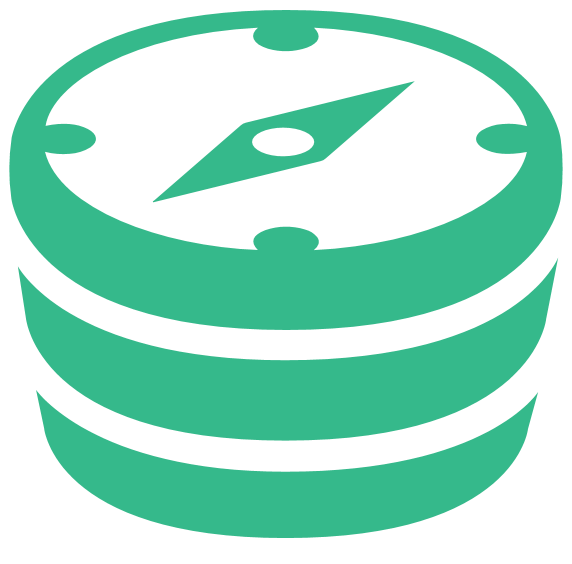
S

Software guidebook

HAN Hogeschool van Arnhem en Nijmegen

DWA-project





Ciyan Cöcelli, Roel de Man, Alex Post

Marijn Martin, Niels Weijers

Table of content

[Introduction 3](#_Toc472593895)

[1. Context 4](#_Toc472593896)

[1.1 Context diagram 4](#_Toc472593897)

[1.2 Users 4](#_Toc472593898)

[1.3 Features 4](#_Toc472593899)

[1.4 External systems 4](#_Toc472593900)

[2. Functional overview 5](#_Toc472593901)

[2.1 Viewing your data 5](#_Toc472593902)

[2.2 Managing your data 5](#_Toc472593903)

[3. Quality Attributes 6](#_Toc472593904)

[3.1 Table 6](#_Toc472593905)

[4. Constraints 9](#_Toc472593906)

[5. Principles 10](#_Toc472593907)

[5.1 Automated testing 10](#_Toc472593908)

[5.2 Code standards 10](#_Toc472593909)

[6. Visual architecture 11](#_Toc472593910)

[6.1 Making a connection 11](#_Toc472593911)

[6.2 Open collection 11](#_Toc472593912)

[6.3 Different views 12](#_Toc472593913)

[6.3.1 Tree view 12](#_Toc472593914)

[6.3.2 Finder view 13](#_Toc472593915)

[6.3.3 JSON view 14](#_Toc472593916)

[6.4 Settings 14](#_Toc472593917)

[7. Folder structure 15](#_Toc472593918)

[7.1 Folder structure diagram 15](#_Toc472593919)

[8. Data flow architecture 16](#_Toc472593920)

[8.1 Data flow diagram 16](#_Toc472593921)

[9. External components 17](#_Toc472593922)

[9.1 Dependency injection diagram 17](#_Toc472593923)

[10. External interfaces 18](#_Toc472593924)

[10.1 Express API 18](#_Toc472593925)

[10.2 MongoDB API 18](#_Toc472593926)

[10.3 Database 18](#_Toc472593927)

[11. Code 19](#_Toc472593928)

[11.1 Generating/rendering HTML 19](#_Toc472593929)

[11.2 Data binding 19](#_Toc472593930)

[11.3 HTTP Request handling 19](#_Toc472593931)

[11.4 Server structure 19](#_Toc472593932)

[11.5 Exceptions and logging 19](#_Toc472593933)

[12. Infrastructure Architecture 20](#_Toc472593934)

[12.1 Infrastructure diagram - local 20](#_Toc472593935)

[13. Deployment 21](#_Toc472593936)

[13.1 Deployment diagram 21](#_Toc472593937)

[14. Operation and Support 22](#_Toc472593938)

[14.1 Starting Mongo 22](#_Toc472593939)

[14.2 Starting the application 22](#_Toc472593940)

[14.2.1 Application 22](#_Toc472593941)

[14.2.2 NodeJS server 22](#_Toc472593942)

[14.3 Updating the application 22](#_Toc472593943)

[14.4 Overall support 22](#_Toc472593944)

[14.4.1 Support of external components 23](#_Toc472593945)

[14.5 Testing 23](#_Toc472593946)

[15. Decision Log 24](#_Toc472593947)

[Literature 26](#_Toc472593948)

# Introduction

This software guidebook provides technical and functional information about the MongoExplorer. The MongoExplorer is a web application in which you can manage and view MongoDB databases.

The purpose of this guidebook is to give future programmers(developers) and people who are interested in the product insight about the application. With this guidebook, we want to give people without knowledge of the application the possibility to understand the application and ensure they are, after reading this document, on a level of experience where they can take the development of the application to a further level.

Besides an overview about the code, functionality and user interface, this document also contains information about choices the development team had to make and how the quality of the application is being ensured.

Another important part of the software guidebook is the operation and support section, this chapter provides every step you need to take in order to run the application.

We hope to inform you in this guidebook as much as possible about every element of the MongoExplorer, and you are able to continue the development of the product after reading this software guidebook.

The application can be found on GitHub, you can either pull or download it to your computer. The URL for the GitHub folder is <https://github.com/HANICA/DWA-nj2016-P-MongoExplorer>.

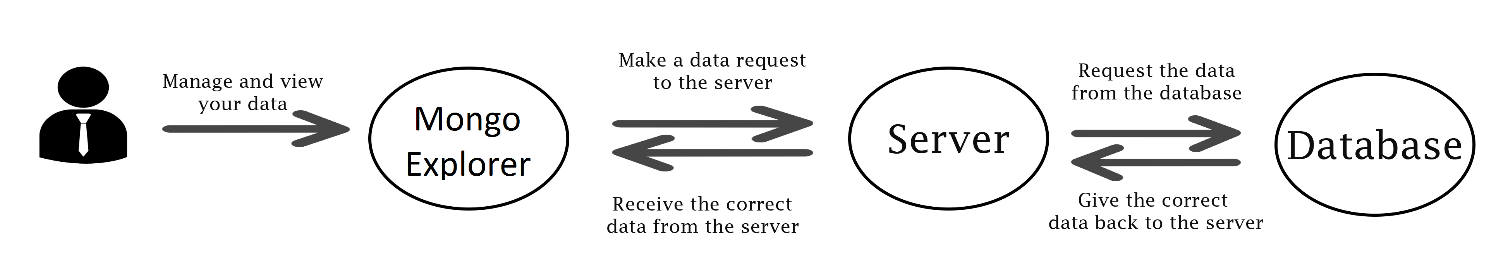
# 1. Context

This document will provide an overview and information about the MongoExplorer.

The MongoExplorer is a web application in which you can display and manage MongoDB databases. The key features of the MongoExplorer are to display, edit, insert and delete data from MongoDB databases.

## 1.1 Context diagram

In diagram 1.1 the global structure of the application is shown. The user interface is being referred to as ‘MongoExplorer’. The database is in this case the combination of the database API and the database itself.



1.1 Context diagram

## 1.2 Users

The MongoExplorer is an application built for programmers, but also for people with their own businesses or office employees who need to work with databases to maintain their data. Current systems like RoboMongo don’t have a web application, only an application which runs locally on a computer. One of the key points of the MongoExplorer is that we do have a web application. The application makes no difference in the type of user, everyone has the same rights.

## 1.3 Features

One of the main features of the MongoExplorer is the simple but elegant user interface. Applications with the same functionality as MongoExplorer usually have a very chaotic user interface, specially designed for programmers. Our goal was to build an application with all the main features but with a cool and elegant user interface.

The application has three ‘main components’: the user interface, the server and the database API. The user interface makes calls to the server, which directs the calls to the database and collects the correct data. The server receives this data and passes it through to the user interface.

## 1.4 External systems

The only external system which applies to our product is the database. The database is the place where the customer stores its data. Until there are written API’s for new databases, the only supported database system is MongoDB.

# 2. Functional overview

The application is built for people who need to manage their data and who have access to a document oriented database. Until now, only MongoDB has been added as supported database management system(DBMS). The goal for the future of the application is to have more DBMS’s added.

The application can be globally divided in two segments, the first is viewing your data and the second one is managing your data.

## 2.1 Viewing your data

The development team has decided to give the user a few different options to view the data. The reason behind this is giving the user the opportunity to create a user interface that works the best for them. There are two different main views to choose from: the tree view and the finder view. The view can be selected in the menu bar.

