A Modular Architecture for a Playful Brainstorming Web Application

Lukas Ameisbichler

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2019

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, September 24, 2019

Lukas Ameisbichler

Contents

Abstract

Brainstorming games were mostly performed offline together with a group of people. The ideas created by the participants are written on small sticky notes, which are placed on a whiteboard and grouped with markers or colors. There has been a transition starting to use online tools for this type of activity. Web-based brainstorming applications allow multiple users to contribute ideas and collaboratively develop a structure for their ideas over the internet, without having to be in the same room. Although such systems typically facilitate the input of a fairly large number of users, they focus primarily on idea generation based on a rather fixed interaction paradigm. However, various applications could utilize the data which was generated by such brainstorming activities. Therefore, the brainstorming web application should be extendable with different modules, as lightweight as possible, without the need for significant rewrites and changes in the underlying architecture and data structure for reusing and changing the data.

In order to achieve this goal, a plugin system has been developed and added to the brainstorming web application. Additionally, a plugin environment with a plugin scaffold and a sample plugin was provided, were external programmers can create third party plugins. These third-party plugins can be uploaded and used in brainstorming sessions to reuse and extend the generated data.

Kurzfassung

Brainstorming-Spiele wurden meist offline mit einer Gruppe von Personen durchgeführt. Die Ideen der Teilnehmer werden auf kleine Haftnotizen geschrieben und auf ein Whiteboard geklebt und mithilfe von Farben und Markern gruppiert. Immer öfter werden nun Online-Tools für diese Art von Aktivität verwendet. Webbasierte Brainstorming-Anwendungen ermöglichen es mehreren Benutzern Ideen einzubringen und gemeinsam über das Internet eine Struktur für diese Ideen zu entwickeln. Dafür müssen sie sich nicht einmal im selben Raum befinden. Obwohl solche Systeme typischerweise die Eingabe einer ziemlich großen Anzahl von Benutzern erleichtern soll, konzentrieren sie sich in erster Linie auf die Ideenfindung auf der Grundlage eines eher festen Interaktionsparadigmas. Verschiedene Anwendungen könnten jedoch diese Daten nutzen, die durch solche Brainstorming-Aktivitäten generiert wurden. Daher war das Hauptziel, ein Online Brainstorming tool so zu verändern, dass es mit verschiedenen Modulen erweiterbar ist, ohne dass wesentliche Änderungen in der zugrunde liegenden Architektur und Datenstruktur für die Wiederverwendung der Daten erforderlich sind.

Um dieses Ziel zu erreichen, wurde ein Plugin-System entwickelt und der Webanwendung hinzugefügt. Zusätzlich wurde eine Plugin-Umgebung mit einem Plugin-Gerüst und einem Beispiel-Plugin bereitgestellt, in der externe Programmierer Plugins erstellen können. Diese Plugins können dann im Anschluss hochgeladen und beim Brainstorming verwendet werden, um die generierten Daten wiederzuverwenden und zu erweitern.

Chapter 1

Introduction

1.1 Motivation

Brainstorming games are mostly performed offline with a group of people together. The ideas, created from the participants, are written on small sticky notes, which are placed on a whiteboard and grouped with markers or colors. Since people are interacting more often with computers and smartphones, there has been a transition starting to use online tools, where participants can collaborate with others over the internet without having to be in the same room. These applications allow to contribute ideas to the online whiteboard and collaboratively develop a structure for their ideas. Although such systems typically facilitate the input of a fairly large number of users, they focus primarily on idea generation based on a rather fixed interaction paradigm. However, various applications could utilize the data generated by such brainstorming activities. Most of these applications focus on mind mapping without the use of some gamestorming techniques for creating a solution, which is one of the main topics in the thesis. Therefore a modular data framework is needed for reusing the data in various playful activities such as votings, quizzes and other games.

1.2 Blobster Features Implemented

The prototype of Blobster was implemented in the SS18 IM590: Project 2 Course. The prototype consisted of the following features.

Grouping Ideas

A moderator can create a session with multiple participants, who can dispatch ideas to the moderator screen. The moderator can group ideas by using a folder element. On hovering over the folder, the moderator sees all the ideas of the folder. When clicking on one of those ideas, the idea is expanded and the optional description is shown. An idea can also be deleted, when there is no need for it or another duplicate exists.

Dragging ideas and folders

Dragging folders and ideas is possible on the moderator screen of the web application. When trying to drag a folder or idea, the item gets tilted and the moderator can drag it around within the custom drag layer. If an item is dragged out of this layer, it gets respawned to its original position before dragging.

1.2.1 Voting of Ideas

In the gamestorming technique Dot Voting, multiple participants can vote for all ideas within a folder on a scale from 0, meaning bad, to 5, meaning excellent. Simultaneously the votings are displayed in real-time on the moderator screen. This data is visualized within a bar chart.

1.3 Extension for the Master Project

As part of the Master Thesis Project, Blobster is used as a base Frontend implementation.

1.3.1 Situation

Starting with the prototype of the Project 2 SS18 Project Blobster, a decision was made that the folder structure is not suitable for this kind of use case. Therefore a change to a clustering approach, where the user can see the ideas without the need of hovering over a folder was selected. The main goal of the project is the extensibility of the data structure and the web application without the need of major changes. Therefore it is needed to introduce some kind of plugin system for the different gamestorming techniques and other modules which can be used in the tool for reusing the generated data.

1.3.2 Topic

In the course of the project, the initial topic was transitioning from a data architecture topic to a software topic, because there was more work needed to make a plugin system, than to create a reusable data architecture with a suitable Frontend. During the project, it became clear that the focus changed to have a good system pattern and a minimalistic Frontend, which will use it as an example. The Frontend could then be adapted in the future.

1.4 Evaluation

As an evaluation, a comparison with other possibilities and frameworks for a plugin-system should be done. In addition, a verification how the code splitting improved the loading time for the end-users should be made.

1.5 Thesis Outline

The following chapter 2 illustrates the fundamentals of brainstorming, mind mapping and gamestorming with some examples of gamestorming techniques. Furthermore, it will shortly describe the base architecture of single-page applications used in the thesis project and the downside of it. Chapter 3 is about the most famous brainstorming and e-learning tool, which are related to the *Blobster* project. Projects using a modular architecture for creating an extensible web application will be shown afterwards. After that, the thesis gets deeper into the thesis project part, with chapter 4. This chapter presents the concept behind the thesis project. It covers the implemented requirements of the system, the use-cases, the system architecture, and the concept of the developed plugin-system. Afterwards, the technical implementation is described in chapter 5. The evaluation part is located in chapter 6 and at the end, chapter 7 will summarize everything up and show the challenges and possible future work.

Chapter 2

Fundamentals

This chapter describes the fundamentals needed for the practical part of the thesis. Therefore the first section is about brainstorming and gamestorming with an in-depth look into two different games which could be utilized in such an environment. The last part of this chapter is about single page applications and one of the main problems of them.

2.1 Brainstorming

Brainstorming is a popular method for creatively solving problems, mostly accomplished in groups, where the ideal group size should be between five and ten people. Osborn defined 4 basic rules for brainstorming sessions [18]:

- *Judgment is ruled out*: Ideas of others should not be criticized, every idea is welcome. Analysis and evaluation should be done after the brainstorming.
- *Freewheeling is welcomed*: The wildest ideas are the best. It is harder to make an idea more exciting than the other way round.
- *Quantity is wanted*: Every idea is important and wanted. The more ideas are spawned, the higher is the chance that an idea triggers other participants, which then results in another one. Also, a bigger pool results in a bigger range to choose from at the end of the session.
- *Combination and improvement are sought*: Suggestions on how to improve an idea, or to combine ideas is highly appreciated in a brainstorming session.

Brainstorming is an approach of problem-solving with lateral thinking, which is a type of thinking to solve problems innovatively and creatively in an open environment, where everyone is encouraged to participate. The aim is to break out of the different old approaches and look at a problem in another way to generate an original and creative solution, while others can give additional feedback and input or construct new ideas out of them [7]. Many people with diverse experience are highly favored because this will increase the richness of ideas generated by the group [1]. In this way, brainstorming can be better than conventional group interaction. They are often undermined by group thinking, a phenomenon occurring when people do not want to jeopardize group consensus rather than presenting an alternative way or give a more unpopular opinion about the topic than others. That means the view of the participants is shadowed by the group notion [13]. The brainstorming rules and the open environment should avoid this kind of behavior.

Although group brainstorming has many positive aspects, there are also some negative ones. In some cases, certain individuals doing brainstorming alone can create more ideas, which could also be more unique than the same amount of people working together in groups [16]. The thinking of the participants is affected by others and could lead them in the same direction, where they cannot create their ideas. Therefore their creativity can get restricted to a certain area, compared to working alone without any impact. Because during the brainstorming session, no one is allowed to criticize or reward the ideas, there is an evaluation session afterward. Discussion and an analysis of gathered ideas should take place and the solution can also be crafted in conventional ways now, with the help of the gathered ideas. A great tool for brainstorming is mind mapping, which helps to create a diagram out of the ideas and their relationships.

2.1.1 Mind Mapping

Mind mapping is a diagram often used for brainstorming, planning and many other use cases developed from Tony and Barry Buzan. According to Buzan [3]:

> The Mind Map is an expression of Radiant thinking and is, therefore, a natural function of the human mind. It is a powerful graphic technique which provides a universal key to unlock the potential of the brain. The Mind Map can be applied to every aspect of life, where improved learning and clearer thinking will enhance human performance. The Mind Map has four essential characteristics:
>
> - The subject of attention is crystallized in a central image.
> - The main themes of the subject radiate from the central image as branches.
> - Branches comprise a key image or key word printed on an associated line. Topics of lesser importance are also represented as branches attached to higher-level branches.
> - The branches form a connected nodal structure.

A mind map can be used as a diagram for capturing ideas on a sheet of paper and is useful because it allows to use both sides of the brain at once. The left one is used for analytical and sequential thinking and language thinking and the right side is for intuitive thinking, spatial understanding, imagination, color and rhythm [2, 17]. One could deduce that the right side is creative, likes to think freely and out of the box. In comparison to lists on a lined notepad, mind maps are allowing more creative thought processes and therefore favors the solution of problems. With mind maps, it is also easier to see the bigger picture, focus on key issues and clarify thinking. In addition, the thinking process can be tracked very easily. Therefore everyone knows how the ideas came up. Figure 2.1 shows a mind map about creativity with multiple instructions on it how a mind map should look like.

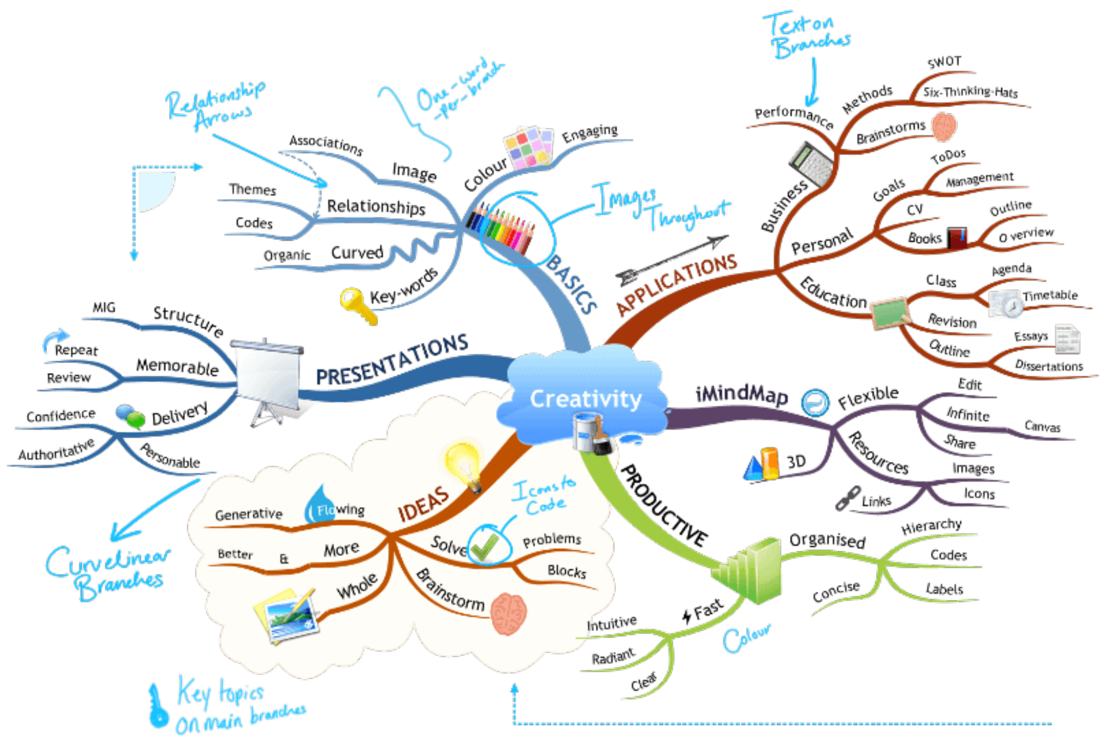


Figure 2.1: Mind map about creativity with instructions for mind mapping [26].

2.2 Gamestorming

Gamestorming is an alternative technique to explore and look into challenges for generating a business result, such as making a decision, solving a problem, or developing a strategy. As described in [11, S. 5]:

> It can be used instead of a traditional meeting, where the games are typically designed to ask questions, visualize ideas, combine ideas, experiment and stimulate creativity.

The human mind is not working in a straight line [30] like it is needed for a process, which is a chain of cause and effect [11, S. 6]. In figure 2.2, it can be seen what radiant thinking looks like. Mind Maps, which are often used in the context of brainstorming, are used for developing a concept and is an example for Radiant Thinking [4]. There is one central idea, which expands in different directions and form sub-topics [9] with more detail.

There are several gamestorming techniques which can be used in combination with brainstorming. On the next pages, two of these techniques will be described in detail to show what gamestorming is about and how it can help to make brainstorming even better.

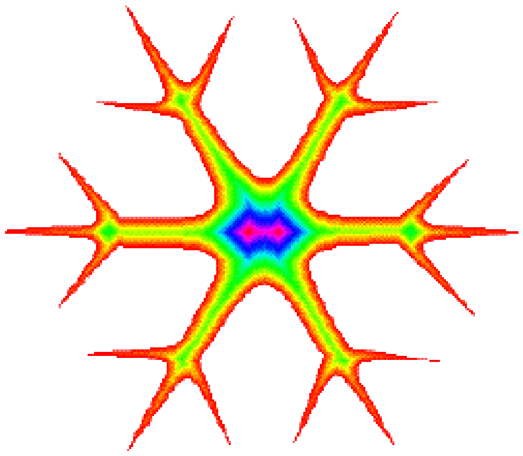


Figure 2.2: Radiant Thinking visualization [30].

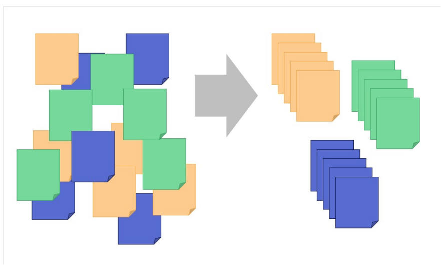


Figure 2.3: A visualization about before and after affinity mapping [34]

2.2.1 Affinity Map

In a brainstorming session, a group generates as many ideas as possible around a topic within a limited time frame. After the time ended, there is a high quantity of mixed information on the board. The question is now, how to get a meaning out of a large amount of data. Affinity mapping is a technique that solves this issue and helps to discover embedded patterns of thinking by sorting and clustering and creating categories from all the notes created. It also points out which ideas occur more frequently by the participants and are therefore more prevalent within the group [11].

