Modularization of learning tools

Final Report

by

R. G. Machielse
D. Vliegenthart

in partial fulfilment of the requirements for the degree of

Bachelor of Science
in Computer Science

at Delft University of Technology

TU Delft coach: Dr. A.E. Zaidman
Client coach: Ir. S.B. Kok
Bachelor Project Coordinator: Dr.ir. F.F.J. Hermans
Our client is a company that focuses on the improvement of education. They do this by providing tools for students and teachers, with the ultimate goal to improve the way they learn. One of the challenges is the discovery of these new ways to learn, which should be possible in an agile manner by building and testing separate functionalities for the new platform.

We are assigned the task to develop a modular structure and to build an example module alongside it as a proof of concept. The goal is to enable developers at our client company to develop new modules rapidly and without interfering with other parts of the new platform.

This prevention of interference, or the isolation, of different parts of the overall platform is done by building a modular system. Each module built using this modular system can be integrated in the core platform, all while running isolated on a separate server, functioning like a plugin so to say. These modules are also self runnable, meaning they can function even without being mounted into the core platform.
Preface

This is the final report describing our bachelor project "Modularization of learning tools" submitted in partial fulfillment of the requirement for the Bachelor degree in computer Science at the Delft University of Technology. This project ran from April 2015 to June 2015 and has been carried out by Richard Machielse and Daniël Vliegenthart.

We would like to thank our TU Delft Coach and supervisor, Dr. A.E. Zaidman, for not only providing us with the guidance we needed, but also for giving us the flexibility to carry out this project in a way we felt would lead to the best results.

R. G. Machielse  
D. Vliegenthart  
Delft, Zuid-Holland, The Netherlands, June 19th 2015
Our client is a growing business, which leads to less flexibility and more problems in development. Also, there is a need to be able to easily test out functionality on a small group of customers, without breaking other parts of the product. Therefore our client would like to modularize their application.

As the client wants to develop a new modular platform next to their current platforms, the choice of frameworks and techniques was still open. We split up the backend and frontend, because they wanted to have a static frontend communicating with an API server. We used Ruby on Rails for the backend and Ember.js as a frontend framework. The modular structure has been implemented by building plugins for these frameworks, being a Rails Gem and an Ember Addon. Also, we built a command line tool to generate new modules to make it easier to work with the new structure.

The resulting plugins allow for the generation of both frontend and backend modules, being able to communicate with each other. They can be executed both standalone, and integrated into the new platform.

During this process we wanted to keep high software standards in mind. We continuously tested the product and measured code coverage, quality and other analytics. We have used the Scrum methodology for our agile workflow.

During the development it became clear that the core application of the client wouldn’t be ready in time. Therefore we adapted our plugins to be completely standalone.
Introduction

The influence of the internet has changed every aspect of life as we know it: from the way we interact to the way we learn. FeedbackFruits is a young company of students that wants to improve the ability to learn using this technology.

Founded in 2012, a lot of experience has already been gathered. From the beginning, when the company intended to be a platform to let students give feedback to teachers, the strategy has emerged to a full platform that enables teachers to structure their courses and helps students to learn using interactive tools.

Being a startup, they often change ideas quickly. New functionality gets added and the number of users increases continuously. However, while the amount of code and developers grows, the platform becomes less maintainable and bugs appear more often. At the same time, due to the larger amount of customers, requests and ideas keep coming in.

To enable the team to keep on developing at a high pace, the need for a different architecture increased. The solution to this problem: modularity. Inspired by success stories from the industry, our client would like to invest in a new modular structure that theoretically allows anybody to add modules to the platform. Testing them out on a small audience, without having to go through the often time intensive and problematic merging of features will allow them to keep that flexibility without losing stability or quality.

Our job is to investigate the options and implement the best approach in our opinion. The result will be a proof of concept, that allows them to create and run isolated modules, while keeping security, efficiency and adoptability in mind.

This document is structured as follows; chapter 2 will give a detailed description of the problem to solve, the domain, the deliverables and the requirements for our product. The process for our project is described in chapter 3, consisting of the software development methodology, communication, testing methods, etc. Chapter 4 describes the first phase of the project; the
research phase. During this phase we made the majority of the decisions about our project, like the architecture and frameworks to be used. After this research phase, we started the head phase of our project, which is the implementation of our product. The results of this phase can be found in chapter 5. Next up is the evaluation of our project. This is described in chapter 6 and consists of an evaluation of our process, code, division of labor, communication and requirements. Chapter 7 contains a discussion about our product and other subjects. Finally, chapter 8 describes our conclusion about the final product and the future.
2

Problem

2.1 Domain description

The client company is an e-learning company, providing a platform with the goal to improve the learning methods of their users and show them new ways to do so. The platform is split up into four main learning tools, being Dialog, Share, Live and Learn. Every learning tool has a different purpose for different situations while studying for a course. All together, the current platform offers a great variety of features.

The domain of this project is made up of clients, users and employees of the company. The clients of the company are the universities they sell the platform to. The users are the students and teachers utilizing the platform.

2.2 Domain analysis

As described in the previous section, the amount of features the platform offers is large, hence the problem that we have to solve is mainly an engineering problem. So our solution will have the biggest impact on the developers, as they have to adopt the new modular structure. The modular structure is intended to be used by developers and should ensure that developers can work on their own isolated module, without being dependent on other developers to continue with their work.

The users will not directly be affected by our work, actually if our solution meets the requirements, users shouldn’t even notice the underlying modular structure.

2.3 Deliverables

The goal of this project is to deliver a program that allows us, and other developers in the future, to run a module in a separate and isolated environment, so that the functionality and logic is kept apart from the core platform. The module will contain a frontend and a backend part, both
fully tested. Also, an example module will be build, which will be fully documented, to show how our modular system can be used to develop a module.

The project will be considered successful when we are able to run a basic example module using our modular structure, where at least all of the high priority requirements have been met and everything is tested adequately.

## 2.4 Requirements

We have listed and prioritized all the requirements according to the MoSCoW principle, to show the importance of each requirement for this project.

