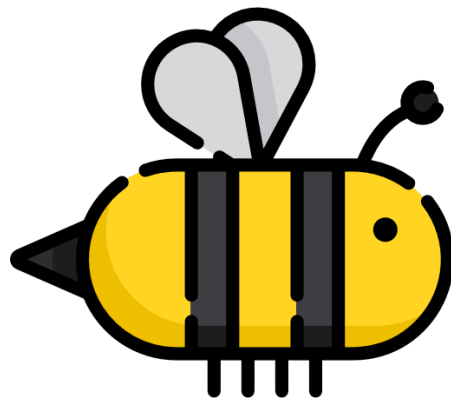


Status Update Report

Microservice 3 (MS3)



Busy Bee

Team 301

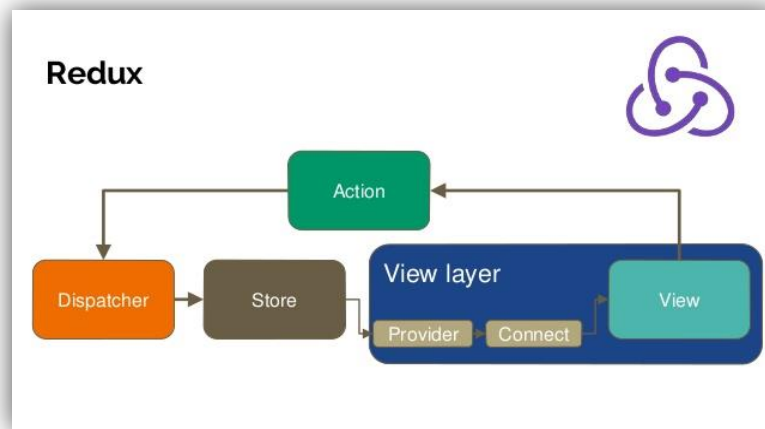
**Mersi Stafa
01276603**

The core design decisions of your Microservice: How does the service's external interface look like? How can your colleagues use your service? Which protocol(s) and technologies are used?

For the User Interface was used on ReactJS and the Redux framework.

React is an open-source JavaScript library for building user interfaces and allows the creation of applications that use data and can change over time without reloading the page. It aims primarily to provide speed, simplicity, and scalability. It processes only user interfaces in applications. This corresponds to View in the Model-View-Controller (MVC) pattern, and can be used in combination with other JavaScript libraries or frameworks.

Redux is a predictable state container for JavaScript apps. It helps you write applications that behave consistently, run in different environments as client, server or native which offer an easy way of testing. It also provides live code editing combined with a time traveling debugger. It is suggested to be used with React as in this case.



Here is a scheme about the workflow of both ReactJs and Redux. The functionality of Redux revolves around a “store” where the application lays. There is no direct way to modify this store but through reducers. The store contains multiple reducers. Every reducer contains commands and states. To trigger reducers it is needed to dispatch

actions. So when changing data, a dispatch of an action happens. When retrieving data, is used the state of the store.

On the other hand, it will be used Webpack. Webpack is an open-source JavaScript module bundler. Webpack takes modules with dependencies and generates static assets representing those modules. Webpack works best with NPM. The configuration file was taken from: <https://github.com/alicecoding/react-webpack-babel> because it pretty complicated to configure by itself.

Webpack allows hot redeployment. It means that as soon something is changed in code, it automatically will be applied on the development version of the latest build.

The Webpack-dashboard will be loaded which shows if the code can be compiled and checks if there are some syntax errors in the code. Hot redeployment happens by comparing bytes of changed files. The screenshot on the next page shows exactly the

issues if there are any errors.

At the end, everything is wrapped into Webpack meaning that it will be controlled by it. The used protocol is http using Rest Clients which are making calls via the http protocol to reach the other implemented services from the other team members.

Your thoughts on the service's deployment: How are you deploying your Micro-service? How can it be reached via the network (i.e. IP/hostname and port(s))?

Before deployment, the necessary installation of modules and libraries should be done. First of all, fetching all node modules is a must. This is done by the command: npm install.

For production purposes it can be used: npm run build, which in this case all React Components will be compiled in the simple JavaScript and HTML files. They will be saved in the public folder and the data can be deployed in any server.

The development version is done by: npm run dev. In this case, the screen with the detailed information about deployment and current build will be opened.

The screenshot shows the Webpack Dev Server interface during a successful build. The top panel displays a log of the build process, including file sizes and build status for various assets. The bottom panel shows a table of installed modules and their sizes, along with a table of assets.