How to Play

The first thing needed is the question of what the participants should think about. As already stated in section 2.1.1, it would be great to have an image matching the question, which will be presented on a whiteboard, projector, or something similar to the whole group. The group size can accommodate up to 20 participants, which will play the game for a maximum of 1.5 hours [11]. After preparation is done, the game can start, each player should write down their ideas according to the question on sticky notes and put them on a flat surface such as a whiteboard, which should be visible to everyone. With guidance from the players, the ideas should be sorted into columns or clusters based on their relationships. After sorting is done, the group should agree on names which represent the columns or clusters which were created by them. Figure 2.3 shows the start status and the result of this technique.

2.2.2 Dot-Voting

In most of the brainstorming sessions, the group has to reduce the amount of good ideas at some point to proceed further. Therefore dot voting is one of the best and most straightforward game to prioritize and converge upon an agreed solution [11] and continue to focus only on the best or most popular ideas of the participants.

How To Play!

The starting point is an ongoing brainstorming session with lots of ideas or an already sorted and clustered wall of sticky notes after an affinity mapping game, as described in section 2.2.1. The dot voting can then be applied to a cluster to get the most valuable approaches out of them. This could be done multiple times with different clusters to reduce all ideas to a manageable quantity. Every participant gets 5 votes, which could be stickers or markers, then they can use them on their favorite notes or feel the most strongly about [11]. It is also possible to use multiple votes on the same idea. After voting, the ranked list will be the new subject of discussion and decision making. Lower voted ideas should not be included in further discussions. Only a short reflection can be made to be sure nothing dropped out without any reason. Dot voting can be used in many cases outside the brainstorming field, for example, to get the most wanted next feature in the development or to choose among strategies and concepts [11] or vote on the next dish to be cooked as it can bee seen in figure 2.4.

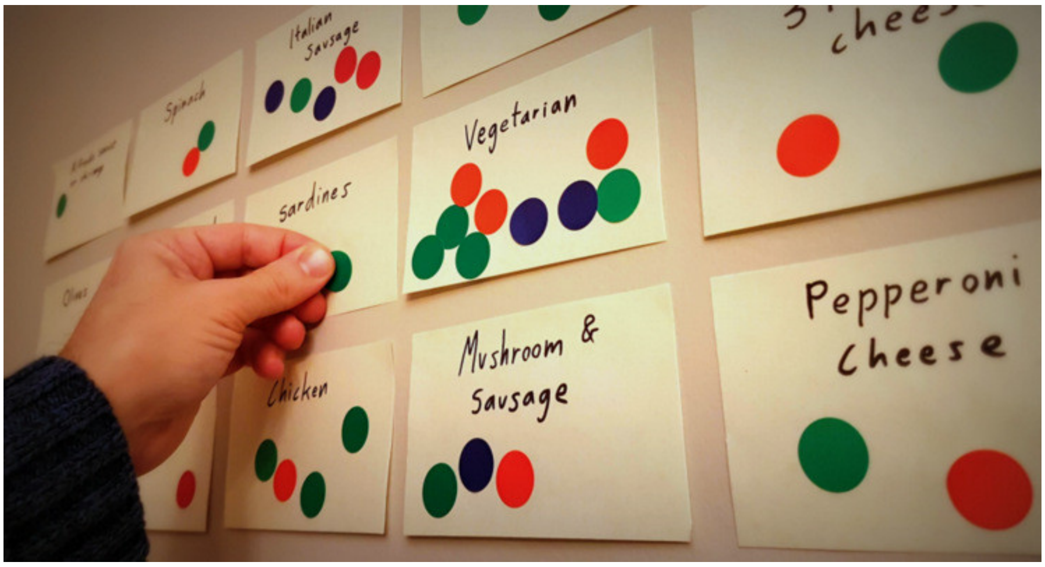


Figure 2.4: Dot-Voting on the next dish to be cooked [32].

2.3 Single Page Application

Single page applications, also shortly called as SPAs are web applications which consists, as the name says, only of one page. The content of the page is organized in components, which can be exchanged and updated dynamically without a page reload [14]. Therefore all the sources of the page, such as HTML, CSS and javascript have to be downloaded at once, at the initial load of the SPA. This behavior is increasing the initial load time of the SPA.

The load time of web applications is very important for users and long loading times result in losing users, even before the application is fully shown to them [8]. Around 50% of page visitors expect a page to load in 2 seconds and are likely to abandon the page when it takes more than 3s to load [31]. That means that a fast site can be associated with a good user experience [25]. One of the major problems slowing down mobile web applications is the file size [31]. Therefore a SPA, which loads everything at once, makes it even worse. Figure 2.5 shows a comparison between a normal website and a SPA. A solution for this problem is code-splitting, which will be explained in the next section.

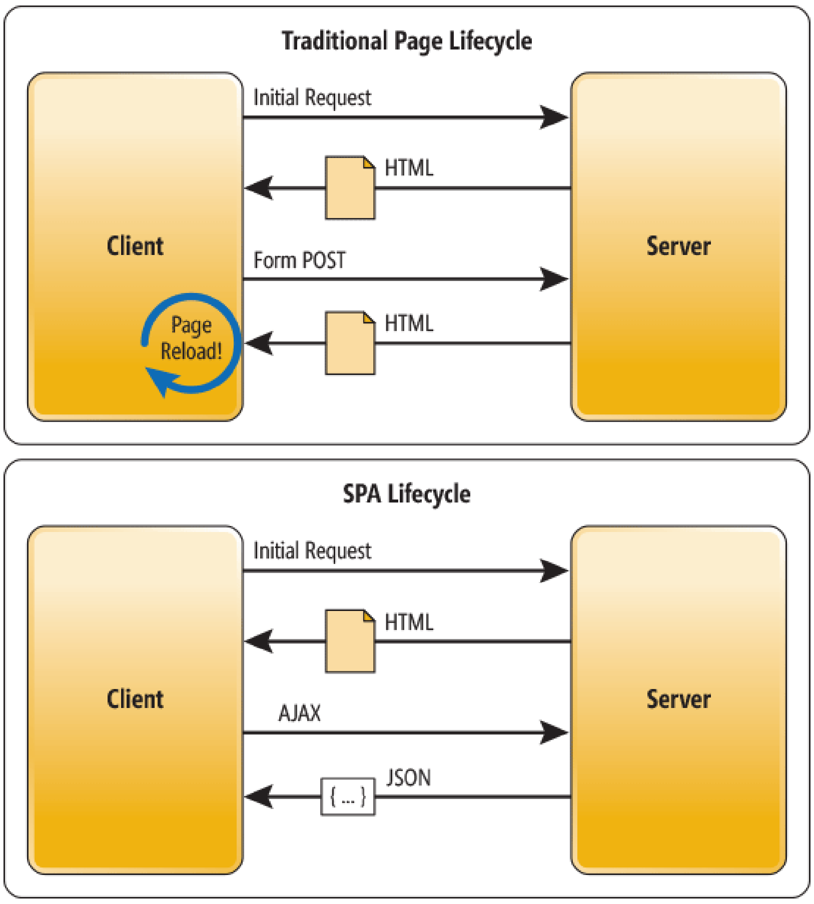


Figure 2.5: Lifecycle comparison between a traditional website and a single page application [42].

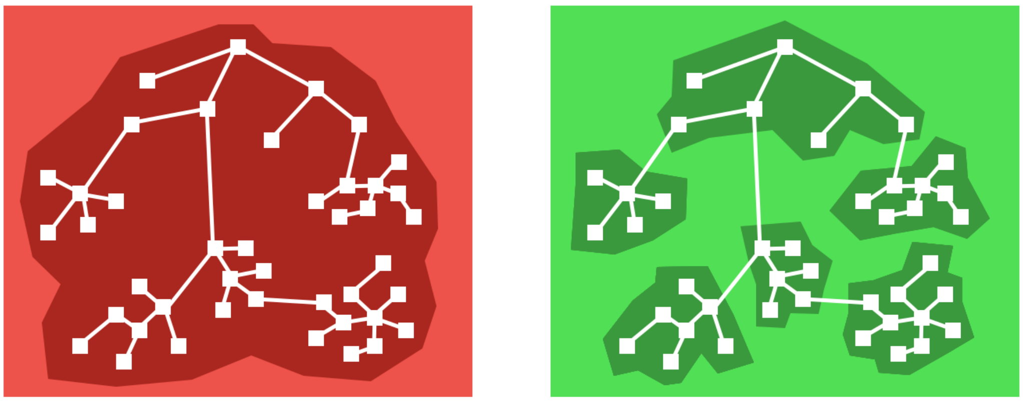


Figure 2.6: Before and after code-splitting [22].

2.3.1 Code-Splitting

A modular environment, where multiple plugins can be added to a single page application, will result in an even more increased bundle size for the client. The bundle has to be downloaded first by the browser before the website can be rendered to the screen and therefore affects the page load time for the user. Code splitting is the solution to this problem. It allows the browser to load the source code in smaller parts [8], which can be seen in figure 2.6. The browser is only loading the chunks, which are needed to show the page, which is currently requested. When the user is going to the next page, other parts of the web page are downloaded. This results in an improved page load time because only the essential needed parts are downloaded [33]. There are three main ways of doing code splitting [33]:

- *Vendor splitting*: In this form of code splitting the source code of all the 3rd party libraries is extracted and shipped in addition to the application source code. This can also lead to better performance when the app cache is invalidated.
- *Entry point splitting*: Splits the source code by the app entry points. Single-page applications are using client-side routing and therefore, entry point splitting is not working that well. It is the best option for server-side routing.
- *Dynamic splitting*: Mostly the best choice for single-page applications, which use dynamic *import()* statements.

2.3.2 Route-based Splitting

Route-based splitting is similar to the entry point splitting for server-side routing, but in the case of a single page application, there is always a single entry point. From this entry point, the routing is handled on the client-side. The routes then can be split away from the main bundle, which can be seen on the left in figure 2.7 and makes sense when certain routes or features have a large bundle size and are not always used from the users. For example, in the context of Blobster, where only the moderator has to load

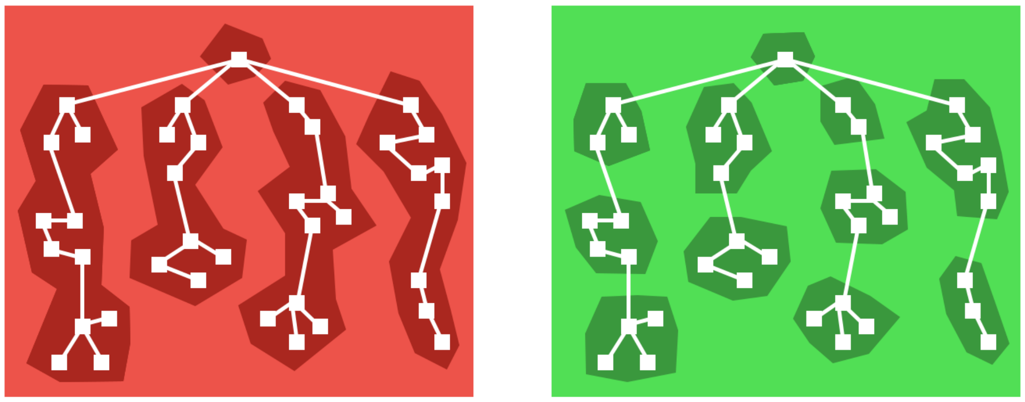


Figure 2.7: Before and after code splitting [22].

the moderator screen. The participants only need to download the mobile screen with their mobile devices, which often have a slow internet connection, resulting in a longer page load time.

2.3.3 Component-based Splitting

Component-based splitting is an addition to the route-based splitting and can be accomplished with dynamic splitting. Therefore components can load chunks of the bundle on demand. For example, when a plugin is not used in Blobster, then also the code is not needed. On the right in figure 2.7 is the result of a web application, which is using component-based splitting of the bundle.

Chapter 3

Related Work

This chapter is about the state of the art in the field of brainstorming tools and modular architectures in the web development area. In the first section, the most valuable brainstorming and e-learning tool will be presented, which are relevant for the practical part of the thesis. Afterwards, it will also explain the different development approaches and frameworks for creating a modular architecture.

3.1 Brainstorming

There is a variety of different brainstorming tools on the market. The following section will present the most popular and powerful tool.

3.1.1 Mural

Mural is an online tool where multiple participants can work together on a whiteboard, without being at the same location. Images, notes and also drawings can be added and seen by others on the whiteboard in real-time. In figure 3.1 the mural whiteboard can be seen with a lot of sticky notes on it. Additionally, it is possible to create mind maps with mural. The web application can be used on nearly every device and there is also a mobile application for smartphones and tablets. On tablets, user can draw with the help of a pen directly on the online whiteboard and show their ideas to other participants.

3.2 E-Learning

3.2.1 Moodle

Moodle is an e-learning system, used by many institutes. It is possible for teachers to create courses where the students can register. Every course has a syllabus and a time schedule with the corresponding slides, which were presented on this date. Additionally, there are exercises with an upload form and more features like a small quiz.

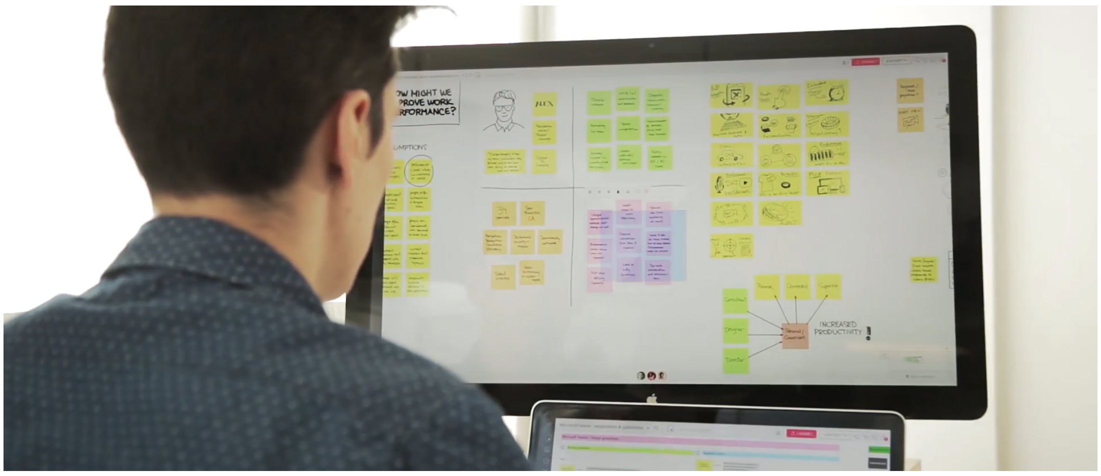


Figure 3.1: Mural whiteboard with many sticky notes on it [38].

3.3 Modular Architecture

A modular architecture in the software field is simply saying the splitting of a complex problem into smaller manageable modules [40], or as Knoernschild would say [15]:

> A software module is a deployable, manageable, natively reusable, composable, stateless unit of software that provides a concise interface to consumers.

3.3.1 Nylas

Nylas was one of the key players in supporting Plugins in React before they abandoned their *Nylas Mail* Client. However, their approaches are still a good starting point.

React is a good foundation to start supporting plugins because components can have an isolated behavior and rendering and can be self-contained [37]. But there are three main issues [37]:

- *Plugin components need access to the state* and database.
- *Plugin components need to appear in the application* and other components will not know what they have to render.
- *Plugin components may throw exceptions* and crash the whole application.

The first issue is nowadays pretty simple, as Redux is now one of the most used state containers and solves the issue. The key for the second issue is to think different, normally components are rendered directly in the *render()* method of react, but it is also possible to create an array of them and store them in Redux and dynamically inject more e.g., button components to a field as it can be seen here:

```
<div>{
    injectedButtons.map(function(button){
        return <button draft={this.props.api}/>;
    });
}</div>
```

Therefore it is also possible to pass some API calls over the props to the injected components. But they have to be clearly documented for other developers [37].