Another nifty feature with the views is the JSON view. When in tree or finder view, documents can be selected through check boxes and by clicking the ‘edit’ button in the left menu, the documents will be opened in the JSON view.

To complete the idea of creating your own user interface there are several options available in terms of styling of the application. These options are: select colors, select the number of documents displayed at once and select the default view to open a collection.

## 2.2 Managing your data

The second part of the application is about managing your data.

There are four main functionalities. The first is the option to select and delete documents. The second functionality is the option to edit documents by updating the JSON. The third option is the export functionality. Multiple documents can be selected and downloaded to your personal computer. The last functionality is the import option. With the import option, you can insert documents by uploading valid JSON documents.

The visual components will be explained in chapter 6. Visual architecture

# 3. Quality Attributes

To guarantee the quality of our product, every piece of code will be reviewed by a team member. In this way, the code is always seen by at least two people, so it is not possible that someone has made a mistake and nobody knows about this.

Another way to maintain the quality of the code and the application is end to end testing. Several end-to-end tests are written which will emulate multiple scenario’s, either successful or with deliberate errors to check how the system deals with them.

## 3.1 Table

This table is divided in categories and every category has several quality attributes. With every quality attribute comes a description that shows how we have ensured the quality of this.

The blueprint of this table comes from the following source: (Meier, et al., 2009)

|  |  |  |
| --- | --- | --- |
| Category | Quality Attribute | Description |
| Design qualities | Conceptual integrity | For the most components Ant.design has been used. These components, like the tree view, tabs bar and breadcrumbs are free to use for everybody and because they’re all used from the same source, the quality and styling is consistent.  In terms of code consistency, we have used code reviewing by team members to ensure that the overall look and naming of components and variables is as consistent as possible. |
|  | Maintainability | The different components that are used in this application don’t depend on each other meaning that if one component crashes, all of the other components keep working. |
|  | Reusability | All the visual components on the website are single file components. Every component is designed in their own file within the ‘Components’ folder. All the components can be imported and used everywhere in the code.  Another part that is reusable are the functions and Redux actions. As long as they are properly exported and imported in the correct file, they can be reused in other components. |
| Run-time qualities | Availability | The system should be working 24/7. There are no reasons why the system would be down at any moment. The application is available to download from GitHub. Even when a new version is being developed, the application still works because you’ve downloaded it to your personal computer. |
|  | Interoperability | The application has been designed with the intention that programmers can write their own document oriented database API’s and include them in the application. Until now only MongoDB is being supported, but the option to add your own API in the application is available. |
|  | Manageability | The code of the application is provided with commentary that describes how the code functions. This ensures that people without knowledge of the product can still maintain the code.  To monitor the state of the application, the official Redux dev tools are available to download in your browser which allows you to see and manage your data. |
|  | Performance | One of the ways to increase the performance of the application is by using React. Another thing we have used for boosting the performance is making sure that for every request a new connection with the server is being made (to decrease server load) and the downloaded JSON files are not stored on the server. |
|  | Scalability | Because our application is run locally and doesn’t run from servers, there is always only one user at the time. Therefore, it is impossible to test how the system would handle a rising number of users and data traffic. |
|  | Security | There is no security in the application because this was not a requirement by the product owner.  The responsibility of data management lays with MongoDB and not with our application. We directly forward the requests to the database and this database checks whether requests are valid or not. |
| System qualities | Supportability | The system provides error messages whenever you try to do something which can’t be handled at that time, for instance when you try to open a collection when there hasn’t been a connection made. You’ll get a message saying ‘please make a connection first’. Every component has its own error messages whenever something is going wrong, either by the system or by the input of the user. Another addition to this errors are the developer tools in your browser, this set of tools show you messages to help you find the errors.  To make life easier for programmers we have used a ‘.map’ file. This file can be used by the web browser to show you where the error is in your JavaScript file instead of showing you where the error is in the Webpack bundle. |
|  | Testability | The application is tested with end to end tests. In these tests, we try to emulate good and faulty situations. There are certain things that are difficult to test for us, for instance performance and availability. The layout and functionality will be tested using end to end tests with commentary to describe the tests. |
| User qualities | Usability | The system is reviewed by the product owner. The input of the product owner has been implemented in the application.  Because there is a variety of different types of users for the application, there is the option to ‘create’ your own user interface by picking the view and colors of the application. |

# 4. Constraints

This section is about the constraints of the product. The constraints are imposed by school. Since the MongoExplorer is a school project, there is no budget for developing the application. Because of this we can’t use paid software and everything will be done with open source software. Below is a short list with the constraints. This will give you, the reader, some insight in the use of different development languages and packages.

4.1 Constraints by school:

- React with Redux: the use of React and Redux was imposed by school to test our knowledge of the course.

- Webpack: same as above.

- MongoDB: the use of MongoDB is mandatory because the whole project resolves around it.

# 5. Principles

This section is about the principles used in the project to design and build the application.

## 5.1 Automated testing

Every functionality of the application will be tested with end-to-end tests. Unit test for the server must also be written. The purpose of this is to test the API of the server and check if it works correctly. These tests will simulate a good flow and a bad flow. The good flow will simulate the flow of the program when everything is going as intended. This is to check whether every part of the application is built and designed in the correct way.

The wrong flow simulates the flow the system takes whenever something goes wrong, this is mostly to see how the system handles these errors. The faulty test will have some wrong values to see how the application handles them.

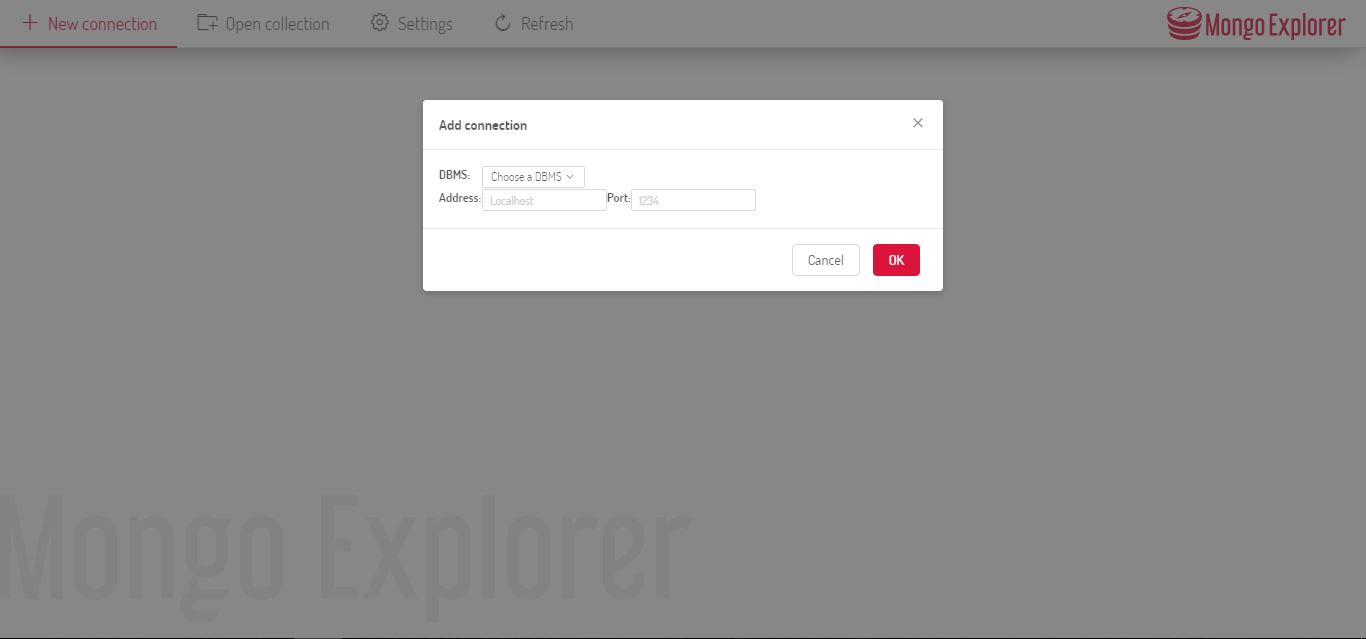
## 5.2 Code standards

As a basis for the code JavaScript will be used. With JavaScript, there comes a lot of variations, in our case we’ve used React with Redux combined with JSX. This means that everything must be done through the redux flow. More information about this is explained in chapter 11. Code. Another attribute we’ve used as a principle is code checking. Every element which has been programmed by only one programmer has been reviewed by team members before it was checked as done, this assures that the programmer has not made decisions by his own and he has not made any significant mistakes.