Log

```
[./node_modules/webpack/hot/emitter.js] (webpack)/hot/emitter.js 75 bytes 0 [built]
[./node_modules/webpack/hot/log-apply-result.js] (webpack)/hot/log-apply-result.js 1.27 kB 0 [built]
[0] multi (webpack)-dev-server/client?http://0.0.0.0:3000 webpack/hot/dev-server react-hot-loader/patch ./src/index.jsx 64 bytes 0 [built]
[./node_modules/webpack/hot/log.js] (webpack)/hot/log.js 1 kB 0 [built]
[./src/components/todo/todo-main-component.jsx] ./src/components/todo/todo-main-component.jsx 1.65 kB 0 [built]
[./src/index.jsx] ./src/index.jsx 1.32 kB 0 [built]
[./src/store.jsx] ./src/store.jsx 859 bytes 0 [built]
[./styles/index.scss] ./styles/index.scss 1.08 kB 0 [built]
+ 547 hidden modules
Child html-webpack-plugin for "index.html":
  Asset      Size  Chunks  Chunk Names
  index.html  1.48 MB           0
[./node_modules/html-webpack-plugin/lib/loader.js!./src/template.html] ./node_modules/html-webpack-plugin/lib/loader.js!./src/template.html 996 bytes 0 [built]
[./node_modules/lodash/lodash.js] ./node_modules/lodash/lodash.js 540 kB 0 [built]
[./node_modules/webpack/buildin/global.js] (webpack)/buildin/global.js 488 bytes 0 [built]
[./node_modules/webpack/buildin/module.js] (webpack)/buildin/module.js 495 bytes 0 [built]
```

Status

Success

Operation

Idle (12s)

Progress

100%

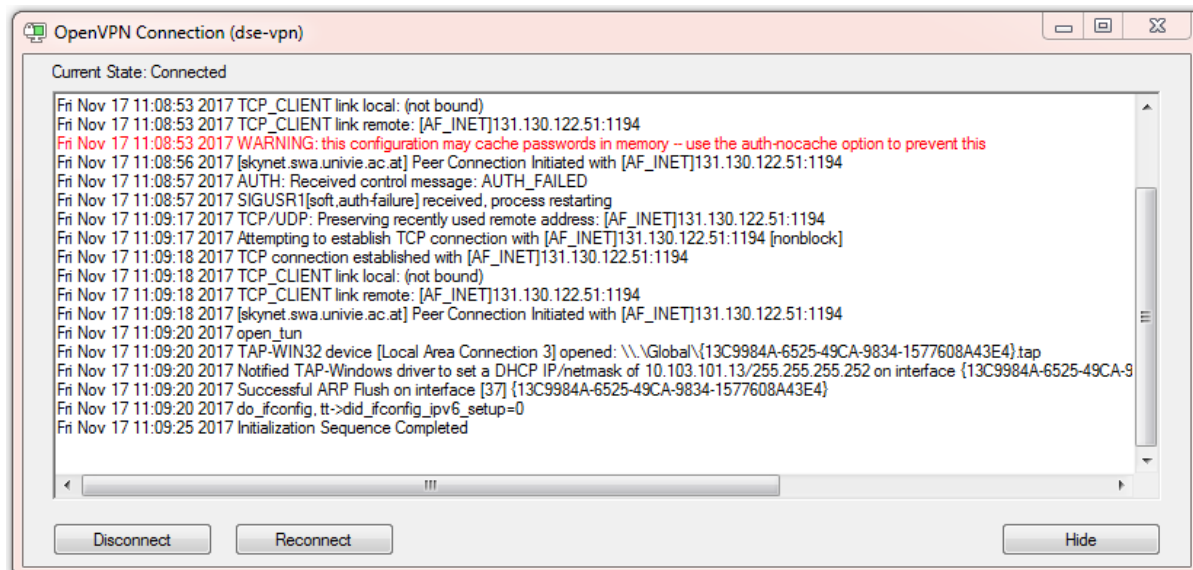
Modules

Name	Size	Percentage
react-dom@15.6.2	523.85 KB	36.7%
sockjs-client@1.1.4	176.47 KB	12.3%
react-router@3.2.0	121.29 KB	8.49%
history@3.2.0	46.82 KB	3.6%
<self>	74.47 KB	61.4%
lodash@4.17.4	186.2 KB	7.43%
react@15.6.2	79.49 KB	5.56%
html-entities@1.2.1	57.42 KB	4.02%
react-redux@5.0.6	39.36 KB	2.75%
hoist-non-react-statics@0.6	1.95 KB	4.96%
<self>	37.41 KB	95.8%
core-js@2.5.1	35.44 KB	2.48%

Assets

Name	Size
bundle.js	4.23 MB
index.html	676 B
Total	4.23 MB

The UI is reachable on <http://10.103.101.13:3000/> if the machine is reachable. The screenshot also shows the OpenVPN connection that is established. In this case the endpoint will be reachable from all team members.