3.3.2 React-Slot-Fill

React-Slot-Fill is another approach where places in the application, which can be used to inject components are called slots. These slots can be filled with other components. The following code snipped shows the principle [23]:

```
import { Slot, Fill } from 'react-slot-fill';

const toolBarLink = (props) =>
  <div>
    <Slot name="Toolbar.Link" />
  </div>

export default Toolbar;

Toolbar.Link = (props) =>
  <Fill name="Toolbar.Link">
    <button>{ props.url }</button>
  </Fill>
```

3.4 Code-Splitting

3.4.1 React Loadable

React Loadable is a small library and facilitates the dynamic imports for the component-based splitting approach [22]. Therefore it provides a Higher Order Component, which takes care of the imports and is also handling loading and error states. Hereafter a sample dynamic import is shown.

```
Loadable({
  loader: () => import('./components/Bar'),
  loading: Loading,
  delay: 300, // 0.3 seconds
});
```

The specified delay is the waiting time before the loading screen is presented to the user. Depending on the size of the chunk and the internet connection, the loading screen is not shown when the download is faster than the specified time in milliseconds.

# Chapter 4

Concept

This chapter is dedicated to the concept and system design of the prototype, which was part of the master project. In addition, a system and context analysis was performed and the corresponding use cases and requirements were defined.

4.1 Initial Goal

The role of the thesis project is not to implement a complete brainstorming tool, a small prototype for a brainstorming tool was already done before and the master thesis project will build upon this project, called *Blobster*. Such brainstorming applications allow to contribute ideas to the online whiteboard and collaboratively develop a structure for their ideas. Although such systems typically facilitate the input of a reasonably large number of users, they focus primarily on idea generation based on a rather fixed interaction paradigm. However, various applications could utilize the data generated by such brainstorming activities. Most of these applications focus on mind mapping without the use of some gamestorming techniques for creating a solution, which is one of the main topics in the thesis. Therefore a modular data framework is needed for reusing the data in various playful activities such as a voting, quizzes and other games.

4.2 System Requirements

Requirements have been defined for the master thesis project to show the scope and what needs to be done to realize the project. In the following listing, the requirements are introduced and give an overview of all the changes and features which need to be implemented.

- Revise of the *Blobster* project as well as stability improvements, bug fixes and the rework of the clustering mechanism.
- Create a plugin system.
- Integrate the plugin system into *Blobster*.
- Create a development environment for plugins.
- Develop a sample plugin.
- Plugins should be able to add and change data from the session.

- Communication between moderator and participant plugin view.
- Possibility to upload the plugin source code.
- Possibility to add a plugin, also called template, to the session and use it in a *Blobster* session.

4.2.1 Blobster

Blobster is a small web brainstorming tool where multiple people can brainstorm together over the internet. The improvement of the *Blobster* project is an essential part of the master thesis because all the other requirements build upon this web application. Therefore, the system must be so flawless and stable that other requirements are not affected or blocked.

4.2.2 Plugin-System

The most significant part of the master thesis project is the plugin-system, which will be integrated into the *Blobster* web application. Third-party developers should be able to create a Plugin, or in the context of *Blobster*, called template. These templates are added to the brainstorming session and can be used from users to evolve their gathered ideas and data to something new, add aspects to the existing data or decide upon one solution with voting. There are no limits, with the plugin system, it should be possible to do nearly everything with the system. It could also be used in schools for teaching and checking if the students are collaborating in the lessons with a small quiz afterward about the transcript created in *Blobster*.

4.2.3 Sample Plugin

The sample plugin will be a combination of a voting and a quiz such as *Kahoot*[1]. There will be a question or topic and the participants can vote or select one of the four different possibilities. It can also be used in the beginning, where the moderator wants to check the knowledge base of the group.

4.2.4 Development Environment

The developers who can create Templates need an environment where they can test and develop their projects with a development version of *Blobster*.

4.2.5 Data Architecture

In most cases, the developed templates also need to save data in the database of *Blobster*. Therefore an interface for the communication with the database has to be developed, which can be used from the templates to save and manipulate data to or from the session.

[1] https://kahoot.com

4.3 Use-Case

The Diagram in figure 4.1 illustrates all the use cases of the *Blobster* web application. It consists of 4 Actors who can interact with the system.

4.3.1 Moderator

The moderator is the actor with the most use-cases. He can create a brainstorming session and invite participants to it. The created ideas from the participants can also be changed with the edit ideas use-case. When enough ideas are gathered, a template can be added to the session. Therefore the template list is shown, where the moderator can select a template, which is then added to the session. Afterwards, the template can be started. When everything is done, the session can be closed from the moderator.

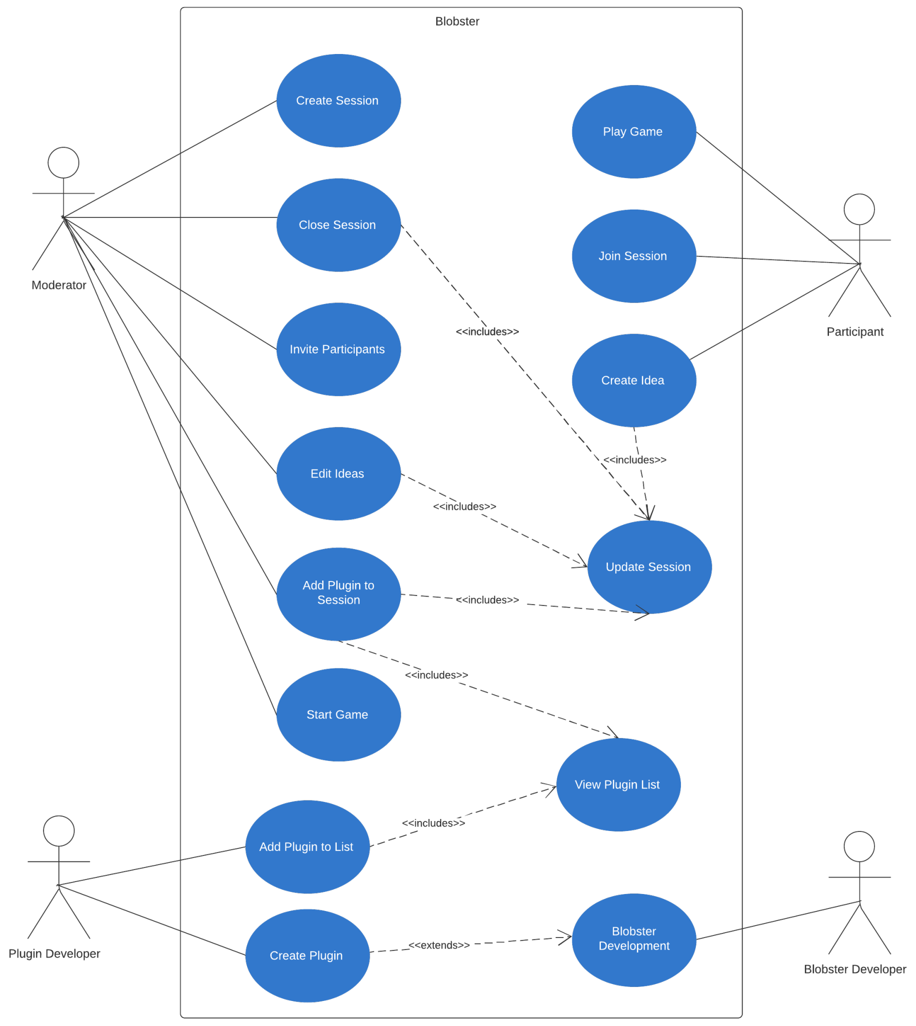


Figure 4.1: Use-Case diagram of the *Blobser* Project.

4.3.2 Participant

At first, a participant has to join a session, which is created by a moderator. After the joining process, he can spawn new ideas and send them to the session. When the moderator decides to start a game, he has to attend an play the game with all other participants.

4.3.3 Blobster Developer

The *Blobster* developer, which is also the author of the thesis, is the actor who will work on the brainstorming tool. He is responsible for adding additional features and for bug fixes in the brainstorming platform.

4.3.4 Plugin Developer

As a plugin developer, templates can be created and added to the plugin list. For developing, he needs the *Blobster* environment.

4.4 System Design

This section describes the main parts of the project in a more detailed way and should give an overview of the whole project as it can be seen in figure Figure 4.2.

4.4.1 Plugin-System

The plugin-system is one of the main parts of the thesis project, because of that, a long research was done to define all the needed functionality and requirements on the system. At the time of development, no solution could have been easily reused in this project, such as a *NPM*[2] package. In addition, there are not many documented approaches out there for *ReactJS* in the world wide web, how someone could create a plugin architecture, or in the case of *Blobster* a Template System. Therefore a concept for a plugin system was developed, which should be as general and straightforward as possible so that it could be easily used from other developers in different projects with other frameworks as *ReactJS*. Therefore it should be exported as a *NPM Module* in the future. For creating such a system, these points were considered as the way to go:

1. Plugins should be separated projects, so the source code is not in the same *Git* repository as *Blobster* or any other project where it is used.
2. Vendor libraries should be reused from the main application. Therefore the bundle should exclude them. This will decrease the bundle size and increase usability because of the lower loading time as nothing is loaded multiple times. Another benefit is that there is no version mismatch between the main project and the plugin projects. All of them are using the same library as provided from the main source.

[2]NPM - Node Package Manager is a registry to share open-source software packages. (https://www.npmjs.com)

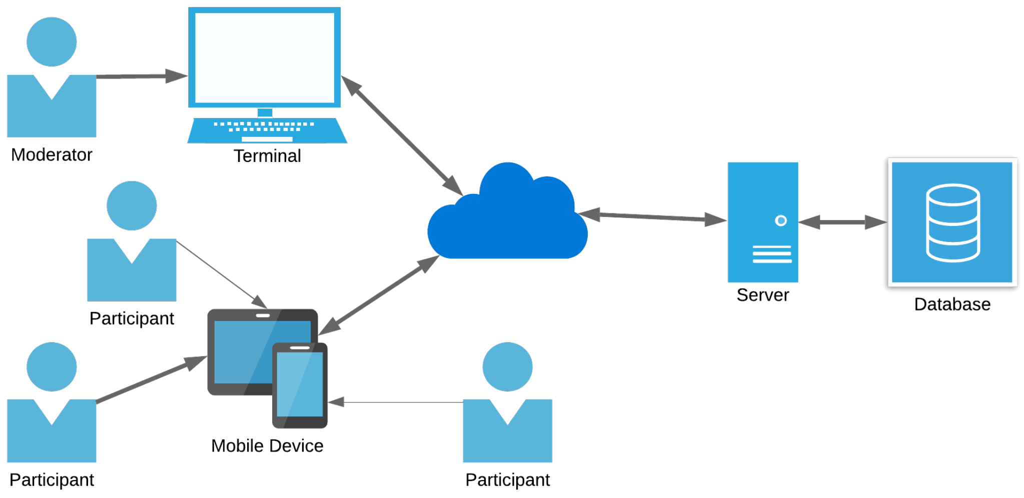


Figure 4.2: Big picture of the whole project.

3. The plugin system has to provide a register function for the loaded plugins, where they will register when they are loaded completely. The window object is the best option to be reusable in other projects too.
4. The control is inverted. Therefore the plugin has to call this function, to register in the plugin system. As a parameter, the plugin will pass its view and additional data like IDs or functions which are passed to the project using the plugin system.
5. For the case a project is using some state management system, such as *Redux*, there is also the option to pass a function to the plugin-system, which is called when a plugin registered in the system. It is also possible to use the plugin-system as a cache. All registered plugins are cached and can be queried from the system.
6. The main parameter the system needs is a list of URLs, where the bundled plugins are saved. The system will then load them and dynamically inject the scripts to the site and call the callback described in Point 5 and cache the components.

In figure 4.3, the flow of the plugin-system can be seen. First, it must query the list of plugins that are needed. In most cases, all the plugin data is saved in a database or something similar. With all the information needed, the system will spawn the plugins in the context of the system. When the plugins are fully loaded, they will call the system that they are ready to use. The system will then return a list of all loaded plugins to the web application, which requested the plugins.

4.4.2 Plugin Environment

Developing a plugin should be as easy as possible. Therefore it is essential to have a pleasant development environment. A third-party developer has to be able to test and develop their plugins without much effort setting up the project and build pipeline. Figure 4.4 shows a sequence diagram of how the procedure should look. Developers can

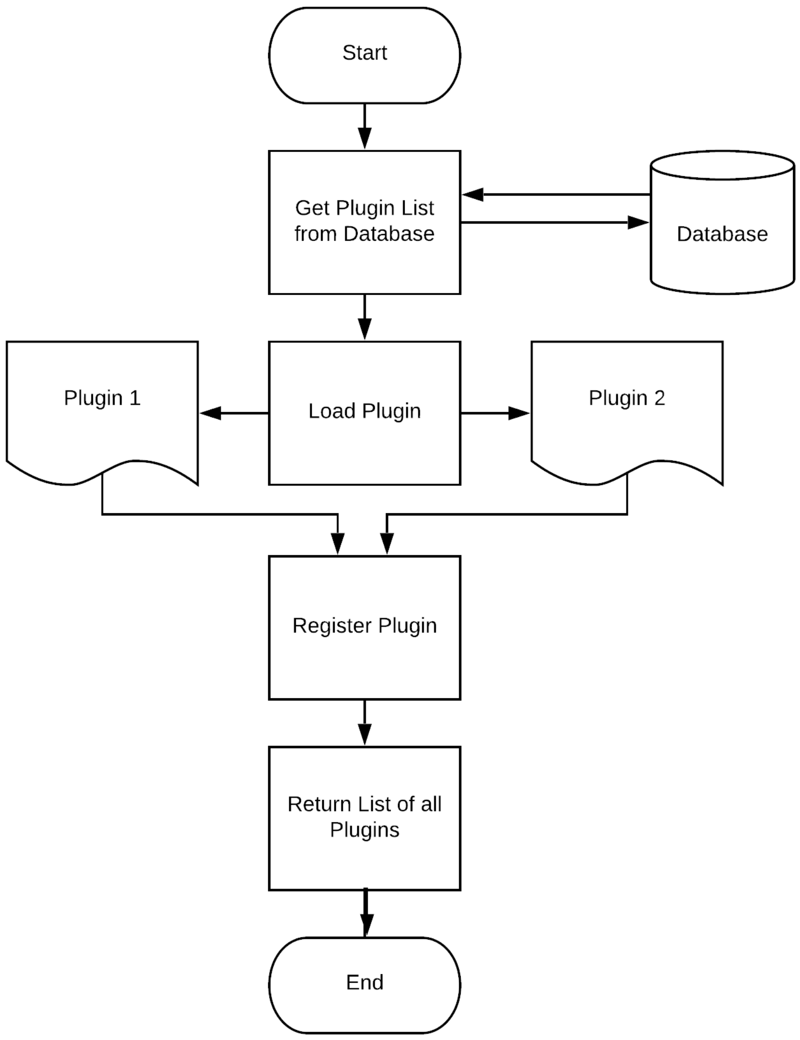


Figure 4.3: Flow chart of the registering from the plugins.

clone the project for developing a plugin from source control. In this project, everything should be set up, including a development version of the *Blobster* project. The only thing they have to do is install the packages, start the environment and start programming. When they save in their favored coding ide, the code should be bundled and updated in the *Blobster* web application. Testing is done after the automatic refresh ob the website.

When everything is correct and the plugin is finished, a release bundle can be created and is returned to the developer. This bundle has to be uploaded to the plugin section of the *Blobster* web application and will be published and usable after the input of a name and a description and the upload of a corresponding image.