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>User story</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLI</td>
<td>As a developer, I want a CLI to create, scaffold and generate the module and parts of it</td>
<td>Should Have</td>
</tr>
<tr>
<td>2</td>
<td>Separate runnable</td>
<td>As a developer, I want to be able to run the module separately</td>
<td>Must Have</td>
</tr>
<tr>
<td>3</td>
<td>Backend and frontend</td>
<td>As a developer, I want to be able to add both frontend and backend functionality to a module</td>
<td>Should Have</td>
</tr>
<tr>
<td>4</td>
<td>Services</td>
<td>As a developer, I want to be able to access user data, comments and more</td>
<td>Must Have</td>
</tr>
<tr>
<td>5</td>
<td>Isolation</td>
<td>As a developer, I want modules to be defined separately from the core platform, to prevent bugs</td>
<td>Must Have</td>
</tr>
<tr>
<td>6</td>
<td>Usage statistics</td>
<td>As a company, I want to track user actions and activity for statistical purposes</td>
<td>Should Have</td>
</tr>
<tr>
<td>7</td>
<td>Easy adoption</td>
<td>As a developer, I want to be able to adopt the modular design easily</td>
<td>Should Have</td>
</tr>
<tr>
<td>8</td>
<td>Tests</td>
<td>As a developer, I want to be able to test every module both frontend and backend</td>
<td>Should Have</td>
</tr>
<tr>
<td>9</td>
<td>Hidden</td>
<td>As a user, I don't want to notice the modular structure at all</td>
<td>Must Have</td>
</tr>
<tr>
<td>10</td>
<td>Views</td>
<td>As a developer, I want to be able to add multiple views</td>
<td>Must Have</td>
</tr>
<tr>
<td>11</td>
<td>Module data</td>
<td>As a developer, I want to be able to save module-specific data</td>
<td>Could Have</td>
</tr>
<tr>
<td></td>
<td>Code quality</td>
<td>As a developer, I want to be able to see automatically generated code quality statistics</td>
<td>Could Have</td>
</tr>
<tr>
<td>---</td>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>

Table I: Requirements
3

Process

3.1 Scrum

We will be adhering to the Scrum methodology during the development process of our product. The Scrum roles we have assigned and conventions we have determined are described below.

Scrum roles

- Product owner: Siem Kok, representing the client company
- Scrum Master: Daniël Vliegenthart
- Development team: Richard Machielse, Daniël Vliegenthart
- Supervisor: Andy Zaidman

Conventions

- Sprint duration: 2 weeks.
- Backlog: A Trello board with the appropriate lists, being Committed Items, Tasks not Started, Tasks in Progress, Tasks Completed, Finished Items and Skipped Items. This setup gives us the convenient tooling to keep track of our progress.
- Planning meeting: At the start of every sprint. Determine what Backlog items to plan for this sprint and prioritize them.
- Retrospective meeting: At the end of every sprint. Discuss what has been done and what items have been finished. This will result in a conclusion about the correctness of the estimated effort. If the estimated effort was incorrect, a new estimation of effort for the next sprint is made.
- Daily Scrum meeting: Standup style meeting to discuss what has happened since the last Daily Scrum and what is planned for the upcoming day.

3.2 Communication

Efficient communication between the different parties in a software development project is an essential ingredient for success. The core of our communication with the client consists of the
Daily Scrum standup meeting, to keep everybody up to date and discuss potential problems or obstacles as swiftly as possible.

Communication with our TU Delft Coach, or supervisor, consists of a combination of Trello backlog cards management and Skype calls or more frequent meetings if desired so. The shared Trello board will ensure that our supervisor can check our progress at any stage of the project. We discuss our progress and potential issues during the Skype calls or meetings, which are planned every week or every two weeks, depending on the amount of material to discuss.

3.3 **Testing**

The choice of a software testing methodology can greatly benefit or harm the progress and final quality of the code written. The described choice is made up of two main parts, being the chronological process of writing the tests and the type of tests written.

First off, the chronological process of test writing has to be determined. A well known and often picked method is Test-driven development, where the tests are written before you start implementing any of the software itself. This offers the big advantage that you write tests to match your requirements and don't write tests to match the already written code. This is an excellent testing method, however we did not pick this exact approach, because of the fact that a big part of our project is to actually find out what the best way is to build this particular modular system. Most decisions will take more time and the answers will be clearer when we have advanced further into this project. This is the reason why testing while coding, an adaption of the Test-driven development method, is best suited for us, because it allows us to ensure that everything is fully tested, all without restricting our ability to be agile in our code structure.

Besides the testing methodology, a decision also has to be made regarding the type of tests written. We are writing unit and end-to-end tests for all of our modular components. The unit tests ensure that every block of code is working and handling errors as intended, while the end-to-end tests verify that our end functionality is as we require it to be.

3.4 **Code coverage**

To make sure that we test all of the code, we will use code coverage tools to determine the level of coverage. Although code coverage represents the cases in your code that have been tested, it doesn’t represent the cases that were left out. Therefore we should strive to have a high code coverage, but just a high coverage is not sufficient. We will try to test all of the edge cases that are relevant to our project.
3.5 **Code quality**

Our goal for this project is not only to create an impressive product, but also to write our code with high code quality and security in mind. One of the ways to maximize this is to utilize tools created with the purpose to check and/or enforce this. We check analysis reports frequently, being multiple times a week, to improve our code quality in an agile and iterative way.

3.6 **Version control system**

We use Git and Github alongside it for our versioning control. Our client uses these tools already and we also feel that it is the best tool out there by far.

3.7 **Code reviews**

One of the great advantages of Github is the convenience with which you can do code reviews. We use branches to develop separate features and pull requests to indicate that this feature is ready to be merged into the master branch. These pull requests provide an excellent opportunity to do code reviewing, it is in fact one of the core features of Github.

3.8 **Timeline**

This section outlines the set deadlines and phases for each week, to give a clear overview of our timeline during this project.

<table>
<thead>
<tr>
<th>Weeknumber</th>
<th>Phase</th>
<th>Deadlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Research</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Research</td>
<td><em>Friday May 1st: Research report</em></td>
</tr>
<tr>
<td>19</td>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Implementation</td>
<td><em>Tuesday May 26th: First submission to SIG</em></td>
</tr>
<tr>
<td>23</td>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Implementation</td>
<td><em>Tuesday June 9th: Second submission to SIG</em></td>
</tr>
<tr>
<td></td>
<td>Documentation</td>
<td><strong>Friday June 19th</strong>: Submit final report</td>
</tr>
<tr>
<td>---</td>
<td>---------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>26</td>
<td>Presentation</td>
<td><strong>Friday June 26th</strong>: Final presentation</td>
</tr>
</tbody>
</table>

Table 2: Timeline of the project
4.1 Architecture

The main part of this project is to come up with a modular structure that makes it possible to create and run parts of an application both isolated and integrated in the platform. The modules have to be as secure as possible and should be easily understandable.