Ideas for testing and presentation: How are you going to test your service when other services you have to depend upon are not available? How can you demonstrate the functionality of your service in such situations?

In such a case, there will be some “dummy data” available. A “dummy” user will log in with the given credentials: Username: “dummy” and Password: “dummy”. It will bypass the user service and the to do list service as well so that the own service can be tested. The data will be fetched from JSON file which is stored on frontend side. This test is possible on the local machine or in the case of a connection via VPN in which it will be accessible by a static IP.

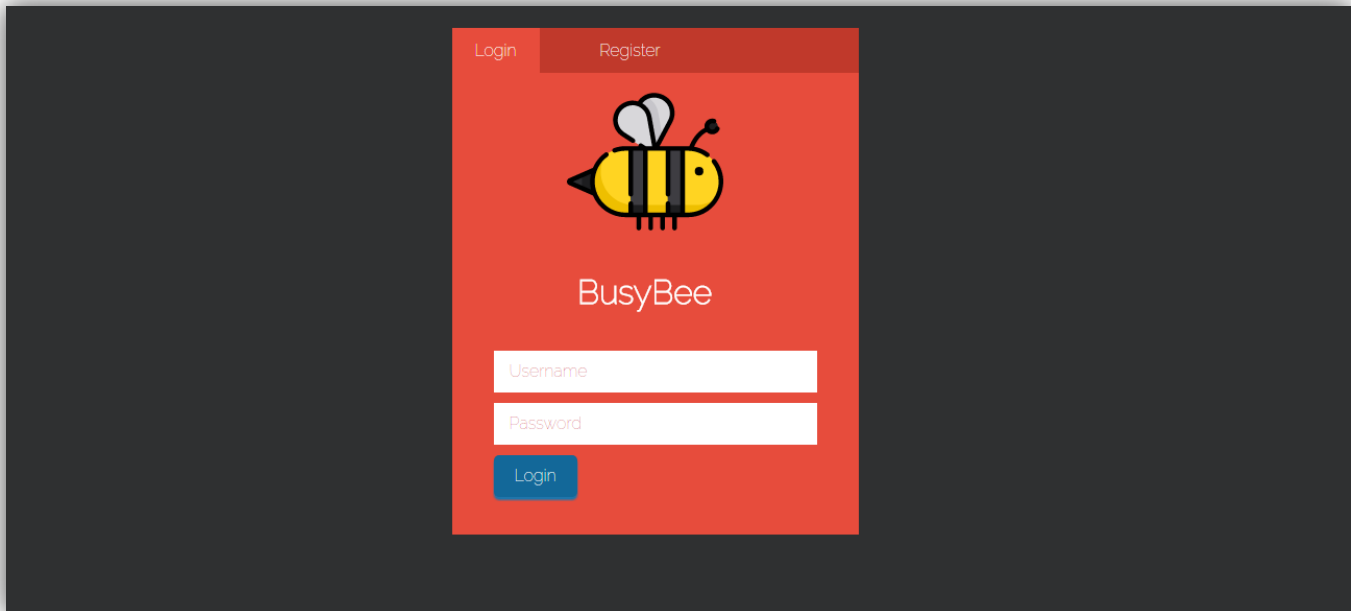
The current state of your implementation: What has been achieved so far? What's missing? What works? What doesn't?