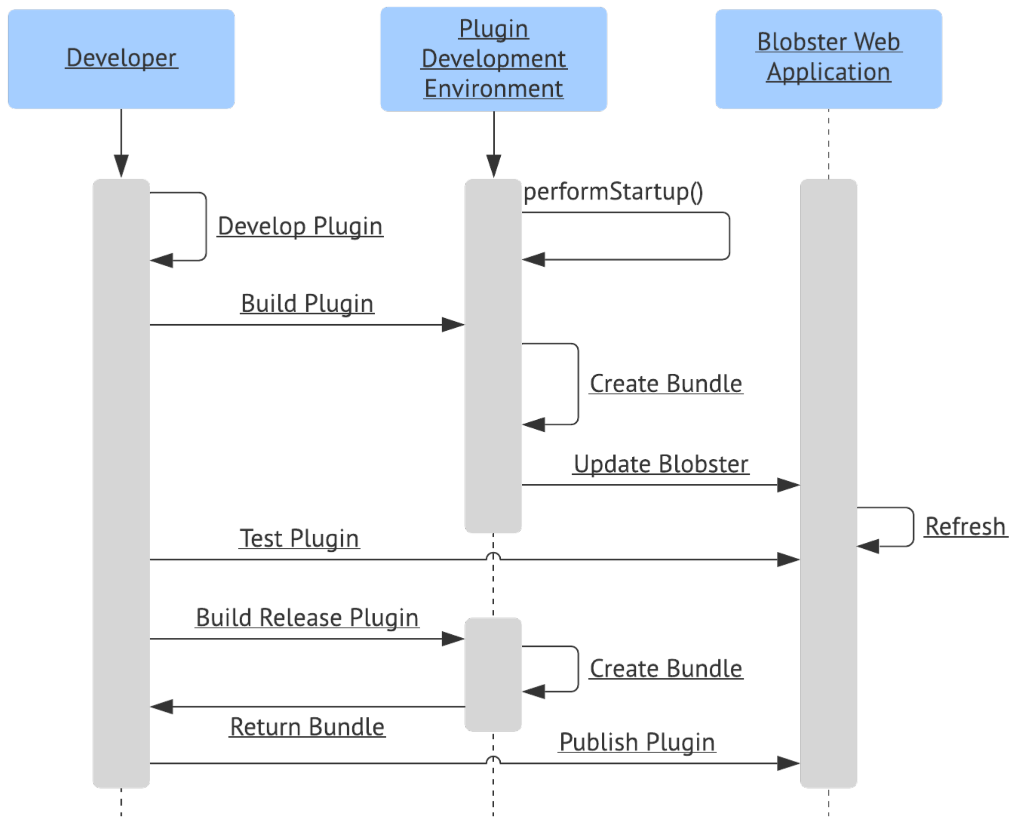


Figure 4.4: Sequence diagram of the plugin development.

4.4.3 Data Architecture

To give plugins access to the database, they will be provided with API methods for manipulating the data they are allowed to change.

Reusable Data

Saving data is not the main problem, but the saved data should also be reusable later on in other templates, for example. Therefore, a particular structure is needed which can be read by the main project and other templates. This structure has to be defined and documented.

4.5 Template-System Mockup

Figure 4.5 shows a mockup of the template system in the *Blobster* project. On the left side, the moderator has a scroll view where all the templates added to the session are

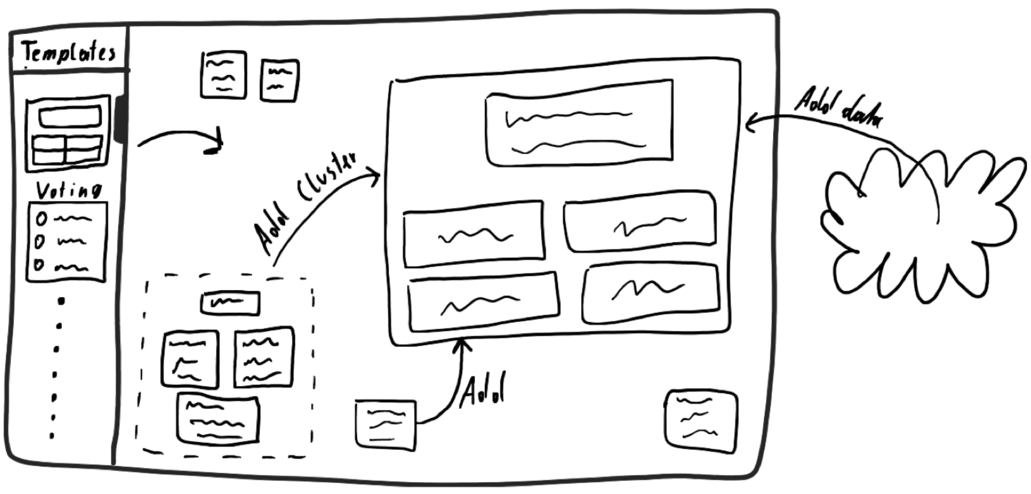


Figure 4.5: Sketch of the template system.

listed. With drag and drop, a template can be added to the board. In this case, a quiz template with one question and four answers is added. The fields can be filled either with dragging a cluster or idea onto the template or writing the content directly into the fields.

Chapter 5

Technical Implementation

5.1 System Architecture

As seen in figure 5.1, the frontend is built in React.js. With the help of Redux and redux-observables. All the inputs are mapped to actions, which are going through the reducer, the epic middleware and create a new state or fetch data from the rest API. The features like sending a post-it to the moderator screen or the voting is sent over the socket.IO connection between the moderator and the participants. The backend is built in Node.js (using Express). In combination with a Rest API and Mongoose, the backend stores data in the MongoDB database.

5.2 Frontend

5.2.1 ReactJs

React, or also called *ReactJs*, at which Js stands for Javascript, is an open source JavaScript library for building user interfaces developed by *Facebook* in 2003 [10]. *React*

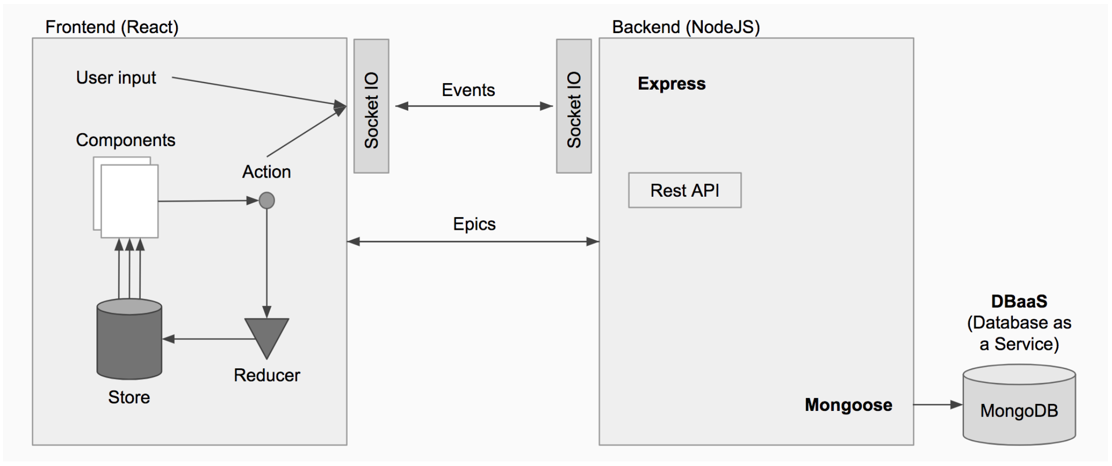


Figure 5.1: Basic concept of the application procedure.

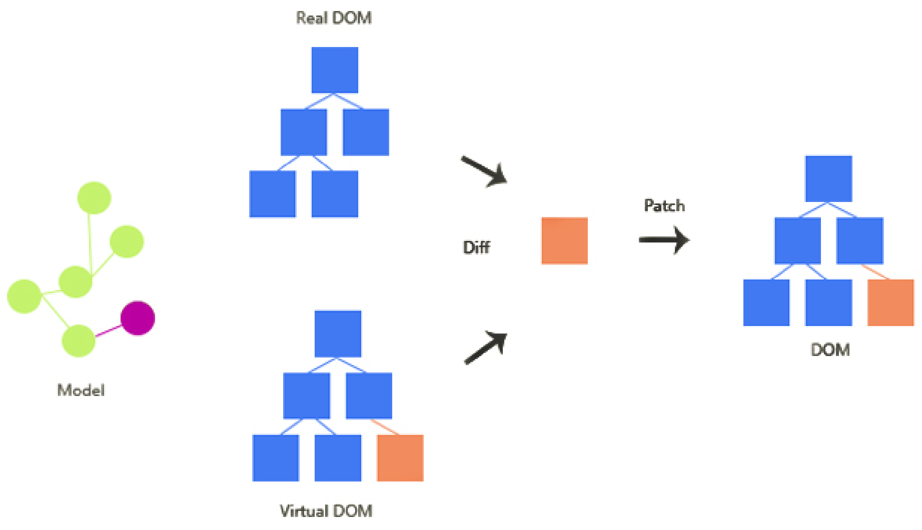


Figure 5.2: Visualization of how the real DOM is updated [35].

is basically a declarative wrapper for the imperative DOM[1] API [10]:

- Imperative describes how something has to be done.
- Declarative describes what has to be done without telling how.

React is only used for the view. Therefore the view is broken down into smaller parts, which are called components. Each component has props passed by the parent and manage their own state. Changing either the state or the props will result in re-rendering this single component and not the whole view [28]. All the DOM manipulation is done by *React* itself because these operations are expensive and imperative. Therefore *React* provides a virtual DOM, which the developers use to render. The virtual DOM and the real DOM are compared and *React* conducts the minimum number of DOM operations required to patch the real DOM to the new state. [19].

5.2.2 Redux

Redux[2] is a small library for JavaScript, which is used to maintain the state of a web applications in an predictable state container. Thereore *Redux* is using these basic principles [29]:

- Single source of truth, a project has only one store which handles the complete state of the application.
- State is read-only. It can only be changed when dispatching a new action.
- Changes to the store are made with pure functions, at which "pure" means, the same input will result in the same output every time.

[1]DOM, Document Object Model is a representation of a website as nodes and objects. (https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction)

[2]https://redux.js.org

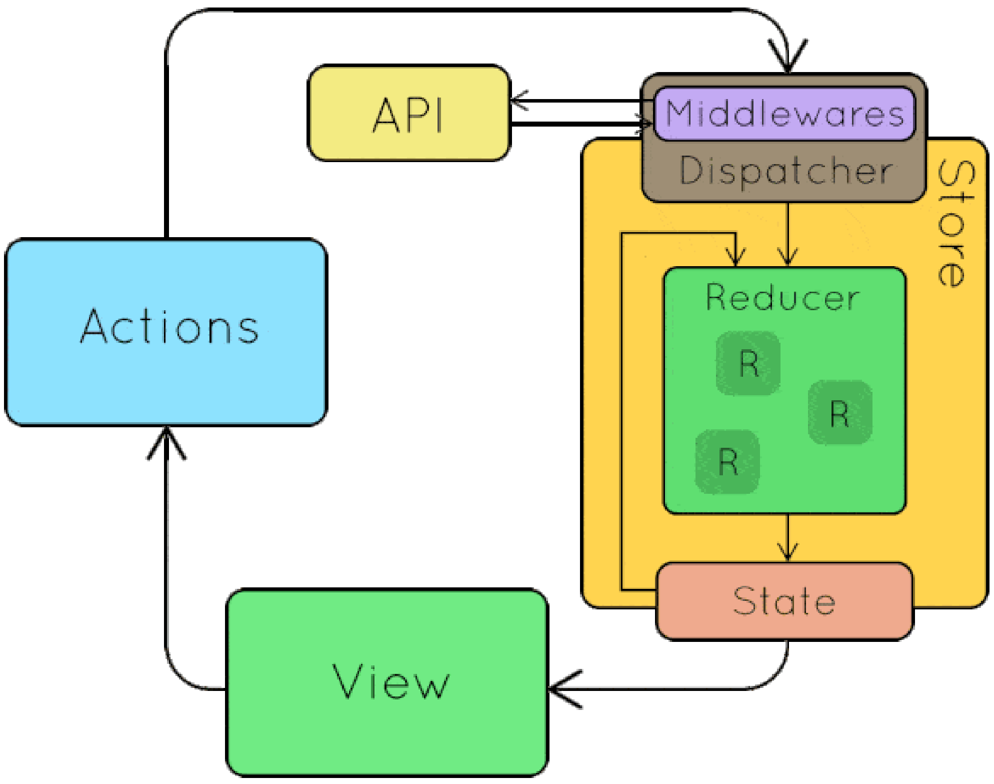


Figure 5.3: Illustration of the redux flow [41].

In figure 5.3 the redux data-, action- and state flow is illustrated. The view is rendered by *ReactJs* events such as a button click will be transformed to actions. An action is simply a JSON Object with a type and a payload, as it can be seen in the following snippet.

```
{
    type: "INCREASE_COUNTER"
    payload: 1
}
```

This action is then sent to the store. At the store, the action is first passed through all the middle wares, which could produce side effects and query an Rest API for example. The reducer is then generating a new state with the old stat and the passed action. The state will then be returned back to the view Changes in the state will result in a re render of the view or component. Side effects, such as a API call can also spawn new actions and change the state.

5.2.3 React-DnD

React-DnD[3] is a library for handling all the drag and drop features on the brainstorming board in the *Blobster* project.

5.2.4 Code-Splitt

For splitting the source code *bundle.js* into multiple chunks, *React-Loadable* was used. *React-Loadable* is a wrapper around components with dynamic imports. *Webpack*[4] is then splitting the bundle into chunks, which are loaded when the application needs them. Therefore, the initial page load time is decreased. The snippet below is showing one of the loadable components, which will be outsourced into a chunk.

```
const loadablePublicScreen = Loadable({
  loader: () => import('./admin/PublicScreen'),
  loading: () => <div>Loading Screen ... </div>
});
```

In this example, the public screen from the moderator is only loaded when the client is a moderator. Participants will not load this chunk.

5.2.5 Plugin-System

The template or plugin system is an external module, developed in his own project. The system is bundled and uploaded to an S3 Bucket and loaded into the application with a script tag.

```
export default class PluginSystem {
  constructor(pluginUrlList, pluginUpdateCb) {
    console.log('New Plugin System')
    this.pluginUrlList = pluginUrlList
    this.pluginUpdateCb = pluginUpdateCb
    this.pluginCache = []
    window.registerPlugin = this.registerPlugin
    pluginUrlList.forEach((pluginUrl) => {
      this.dynamicallyInjectScript(pluginUrl)
    })
  }
}
registerPlugin = (plugin) => {
    this.pluginCache.push(plugin)
    this.pluginUpdateCb(this.pluginCache)
}
```

The system is passing the `registerPlugin()` function to the `window` object. Afterward, all the requested bundles are downloaded and injected into the web application.

When the plugin is loaded into the application, it will call the `registerPlugin()` function, which can be seen in the following code snippet.

```
let registerPlugin = () => {
  if (window.registerPlugin) {
    window.registerPlugin({
```

[3] https://github.com/react-dnd/react-dnd

[4] Webpack is a tool, which creates a bundle and chunks out of the source code. (https://webpack.js.org)

```
      pluginId: PLUGIN_ID,
      pluginAdmin: connect(
        state => ({
          topic: state.gui.topic,
          session: state.session.session
        }),
        { goToFullScreenPlugin }
      )(ExamplePlugin),
      pluginMember: ExampleMember,
      reducer: [hoodmood],
      epics: [hoodmoodEpic]
    })
  } else {
    setTimeout(() => registerPlugin(), 100)
  }
}
registerPlugin()
```

The template is trying to call the `window.registerPlugin()` function, if it is not set yet, it will wait 100ms and try again. As soon as the function is ready, the template object is passed to the plugin system. This object contains a plugin-id, which is used to distinguish between the different templates. The template provides a moderator component and a participant view, which are used from *Blobster* to present the template on the board. The figure 5.4 sequence diagram shows the whole process. There is a Plugin Overview page, in which Moderators can select plugins for their sessions. When a developer has registered his plugin on the screen shown in figure 5.5, it is shown in the Plugin List, like it can be seen in this figure 5.6. When the moderator adds a Plugin to the session, it is added to the template system and a *Socket.IO* event is fired to inform the participated clients to add the plugin, too.

