpasterkamp@hotmail.com
Contribution: Focus on server side implementation

The team worked together most of the time on the implementation, the report and the final presentation.

Client Supervisor:
ir. R.J.F. Hendriks
Fenêtre online solutions

TU Delft Coach:
Dr. C. Lofi
Assistent Professor
Web Information Systems group

TU Delft BEP Coordinator
[To be added]

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

Presentation date: June 29, 2016