The implementation is not completed but so far it has been achieved the following points:

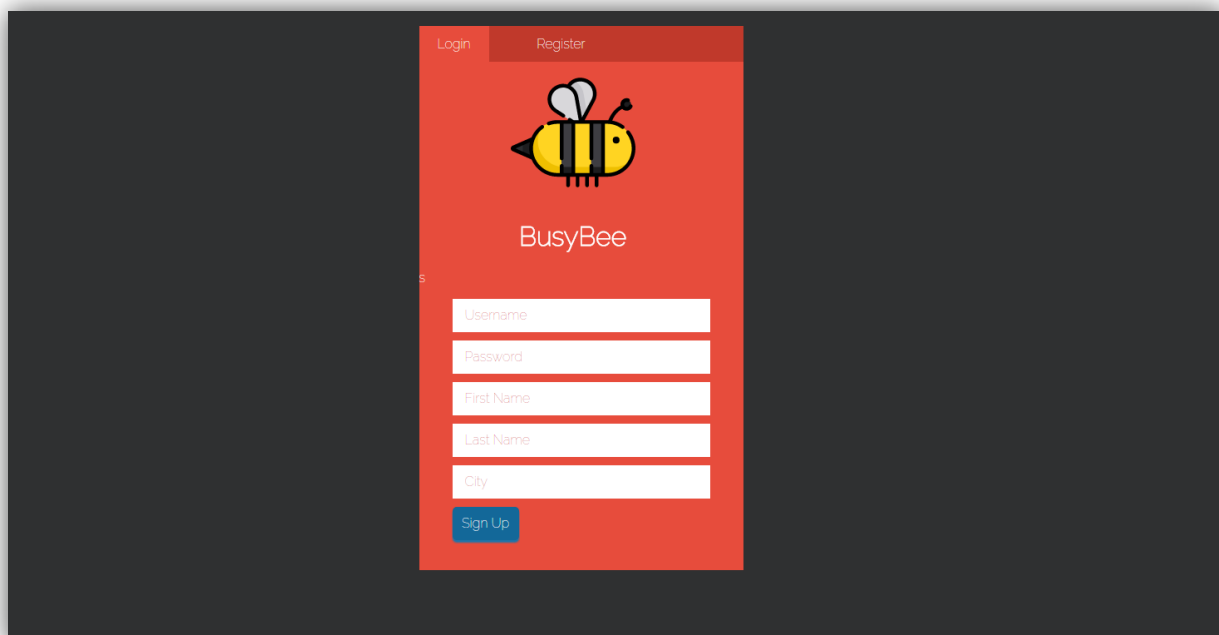
- 1- User Interface for login and response handling.
- 2- User Interface for registration and response handling.
- 3- Validation method.
- 4- User Rest Client which calls the first implemented Micro-service.

The screenshots below show the actual user interface which is until now implemented in the application.

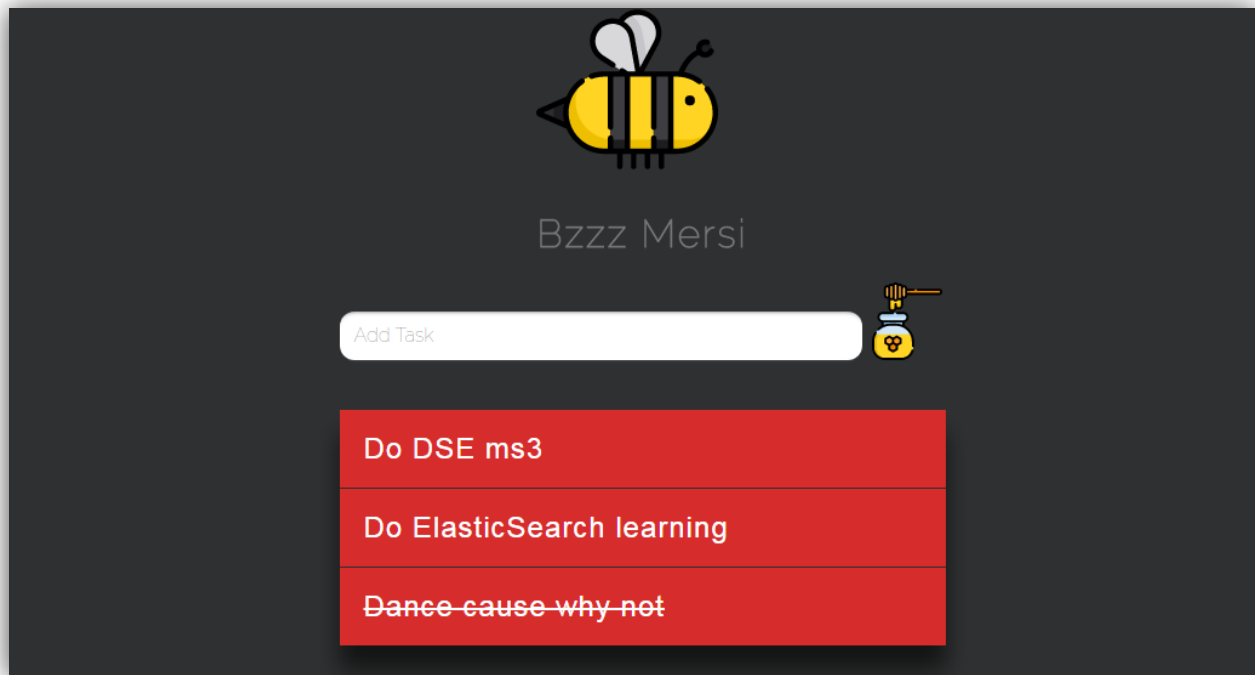
Log in: When the user opens the application, the login form will appear. By writing the right username and password, the user will be redirected to his lists.