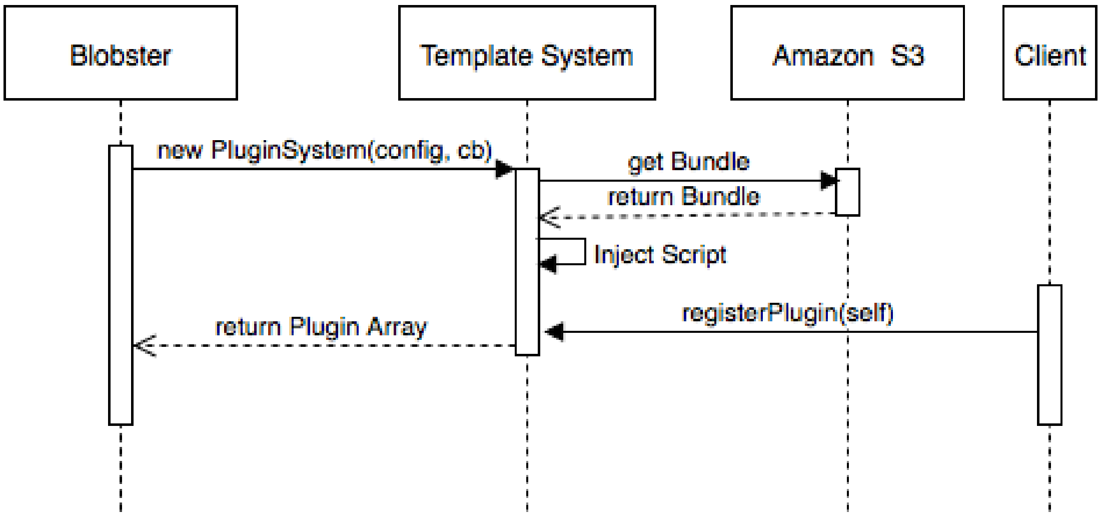


Figure 5.4: Sequence diagram of the template loading process

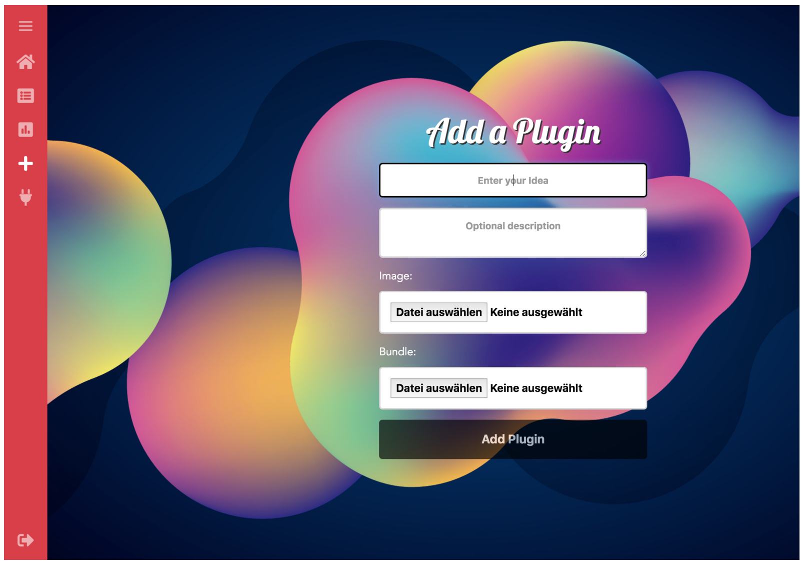


Figure 5.5: Screen for adding templates to *Blobster*.

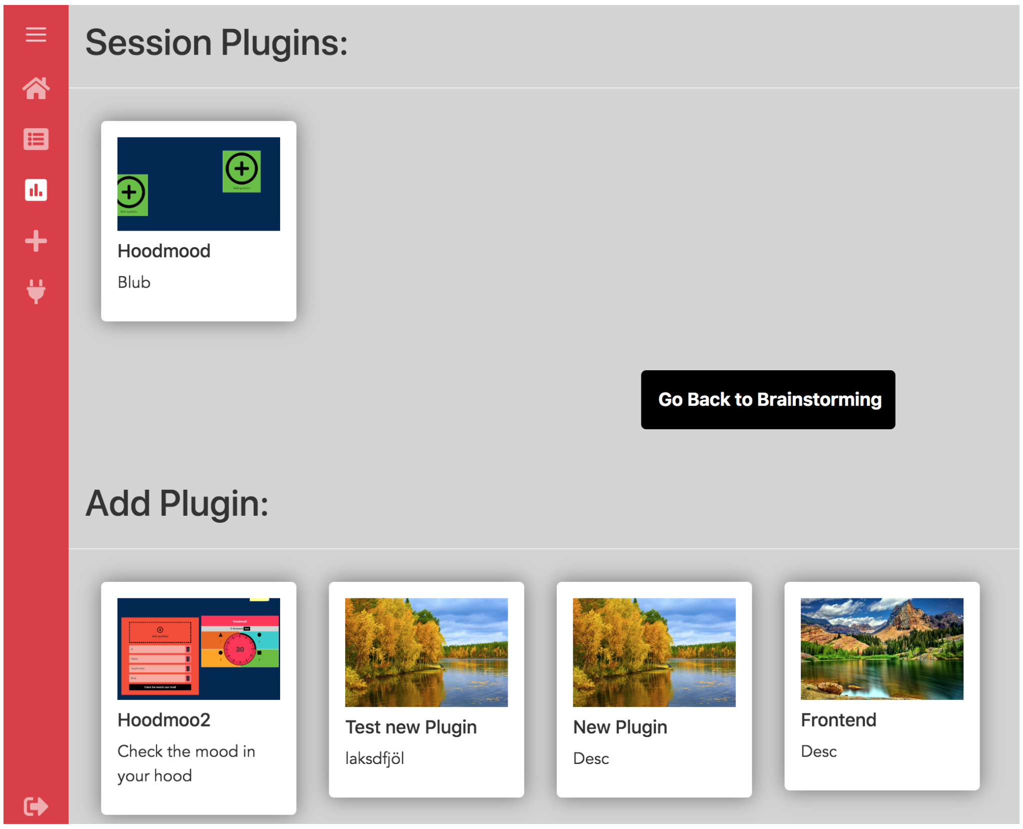


Figure 5.6: Template List with one template added to the session.

Redux

In order to use *Redux*, the reducer has to be injected into the store, the template system will take care of this. The store configuration has to be adapted to this approach [39]:

```
function configureStore(initialState) {
  const store = createStore(createReducer(), initialState)

  // Keep track of the registered template reducers
  store.templateReducer = {}

  // Create an inject template reducer function
  // This function adds the template reducer which is loaded async
  // and creates a new combined reducer
  store.injectReducer = (key, templateReducer) => {
    store.templateReducer[key] = templateReducer
    store.replaceReducer(createReducer(store.templateReducer))
  }
  return store
}

export default configureStore(composedEnhancers);
```

The template system then calls the `injectReducer()` function with the passed reducer. After this has been done, the state from the store can be used in the template and also actions can be dispatched.

Epics

For the epics, it is pretty much the same as with the reducer in section 5.2.5. Only the injection method is different [20]:

```
export const epic$ = new BehaviorSubject(combineEpics(
  userEpic,
  apiEpic,
  sessionEpic,
  gameEpic,
  guiEpic,
  pluginEpic
));
export const rootEpic = (action$, state$) => epic$.mergeMap(epic => epic(action$,
    state$));
plugin.epics.forEach(epic => epic\$.next(epic))
```

5.2.6 Integration into Blobster

In this section, the integration of the plugin system into the *Blobster* project is described. The first step is to load the system itself, which is also an independent project and is bundled and uploaded to a web-server hosting the file. The system is loaded via a script tag in the *HTML*.

```
<script src="https://s3.eu-central-1.amazonaws.com/plugins.blobster.it/
    pluginSystemNew.js"></script>
```

For creating the system, the following code is used:

```
createPluginSystem(session) {
  this.pluginSystem = new window.PluginSystemMeisi(session.plugins.map((it) =>
    it.bundle), (plugins) => {
    console.log('PLUGINS_CB: ', plugins);
    plugins.forEach((plugin) => {
      plugin.epics.forEach(epic => epic.next(epic))
      plugin.reducer.forEach((reducer, index) => store.injectReducer(plugins[index
    ].pluginId, reducer))
    });
    this.props.receiveComponents(plugins);
  })
}
```

The system is created with the plugin list from the session object as a parameter. The session object contains all the information of the brainstorming session, such as the ideas, cluster and added templates or also called plugins. The second parameter is an anonymous function, which is called when the plugin is loaded and registered in the system, as described in section 5.2.5. In this function, all the epics and reducer functions are added to the *Redux* store, with the help of the functions created in section 5.2.5 and 5.2.5. Afterwards, the plugin components are added to the redux store with the `receiveComponents()` function. When a plugin is added on the fly from the plugin list(Figure 5.6), the `addPluginToSession` action is called with the plugin and session id and the plugin itself.

```
this.props.addPluginToSession({pluginId: plugin._id, sessionId:
    this.state.sessionActive, plugin: plugin});

  action =>
  action
    .ofType(String(addPluginToSession))
    .pluck("payload")
    .do((x) => {
      addNewPluginOnClient(x.plugin)
    })
    .mergeMap(data =>
      api.post(
        {
          path: SESSION_PATH + '/addPluginToSession',
          body: data
        },
        obs => obs.pluck("response").do((x)=> console.log(x)).map(receiveSession)
      )
    ),
```

One of the epic is listening for this action and performs multiple side effects for this action:

1. Send a socket event to the client. Therefore, the method `addnewPluginOnClient` is called with the plugin and a *Socket.io* event is emitted with the same name and the plugin data. The backend is then broadcasting an `addNewPlugin` event to all the connected participants:

```
    socket.on('addNewPluginOnClient', function(plugin) {
        socket.broadcast.to(socket.sessionRoom).emit('addNewPlugin', plugin);
    });

    socket.on("addNewPlugin", data => {
        if(this.pluginSystem) {
            this.pluginSystem.addPlugin(data.bundle)
        }
    })

```

The clients receiving this broadcast will then add the plugin to their plugin-system.

2. Create a post request and add the plugin to the session object saved in the database with the route `addPluginToSession`.

Draggable Extension

An extension is an instance of a template, because a session can have multiple instances of a template at the same time. These Extensions need to be draggable at the brainstorming board. Therefore, a draggable Extension component is created. The source code of this extension is shown in program 5.1. The drop target object defines which types of draggables can be dropped onto the extension. The render method connects everything up and renders only the children. In the case of an extension, it will render the components, which are passed from the plugin into the plugin-system.

5.3 Backend

5.3.1 Node.js

In the backend, Node.js is a Javascript interpreter and runtime system, which has made *JavaScript* an attractive option for server-side programming. It is a combination of an interpreter with an event-driven library. This makes the library particularly suitable for web applications that deal with many requests [5].

5.3.2 Express

Express is a web application framework built on top of *Node.js*. It makes it easier to handle routing and adds helpful utilities to *Node.js*, such as a middle ware approach. In addition, *Express* facilitates the rendering of dynamic views [12].

5.3.3 MongoDB

As a database, *MongoDB* is used in the *Blobster* project. *MongoDB* is a powerful, flexible and general-purpose database and is document-oriented. Therefore, the concept of columns and rows is replaced by a more flexible object, the document [6].

5.3.4 Mongoose

Mongoose is an extension library for *Node.js*, which provides a schema-based solution to model the application data in a *MongoDB* database. The main features are according to [36]:

> Built-in type casting, validation, query building, business logic hooks and more.

Program 5.1: The draggable extension component drawn onto the brainstorming board.

```
        const dropTarget = {
          drop(props, monitor) {
            const item = monitor.getItem();
            if (monitor.getItemType() === ItemTypes.IDEA ||
                monitor.getItemType() === ItemTypes.FOLDER ||
                monitor.getItemType() === ItemTypes.IDEA_PREVIEW) {
              if(window[props._id]) window[props._id](item, monitor.getItemType()
                === ItemTypes.FOLDER);
            }
            return { name: 'extensionDropTarget' }
          }
        }
        class DraggableFolder extends Component {
          static propTypes = {
            connectDragSource: PropTypes.func.isRequired,
            connectDropTarget: PropTypes.func.isRequired,
            left: PropTypes.number.isRequired,
            top: PropTypes.number.isRequired
          };
          render() {
           const { connectDragSource, connectDropTarget } = this.props
           return connectDragSource(connectDropTarget(
               <div style={getStyles(this.props)} className={'extension'}>
                    {this.props.children}
                </div>
            ))
          }
        }
        export default compose(
          DragSource(ItemTypes.EXTENSION, extensionDragSource, (connect, monitor) =>
    ({
            connectDragSource: connect.dragSource(),
            // connectDragPreview: connect.dragPreview(),
            isDragging: monitor.isDragging()
          })),
          DropTarget([ItemTypes.IDEA, ItemTypes.FOLDER, ItemTypes.IDEA_PREVIEW],
   dropTarget, (connect, monitor) => ({
            connectDropTarget: connect.dropTarget(),
            isOver: monitor.isOver(),
            canDrop: monitor.canDrop()
          }))
        )(DraggableExtension)

```

5.3.5 Database as a Service

Database as a Service (DBaaS) is a service, which provides a database without the need for buying physical hardware or installing a database software on a machine in the network. The developer only has to connect to the database over an specific address [27].

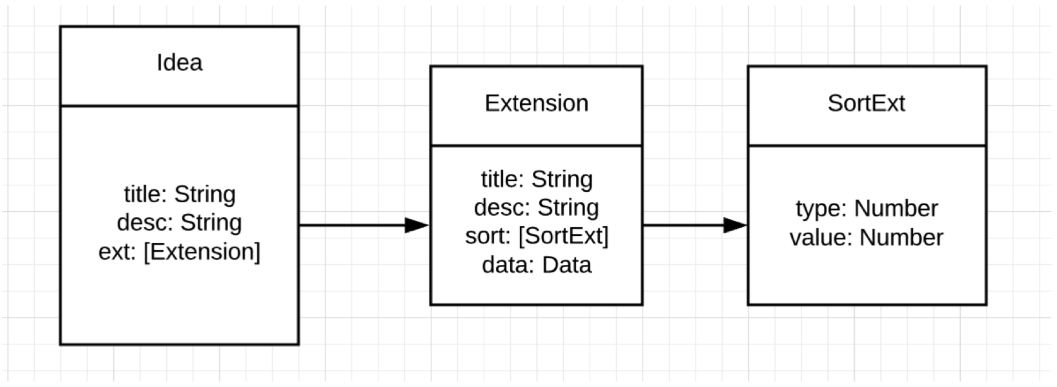


Figure 5.7: Extension Mongoose schema.

5.3.6 Routes for the Extensions and Plugins

Extension Route

Program 5.2 is showing how an extension is created and updated in the database.

Plugin Routes

Program 5.3 is showing the route for adding a template to the session object by the moderator. The second route is for the participant, which also need to fetch the plugins from their current session.

Data

Templates can also store data in the *Blobster* MongoDB database. An extension field is created as soon as a template is added to the board, which has a data field where the template can save any kind of JSON data. Therefore they only need to `dispatch` an `updateExtension` action to the store. After it is saved, the refreshed data object is instantly passed back to the template. In addition to that, there is also an option to add an extension to an idea. E.g., when an idea is dragged into the question editor, it will be used as an answer. Then there is also an extension added to the idea, which gets a sort extension after the quiz is finished, as it can be seen in figure 5.7. This sort of extension could then be used in other templates or in *Blobster* to sort or filter. Currently, this is only implemented in the backend and can not be used in the frontend. Program 5.4 is showing the database schema.

