# Mini project for WebEng 17-18

Table of Contents

[Mini project for WebEng 17-18 1](#_Toc501627872)

[1.1Pedagogical goals and organization 1](#_Toc501627873)

[1.2Evaluation 2](#_Toc501627874)

[1.3Final report structure (to be respected) 2](#_Toc501627875)

[Final Report 3](#_Toc501627876)

[2.1 Introduction 3](#_Toc501627877)

[2.2 Development Platform 3](#_Toc501627878)

[2.3 Development Environment 4](#_Toc501627879)

[2.4 Specific parts of our application 4](#_Toc501627880)

[2.4.1 Crawler 4](#_Toc501627881)

[2.4.2 Instant Data Retrieval 5](#_Toc501627882)

## Pedagogical goals and organization

The goal of the mini project work is to apply technological skills presented during the lectures and to also make a survey activity on newest technological platforms in order to share them with the whole class. The final deliverable concerns the development of a proof-of-concept application using the technologies presented at the intermediary presentation and a final report:

* **Proof-of-concept application**
* **Final report**
* **USB Key** with a copy of all documents: report + user guide report + installation report + final project presentation + source code + eventually external resources (software, articles, etc.). The USB key has to be provided to the supervisors the day of the final presentation

## Evaluation

Final mark of the WebEng Module = (70% oral examination) + (30% mini-project)

## Final report structure (to be respected)

1. Introduction
2. Development Platform
3. Development Environment
4. Specific parts (Presentation, Logic, Database, etc.)
5. Code documentation tools
6. Installation guide
7. User guide
8. Conclusion

# Final Report

## 2.1 Introduction

At the beginning of the semester we were assigned the subject “University Sport” for this mini-project. We quickly chose “UniSport” as the name for our project, realising only later that this name was already being used.

It was a bit unlucky that in our group two out of three members were rather unexperienced programmers, not having done anything related to web development before.

## 2.2 Development Platform

Due to our unexperienced nature, it was quite difficult to choose a certain development platform, because we couldn’t really recognise advantages or disadvantages of the different platforms. After some research, we decided to go with the full original MEAN-stack: Node.js, Express.js, AngularJS and MongoDB. We did so because we thought this platform would offer us the most documentation and guidance. It was only after we had started, that we realised most people were already working with Angular 5. We decided to keep using AngularJS, rather than starting over with the more recent technology.

At the start of this project we chose to use the MEAN-stack because everything is in the same language, it is well documented and it is said to be fairly simple. Now that we have been using these technologies for the past months, we can look back and shine some light on them from our own experiences.

First of all, we liked the MVC structure. It was quite easy to understand and get a hang of, even without any prior experience. We think it’s very useful to produce a crude application in a short time. But it is also very scalable for fast growing programs. As expected we found a very large community for all of the MEAN technologies, which was a very big help for us. This was especially the case for AngularJS. We think that the package manager of Node.js, npm was very helpful and easy to use, especially when the alternative is Windows CLI. On the other hand, we experienced that having all parts of your code in the same language can become a bit confusing. We hadn’t thought of that in the beginning. This might be something to think about for more experienced coders but, this complication doesn’t weigh up against the ease of only having to learn one language on a short notice. Another consequence of everything being in JavaScript, is that there is a lot of overlap between technologies. That is, pieces of code could achieve more or less the same result in different places. When this is described as an advantage it is called “flexible coding”. We also perceived this as confusing from time to time. Because there are so many ways to do things, it becomes hard to find the ‘correct’ one. Lastly, we found that the AngularJS directives were a bit more difficult to grasp than we had hoped for. Maybe this is clearer in the later Angular versions.

Other technologies we used next to or on top of the MEAN-stack are: Heroku to deploy our app, BeautifulSoup to do a crawler for the real data, Bootstrap for the frontend and Nodemailer being used in the contact form.

We can recommend Heroku if you want to deploy anything quickly; it was very easy to set up. BeautifulSoup was also very user friendly. To get the data from the respective websites, there wasn’t much coding required. We used Bootstrap on top of the MEAN-stack to give our program a bit more shape. We found it to have a nice look and it was quite easy to understand and use, thanks to the very rigorous documentation provided on their website. To be able to receive messages from our users, we set up a contact service using Nodemailer. It wasn’t too difficult to set up and it sends all the messages left by user directly to our e-mail accounts.

## 2.3 Development Environment

We used a few different environments. Concerning the OS, we worked with both Windows 10 and Mac OS X High Sierra. The overall consensus about this is that the CLI available in Windows does a very poor job, compared to the CLIs available in Mac OS X or Linux distributions.

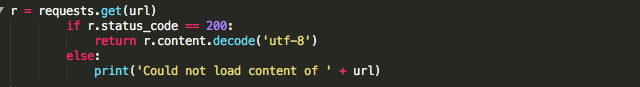
Further we used WebStorm and Sublime Text to code in. WebStorm is a very useful IDE for web development with the MEAN-stack because of its focus on JavaScript. The auto completion and style aid are nice for people who like that. Unfortunately, we weren’t able to figure out how we can use the debugger for our application with Node.js. Sublime Text is of course a very useful text editor that almost everybody knows.