# 6. Visual architecture

This chapter contains a list of all the important visual components of the application.

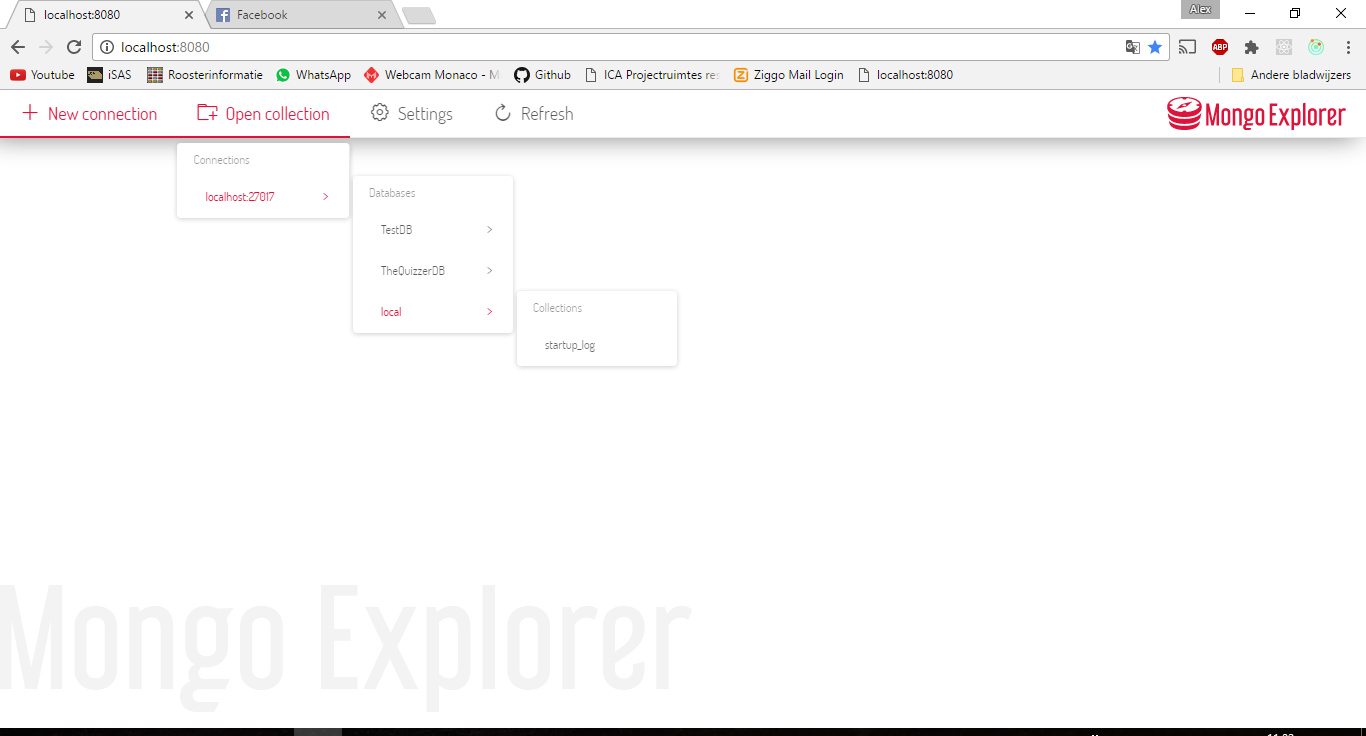
## 6.1 Making a connection



6.1 Making a connection

When clicking the ‘New connection’ button, a modal window is opened in which you can fill in the address and port number on which your database is running.

## 6.2 Open collection



6.2 Open collection

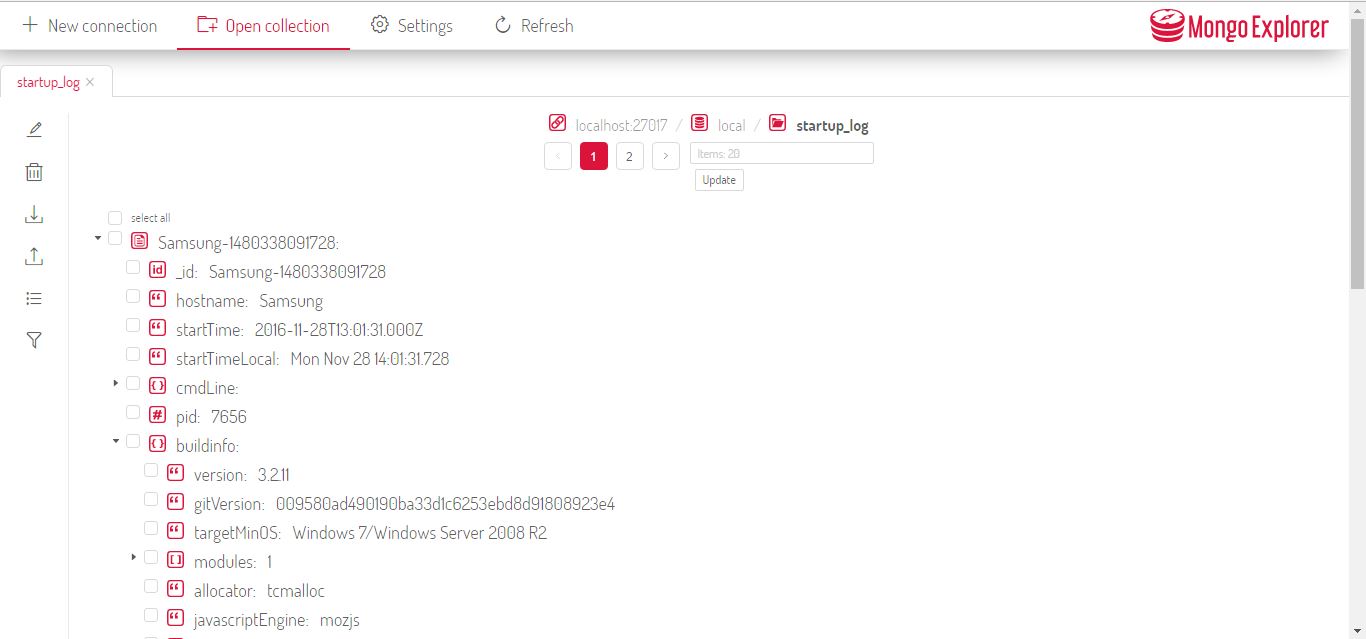
If the connection is made successfully, the ‘Open collection’ button is now clickable. This button opens a dropdown menu. This menu contains all the databases and collections within the current connection. You can open a collection by clicking it in the dropdown menu.

Another thing you can see in this image is the header. The header contains buttons for global actions (not collection related) and the logo of the MongoExplorer.

## 6.3 Different views

There are three views available to choose from. The two main views are the tree view and the finder view. The third view is the JSON view, this view is used to edit documents.