5.4 Problem with the Existing Code Base

5.4.1 Local Storage

The local storage of the browser was used for reconnects and storing session data, which was a wrong approach and in some situations, the sessions were mixed or side effects

Program 5.2: Extension route for adding and updating of the extension in the session object.

```
 router.post('/addExtensionToSession', function(req, res) {
  if(req.body.sessionId && req.body.extension) {
    Session.findOneAndUpdate(
      { _id: req.body.sessionId },
      { \$push: { extensions: req.body.extension } },
      {new: true},
      function(err, session) {
        if(err || !session) return res.status(400).send({
          "error": "400026",
          "message": "Problem adding the extension to the database"
        })
        return res.status(200).send(session)
      }
    );
  } else {
    return res.status(400).send({
      "error": "400027",
      "message": "Problem parsing JSON, probably Missing Fields",
      "required": "sessionId, extension"
    });
  }
});

router.put('/updateExtension', function(req, res) {

  if(req.body.sessionId &&
    req.body.extension) {

    Session.findOneAndUpdate(
      { _id: req.body.sessionId, 'extensions._id': req.body.extension._id },
      { \$set:  {
          'extensions.\$': req.body.extension
        } },
      {new: true},
      function(err, session) {
        if(err || !session) return res.status(400).send({
          "error": "400028",
          "message": "Problem updating the extension"
        });
        return res.status(200).send(session)
      }
    );
  } else {
    return res.status(400).send({
      "error": "400029",
      "message": "Problem parsing JSON, probably Missing Fields",
      "required": "sessionId, extension"
    });
  }
})

```

Program 5.3: Plugin route for adding fetching the plugins of a session.

```
  router.post('/addPluginToSession', function(req, res) {
  if(req.body.sessionId && req.body.pluginId) {
    Session.findOneAndUpdate(
      { _id: req.body.sessionId },
      { \$addToSet: { plugins: mongoose.Types.ObjectId(req.body.pluginId) } },
      {new: true},
      function(err, session) {
        if(err || !session) return res.status(400).send({
          "error": "400012",
          "message": "Problem adding the idea to the database"
        })
        session.populate('plugins', (err, sessionPopulated) => {
          if(err || !sessionPopulated) return res.status(400).send({
            "error": "400012",
            "message": "Problem adding the idea to the database"
          })
          return res.status(200).send(sessionPopulated)
        });
      }
    );
  } else {
    return res.status(400).send({"error": "400011"});
  }
});
router.get('/getPluginsFromGame/:gamePin', function (req, res) {
  Session.findOne({ 'pin': req.params.gamePin}, function (err, session) {
    if (err) {
      return res.status(500).send({
        "error": "5001",
        "message": "Problem finding game"
      });
    } else {
      if(session) {
        if(session.active) {
          return res.status(200).send(session.plugins);
        } else {
          return res.status(500).send({
            "error": "5002",
            "message": "Game is not open!"
          })
        }
      } else {
        return res.status(500).send({
          "error": "5002",
          "message": "Wrong Game Pin"
        })
      }
    }
  });
});

```

Program 5.4: Extension with the SortExtension field.

```
 var mongoose = require('mongoose');
var SortExtSchema = new mongoose.Schema({
  type: {
    type: String,
  },
  value: {
    type: String
  },
}, { timestamps: true});

module.exports = SortExtSchema;
var ExtensionSchema = new mongoose.Schema({
title: {
    type: String,
    required: true
  },
  pluginId: {
    type: String,
    required: true
  },
  top: {
    type: Number
  },
  left: {
    type: Number
  },
  data: {
    type: Object,
    default: {}
  },
  sortExt: [SortExt],
}, { timestamps: true});
module.exports = ExtensionSchema;

```

occurred. Therefore, all the data stored in the local storage got transferred to the query part in the URL, which is a much cleaner and manageable approach.

5.4.2 Idea position after Dragging out of Folder/Cluster

There was a problem, where the drop position was wrong calculated when dragging ideas from one Drop-Target to another(e.g., When an idea is dragged out of a cluster on the board). To narrow this problem down to its fundamentals, an environment to study which factors are involved had to be created. React DnD supplies different offsets when dropping an item to a target, these offsets are described in the documentation of the library as followed [21]:

- `getInitialClientOffset()`: Returns the x, y client offset of the pointer at the time when the current drag operation has started. Returns null if no item is being dragged.

- `getInitialSourceClientOffset()`: Returns the x, y client offset of the drag source component's root DOM node at the time when the current drag operation has started. Returns null if no item is being dragged.
- `getClientOffset()`: Returns the last recorded x, y client offset of the pointer while a drag operation is in progress. Returns null if no item is being dragged.
- `getDifferenceFromInitialOffset()`: Returns the x, y difference between the last recorded client offset of the pointer and the client offset when current the drag operation has started. Returns null if no item is being dragged.
- `getSourceClientOffset()`: Returns the projected x, y client offset of the drag source component's root DOM node, based on its position at the time when the current drag operation has started, and the movement difference. Returns null if no item is being dragged.

For dragging the ideas on the board, the `getDifferenceFromInitialOffset()` method is used, but in case of dragging an item out of another source, such as dragging an idea out of a cluster, there is the initial position missing, because they are not on the board, they are part of the cluster. After testing and thinking about the drag and drop context, the best suitable one from these methods is `getSourceClientOffset()`. However, the result was still wrong. After the investigation, the factors playing along with this problem are:

- The Dom-Node,
- the Position of the Dom-Node and
- the scroll position of the Dom-Node.

The source client offset and the drag and drop context container node is needed. Removing the offsets of the container, which includes *margin*, *padding*, *position* offset of the container relative to the overall page and add the scroll offset to the item, will result in the correct drop position.

```
const newPosition = monitor.getSourceClientOffset();
store.dispatch(deleteCategoryFromIdea({
    _id: item._id,
    left: newPosition.x-containerNode.offsetLeft+containerNode.scrollLeft,
    top: newPosition.y-containerNode.offsetTop+containerNode.scrollTop}));
```

5.4.3 Post-It

The post-it, which was used until now was only a picture with a title field and a drop-down text field for the description, which was not editable from the moderator. Because this is not sufficient an HTML and CSS post-it was added to the project and replaced the old picture post-it. This new post-it can also be colored in any color the user wants, as it can be seen in figure 5.8.

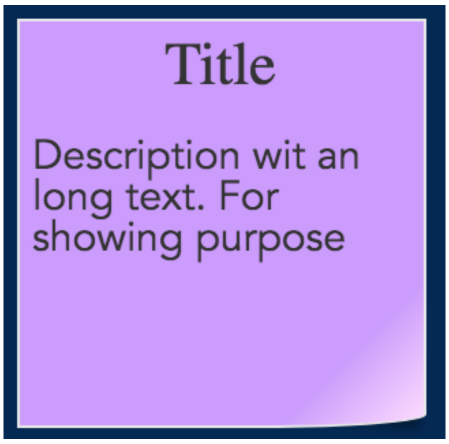


Figure 5.8: New Post-It with title, description and a different color.

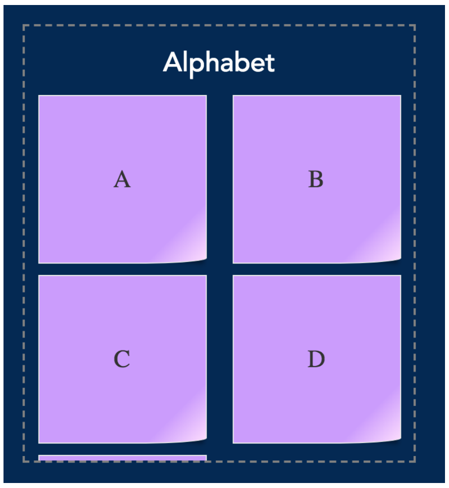


Figure 5.9: Cluster with colored post-it.

5.5 Adding Color to the Project

Up to now the whiteboard only has used yellow post-its, with the newly created ones, the color is used that someone can see that they are selected instead of a slight rotation of the note. This is a much better user experience. Moreover, clusters do now have different colors as it can be seen in figure 5.9. Every time a cluster is created, a color from a defined list, or a random color when all colors are used, is selected.

5.6 Screen Extension

When multiple users are generating ideas, the space is getting too small if there are multiple clusters. Therefore a screen extension is needed. It is now possible to add more screens with a click on the small plus symbols to the board, like adding a row to a

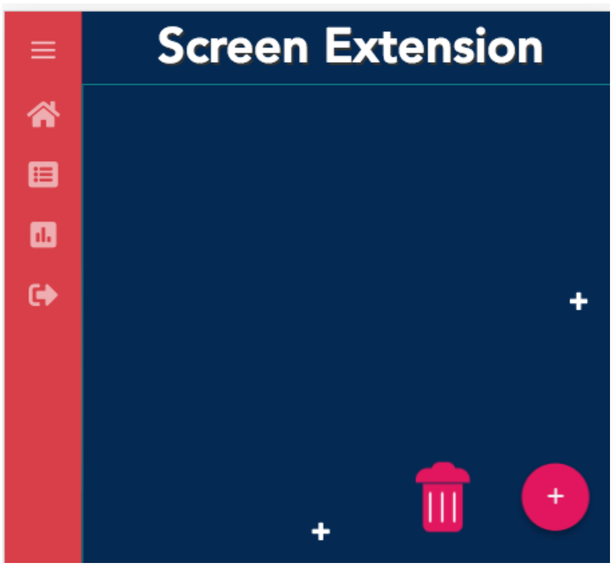


Figure 5.10: Screen extension icons

table. Figure 5.10 is showing the symbols on the brainstorming screen. The screen can get one extra size to the right and 6 to the bottom until it is a 6 × 2 grid, which is the max size currently. Because this was not part of the semester project, there was a lot of rewriting required. A first step was fixing the problem mentioned in section 5.4.2, which was needed for this extension. Also, the position calculation, when dragging items longer distances, needed some adaption. (e.g., whenever an item is dragged from top to bottom and the page is scrolling down) There was a method created, which takes the scroll and offset position before and afterwards in order to calculate the correct offset based on the different scroll positions:

```
function calculateOffset(offB, offA) {
  if(offB === offA) return 0;
  if(offB < offA) return offA-offB;
  if(offB > offA) return (offB-offA)*-1;
}
```

`OffB/offA` is the scroll position offset before and after the dragging. Then it has to be considered if it was not scrolling or it was scrolling left/right or up and down. Depending on the scroll direction, the calculation has to be adapted to the offset. Furthermore, the selection library, which is used (React Selectable) was not working as expected. Because the functionality of this lib was needed, a dive into the library had to be done. After investigating and debugging what could go wrong in the library, it was clear that the library is missing some offsets. Therefore the calculation of the correct offset is now done in the *Blobster* application and handed over to the library, which was changed to use the values provided by our project:

```
const boxLeft = this.applyContainerScroll(
      Math.min(leftContainerRelative - bowWidth, leftContainerRelative),
      +this.props.nodeForOffset.scrollLeft
```

After that, the library had to be built, transpiled and replaced in the node modules folder.

5.6.1 Socket.io

In the existing code base, which was used for the master thesis project, were multiple problems with the real-time session handling. One of the most notable problems was the duplication of the user name in the session invite screen, as well as the duplication of ideas on the moderator screen. Debugging is pretty hard in this real-time scenarios, so the search time for finding those mistakes was a long term investigation with multiple changes and testing phases, because some mistakes occurred only after disconnects or after a specific sequence of actions. For example, a user who disconnected because of an internet issue joined back and was invited to another session without closing the old session from the moderator ended in a user sending ideas to 2 sessions at the same time or duplicating the ideas. Because of this problem, multiple changes were needed in the frontend, as well as in the backend. In the frontend, the registration to the *Socket.Io* listeners for a specific real-time event could be executed more than once, for example when the registration is in the wrong life-cycle method or is not unregistered on a session change, or it is executed again after a reconnect.

5.7 Real-Time-Events

One of the biggest challenges was the communication between the moderator and the clients. Because developers cannot add *Socket.IO* events on the server dynamically. Therefore, another method had to be considered. There are two events provided on the server, which can be used for sending data from the moderator/client to the client/-moderator:

```
socket.on('sendDataToModerator', function(data) {
    socket.broadcast.to(socket.sessionRoom).emit('receivePluginDataFromClient', data
    );
});

socket.on('sendDataToClient', function(data) {
    socket.broadcast.to(socket.sessionRoom).emit('receivePluginDataFromModerator',
    data);
});
```

When the client or moderator receives data from the other part, an action is called. In the template epic, two epics are listening for these events. The template developer can then think about a protocol and send actions over the socket and merge these actions into the epic, which then gets dispatched on the other side, as it can be seen in the following sample epic:

```
(action$, store) =>
    action$
      .ofType(String(receiveDataFromMod))
      .pluck("payload")
      .do(hood => console.log("hoodmoodepic", hood))
      .mergeMap((data) => {
        switch(data.action) {
```

```
        case 0:
          return Rx.Observable.of(goToFullScreenPlugin({pluginId: data.pluginId,
      extensionId: data.extensionId, mod: false, sessionId: data.sessionId}))
        case 1:
          return Rx.Observable.of(goBackToPrivateScreen(), resetState());
        case 2:
          if(store.getState()[PLUGIN_ID].waiting) return Rx.Observable.of(
      toggleWaiting())
          return Rx.Observable.empty()
      }
    })
```

Styles

Importing styles the normal ReactJS way is currently not supported, the style is imported and can then be used with the *classnames* library.

```
//ReactJS:
import './Cluster.scss'
//Template:
import qes from './QuestionsEditor.scss';
import cx from "classnames";

<div className=\{cx(qes\["answers"\])\}\>
```

Images

Images will be *Base64* encoded by *rollup*. Because this is increasing the bundle size heavily and therefore makes the usability worse, it is recommended to host the pictures online and set the URL in the image tag.

5.8 Development Environment

A test environment was created for the developers, where the *Blobster* project is added as a submodule. The test environment provides a template dummy project, which is automatically added to the template system. Changes in the template section will automatically trigger a rebuild from the *Blobster project*. Therefore *rollup* is added for building the template. When the template is finished, it can be bundled with *browserify*:

```
browserify index.js -o hoodmood2.js
```

Afterwards it can be uploaded with the form in figure 5.5.

5.9 Build and Deployment

5.9.1 Amazon S3

The web application *Blobster* is hosted on an Amazon S3 bucket. To open the application, one can enter the URL http://www.blobster.it into the web browser.

Amazon Simple Storage Service (Amazon *S3*) is a web-based cloud storage service designed for online backup and archiving of data. In addition, it is also possible to host a website or web application with an static URL on a *S3* bucket. Each file stored in the bucket has his own personal ID, which is used by an application to retrieve the file. Over a rest interface it is also possible to add and access data [24].

Chapter 6

Evaluation

In the previous chapter 5, the implementation and the system design of the prototype have been described in more detail. The following chapter is about the evaluation of the created prototype. The evaluation consists of the fulfillment of the requirements and a comparison to other systems with a modular architecture. In the end, the new *Blobster* web application will be compared to the old code base and it will be verified whether the loading time has improved due to the modular structure.

6.1 Fulfillment of the Requirements

In this section, the four main requirements defined in section 4.2 will be repeated in more detail and explained how they were accomplished.

6.1.1 Correction and Improvement of *Blobster*

The first requirement, essential for all of the other requirements was to fix all the problems in the existing code base and make multiple improvements. As described in section 5.4, all mature problems such as the intersection between the sessions, the real-time event duplication and the positioning problem after dragging the ideas, were fixed.

Moreover, some improvements were done, figure 5.9 is showing the new cluster format in which the ideas are visually combined. Because of the space problem coming along with the cluster showing the ideas without hiding them and therefore using a lot more space and multiple templates on the board, the screen extension was also a required functionality.

6.1.2 Plugin-System and Integration into *Blobster*

The actual point of the master thesis project was the development of a modular architecture, where plugins can be added without significant rewrites of the main project. Therefore the plugin system described in section 5.2.5 was successfully developed and integrated into the *Blobster* project. A view, where a moderator can select templates for his session and also a navigation bar on the left side, was added for better usability. Dragging a template from the left bar onto the board lets a template appear in the actual session. For the developers, an add plugin screen was designed, where they can upload their self-developed templates to *Blobster*.