4.1.1 Separation

Modules have to be able to access and modify data, provide data from their own and communicate with other platforms. Also, modules should be able to offer an interface in any location in the platform. Therefore we separate the project into backend and frontend modules.

Figure 1: The full platform and module architecture. The black lines represent links with child components while the dotted lines reflect data communication.
Backend modules will be able to access data from the core API, they will become API modules. They will run isolated from the core API, in an API Module Runner, to improve security, because this way they can’t access the database or core code directly. All communication will go through the API and will therefore be easier to secure and test.

Frontend modules will provide visual aspects and frontend communication to the core UI of the platform, and will therefore be UI modules. UI modules can be included in the UI, since browsers don’t allow much isolation in one tab. For more information on this see section 4.2.5. UI modules should only be able to communicate with the UI or their API module. Therefore we came up with the architecture visualised in figure 1.

### 4.1.2 Backend

An API Module has to be able to communicate with the core API and UI module. This communication will usually be done using a RESTful JSON API\(^\text{16}\). This will be provided by a Rails gem. However a backend module is also able to provide other export formats, like html and xml.

![API Module architecture](image)

Figure 2: The API Module architecture. The black lines represent links with child components while the dotted lines reflect data communication.

Figure 3 below shows the architecture of the Rails gem after our research phase. We will setup this architecture with a focus on strongly decoupled and flexible services\(^\text{22, 27, 28, 30}\). This will result in high agility and centralized settings, like the fact that any endpoint can be easily changed by just editing the environments endpoint configuration file, which is essentially just a dictionary of settings. This configuration architecture also provides the possibility to add a brand new environment by just editing one configuration file. Adapters\(^\text{24}\) can be added or removed when desired, enabling a new request or authentication method, while leaving the services code untouched. All of this accommodates the quick and agile feature development and testing flow.
4.1.3 **Backend runner**

Each API module will be mounted into a backend runner that runs separately from the core API.

Figure 3: The Rails modules gem architecture 0.1.0. The black lines represent links with child components while the dotted lines reflect data communication.

Figure 4: The API module runner architecture. The black lines represent links with child components while the dotted lines reflect data communication.
4.1.4 **Frontend**

The UI module will consist of the module specific code and an ember addon. This addon will provide the communication between the core UI and the UI module. See figure 5 for the corresponding diagram.

![Diagram of UI module architecture](image)

Figure 5: The UI module architecture. The black lines represent links with child components while the dotted lines reflect data communication.

4.1.5 **Command line interface**

In order to make it easy for developers to get started on a new module, as required according to requirements #1 and #7, we have to make a command line tool that you can use to generate and set up a new module easily. This tool will be able to check if a developer's' environment is set up correctly and scaffold both frontend and backend modules.

4.2 **Decisions**

This section outlines the biggest decisions we have made during our project. Every decision will be based on a comparison of the pros, cons and mitigations for all the available alternatives for that particular purpose.

4.2.1 **Backend framework**

The company has been using Ruby on Rails as its backend framework since the beginning. Our client has made it clear to us that they would like to continue to use Ruby on Rails as a backend, also for the modules. We fully support this decision, because we are both familiar with the framework and have experienced how powerful it really is.
Rails combines the Ruby programming language with HTML, CSS, and JavaScript to create a web application that runs on a web server. It's an excellent backend framework to create a RESTful API and has a MVC structure, with all the advantages this model brings, such as the fact that this MVC model ensures that little mistakes are made regarding where code should be put and how different files should interact. Finally, Ruby on Rails has built in speed optimization and security layers.

Besides the previously described advantages that Ruby on Rails has, it also matches our backend requirements closely. Ruby on Rails has a strict architecture, similar across basically all projects written using the Rails framework, so the adoption for the developers will come naturally. Rails also makes it possible to save module-specific data. The use of a Rails Engine, basically a mountable plugin for a Ruby on Rails application, makes it runnable separately by mounting it in the backend runner. Finally, the implementation of services can be done in the way Rails is intended to be used. being a gem for the services. A gem is basically an installable plugin for any Rails application, making it perfect for adding a service layer between the module data layer and the core API.

4.2.2 Frontend framework

During the orientation phase we had a clear idea of what kind of features a frontend framework should support. This led to the following candidates list; EmberJS, Backbone/Marionette and AngularJS. In this section we will discuss each framework, compare the pros and cons and mitigate cons where possible, which enabled us to make a well substantiated decision.

EmberJS

EmberJS is a framework for creating rich JavaScript web applications. It has the philosophy of "Convention over Configuration", basically meaning that it makes a lot of decisions for you, which helps developers to be more productive and write better code. Finally it has excellent definitions on routers, templates, components and data loading.

<table>
<thead>
<tr>
<th>EmberJS</th>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Convention over Configuration&quot;, so it can infer a lot as long as you follow the set conventions, so minimal configuration.</td>
<td>In the past API changed a lot, confusing to see old answers not compatible with current release.</td>
<td></td>
</tr>
<tr>
<td>Clear router and separate data layer (called Ember Data), which integrates very well with Ruby on Rails backend.</td>
<td>Since so many conventions exist, there is not a lot of freedom.</td>
<td></td>
</tr>
</tbody>
</table>
Major focus on performance in design of this framework, ensuring optimal loading times.

Beautiful and clean code, separated in Controllers, Models, Routers, etc.

Table 3: Pros and cons of EmberJS

**Backbone/Marionette**

Backbone is a super light-weight Javascript framework that allows for Javascript code to be structured in an MVC structure. In this case, the M stands for a Model, which handles all data fetching and population. V is the View and C the Controller, which handles the application’s logic. Marionette is defined as a "Backbone Framework" which simplifies the Backbone application code with more robust views and architecture solution.

<table>
<thead>
<tr>
<th><strong>Backbone/Marionette</strong></th>
<th><strong>Pro</strong></th>
<th><strong>Con</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely lightweight, fast and small memory footprint.</td>
<td>No set structure, decisions have to be made by the developer, increased chance of mistakes.</td>
<td></td>
</tr>
<tr>
<td>All source code documented and annotated.</td>
<td>No support for two-way binding.</td>
<td></td>
</tr>
<tr>
<td>Few concepts to learn (Models, Collections, Views and Routes).</td>
<td>Difficult to unit-test, because it manipulates the DOM directly.</td>
<td></td>
</tr>
<tr>
<td>No set structure, allows for a lot of freedom, a good fit for a wide range of projects.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Pros and cons of Backbone/Marionette

**AngularJS**

AngularJS is a structural framework for dynamic web applications, which lets you use the HTML itself as a template language and allows you to extend HTML’s syntax to express any application's components. AngularJS also supports data binding and dependency injection, which eliminates a lot of code you would otherwise have to write yourself.