Registration: To register in the application the user needs to give some information. The needed information data are: username, password, first name, last name and city. After the registration is complete, the user will be automatically logged in and redirected to the dashboards of the lists.



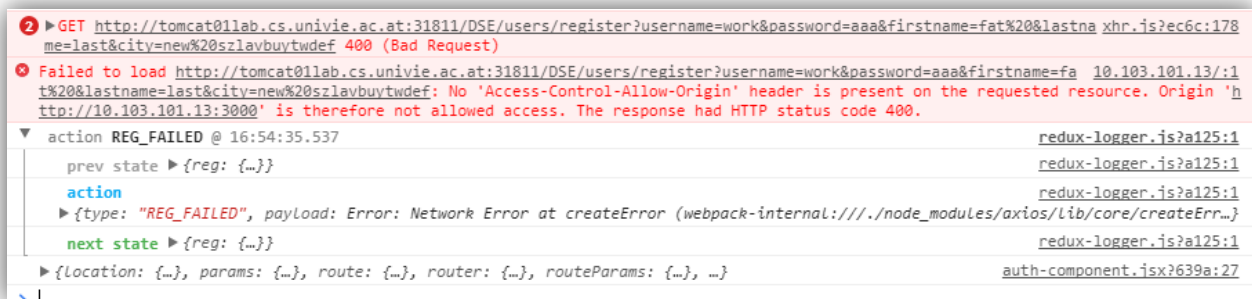
To Do List: The to do list page looks like this. On top of the page, a message appears which salutes the user: in this case “Bzzz Mersi” in which Mersi is the name of the logged in user. After that the user may add tasks in the empty space by simply writing them and clicking the “honey” button to add them. Below that, each added task will appear. After completing a task, the user may cross it off simply by clicking on it or make it appear once more with another click.



Some missing points of the application would be:

- 1- the category choice should be also added (where the user can give in which category he wants the tasks to be into),
- 2- the creation of multiple to do lists (a user may have more than one to do list),
- 3- removing, creating or managing lists (to do list rest client).

Something that does not work would be that unfortunately the user service doesn't accept requests except a localhost.



```
2 GET http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/register?username=work&password=aaa&firstname=fat%20&lastna xhr.js?ec6c:178
me=last&city=new%20szlavbuytwdef 400 (Bad Request)
Failed to load http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/register?username=work&password=aaa&firstname=fa 10.103.101.13/:1
t%20&lastname=last&city=new%20szlavbuytwdef: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'h
ttp://10.103.101.13:3000' is therefore not allowed access. The response had HTTP status code 400.
action REG_FAILED @ 16:54:35.537 redux-logger.js?a125:1
prev state ▶ {reg: {...}} redux-logger.js?a125:1
action ▶ {type: "REG_FAILED", payload: Error: Network Error at createError (webpack-internal:///./node_modules/axios/lib/core/createErr...} redux-logger.js?a125:1
next state ▶ {reg: {...}} redux-logger.js?a125:1
▶ {location: {...}, params: {...}, route: {...}, router: {...}, routeParams: {...}, ...} auth-component.jsx?639a:27
```

Suggestions of solving this issue:

Add Access-Control-Allow-Origin accessible to all.

Sources:

[https://en.wikipedia.org/wiki/React_\(JavaScript_library\)](https://en.wikipedia.org/wiki/React_(JavaScript_library))

<https://redux.js.org/>

<https://www.slideshare.net/binary-studio/academy-pro-react-js-redux-tooling>

<https://en.wikipedia.org/wiki/Webpack>

<http://www.sohamkamani.com/blog/2017/03/31/react-redux-connect-explained/>