## 2.4 Specific parts of our application

### 2.4.1 Crawler

The first element of our application we want to highlight, is the fact that we used the real data from the unisport-websites of the three universities. At first, we tried contacting the people responsible for the unisport-websites, to try to get access to the data in a simple way. Because they did not want to cooperate by giving us access to an API or in another way, we had to improvise. That is how the crawler-idea originated. For this we joined forces with group 2, who also worked on a unisport-project. Luca Liechti and Joel Niklaus developed a crawler which visits the unisport-websites, scans them and then downloads the data and outputs it as a JSON array of courses. The library that was used to do this is Beautiful Soup (<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>). This library is based on Python, so python was used as programming language for the crawler. The full code is available on GitHub (<https://github.com/JoelNiklaus/UniSport-Crawler>). As mentioned before, in our experience the library was pretty easy to use and very practical. Therefore, our codebase could be kept quite small. The entire crawler code for one unisport page, is only 100 lines on average. Sadly, the final version was finished a bit too late to be able to include the data from Bern and Neuchatel. Our application now displays the present data of the courses offered by the University of Fribourg.

Below we have added some code snippets from the crawler. This first example shows how the request is made:



The second snippet shows how the links to the courses are extracted:



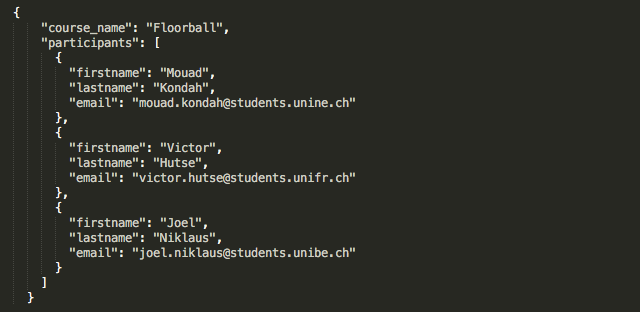
### 2.4.2 Instant Data Retrieval

The second element of our application we want to draw attention to, is the instantaneous data retrieval from our MongoDB database. To realise this, we used Mongoose. Mongoose is a very useful technology to have easy instantaneous access your database, a job which would be rather tedious to do without it. It acts as a gateway between the Node.js server and the MongoDB server. Mongoose uses schemas to model the data it handles.

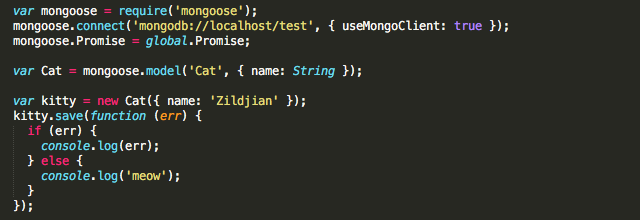
We imported the day from the crawler for Fribourg as a JSON file and created a collection for it in our MongoDB database. Afterwards, we created a data model which can be used to query the course data from the Fribourg collection. Of course, other data models were also created for other actions. This is the model we created for saving reservations:



This would then be how the saved data looks like:



Just to shortly present how this works, this a snippet taken from the mongoose website, below:



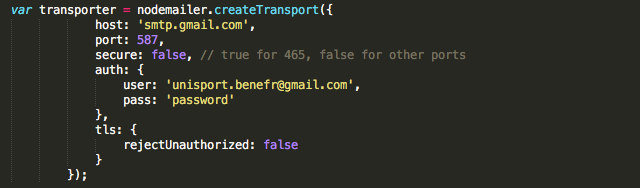
In the first block of code we import the module and set it up to be connected to the database. In the middle block, we create an instance of one of the mongoose data models we created. The last block then shows how we can write something away to our database.

### 2.4.3 Contact Us

The last part of our application, we will shortly mention, is the contact page. For users to be able to contact the administrators of the UniSport web application with any remaining questions, we set up a small contact page. In the front-end this is implemented as a form, which is handled by AngularJS in the controller. The data retrieved from the form is the sent with a HTTP POST-method. This post uses the module Nodemailer.

Nodemailer uses an e-mail address and smtp-server you provide to send the data to another e-mail address you wish to send it to. This is also a very easy and practical module.

First, we create a transporter instance:



In this case we only use an unsecure connection, because it is just a proof of concept. When implementing this in a real-life case, it would be necessary to use OAuth2 identification. After creating the transporter instance, we the send the mail, using it:



## 2.5 Code Documentation Tools

## Concerning code documentation, we added comments in the code where we saw fit. These short comments explain what a certain function does, why we take certain steps or what something is used for. We believe that proper naming of variables and functions is also a critical part of code documentation, the same is true for code structure and format.

We tried to write clean, readable code with enough comments to help a reader, but the overall focus was on experimenting with functionalities.

## 2.6 Installation Guide

## 2.7 User Guide

## 2.8 Conclusion

Overall, we are pretty happy with the outcome. Our major advantage was choosing such well-documented technologies. This allowed us to look up an endless stream of examples, tutorials etc., which enabled us to make this application even without experience.

The result is still more of a proof of concept than a finished web application. But if we look back at the goals we set for ourselves at the beginning, we think we did pretty well. We struggled with the languages from time to time. But in the end, we have a proper front- and back-end with the functionalities we set out to create and maybe even a bit more. We would have liked to create a home page that is a bit more attractive, however.

If we look back at wat we learnt during this project, we conclude that both Mouad and Victor now have a proper idea of how web development works. They gained experience with new languages and technologies and have a greatly improved view on the possibilities, the restrictions and the structure of web development. Joël, who already had a stronger basis concerning web development, learnt more about the possibilities and restrictions of certain technologies. On top of that, he gained experience leading a (small) developing team.