### 6.3.1 Tree view

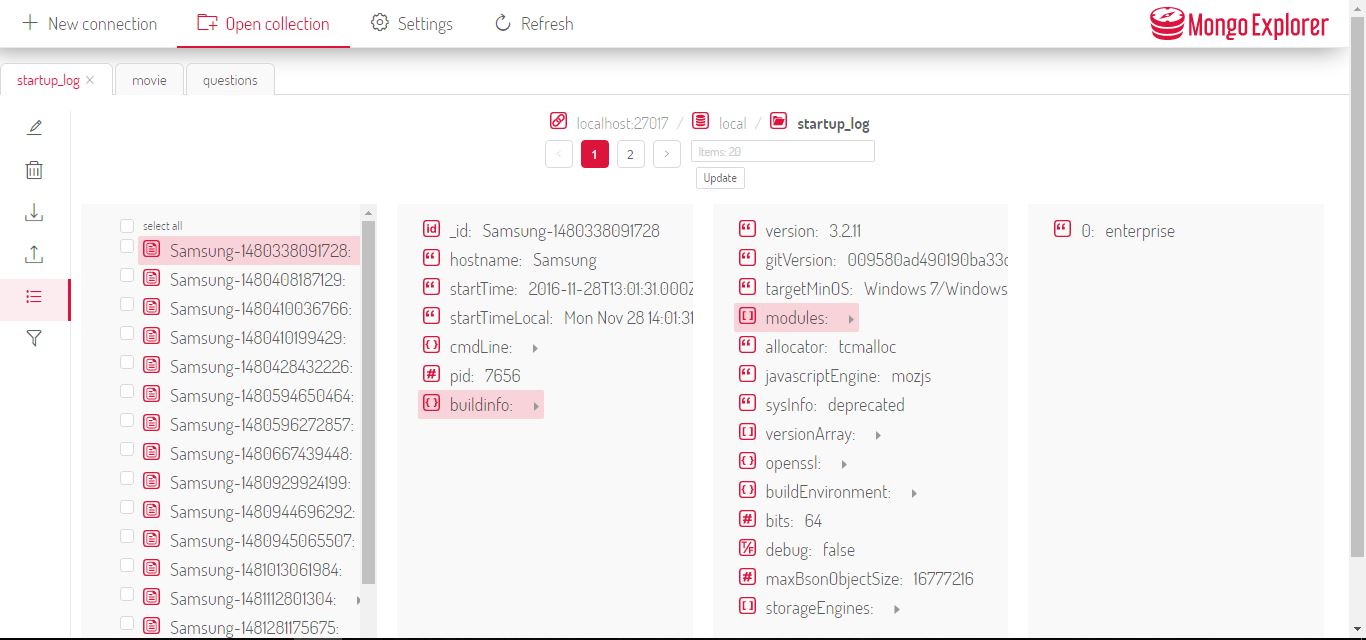


6.3 Tree view

The tree view opens the collection like a tree. At first, a list with all documents is being displayed. If a document contains data, there is an arrow available in front of the document name. By clicking this arrow, you can open the document and view all the data inside.

Other visual components (not linked to the tree view) you can see in this picture are the tab menu bar on the left side and the pagination component. The side menu contains all the actions that are available for an opened collection, these actions are edit, delete, import, export, select view and search query.

### 6.3.2 Finder view



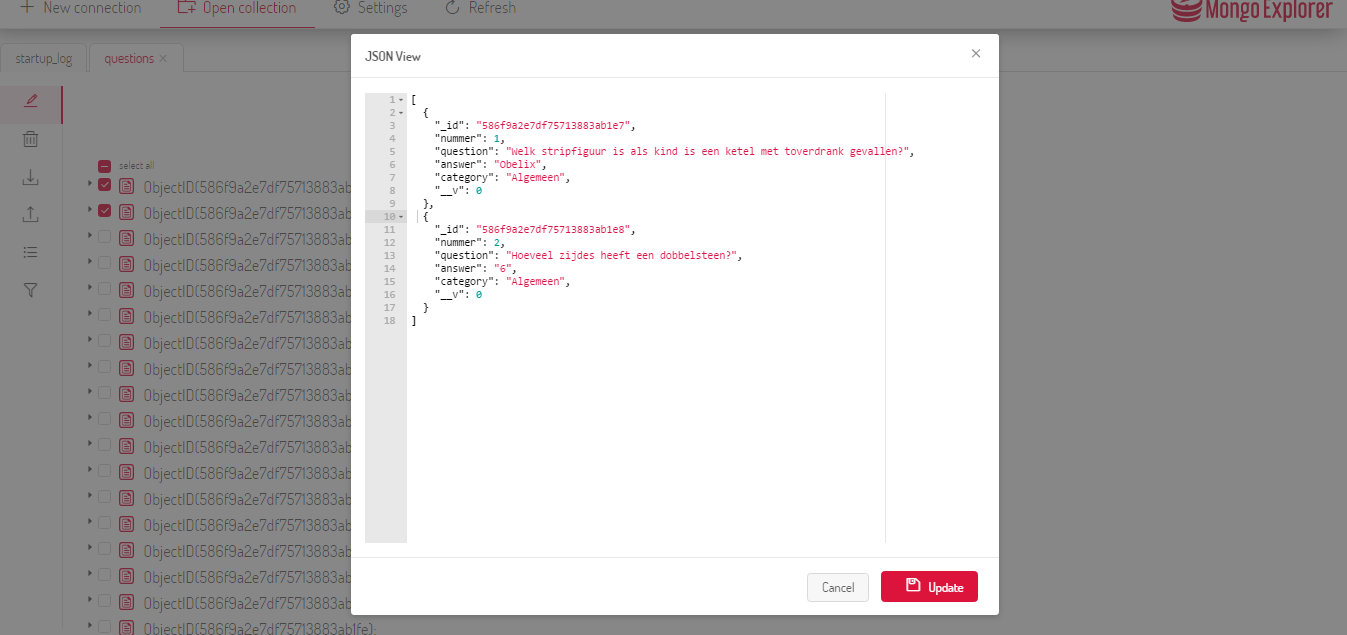
6.4 Finder view

Another view that is available to use is the finder view. The finder view contains a list of all the documents within the opened collection. By clicking a document name, a second panel appears with the correct data. If a value in the second panel contains more data, you can click the little arrow on the right side of the name. The application shows a maximum of four panels with data. If a document contains more than four layers, four will be displayed and a horizontal scroll bar appears to let you scroll through the other panels.

Another thing you can see in this picture is the tabs bar. When a collection is clicked, it will be opened in a new tab. New collections can be opened in tabs and these tabs can also be closed.

The last thing you see in this image is the pagination component. The standard number of displayed documents per page is set at 20. If a collection contains more than 20 documents, the pagination component can be used to navigate to other pages. In the text input box, you can set the number of documents you want to see per page, by clicking ‘update’ you can apply this.

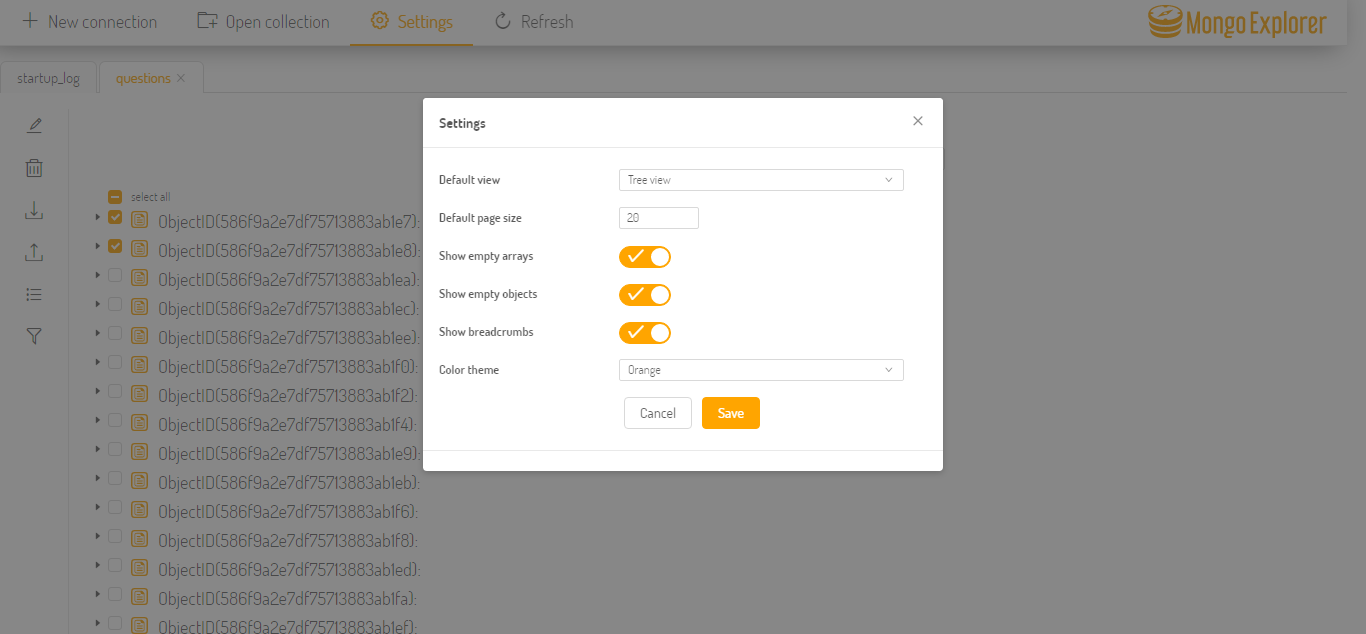
### 6.3.3 JSON view



6.5 JSON view

When a document is selected (by clicking the arrow in front of it), it can be opened in the JSON view. The JSON view show the exact JSON structure of the selected documents. The JSON view also contains syntax highlighting. In the JSON view, you can edit the text and click ‘update’ to update the edited document.