<table>
<thead>
<tr>
<th><strong>AngularJS</strong></th>
<th><strong>Pro</strong></th>
<th><strong>Con</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy two-way-binding.</td>
<td>High complexity of Directives API.</td>
<td></td>
</tr>
</tbody>
</table>
Decision
In the end, we weighed the advantages and disadvantages of all the requirements for the frameworks, as can be seen below in table 6, and concluded that EmberJS is the clear winner here. Most of the disadvantages of EmberJS could be mitigated, because the way EmberJS is intended to be used closely matches the way we want to build our software. Also, the templating tool for EmberJS has been improved drastically over the past few versions, with the release of HTMLBars, the replacement for the previously built in Handlebars. Finally, the Ember CLI is a very powerful tool to manage any packages associated with the frontend application.

<table>
<thead>
<tr>
<th>Frontend framework⁴, ⁹, ³²</th>
<th>EmberJS</th>
<th>Backbone/Marionette</th>
<th>AngularJS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance optimizations</td>
<td>+ +</td>
<td>+ +</td>
<td>+</td>
</tr>
<tr>
<td>Source code size</td>
<td>- -</td>
<td>+ +</td>
<td>-</td>
</tr>
<tr>
<td>Code structure (MVC)</td>
<td>+ +</td>
<td>- -</td>
<td>+</td>
</tr>
<tr>
<td>Templating engine</td>
<td>+ +</td>
<td>- -</td>
<td>+</td>
</tr>
<tr>
<td>Two-way data binding</td>
<td>+</td>
<td>- -</td>
<td>+ +</td>
</tr>
</tbody>
</table>

Table 6: Rating of frontend frameworks

4.2.3 Mock API server

A mock API server is an essential tool to enable a smooth development flow for backend modules. A mock API server is a server that provides the stubbing and mocking of webservices and resources, or in other words, gives you the possibility as a developer to make requests to this server in the same way you would when using the production API server. The existence of a correctly setup mock API server ensures that the data flow for the development environment is constant and predictable, so no data flow related bugs exist. It also allows you to specify error codes for certain requests, to test the error handling by the backend module in development.
This section compares the candidate mock API servers we have found to be most likely to fit our requirements, which are Apiary/API Blueprint, JSON Server, Sandbox and a Rails fixture environment. This comparison has been conducted in the same way as we made our decision for the frontend framework.

**Apiary/API Blueprint**

The motto of API Blueprint is "API Documentation with powerful tooling". As a user you have to create a custom API Blueprint file, which specifies all the properties, descriptions, resources, endpoints and hard coded responses (including error codes). This file is used to automatically generate an API server with the documentation alongside it. This tool is powerful by itself, but its functionality is strongly enhanced by combining Apiary with it. Apiary provides the user with an editor and inspector for the created API Blueprint file. The editor shows the API Blueprint file and generated interactive documentation side by side and applies every edit to the API Blueprint file instantly to this documentation. The inspector lists all the requests made to the mock API Server.

<table>
<thead>
<tr>
<th>Apiary/API Blueprint&lt;sup&gt;5, 6, 14, 18&lt;/sup&gt;</th>
<th><strong>Pro</strong></th>
<th><strong>Con</strong></th>
<th><strong>Mitigation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Beautiful and interactive documentation is automatically generated.</td>
<td>It is just a static file, so every request response has to be hardcoded.</td>
<td>Static validations are possible, since error codes can be returned for certain urls.</td>
<td></td>
</tr>
<tr>
<td>Inspector of requests shows all the essential information and details of every single request made.</td>
<td>Dynamic validations are not supported.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The URL to the project contains a private hash, ensuring that it is private. There is an option to make the project public as well, giving us the possibility to publish the documentation if we so desire at a later time.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validators implementation through the usage of a JSON Schema&lt;sup&gt;16&lt;/sup&gt;.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Pros and cons of Apiary/API Blueprint
JSON Server

JSON Server\textsuperscript{17} is a Node package that provides a quick backend for prototyping and mocking API servers. It’s a minimalistic JSON RESTful API server\textsuperscript{15}, which uses a JSON file as its provider of the endpoints and responses. As a user, you have to create this JSON file and the server uses this to automatically generate RESTful urls and responses.

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
<th>Mitigations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic generation of urls and appropriate responses from provided JSON file.</td>
<td>No documentation generated.</td>
<td>Generate documentation separately.</td>
</tr>
<tr>
<td>Very lightweight.</td>
<td>A lot of effort to include authentication and validations (separate Node packages).</td>
<td></td>
</tr>
</tbody>
</table>

Table 8: Pros and cons of JSON Server

Sandbox

Quick and easy mock RESTful API and SOAP web services\textsuperscript{29}. Sandbox\textsuperscript{12} is a Javascript mock API provider, which offers a UI consisting of an overview of the routes, a raw view of the editable code, a console and a state overview. There are two ways of adding a new route or resource, by either using the user interface or by writing it in the raw Javascript editor.

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
<th>Mitigations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication and validations are easily controlled and customized, because the underlying structure is basic Javascript.</td>
<td>No documentation generated.</td>
<td>Generate documentation separately.</td>
</tr>
<tr>
<td>Since it’s written in Javascript, customization is convenient.</td>
<td>Authentication is really basic and does not support our needs.</td>
<td></td>
</tr>
</tbody>
</table>

Table 9: Pros and cons of Sandbox
Rails fixture environment

The last option to consider for the mock API server would be to add a new fixture Rails environment to the backend core code, alongside the other Rails environments, such as development, production and staging.

<table>
<thead>
<tr>
<th></th>
<th>Pro</th>
<th>Con</th>
<th>Mitigations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full control of every aspect</td>
<td>No documentation generated.</td>
<td>A dependency on the state of the backend code to get the proper data and request handling.</td>
<td>Generate documentation separately.</td>
</tr>
<tr>
<td>of the mock API server.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom validators for the</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resources are possible,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ensuring that errors are</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>handled as strictly as you'd</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>like.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10: Pros and cons of Rails fixture environment

Decision

See table 11 below for the summary of requirements for the mock API Server. The main priorities that have to be considered are the timeframe we have to implement this mock API server and the dependency on the development of the backend core. Both these priorities strongly oppose towards the Rails fixture environment, even though it looks the best in the table below, but the development of the backend core has not advanced far enough yet to make this possible. The Apiary/API Blueprint service adheres to the most requirements out of the other candidates and therefore offers us the most applicable solution.