6.1.3 Development Environment

Section 5.8 showed that there is a complete development environment where third party developers have a basic scaffold of a plugin with the developer version of the *Blobster* project for testing purposes. Editing the plugin source code will result in a refresh of the web applications. Therefore the plugin developers have instant feedback about how their template is looking in the *Blobster* brainstorming board.

6.1.4 Sample Plugin

As a sample plugin for testing the whole system, a small quiz was developed. Dragging the quiz template from the sidebar onto the board will result in a question editor on the board. With a full-screen option, the moderator can switch to a view where only the active template is shown. There he can enter a question and four answers or possibilities to choose from. Cluster and ideas can also be dragged into the template, after starting the quiz/voting. The participant view will change to the corresponding template voting screen where they can answer or vote on the different options. When the quiz is done, the result will stay on the brainstorming board.

6.2 Comparison to Other Solutions

In addition to the fulfillment of the requirements, there will also be a comparison between other frameworks and libraries for adding modularity to web applications and other projects with a similar system that is not reusable. Some of these projects were already introduced in chapter 3.

6.2.1 Evaluation-System

For comparison, important requirements were defined:

- Developed for web applications?
- Is *ReactJs* supported?
- Third-party plugin development, can every developer add a plugin to the system?
- Add on the fly, is it possible to add plugins without compiling and reloading the page?
- Is the integration into other Applications possible?

6.2.2 Comparison

The projects compared to the *Blobster* modular architecture will be split into three types, as it can be seen in table 6.1. Plugin-system is the first type, which will only focus on the system itself, without any additional framework for creating a web application. The second type is frameworks, which have a plugin system as part of their application framework including multiple functionalities. The last type are products which also include a modular architecture where plugins are involved. Some of them also support third-party plugins, but the system they use is not a standalone package. Therefore, it cannot be reused from other applications.

	Web App	ReactJs	Third party Plugin	Add on the fly	Integrateable
Plugin-System					
Thesis Project	✓	✓	✓	✓	✓
react-plugin-system	✓	✓	✓	X	✓
Frameworks					
Fusion.js	✓	✓	✓/X	X	✓
UmiJS	✓	✓	✓	X	✓
Projects					
Grafana	✓	✓	✓	✓	X
React Static	✓	✓	✓	X	X
Nylas Mail Client	X	✓	✓	X	X
Figma	✓	✓	✓	✓	X
Wordpress	✓	X	✓	✓	X

Table 6.1: Overview and comparison of libraries, frameworks and projects using a modular architecture.

Plugin-System

At the time of writing, there is only one library on the market, which is comparable to the plugin-system developed in the thesis project. The *react-plugin-system*[1] is online since 02/19/2019. Around this time, the thesis project was also developed and it was not known that this system was existing or was developed at the same time. It is similar to the thesis project, but the main focus is on the modularization of a project and its main features in the development and not like *Blobster* for adding additional content on the fly for different use-cases without changing the main web application.

[1] https://github.com/siemiatj/react-plugin-system

Framework

UmiJS[2] and *Fusion.js*[3] are frameworks which provide a pluggable enterprise react application and also include a plugin-system.

Fusion.js is only focused on modules used for the application, such as an internationalize plugin to make it simple to provide the application in different languages. Therefore, most of the plugins are official plugins developed by the *Fusion.js* team to simplify developing and having all the basic system functionality grouped in modules.

UmiJS puts the focus not only on basic web application functionalities, but they also give a possibility to add community plugins.

Projects

The last category are the projects which use a modular data architecture, which is not extracted from their solution and only applicable in their environment. It is a fascinating topic and because of the lacking solutions out there, many of these companies created blog entries on which they talk about how they managed to develop and integrate their plugin-system approach, which gave an initial entry into the topic.

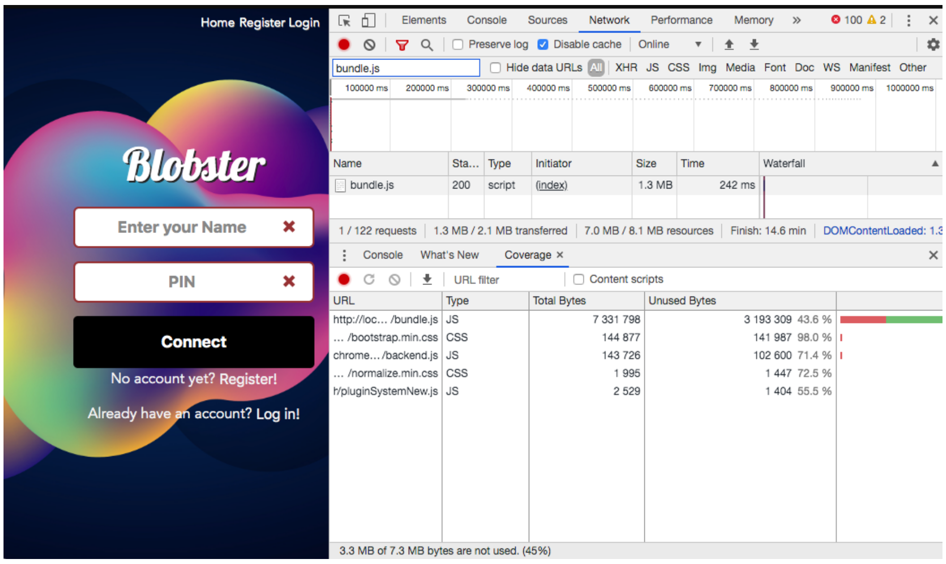
6.2.3 Conclusion

It was a hot topic in the last year since the beginning of this thesis in October 2018. Many projects came up, facing the same problem as the thesis project and discovered new ways to solve this, but they did not fulfill the requirements defined for this project. Most of the projects made a system integrated into their products, nothing which could be reused. Also adding modules on the fly was not covered by many others. Therefore, developing a stand-alone system was the right choice.

6.3 Loading Time

Loading time is an important topic, as it has already been described in section 2.3. It is a big problem in single page applications, because the complete bundle is loaded at the initial page load. Most parts of the loaded *Javascript Bundle* are unused, but will increase the loading time and therefore decrease the usability, because all the users have to wait longer until the page is fully loaded. Even though they do not even need around 50% of the code. Figure 6.1 is showing a loading time and code coverage analysis for the initial page of the *Blobster* project. The *bundle.js*, which includes the complete source code of the project, takes on average 273.2ms to load, as it can be seen in table 6.2. Depending on the *CPU* load and network traffic. The time sounds as if it would not be that bad in most cases but it is directly from the development server hosted by the same machine. Loading a bundle from a server over the internet with a mobile connection can increase this loading time up to multiple seconds as most of the participants use their phone in the brainstorming session. By use of the google developer tools, a simulation of such bad conditions can be done. Figure 6.2 is showing the production build from

[2]https://umijs.org
[3]https://fusionjs.com



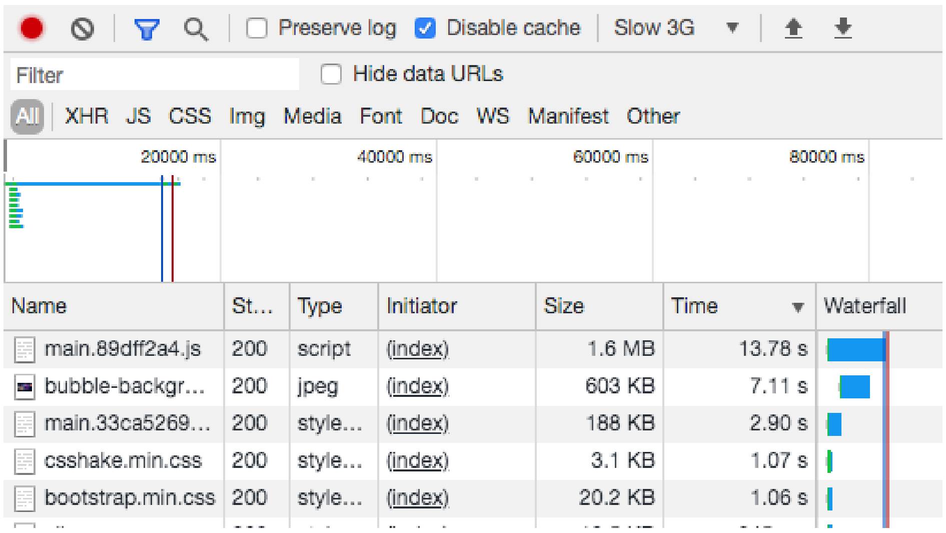
Name	Sta...	Type	Initiator	Size	Time	Waterfall
bundle.js	200	script	(index)	1.3 MB	242 ms	

1 / 122 requests | 1.3 MB / 2.1 MB transferred | 7.0 MB / 8.1 MB resources | Finish: 14.6 min | DOMContentLoaded: 1.3

URL	Type	Total Bytes	Unused Bytes
http://loc... /bundle.js	JS	7 331 798	3 193 309 43.6 %
... /bootstrap.min.css	CSS	144 877	141 987 98.0 %
chrome... /backend.js	JS	143 726	102 600 71.4 %
... /normalize.min.css	CSS	1 995	1 447 72.5 %
h/pluginSystemNew.js	JS	2 529	1 404 55.5 %

3.3 MB of 7.3 MB bytes are not used. (45%)

Figure 6.1: Loading time and coverage analysis from the Google Developer tools.



Name	St...	Type	Initiator	Size	Time	Waterfall
main.89dff2a4.js	200	script	(index)	1.6 MB	13.78 s	
bubble-backgr...	200	jpeg	(index)	603 KB	7.11 s	
main.33ca5269...	200	style...	(index)	188 KB	2.90 s	
csshake.min.css	200	style...	(index)	3.1 KB	1.07 s	
bootstrap.min.css	200	style...	(index)	20.2 KB	1.06 s	

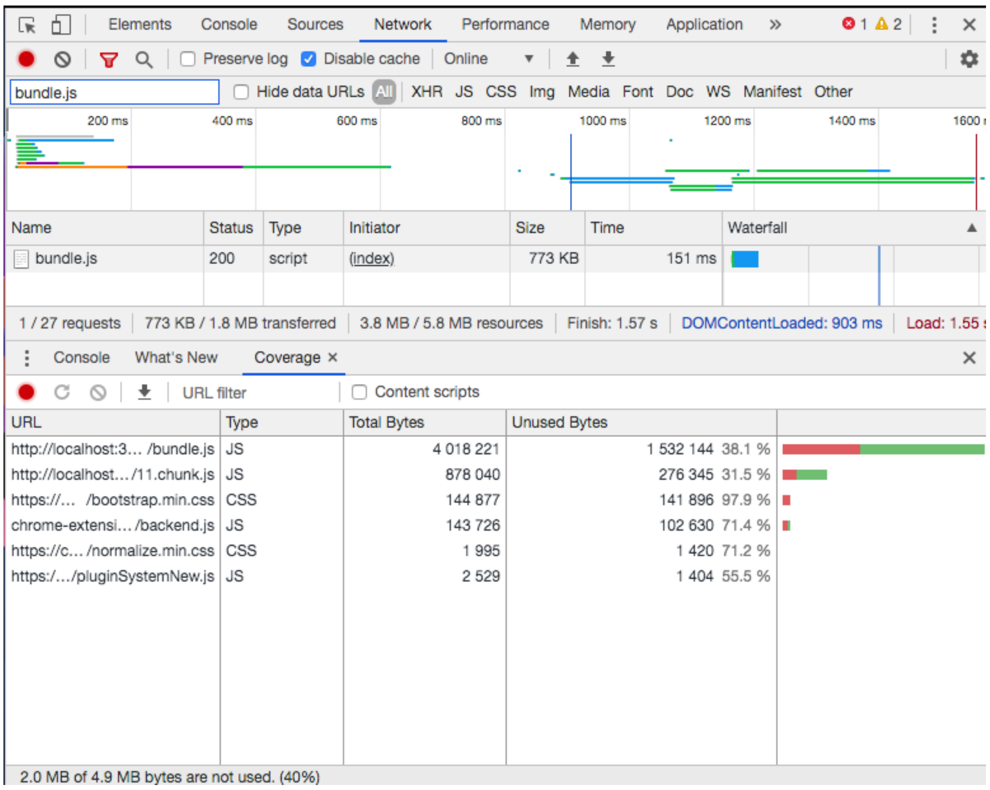
Figure 6.2: Loading time simulated from a Slow 3G connection.

a server with a bad internet connection. The loading time of the *bundle.js* alone took around 14 seconds and calculated that half of the bundle is unused, would result in a time save of 7 seconds, when different modules of the web application are not loaded at the beginning. They should be loaded only when needed. In the case of *Blobster*, where the view is split up into two parts. The moderator and the participant view, they only need to load the part which is actually used by them. Also, the plugin system is only loading the plugin bundle when the moderator is using it, to decrease the bundle as much as possible. Another benefit of the plugin system is when plugins used in the brainstorming session are loaded asynchronously. They are loaded after the initial page

	Time Before 1 [ms]	Time After 1 [ms]	Time Before 2 [ms]	Time After 2 [ms]
1	270	150	292	171
2	253	175	253	184
3	263	160	291	174
4	257	151	269	160
5	298	163	243	154
6	246	149	293	163
7	282	157	259	170
8	266	162	267	158
9	330	180	253	169
10	267	152	278	175
∅	273.2	159.9	270.3	167.8
Bytes(mB)	7.3	4	7.3	4.5
Unused Bytes(%)	43.6	38.1	41.5	37.1

Table 6.2: Overview of the measurements with(After) and without(Before) code splitting. Measurement number one is about the initial page, where participants can join a session. Measurement 2 was done for the moderator screen. A MacBook Pro Retina, 15-inch, Mid 2014 with an i7 of the 4th generation with 2.5 GHz and 16Gb ram was used as a test machine.

load is finished and appear a little bit delayed on the board. This behavior creates a much smoother loading, where the user is seeing the first parts loaded. This will result in the feeling, that it did not take that long as when the user is staring at a blank white page, waiting for all at once. After adding the route-splitting approach described in section 2.3.2, the loading size and time was reduced by a great amount. Figure 6.3 is another test with the google dev tools. The loading time is reduced on average to 159.9ms and the total bytes have decreased considerably as well as a slight change in the unused percentage. This is already a success as the overall bundle size nearly halved. However, there is also more potential in splitting it up. Still, there is around 40% code unused.



Name	Status	Type	Initiator	Size	Time	Waterfall
bundle.js	200	script	(index)	773 KB	151 ms	

1 / 27 requests | 773 KB / 1.8 MB transferred | 3.8 MB / 5.8 MB resources | Finish: 1.57 s | DOMContentLoaded: 903 ms | Load: 1.55

URL	Type	Total Bytes	Unused Bytes	
http://localhost:3… /bundle.js	JS	4 018 221	1 532 144 38.1 %	
http://localhost…/11.chunk.js	JS	878 040	276 345 31.5 %	
https://… /bootstrap.min.css	CSS	144 877	141 896 97.9 %	
chrome-extensi… /backend.js	JS	143 726	102 630 71.4 %	
https://c… /normalize.min.css	CSS	1 995	1 420 71.2 %	
https:/…/pluginSystemNew.js	JS	2 529	1 404 55.5 %	

2.0 MB of 4.9 MB bytes are not used. (40%)

Figure 6.3: Loading time and coverage analysis from the Google Developer tools after code-splitting.

Chapter 7

Final Results and Future Work

7.1 Final Results

The final result is a reusable plugin-system, which is integrated into the *Blobster* project. As a proof of concept and example, there is a Quiz sample plugin added. The plugin was developed and tested in the plugin development environment. The plugin and his functionality can be seen in figure 7.1 and figure 7.2. The main functionalities are:

1. A question list with the option to add questions. Ideas and a cluster can be inserted via drag and drop into the question editor.
2. The moderator voting view, which can also be expanded to full screen as shown in figure 7.4.
3. A list of finished questions, which will stay on the brainstorming board.
4. Clicking on the finished question will show a statistic, as it can be seen in figure 7.2.
5. A participant view, where they can select their answers, shown in figure 7.3.