## 6.4 Settings



**6.6** Settings

In the menu bar is a button for the settings. This buttons opens a window in which you can edit multiple settings as seen above. Another thing you see is that the color is now changed to green. There are numerous colors available to choose from in the ‘color theme’ option.

# 7. Folder structure

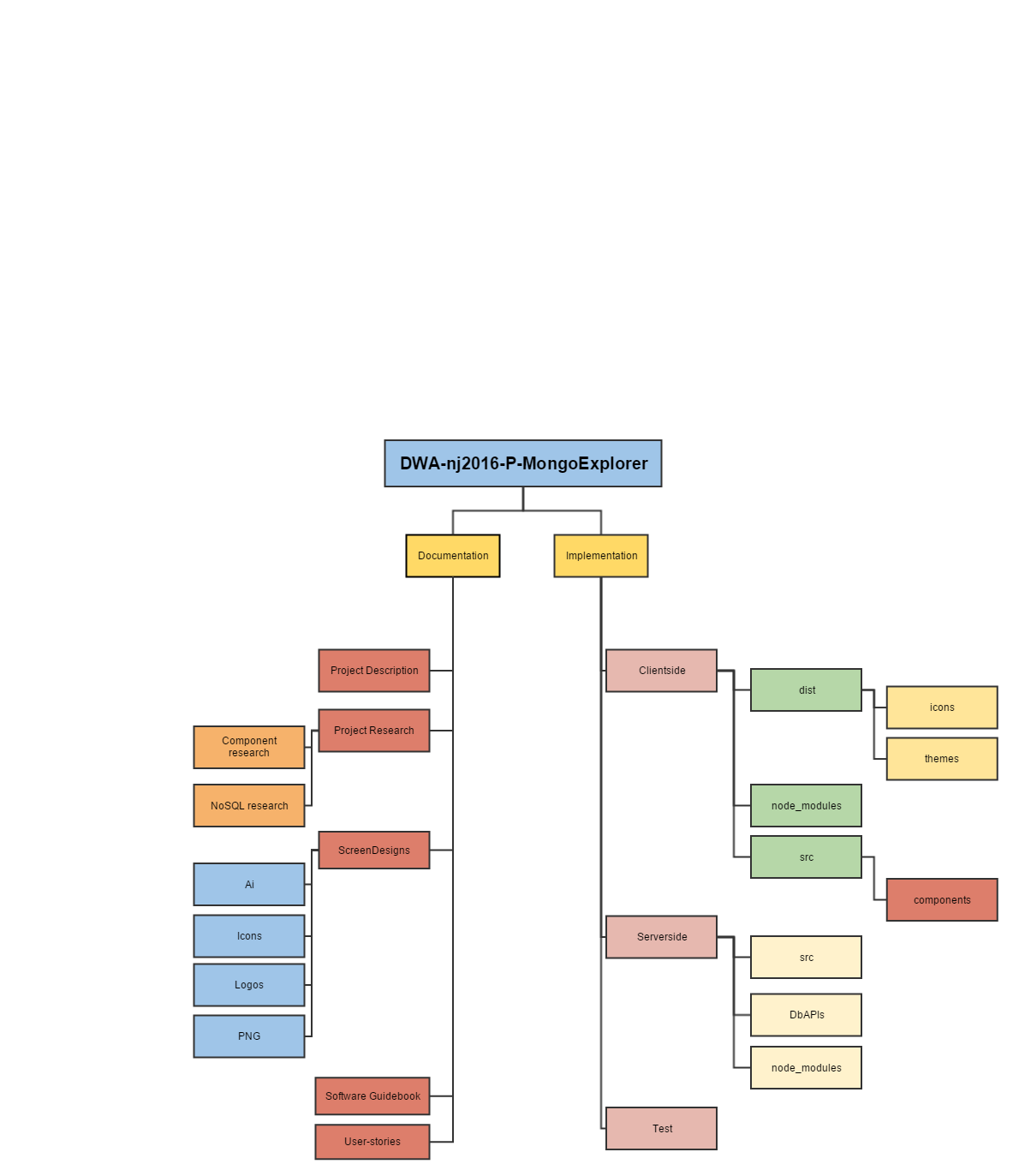
We have chosen to make the folder structure of the application as clear as possible. First, the documentation and implementation are separated from each other. This prevents that the code and text files are in the same folder. For the implementation, we have chosen to split the client side and the server side to keep different parts of the application separated from each other.

## 7.1 Folder structure diagram

The project folder is divided in two different sub folders: Documentation and Implementation. In the Documentation folder includes all the documentation of the project, including this guidebook. Other parts of the documentation are the Project Research and the ScreenDesigns.

The implementation part is divided in Clientside, Serverside and test. In the test folder are the end-to-end test and unit tests. The client side folder contains the ‘src’ folder, this is the location where all the components of the application are stored. Besides the src folder there are the dist and node\_modules. The node\_modules will be further explained in chapter 9.1 Dependency injection diagram.

The Serverside folder contains all of the things that are needed for the server side. The most important sub folder is ‘DbAPI’s’. This folder contains the database API’s.



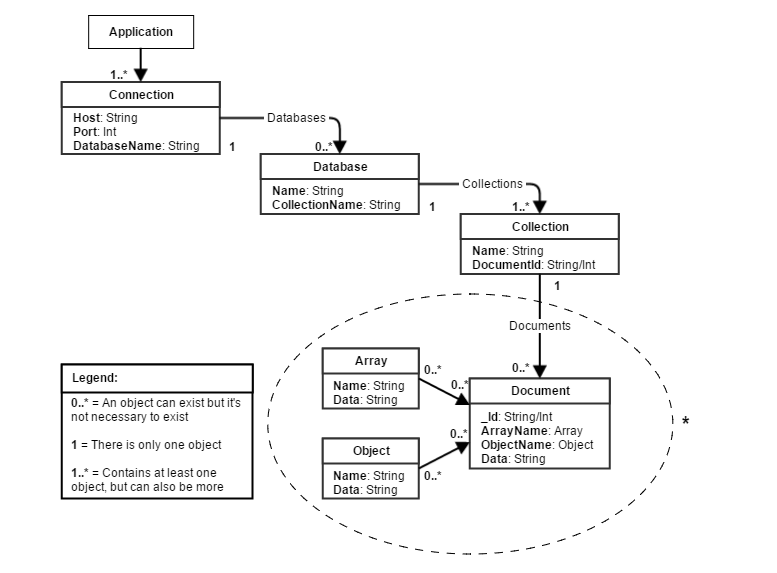
7.1 Folder structure

# 8. Data flow architecture

This chapter describes the data flow of the application of the MongoExplorer.

## 8.1 Data flow diagram

This diagram shows the data flow within the application. With this step by step diagram, we show the different layers of data and how they are connected with each other.



8.1 Data flow diagram

\* It may happen that an arrays or objects has a field that also is an array or an object. Because this process can happen endlessly it is not drawn in the illustration.