<table>
<thead>
<tr>
<th>Mock API server</th>
<th>Apiary</th>
<th>JSON Server</th>
<th>Sandbox</th>
<th>Rails environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versioning</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+ +</td>
</tr>
<tr>
<td>Automatic URL creation</td>
<td>- -</td>
<td>+ +</td>
<td>- -</td>
<td>+ +</td>
</tr>
<tr>
<td>Authentication</td>
<td>+</td>
<td>- -</td>
<td>- -</td>
<td>+ +</td>
</tr>
<tr>
<td>Validations</td>
<td>+ +</td>
<td>- -</td>
<td>+ +</td>
<td>+ +</td>
</tr>
<tr>
<td>Stability of data</td>
<td>+ +</td>
<td>+ +</td>
<td>+ +</td>
<td>+</td>
</tr>
<tr>
<td>Documentation generated</td>
<td>+ +</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 11: Rating of mock API servers
4.2.4 Command Line Interface framework

For us, the decision to use the NodeJS framework for the development of our CLI was very clear. NodeJS is an open source and cross-platform runtime environment, where applications are written in Javascript. NodeJS is modular, meaning that node modules, or packages, can be required in a project and used out of the box with little to no effort. This provides an extremely agile and powerful way of developing applications, or in our case, a Command Line Interface tool. NodeJS also comes with the NPM package manager out of the box, making the management of these NodeJS modules or packages a breeze. All in all, the combination of NodeJS with NPM and Nash, a library to craft command line interfaces, is the most powerful tooling for the development of our CLI.

4.2.5 Frontend isolation method

The backend modules will run isolated by separating the servers. In the frontend, we have to think of another approach to make sure modules can’t access core code or data, as all code runs in just one browser tab.

Iframes

The first option to consider is to use iframes. Iframes are parts of a browser tab and fully run in their own context. They can technically be treated as different tabs and have their own url endpoint. Communication between iframes and the tab can be done by posting plain text messages.

<table>
<thead>
<tr>
<th></th>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code is 100% isolated.</td>
<td></td>
<td>Mutual code has to be included twice.</td>
</tr>
<tr>
<td>Interfacing works like a normal browser tab.</td>
<td></td>
<td>Data communication can only be done with messages.</td>
</tr>
</tbody>
</table>

Table 12: Pros and cons of iframes

Workers

Workers are background processes for browser tabs. They don’t have any visual aspects and are just meant to run asynchronous code. Data can only be exchanged by posting messages between the tab and the worker.
### Workers

<table>
<thead>
<tr>
<th><strong>Pro</strong></th>
<th><strong>Con</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Code is 100% isolated.</td>
<td>Mutual code has to be included twice.</td>
</tr>
<tr>
<td>Data communication can only be done with messages.</td>
<td>Templates have to be rendered and sent to the core each time, since workers can’t be included in the interface directly.</td>
</tr>
<tr>
<td>Browser events have to be communicated using messages.</td>
<td></td>
</tr>
</tbody>
</table>

Table 13: Pros and cons of workers

### Dependency injection

Dependency injection\(^\text{11}\) is a method in that makes sure that different components of the framework aren’t able to access other components that they aren’t supposed to access. There will not be a root variable or something so components will just “float” in the memory and can only be linked by the framework. While they live in the same session, if modules would be namespaced, they won’t be able to access parts of the core if they aren’t injected.

<table>
<thead>
<tr>
<th><strong>Pro</strong></th>
<th><strong>Con</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Template rendering can follow the framework.</td>
<td>Breaking out of scope is hard but it is not 100% isolated.</td>
</tr>
<tr>
<td>Data communication can follow the framework.</td>
<td></td>
</tr>
<tr>
<td>Easy adoption and good documentation.</td>
<td></td>
</tr>
<tr>
<td>Supported by chosen frontend framework.</td>
<td></td>
</tr>
</tbody>
</table>

Table 14: Pros and cons of injection
Decision

The above considerations are summarized in table 15, which shows an overall rating for each method on each topic. All in all we conclude that injection is the best way to go, because it integrates best with the frontend framework and still suffices the isolation requirements we have.

<table>
<thead>
<tr>
<th>Isolation method</th>
<th>Iframes</th>
<th>Workers</th>
<th>Injection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core code isolation</td>
<td>+ +</td>
<td>+ +</td>
<td>+</td>
</tr>
<tr>
<td>Template rendering</td>
<td>+</td>
<td>-</td>
<td>+ +</td>
</tr>
<tr>
<td>Data communication</td>
<td>-</td>
<td>-</td>
<td>+ +</td>
</tr>
<tr>
<td>Easy adoption</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Frontend framework support</td>
<td>-</td>
<td>-</td>
<td>+ +</td>
</tr>
</tbody>
</table>

Table 15: Rating of frontend isolation methods
5.1 Scope of project

During our research phase we have determined what the architecture of our final product will be, as seen in section 4.1. Another important aspect of the project that has to be determined is the scope of the project. This will ensure that there are clear boundaries of what we have implemented for the client company and what components of software were already available within the client company.

For our project we have built a modular structure, consisting of three main components, being the backend module, frontend module and the command line interface. This modular system will be the foundation for the developers of the new platform for our client, to accommodate a range of requirements, as described in section 2.4.

Some aspects of our implementation were dependent on software components under development by the client company itself, being the core API and core UI. During the research phase of our project we were not certain yet how much the development of these core components would progress during the weeks to come. So this means, that when we made a decision based on the state of these core software components, we are referencing the core API and core UI within the client company and not a part of our own modular system.

5.2 Final product

The result of our project consists of three repositories, being the frontend modules repository, the backend modules repository and the command line interface repository. The code of the project is not publicly available.
5.2.1  **Frontend modules**

The goal of frontend modules is to be able to add visual parts to the core application. This can be done using components.

**Components**

A developer that is going to build a UI module is able to add visuals by adding components. Components are isolated parts of a website that can contain both template and application logic. These components can be loaded hard coded or dynamically in the core UI. They can retrieve data from their corresponding backend module by using the existing models in the core UI, extending them, or adding new models. For a more detailed explanation of how this works see section 5.3.1.