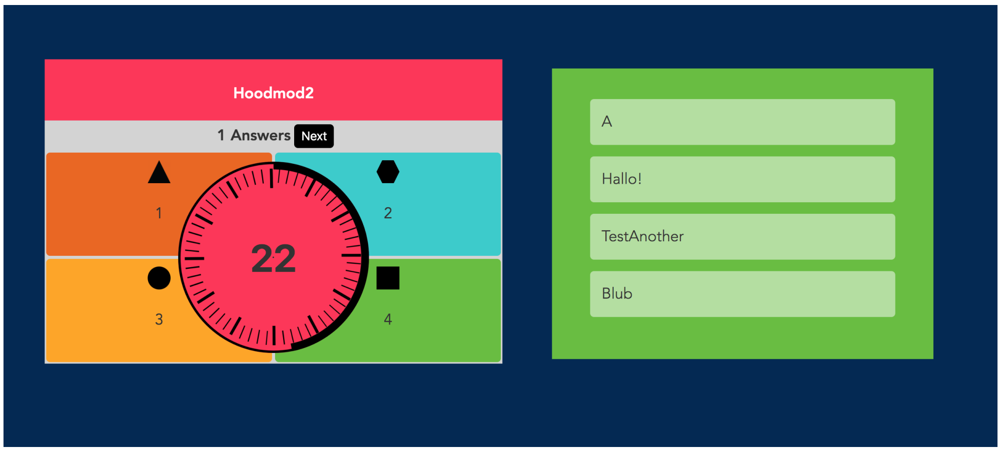


Figure 7.1: On the left side is the moderator view for the voting sequence. The right hand side shows the answered questions after the quiz.

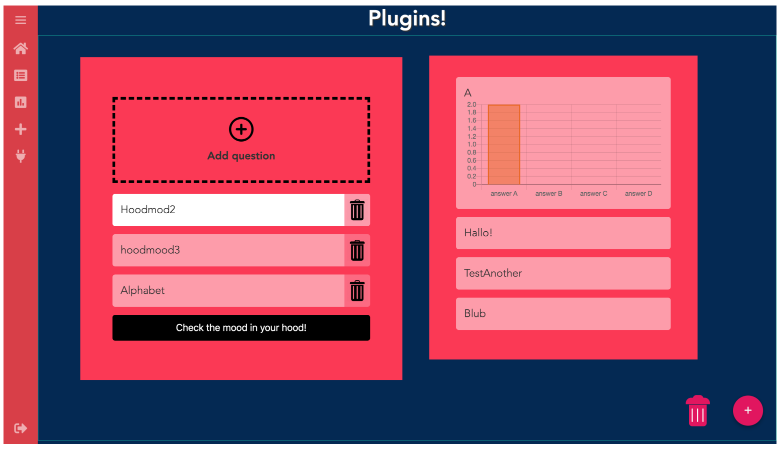


Figure 7.2: Question Editor and the statistic of the quiz.

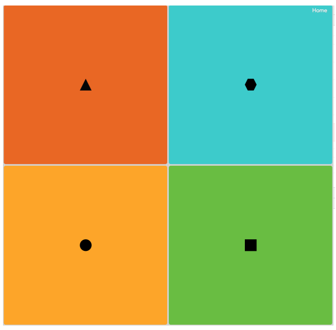


Figure 7.3: Client screen, where the participants can vote on their device.

7.2 Challenges

7.2.1 Communication Between Moderator and Participant View

In the case of *Blobster*, a Template has a moderator and a participant view. These two different views, moderators mostly on a desktop pc with a beamer and the clients with smartphones need to communicate, without changing anything on the backend side of the application.

7.2.2 Reusable Data

Saving data is not the main problem, but we also want to reuse it later on in other templates, for example. Therefore we need a unique structure, which can be read by the main project and other templates.

7.2.3 State

Templates should also be able to get the state from Redux and fire some actions or maybe a template should have his own state. Therefore there was the need for an

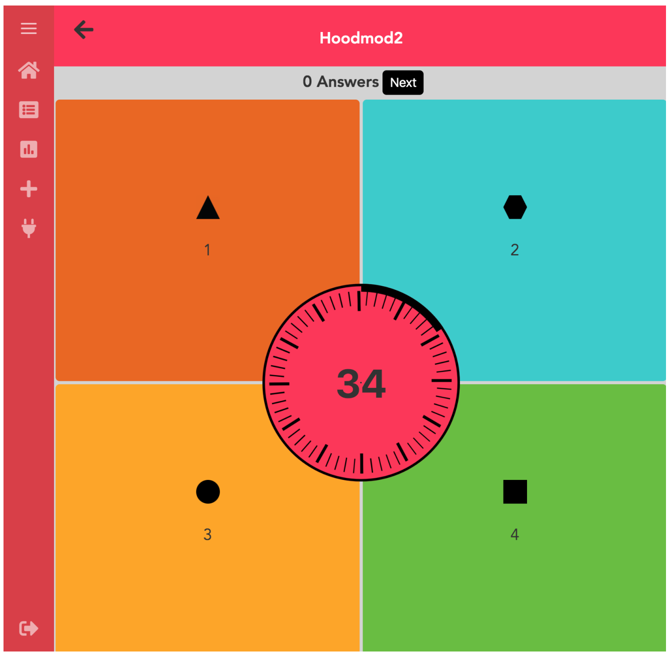


Figure 7.4: Full screen of the moderator screen, because during the quiz, the brainstorming board is not needed.

approach, where this kind of thing has to be dynamically editable.

7.2.4 Bundle Everything Correctly

The biggest challenge overall was to include everything correctly in the bundle, created by *rollup* and *browserify*. Therefore, multiple changes to the config of *rollup* had to be done. E.g., adding some named exports because they are not correctly imported. Also, the URL limit had to be increased because the images were not encoded correctly:

```
plugins: [
  external(),
  postcss({
    modules: true
  }),
  url({
    limit: 28672
  }),
  svgr(),
  babel({
    exclude: 'node_modules/**',
    plugins: [ 'external-helpers' ]
  }),
  resolve(),
  commonjs({
    include: 'node_modules/**',
    namedExports: {
      'node_modules/react-is/index.js': ['isValidElementType'],
      'node_modules/react-dnd/lib/index.js': ['DropTarget']
      }
    })
  ]
```

7.3 Improvements and Future Developments

7.3.1 Security

The security side of this topic has been left out of the equation due to a lack of time.

7.3.2 Full Screen Application

An additional feature or improvement would be a possibility to use the *Blobster* project without the brainstorming tools. Developers can evolve full-screen applications, which only use the environment and the moderator participant pattern implemented. *Blobster* provides user authentication, session management and the communication between two parties, which could be reused for many use-cases and applications, without the need to invest time into things like user handling and session management, which would be working out of the box. Therefore, the developer could focus on their application, without being worried about the boilerplate.

Appendix A

Content of the CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

A.1 PDF-Files

Path: /

MasterThesis.pdf	Master's Thesis

A.2 Project Data

A.2.1 Blobster Backend

Path: /implementation/blobster_backend

blobster_backend.zip .	Backend Project of *Blobster*

A.2.2 Blobster Frontend

Path: /implementation/blobster_frontend

blobster_frontend.zip .	Frontend Project of *Blobster*

A.2.3 Blobster Plugin Environment

Path: /implementation/blobster_plugin

blobster_plugin.zip . . .	Plugin Development Environment Project

A.2.4 Blobster Plugin System

Path: /implementation/blobster_plugin_system

blobster_plugin_system.zip	Plugin System Project

A.3 References

Path: /references

literature/[reference_title].pdf	Files of Referenced Literature
online/[reference_title].pdf	Files of Referenced Online Sources
software/[reference_title].pdf	Files of Referenced Software Sources

A.4 Miscellaneous

Path: /images

*.png	PNG Images
*.jpg	JPG Images

References

Literature

[1] Rudolf Beger. *Present-Day Corporate Communication: A Practice-Oriented, State-of-the-Art Guide*. Springer Singapore, 2018 (cit. on p. 4).

[2] Tony Buzan. *The Ultimate Book of Mind Maps: Unlock Your Creativity, Boost Your Memory, Change Your Life*. Thorsons, 2006 (cit. on p. 5).

[3] Tony Buzan and Barry Buzan. *The Mind Map Book*. Mind set. BBC Active, 2006 (cit. on p. 5).

[4] Tony Buzan and Barry Buzan. *The mind map book: How to use radiant thinking to maximize your brain's untapped potential*. Plume New York, 1996 (cit. on p. 6).

[5] Jenny Chapman. *Javascript on the Server Using Node.js and Express*. Macavon Media, 2013 (cit. on p. 31).

[6] Kristina Chodorow. *MongoDB. The Definitive Guide*. O'Reilly Media, Inc., 2013 (cit. on p. 31).

[7] Edward De Bono and Efrem Zimbalist. *Lateral Thinking*. Penguin London, UK, 1970 (cit. on p. 4).

[8] Rasmus Eneman. "Improving load time of SPAs : An evaluation of three performance techniques". Bachelorarbeit. Linnaeus University, Department of Computer Science, 2016 (cit. on pp. 9, 10).

[9] Martin J Eppler. "A comparison between concept maps, mind maps, conceptual diagrams, and visual metaphors as complementary tools for knowledge construction and sharing". *Information visualization* (2006) (cit. on p. 6).

[10] A. Fedosejev. *React.js Essentials*. Packt Publishing, 2015 (cit. on pp. 23, 24).

[11] Dave Gray, Sunni Brown, and James Macanufo. *Gamestorming: A playbook for innovators, rulebreakers, and changemakers*. O'Reilly Media, Inc., 2010 (cit. on pp. 6–8).

[12] Evan M. Hahn. *Express in Action. Writing, building, and testing Node.js applications*. Manning, 2016 (cit. on p. 31).

[13] Golkar Hassan. "Groupthink principles and fundamentals in organizations". *Interdisciplinary journal of contemporary research in business* 5.8 (2013), pp. 225–240 (cit. on p. 5).

[14] Madhuri A Jadhav, Balkrishna R Sawant, and Anushree Deshmukh. “Single Page Application using AngularJS”. *International Journal of Computer Science and Information Technologies* 6.3 (2015), pp. 2876–2879 (cit. on p. 9).

[15] Kirk Knoernschild. *Java application architecture: modularity patterns with examples using OSGi*. Prentice Hall Press, 2012 (cit. on p. 13).

[16] Helmut Lamm and Gisela Trommsdorff. “Group versus individual performance on tasks requiring ideational proficiency (brainstorming): A review”. *European Journal of Social Psychology* 3.4 (1973), pp. 361–388 (cit. on p. 5).

[17] Ulf Larsson. *Cultures of Creativity: The Centennial Exhibition of the Nobel Prize*. Archives of the Nobel Museum. Science History Publications, 2001 (cit. on p. 5).

[18] Alex Osborn. *Applied Imagination-Principles and Procedures of Creative Writing*. Read Books Ltd, 2013 (cit. on p. 4).

[19] AM Vipul and Prathamesh Sonpatki. *ReactJS by Example-Building Modern Web Applications with React*. Packt Publishing Ltd, 2016 (cit. on p. 24).

Software

[20] *Adding New Epics Asynchronously/Lazily*. URL: https://github.com/redux-observable/redux-observable/blob/master/docs/recipes/AddingNewEpicsAsynchronously.md (visited on 09/05/2019) (cit. on p. 29).

[21] *React DnD*. URL: https://react-dnd.github.io/react-dnd/about (visited on 09/06/2019) (cit. on p. 36).

[22] *React Loadable*. URL: https://github.com/jamiebuilds/react-loadable (visited on 04/05/2019) (cit. on pp. 10, 11, 14).

[23] *react-slot-fill*. URL: https://github.com/camwest/react-slot-fill (visited on 04/05/2019) (cit. on p. 14).

Online sources

[24] *Amazon Simple Storage Service (Amazon S3)*. URL: https://searchaws.techtarget.com/definition/Amazon-Simple-Storage-Service-Amazon-S3 (visited on 09/07/2019) (cit. on p. 42).

[25] Shaun Anderson. *How Fast Should A Website Load in 2019?* Mar. 9, 2018. URL: https://www.hobo-web.co.uk/your-website-design-should-load-in-4-seconds/ (visited on 04/30/2019) (cit. on p. 9).

[26] Giulia Comba. *How can Mind Mapping put your thoughts in order and free your creativity?* URL: http://www.themindmappinglady.com/themindmappinglady-english/ (visited on 09/01/2019) (cit. on p. 6).

[27] *Database as a Service (DBaaS)*. URL: https://www.techopedia.com/definition/29431/database-as-a-service-dbaas (visited on 09/10/2019) (cit. on p. 32).

[28] Facebook. *React; A JavaScript library for building user interfaces.* URL: https://reactjs.org (visited on 09/04/2019) (cit. on p. 24).

[29] Hamza Firoz. *React.js and Front-End Development.* 2018. URL: https://dzone.com/articles/why-choose-react-for-front-end-development (visited on 09/04/2019) (cit. on p. 24).

[30] Flemming Funch. *Radiant Thinking.* 1995. URL: http://www.worldtrans.org/essay/radiantthink.html (visited on 01/09/2019) (cit. on pp. 6, 7).

[31] Google. *The need for mobile speed: How mobile latency impacts publisher revenue.* Sept. 1, 2016. URL: https://www.thinkwithgoogle.com/intl/en-154/insights-inspiration/research-data/need-mobile-speed-how-mobile-latency-impacts-publisher-revenue/ (visited on 04/30/2019) (cit. on p. 9).

[32] *How to Use Dot Voting Effectively.* URL: https://dotmocracy.org (visited on 08/23/2019) (cit. on p. 8).

[33] Addy Osmani Jeremy Wagner. *Reduce JavaScript Payloads with Code Splitting.* URL: https://developers.google.com/web/fundamentals/performance/optimizing-javascript/code-splitting/ (visited on 04/05/2019) (cit. on p. 10).

[34] Steve Johnson. *Affinity Mapping: A Powerful Tool.* May 2017. URL: https://www.under10playbook.com/blog/affinity-mapping-a-powerful-tool (visited on 08/27/2019) (cit. on p. 7).

[35] Aditya Modi. *The Fundamentals of Redux.* 2019. URL: https://dzone.com/articles/basic-fundamentals-of-redux (visited on 09/04/2019) (cit. on p. 24).

[36] *Mongoose, elegant MongoDB object modeling for Node.js.* URL: http://mongoosejs.com/ (visited on 09/10/2019) (cit. on p. 31).

[37] Evan Morikawa. *Building Plugins for React Apps.* Dec. 3, 2016. URL: https://www.nylas.com/blog/react-plugins/ (visited on 04/10/2019) (cit. on pp. 13, 14).

[38] *Mural, think and collaborate visually, anywhere at anytime.* URL: https://mural.co (visited on 01/09/2019) (cit. on p. 13).

[39] *Redux Code Splitting.* May 2019. URL: https://redux.js.org/recipes/code-splitting (visited on 09/05/2019) (cit. on p. 29).

[40] Param Rengaiah. *On Modular Architectures What they are and why you should care.* Feb. 25, 2014. URL: https://medium.com/on-software-architecture/on-modular-architectures-53ec61f88ff4 (visited on 04/07/2019) (cit. on p. 13).

[41] Ethan R. Roberts. *Stuck in the Middleware With You.* Aug. 2018. URL: https://medium.com/@ethan.reid.roberts/stuck-in-the-middleware-with-you-c667acb01fc (visited on 09/11/2019) (cit. on p. 25).

[42] Mike Wasson. *ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET.* Nov. 2013. URL: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx?f=255&MSPPError=-2147217396 (visited on 06/27/2019) (cit. on p. 9).

Check Final Print Size

— Check final print size! —

width = 100mm
height = 50mm

— Remove this page after printing! —