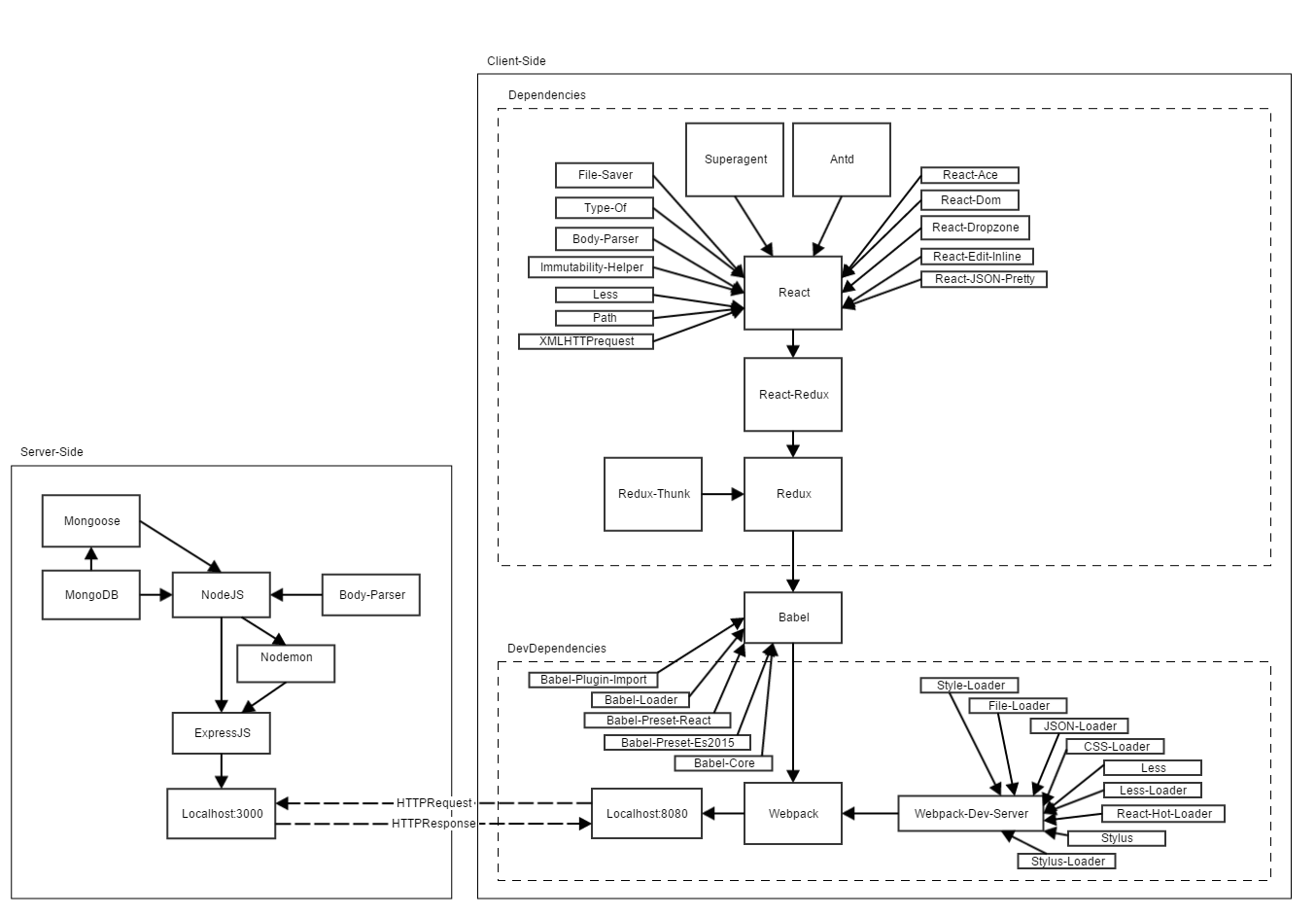
# 9. External components

The application won’t work without the ‘node\_modules’ folder. In the node\_modules, all the dependencies are installed which are located within the package.json file. Client side and server side both use an own node\_modules folder (and package.json).

## 9.1 Dependency injection diagram

The client side and server side package.json are combined in a diagram to show the relation between the server side and the client side. This is being displayed in a dependency injection diagram to give the developers an overview about the relations between modules.

In figure 9.1, both client side and server side are displayed.



9.1 Dependency injection

# 10. External interfaces

The MongoExplorer uses 3 external interfaces; the MongoDB API, the expressAPI and the database that the client needs the data from. In this chapter, these 3 interfaces will be explained. There is added a bullet list with all API functions.

## 10.1 Express API

The Express API listens to a number of routes on the localhost. When one of these routes is requested the API runs the corresponding command and calls the MongoDB API with the right action. The result of this action is directly send to the client with the correct status codes. The API doesn’t use any validation, and can be used by anyone as long as they have the proper routes and parameters.

Originally the idea for the Express API was that it could use a number of different API’s, one for each different database. However, because of various reasons only one database API was written (the MongoDB API). Because of this we can’t guarantee that the Express API works with different database API’s. That is why currently the application only works with MongoDB databases. The version of the application that should work with different API’s can be found on a different branch on GIT.

API functions:

* Get all API’s
* Create connection
* Find documents
* Delete documents
* Insert imported data
* Update document

## 10.2 MongoDB API

The API was created specifically for this application and only works for MongoDB databases. The API is attached to the server and communications between the server and the API happen only on the server. The API will simply run any command it gets without validation, and will return either the result of that command or an error directly to the server.

API functions:

* Create connection
* Find documents
* Delete documents
* Import documents
* Update document

## 10.3 Database

The database is the only interface that doesn’t belong to the application and ownership lays with someone else. Because of this the way this interface behaves differs per database and cannot be guaranteed.

# 11. Code

This chapter provides a detailed view on how the code works in the application and how different pieces of code communicate with each other in order to render the MongoExplorer on your display.

### 11.1 Generating/rendering HTML

For our application, the React framework was used. For our application, we used the JSX language which is almost identical to JavaScript but allows some HTML syntax in the code. This code has to be transpiled to JavaScript in order for it to work.

In React, the application is split into individual pieces, each piece is called a component. JSX is being used to render HTML within components. Each component can contain props and a state. The props are usually given to the component on creation and are immutable. The state of a component can be edited by the component itself, as soon as this is done the component is updated. Each component can contain a render method, this method has to be called ‘render’ or React will not call it. This method can do whatever, and any returned text, HTML or other components will be rendered to the screen. This method will also be called whenever the state of a component changes using the ‘setState’ function.

### 11.2 Data binding

Because it is inconvenient to have the data within your application be spread out across multiple components we used a framework called Redux. In the Redux framework, all data is stored in an immutable object called the ‘store’. In the Redux framework, each component can read some data from the store using the ‘mapStateToProps’ function. When a value in the store is updated, each mapped component will also be updated using the new data. In Redux, the only way to update the store is by using reducers. A reducer will listen to a certain action and update the store when that action is fired. So, if a component needs to update something it will have to dispatch an action. In order to give a certain action to a certain component the ‘mapDispatchToProps’ function can be used.

### 11.3 HTTP Request handling

All of our HTTP requests are done in Redux actions using the ‘superagent’ library. For this a special type of actions is used, ‘thunk’ actions. These actions can be finished at will. When the request is finished, the action also finishes and dispatches the received data. The store is then updated using this new data, so all mapped components will also be updated.

### 11.4 Server structure

On the server, the Express library was used. Express is a JavaScript framework that can bind certain actions to certain routes on the server. These actions will call other actions in the MongoDB API. This API can talk to a MongoDB server and return any results/errors to the server. Depending on the result the server will return a response to the client.

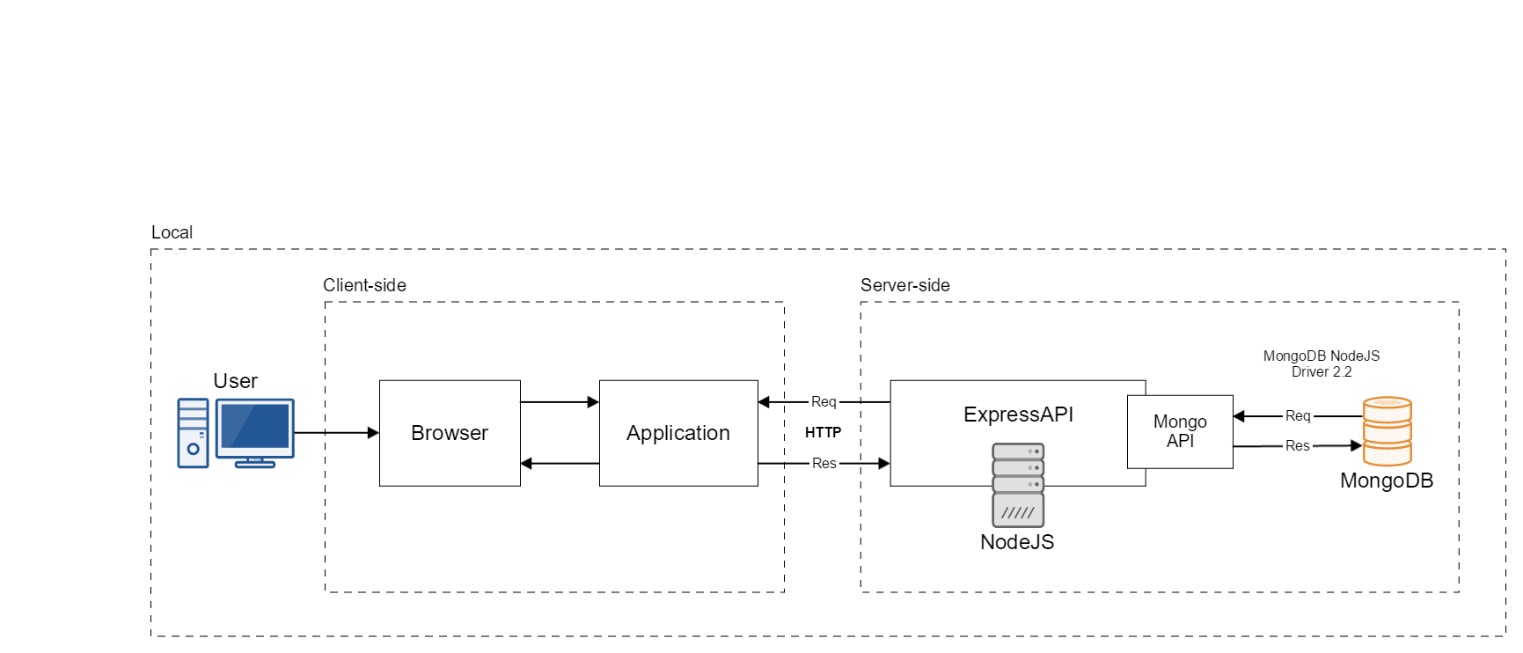
### 11.5 Exceptions and logging

On the client, whenever an error happens a message box is displayed telling the client that something went wrong. On the server, any errors are immediately logged so these can be further investigated. No exceptions are thrown in order to prevent the server from shutting down.