These components can be injected into the core application hard-coded or dynamically. During development, developers can start a server running just the module as well. This will visualize the module’s contents and run each component, by simulating the core UI.

5.2.2  **Backend modules**

The main functionality of a backend module, or so called API module, is to provide the communication with the core API and UI module. This is achieved by building a Rails gem, basically a plugin for a Rails application.

This gem imports the settings as specified in the manifest file, described in section 5.3.1. It uses the data permissions from the manifest to inject the correct core API models into the backend module, so that the developers can use these, and only these, models. This offers a great deal of security, because it allows for a multi-layer authorization. First off, a developer can't make a request using a model he hasn't asked permission for in the manifest. And even if this developer found a work-around to do so, the core API would catch this and return an error code, which is handled by the gem as well.

The Rails gem also overwrites certain Rails generators in the backend module Rails project, to enable the creation and scaffolding of models, controllers and resources with the correct modular syntax and dependencies.

5.2.3  **Command Line Interface**

The command line interface is basically nothing more than a simple generator that bootstraps a new plain module. It can generate both frontend and backend modules. Also, it can verify a developer’s environment. The available commands are listed in table 16.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbf ui:check</td>
<td>verify ui environment</td>
</tr>
<tr>
<td>fbf ui-module:new NAME</td>
<td>scaffold new ui module</td>
</tr>
<tr>
<td>fbf api:check</td>
<td>verify api environment</td>
</tr>
<tr>
<td>fbf api-module:new NAME</td>
<td>scaffold new api module</td>
</tr>
</tbody>
</table>

Table 16: List of available commands in the Command Line Interface

5.3 Architecture

The proposed architecture as a result of the research phase can be seen in section 4.1. Here we describe our process to convert this architecture from schema's to the actual implementation. Any changes made to the architecture during our implementation will be discussed and the final architecture of components will be explained with the help of diagrams.

5.3.1 Manifest

Both frontend and backend modules will contain a JSON manifest file that contains information about the name, description, version, permissions and dependencies of the module. Also it will register the data that a module wants access to, and in the case of frontend modules, which components are available.

5.3.2 Frontend modules

As Ember has been chosen as the main framework for the Core UI, we choose to make an Ember Addon, being a plugin for the framework, to provide modularity to the frontend. The addon is added to the core UI and to each module to provide its logic.
The UI Module Addon for the Ember framework consists of three main parts, that are described below. A visualisation of the architecture of the addon can be found in figure 6.

**Services**

The addon provides ways of communicating with the API. In Ember, data is typically retrieved from a store that is injected into all components. This store uses an adapter to handle authentication and endpoint/namespace configuration. Therefore we added both a “ModuleStore” and a “ModuleAdapter” that handles requests from components. Models from the core can be used, extended or overwritten and new models can be added. However, all communication will be namespaced to the corresponding backend module, so the module’s backend has to provide the model data.

**Isolated Runner**

Each module can be run both integrated within the core UI and isolated as a standalone application. Therefore the addon contains a simple demo application that visualises the module and its components. Also, it will simulate the context that a component can run in, so the module will represent its final result as well as possible.

**Generators**

To make it easier for developers to adopt the module’s workflow and create components, some generators are available to scaffold and register a new component or model.
### 5.3.3 Backend modules

Throughout the development process we have made several changes compared to the original architecture of the API Modules gem as seen in figure 3. These changes can be seen in figure 7 below. The main reason for these changes was the discovery of the Active Resource\(^3\) gem, which is described by the creators of the gem as:

"Active Resource (ARes) connects business objects and Representational State Transfer (REST) web services."\(^3\)

This definition states that Active Resource provides the connection between local models and a RESTful API, like the RESTful JSON API\(^{16}\) for example.

---

**Figure 7:** The Rails modules gem architecture 0.2.0. The black lines represent links with child components while the dotted lines reflect data communication.

The API Modules backend Rails gem provides two main functionalities, being services and generators.

**Services**

A service inherits a big portion of its functionality from the Active Resource gem, which handles the connection between the local service and the data source, being a RESTful JSON API\(^{16}\).
Generators
We have implemented multiple generators, which handle the generation of certain files in the Ruby on Rails project. These generators inherit from the Rails generators and overwrite the standard of these Rails generators where necessary. These generators are used by the Command Line Interface to customize the generated Rails project and can also be used by the developers of a backend module.

5.3.4 Command Line Interface

The command line interface has two purposes: the first is to be able to verify your systems environment and the second is to generate plain modules. Both of these tasks should be possible for both UI and API. Therefore we came up with the architecture demonstrated in figure 8.

We decided to have four commands that a user can run. All of these commands define tasks that will be executed sequentially.

The CLI has been implemented using the Node platform, which means that the code is written in Javascript.
5.3.5 Backend modules runner

The backend modules runner is meant to be a separate Rails project, that mounts modules dynamically from a database of available and approved modules. Unfortunately, because the Core API was not yet ready to contain such data, we left out the runner from our project.

Without the integration with the Core API, the runner wouldn’t differ much from the dummy project that is included in each module, so we didn’t think it was useful to build it.
6.1  **Process**

We have been using Scrum during our project like we described in section 3.1. The Daily Scrum meeting helped us to keep our work aligned and the sprints of two weeks gave us the required structure to split up the development of our final product in multiple clearly separated stages. During our retrospective meetings, we discussed what had been done at what pace and how this could be improved for the next sprint if necessary.

The usage of Git and Github was exactly how we expected it to be, because we have loads of experience with these tools, since we have been using this version control system in previous projects during our studies and privately. Also, the code reviews were an excellent method of giving each other well structured feedback on our written code.

The other aspects of our process are evaluated in the upcoming sections of this chapter.

6.2  **Code coverage results**

In order to see if the test code represents all of the use cases in the project, we generated code coverage reports using coverage tools that work with the languages/frameworks of the different parts of the project, being Blanket for Ember, SimpleCov for Rails and Covert for Node. The results can be found in table 17.

<table>
<thead>
<tr>
<th>Project</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend</td>
<td>96,40 %</td>
</tr>
<tr>
<td>Backend</td>
<td>82,43 %</td>
</tr>
</tbody>
</table>
### Code quality results

From the beginning of the project we wanted to analyze our code to make sure that the code would be of high quality. Therefore we used some tools to analyze the style and detect problems in our code.

#### 6.3.1 JSHint

JSHint\(^1\) is a tool that can statically detect errors and potential problems in javascript code. We used it to verify the quality of the frontend and the command line interface, since they are written in javascript. The output of the tool was quite valuable, we were able to eliminate the amount of JSHint violations in both projects to zero.

#### 6.3.2 Rubocop

Rubocop\(^2^6\) is a static analyzer for code written in the Ruby language\(^2^4\). We used it to improve our backend code. However, since rubocop is quite strict, we weren’t able to fix all of the violations. Even an out-of-the-box plain ruby on rails project results in about forty violations. We went over the violations however and fixed the ones that were relevant.

#### 6.3.3 Code Climate

Next to analyzing the code ourselves with the above tools, we rated our code with an online tool called Code Climate. This tool analyzes not only the quality, but also the style and security of the project, by giving it a “grade point average” rating between 0 and 4. The results can be found in table 18.

<table>
<thead>
<tr>
<th>Project</th>
<th>Lines of code</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend</td>
<td>719</td>
<td>GPA 3.81</td>
</tr>
<tr>
<td>Backend</td>
<td>394</td>
<td>GPA 3.88</td>
</tr>
</tbody>
</table>

---

1. JSHint is a free and open source static code analysis tool for JavaScript.
2. Rubocop is a static code quality tool for Ruby.

---

Table 17: Code coverage results

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI</td>
<td>97.42 %</td>
</tr>
<tr>
<td>Total</td>
<td>94.87 %</td>
</tr>
</tbody>
</table>

---

Table 18: Code quality results
From Code Climate we found some more styling issues that we fixed. The only issues that remained are duplication issues, mainly in tests. We felt the code would actually become more cluttered if we would fix these issues, so we decided to keep some of the duplications.

### 6.3.4 Software Improvement Group

Next to the quality checks we did ourselves, we got feedback from the Software Improvement Group on our whole project twice.

**SIG Feedback 1**

From the first round we received the following feedback:

“De code van het systeem scoort net vijf sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is.

De omvang van het systeem is echter klein (539 regels code) en de meeste bestanden bevatten slechts een paar regels code. Het lijkt er dan ook op dat dit gaat om een gegenereerde template met slechts een paar handmatige aanpassingen.

Op dit moment is er een klein duplicaat te vinden tussen de 'request_adapter' en de 'fixture_adapter', maar met het implementeren van de methoden zal dit probleem opgelost worden. Dit is wellicht wel iets om goed in de gaten te houden.

Hopelijk lukt het om het onderhoudbaarheidsniveau te behouden tijdens de rest van de ontwikkelfase. Als laatste nog de opmerking dat er wel een outline voor test-code is aangetroffen, maar de tests lijken niet geen functionaliteit te testen. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.”

From this feedback we tried to decrease the amount of duplication in our project. Also, we added tests that were on our schedule regardless.

As we are building a modular structure, our project is kind of a template for modules, so it is actually correct that there are a lot of small files.

<table>
<thead>
<tr>
<th>Command line interface</th>
<th>412</th>
<th>GPA 3.76</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total</strong></td>
<td>1525</td>
<td>GPA 3.81</td>
</tr>
</tbody>
</table>

Table 18: Code quality results according to codeclimate.com
SIG Feedback 2
From the second round we received the following feedback:

“In de tweede upload zien we dat het codevolume sterk is gegroeid, terwijl de score voor onderhoudbaarheid is gedaald. Dat is vrij normaal, het is erg moeilijk om een systeem goed onderhoudbaar te houden als het blijft groeien. Jullie scoren nu 4 sterren, wat nog steeds bovengemiddeld is.

Jullie hebben de eerder geconstateerde duplicaten weggewerkt, maar de deelscore voor Duplication is uiteindelijk toch gezakt omdat er nieuwe duplicaten zijn geïntroduceerd (bijvoorbeeld in de callback functies in check.js).

In tegenstelling tot bij de eerste upload hebben jullie nu wel echt testcode geschreven. De inhoud van de tests is nog vrij beperkt, eigenlijk zou je wat meer concrete scenario’s willen testen in plaats van alleen de basisfunctionaliteit, maar het is zeker een verbetering ten opzichte van de eerste upload.

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

The second feedback from SIG reflects our improvements. We tried to minimize the amount of duplication, however we thought it was cleaner to have some code appear in multiple places then to abstract this away.

We made a decision regarding the method of testing, which does reflect the findings by SIG about our testing. We are building a modular structure which facilitates the creation and development of modules. We provide the tools to create clean and structured modules, but not the actual modules themselves, these will be developed by the developers. So for this reason, we feel that the most important tests actually test the functionalities that we provide in our modular structure. Most scenarios would have to be tested on a module specific level, because these modules would provide the actual functionality for the platform. This being said, we will expand our tests cases with more edge cases during the last weeks of our project.

6.4 Division of labor
The separation of our software architecture into a frontend module, backend module and CLI, has made the division of labor pretty straight forward. We felt that the most efficient way to work, would be to each focus on a separate part of the total product. This enabled us to maximize our focus, knowledge and research for each part, because it wasn’t necessary for us to keep on
switching between the frontend, backend and CLI. The following sections describe this division of labor.

6.4.1 Frontend

The frontend module was built by Richard Machielse, because he knew most about the frontend framework we ended up using. His progress can be seen in figure 9 below.

![Figure 9: Github contribution to the frontend repository](image)

6.4.2 Backend

The backend module has been developed by Daniël Vliegenthart. His contribution can be seen in figure 10 below. This task was assigned to him, because he's had a lot of experience with the backend framework we decided to use.

![Figure 10: Github contribution to the backend repository](image)

6.4.3 Command line interface

We have both contributed to the development of the CLI. This seemed like the most logical division of labor to us, because it allowed us to keep our efforts balanced and enabled us to both work on our own expertise within the development of the CLI, being backend or frontend commands. As can be seen in figure 11, our contributions were quite similar, where the difference is caused by the fact that certain frontend or backend commands were more complicated or took more steps to implement.
6.5 Communication

We have described our planned methods of communication before we started this project in section 3.2. We feel like this setup of communication was the most efficient and correct method of communication with both our client and supervisor.

First off, we have had several meetings and Skype calls with our supervisor, which provided us with all the information, tips and feedback we needed. Also, the Trello boards ensured that our supervisor could see our current progress at any point of time.

As for the communication with our client, we have held the Daily Scrum meetings every morning, which helped us align our plans with our client almost daily, giving us the ability to improve our plans for the day in an agile manner.