# 12. Infrastructure Architecture

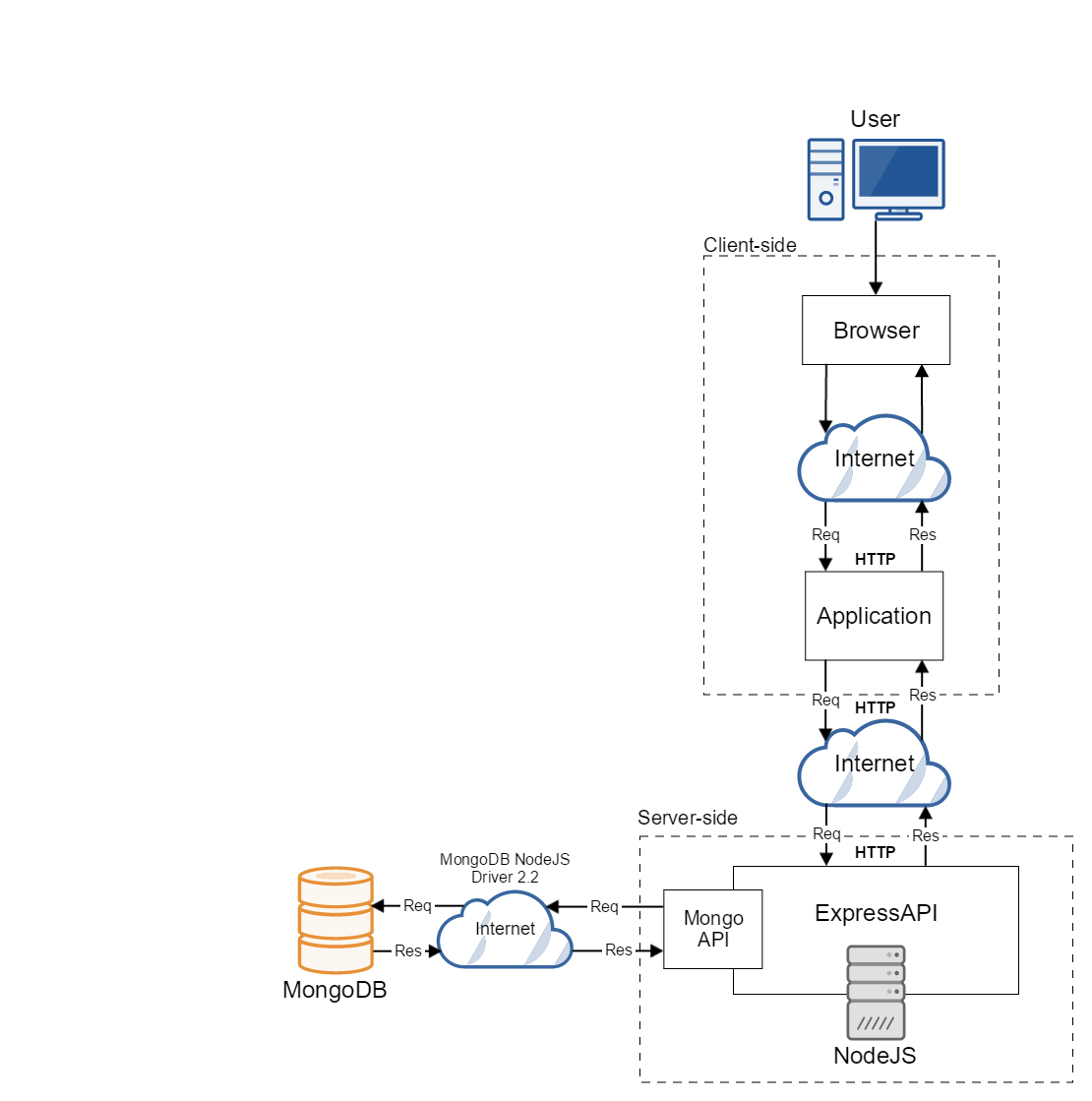
In this chapter, we describe what the connection between the server, database, client side application and user looks like.

## 12.1 Infrastructure diagram - local



12.1 Infrastructure diagram - local

This diagram shows the architecture when the application runs on your localhost.

At first, the user opens the application in the web browser. When clicking something in the application, an action is triggered and a http request is being sent to the server. The server sends a request to the MongoAPI, which transfers the request to the MongoDB database. The database returns the correct data and sends it back to the server. The server then transfers the data to the application.

The image above shows the architecture when the application is run on localhost.

Image 12.2 shows what the architecture looks like if the application

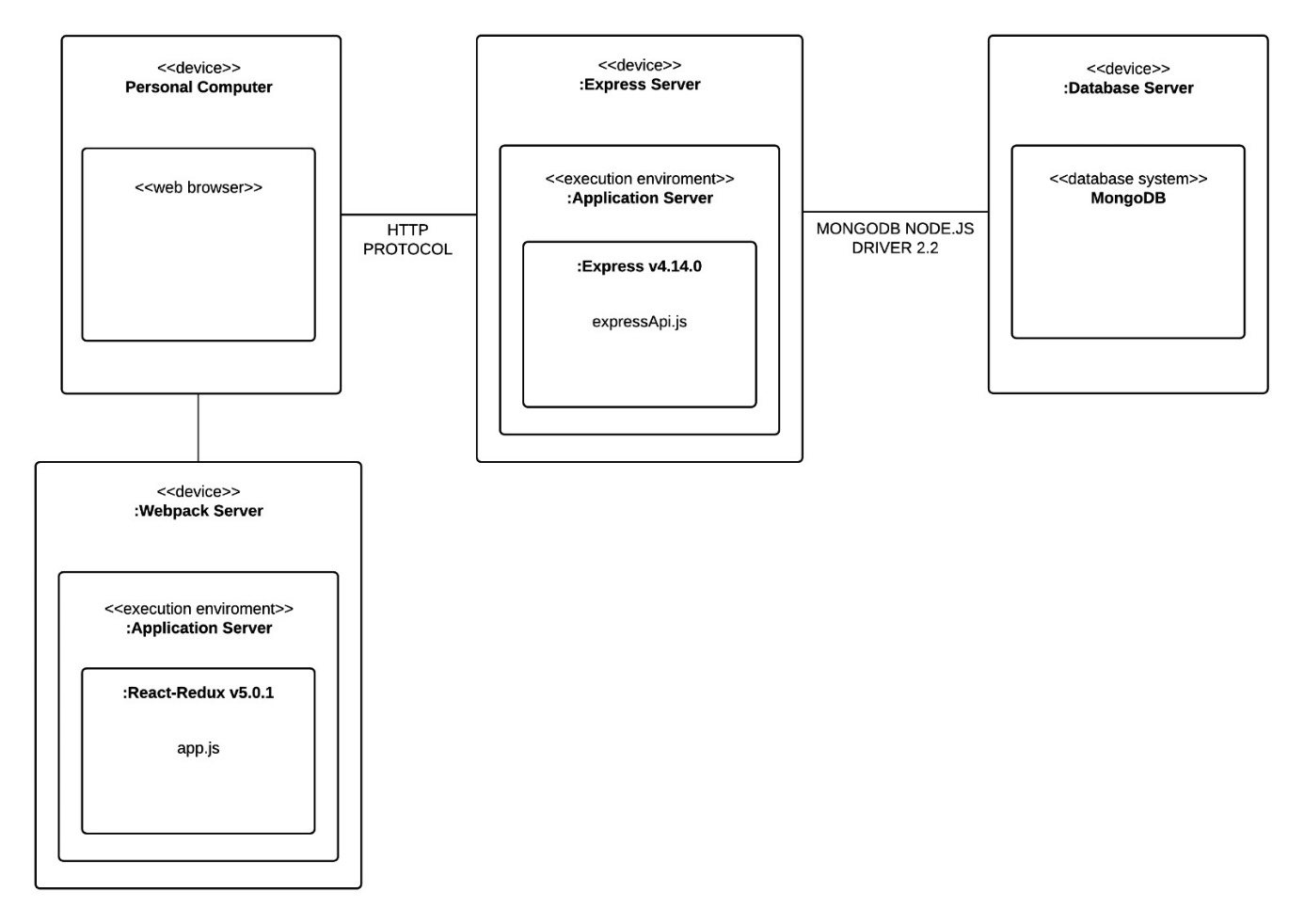
would be on an actual server that is available on the web.