6.6 Requirements analysis

The list of requirements can be found in section 2.4. Here we will analyse which requirements we have met and more importantly, describe why it is that we have not met some other requirements. Our final product adheres to all the requirements, except for #6 and #11. This does mean that every single 'Must Have' requirement has been met, which is absolutely essential for the success of our product.

The first requirement that we were not able to meet was #6:

| 6  | Usage statistics | As a company, I want to track user actions and activity for statistical purposes | Should Have |

The reason we have not been able to implement the usage statistics, is because there were no user actions or activity to track yet, since the core API and core UI were not built yet during the course of our project. We could have possibly created mocks for these user actions and activity, just for demo purposes, but we felt that it would be more beneficial to concentrate on improving other parts of the product. If the core application would implement these user actions
and activity differently from our mocks, this would even mean we would have to change the code itself, which we don't consider to be very efficient.

The second and last requirement we have not met is #11:

<table>
<thead>
<tr>
<th></th>
<th>Module data</th>
<th>As a developer, I want to be able to save module-specific data</th>
<th>Could Have</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We have made the decision to omit this requirement, because we did not find the time to implement this functionality. This requirement also did not have a high priority, as can be seen by the 'Could Have' label.
7 Discussion

7.1 Result

Both of us are confident with the final product we delivered. We believe we have provided a sufficient solution for the problem of the company. Also we are very content with the techniques we have learnt and used. We feel like this project has truly enabled us to apply the knowledge and skills we have gained during our Bachelor of Computer Science over the past three years.

7.2 Dependency on core development

In the beginning of the project, we were a bit too optimistic about the progress of the core application. We had expected both the core UI and core API to have evolved further then it turned out to have. Therefore we had to change our plans a bit. We left out the integration with the core and instead adapted it to be a more standalone project, as a proof of concept.

7.3 Future recommendations

In the future, it might be a bit more useful to let certain projects be standalone proof of concepts from the beginning. The company can then integrate them when they agree on the approach and when they’re ready.
8 Conclusion

8.1 Final product

To conclude this report, we are confident that the product we have built over the past two and a half months can be deemed successful, according to our definition of done, as described in section 2.3.

First off, it is possible to run a basic example module, generated with command line interface and containing both a backend (API) and frontend (UI) module, using our modular structure. All of the options, possibilities and details of our implementation of this modular structure can be found in chapter 5.

The second part of our definition of done states that at least all of the high priority requirements have been met. This has been more than achieved, as can be read in section 6.6.

Finally, we stated that everything, being the code for API modules, UI modules and the CLI, has to be tested adequately with a high percentage of code coverage. This as well has been successful, substantiated by the statistics in section 6.2.

We would also like to state that the code quality analysis results, as seen in 6.3, all indicate that we have written code of high quality regarding metrics like duplication, scalability, security and styling.

In conclusion, we have delivered a product that adheres to all of the conditions of our definition of done, all while keeping the maximization of code quality in mind.
8.2 Future

The result of our project can easily be integrated in the full application, once it’s ready. As we integrated our solution with the frameworks used by the client, they can use our architecture hard-coded or dynamically in the future. The architecture we described is still applicable and usable, and the company might use it the way we designed it.
Bibliography

1. About JSHint
   http://jshint.com/about/

2. About Node.js
   https://nodejs.org/about/

3. Active Resource
   https://github.com/rails/activeresource

4. AngularJS vs. Backbone.js vs. Ember.js
   https://www.airpair.com/js/javascript-framework-comparison

5. API Blueprint Specification
   https://www.airpair.com/js/javascript-framework-comparison

6. Apiary - How It works
   https://apiary.io/how-it-works

7. Automatic code loading: lazy loading of assets when navigating to a route
   https://github.com/ember-cli/ember-cli/issues/402

8. Building a Rails Generator, and packing it into a gem
   http://agmprojects.com/blog/building-a-rails-generator-and-packing-it-into-a-gem

9. Client-side MVC: Is Angular.js or Ember.js the better choice for JavaScript frameworks?
   http://www.quora.com/Client-side-MVC/Is-Angular-js-or-Ember-js-the-better-choice-for-JavaScript-frameworks

10. Creating and Customizing Rails Generators & Templates
    http://guides.rubyonrails.org/generators.html

11. Dependency Injection & Service Lookup - Ember.js
    http://guides.emberjs.com/v1.11.0/understanding-ember/dependency-injection-and-service-lookup/

12. Getting Started - Documentation - Sandbox
    https://getsandbox.com/docs/getting-started
13. Getting Started with Engines
   http://guides.rubyonrails.org/engines.html

14. Handling multiple actions on a single resource
   http://support.apiary.io/knowledgebase/articles/117119-handling-multiple-actions-on-a-single-resource

15. Iframe - MDN

16. JSON API - Format
   http://jsonapi.org/format/

17. JSON Server
   https://github.com/typicode/json-server

18. JSON Schema Validation
   http://support.apiary.io/knowledgebase/articles/147279-json-schema-validation

19. Lazy Loading with Ember
   http://madhatted.com/2013/6/29/lazy-loading-with-ember

20. Making Generators in Rails 3

21. Modules :: Eloquent JavaScript
    http://eloquentjavascript.net/10_modules.html

22. My take on services in Rails
    http://adamniedzielski.github.io/blog/2014/11/25/my-take-on-services-in-rails/

23. Nash - Craft command-line masterpieces in Node.js
    https://github.com/scottcorgan/nash

24. Rails Adapters 101
    http://blog.arkency.com/2014/08/ruby-rails-adapters/

25. RFC: Engines
    https://github.com/emberjs/rfcs/pull/10

26. RuboCop - A Ruby static code analyzer, based on the community Ruby style guide.
    http://batsov.com/rubocop/
27. Service objects in Rails will help you design clean and maintainable code
   https://netguru.co/blog/service-objects-in-rails-will-help

28. Service Objects: What They Are, and When to Use Them
   http://stevelorek.com/service-objects.html

29. SOAP Introduction
   http://www.w3schools.com/webservices/ws_soap_intro.asp

30. Using Services to Keep Your Rails Controllers Clean and Dry
    https://blog.engineyard.com/2014/keeping-your-rails-controllers-dry-with-services

31. What is npm?
    https://docs.npmjs.com/getting-started/what-is-npm

32. Why I Chose Ember.js
    http://ryantablada.com/post/why-i-chose-ember-js

33. Worker - MDN
    https://developer.mozilla.org/en/docs/Web/API/Worker

34. Writing Command Line Tools with Node
    http://javascriptplayground.com/blog/2015/03/node-command-line-tool/