12.2Infrastructure diagram - online

# 13. Deployment

The purpose of this chapter is to explain the mapping between the software and the infrastructure.

## 13.1 Deployment diagram

Picture 13.1 shows us an UML deployment diagram. This diagram graphs the infrastructure and the software it inhibits. There are four different types of devices. The first device is a personal computer which contains a web browser. The purpose of the web browser is to access the web application which is located on the second device. This device is the Webpack server. This server contains all the client side oriented-code. The browser communicates with the third device through HTTP requests. The third device is an Express server. The Express server handles all the requests by communicating with the database server which contains MongoDB. This is done through the MONGODB NODE.JS DRIVER 2.2. After handling a request the Express server will send a response back to the browser which will render a page with the information it received.

13.1 Deployment diagram

# 14. Operation and Support

Starting and updating an application can be quite hard sometimes. Here you can find a quick guide on how to start and update the application in order to make it run on your machine. In this chapter is also the support section included.

The source code can be found on GitHub (<https://Github.com/HANICA/DWA-nj2016-P-MongoExplorer>).

## 14.1 Starting Mongo

For the application to work you need to have MongoDB running. Use your terminal to start the MongoDB server on your local machine.

## 14.2 Starting the application

In order to start the application, two things have to be started: the application and the Webpack dev server, see below how to start them. To start the application, you need to have Node installed (<https://nodejs.org/en/>). Node has a built-in package manager called ‘npm’. The npm command will be used when a dependency needs to be installed.

### 14.2.1 Application

In the project folder navigate to ‘Implementation/Clientside’. Open the terminal in this location and type npm install. This will download every module you need for the application to run. After the load bar is removed from the console, the application is ready to start. In order to start the Webpack Server you need to type npm start, after a short period of time the console gives you webpack: bundle is now VALID. This means the server successfully started and will be running at localhost:8080. The state of the application is being stored in local storage. After something has been changed in the state of the application, the local storage in your browser (in your browser -> developer tools -> local storage) needs to be deleted in order for the application to work.

### 14.2.2 NodeJS server

Navigate to the project folder. Once you’re in the project folder navigate to ‘Implementation/Serverside/src’. First you need to install all the required libraries, this is done by opening the terminal in the location you’re in and type npm install. In this location open the terminal and type node expressApi to start. If the console tells you on which port the server is running, the server has successfully started.

## 14.3 Updating the application

If new modules where added and saved in the package.json, you have to download the newly added modules by navigating to ‘Implementation/Clientside’ and type npm update. This will download every module which is not installed yet. After updating, the application can be started as described in 14.2.2.

## 14.4 Overall support

The support for this application will be provided by the group of students that is working on the application at that time. For now, until week 10 of this semester, the support will be provided by ‘De Boterbloempjes’. After this period, this group will not be responsible and available for the support of the application.

### 14.4.1 Support of external components

There are many external components used for building this application. Most of the components were designed by Ant.design. Because these components have not been built by the development team they will also not be responsible for the support of these components.

## 14.5 Testing

Several tests have been written to ensure that the application functions the way it should function.

The tests can be found in ‘Implementation/Test’. In order to run the test, Selenium should be installed using npm. To install Selenium, open your terminal anywhere and type npm install -g selenium-standalone. After this, type selenium-standalone install.

Another thing that needs to be installed is ‘Mocha’. Install Mocha by opening the terminal and type in npm install -g mocha.

After Mocha and Selenium are installed they need to be started. Start Selenium by opening the terminal in ‘Implementation/Test’ and type in selenium-standalone start to start the Selenium server.

To start the test itself, open a new terminal in the same location and type in mocha. This will run the test; the results of the test will be displayed in the terminal.

# 15. Decision Log

There have been made several decisions that had a major impact during the development of the software, the most important will be explained below.

* **Ant.design**. Ant.design is a website/library that offers many ready to use components that applied to our application. For the consistence and looks we have chosen to use as many as possible components from Ant.design. We have done research to components that were able for us and the result of that research was that Ant.design offered most of the components that were needed for the application.
* **MongoDB.** There has been chosen to only support MongoDB as database management system for now. MongoDB was a requirement by the product owner and we have done research to find out whether there were other databases that we could use in our application. The results of this study were that there are indeed database management systems who look a bit like MongoDB, but are quite difficult to learn in such a short period. So, we decided to skip the idea of creating different API’s for multiple database management systems, and instead design our application in a way that people can build their own API for other databases. Because of this research the name of the application was changed to ‘UniDocExplorer’, but after finding out that time was running out and adding a second database management system didn’t add any new functionality we decided to change the name back to ‘MongoExplorer’.
* **User interface.** We have chosen to build multiple versions of certain components to give users the option to create their own user interface. The components that are included in this decision are the finder view, tree view and different themes. The reason behind this decision is that the users of the application have different background and knowledge of databases, some want a global overview and some might want a very detailed view.
* **JSON-view.** For programmers, we wanted an alternative view that shows the structure of a document. For that reason, we have created the JSON-view, so they can see the underlying structure of documents. The JSON-view gets created with the data that comes from the client and not from the server. The reason behind this is that we already have the data (for the tree structure) and it would be a waste of performance if the data would be requested again from the server.
* **Tabs.** We have created a tabs bar so that different collections can be opened simultaneously. The reason behind this decision is that users want to be able to work in different collections at the same time without losing their current work.
* **Local storage.** We have chosen to use local storage so that the work of the user gets stored and doesn’t get lost when the browser gets refreshed or closed. Another advantage of local storage is that the connection gets remembered and the user doesn’t need to make it repeatedly when closing or refreshing the browser.
* **Breadcrumbs.** When you have opened a collection and either have the tree view or the finder view, it’s difficult to know in what database and on what connection you are. Therefore, the breadcrumbs were added, so the user can keep an overview of the connection, database and collection he/she is currently working in.
* **Pagination.** There has been added a pagination component so that the user can choose the number of document he/she wants to be displayed at one time. The pagination is being managed by the server instead of the client for optimal performance.
* **Multiple files upload.** It is possible to upload multiple JSON-files at once. The requirement by the product owner was that only uploading of one file should be possible, but this has been expanded to multiple files for user friendliness and to save valuable time and work.
* **Delete.** To make the MongoExplorer a ‘data manage application’ instead of only a ‘data view application’ we’ve added the opportunity to delete files. This combined with the upload and update parts makes this application way more usable then without these functions.
* **Update.** For the same reason as the delete option, the update option has been added. This feature makes your database more manageable. Without this functionality, the application would be less useful, that’s why it has been added.
* **Multiple colors.** The product owner wanted to have the opportunity to change the color scheme(theme) of the application. Ant.Design, the library we’ve used, does offer a simple way to change the complete color scheme of the application, but because there are also components from other libraries and components we’ve created our own we decided to add this functionality ourselves. We have chosen for a number of colors the user can choose between, the colors have been picked to work well with the current(white) background.

# Literature

Boes, M., Tijsma, L., Theunissen, T., Leer, S., Haenen, P., & Holwerda, R. (2015-2016). Software guidebook. Arnhem, Gelderland, Netherlands.

Meier, J., Hill, D., Homer, A., Taylor, J., Bansode, P., Wall, L., . . . Bogawat, A. (2009, Oktober). *Chapter 16: Quality Attributes*. Opgehaald van msdn.microsoft.com: https://msdn.microsoft.com/en-us/library/ee658094.aspx