

Web3 Project

To practice what you have learned in *Web Technologien 3*, you will do a project instead of a series of labs. You can do this exercise in a group. Please form groups of 2-3 students on your own. This project is weighed 20% for the final grade.

The goal

Create an Issue Tracker with:

- Responsiveness in look and feel
- Persistent Issues (localStorage)
- Load from and save to a server (REST)
- Back-end to facilitate the above (Optional)
- Write a test-suite to prove correctness (Optional)

Milestones

These are the iterations in which you will create the Issue Tracker. They have been chosen to:

- Be aligned with the theory in the lectures
- Make the scope of the respective technologies explicit to facilitate a better understanding
- Represent a natural succession of steps for a project of this scope

The milestones will each be explained in more detail further down in this document.

1 HTML/CSS Prototype with Web Server

(due: Week 4)

2 Logic, RiotJS, localStorage

(due: Week 9)

2B Tests (Optional)

(due: Week 9)

3 RESTful API

(due: Week 11)

3B Custom back end service (Optional)

(due: Week 14)

4 Deployment

(due: Week 13)

Another Issue Tracker?

The Web3 project might sound familiar to the project of Web2 and this is no coincidence. In fact, the Web3 project builds perfectly on top of your knowledge of Web2.

During your Web2 project you have built an Issue Tracker. On the one hand, it was very simple (only one or two entities), but on the other hand it was already very complex. You might have asked questions like:

- It feels like this jQuery and JavaScript code is just a tangled mess. How would this scale on a more complicated project?
- How should I structure my project?
- Is there a good pattern to build a UI?
- I'm missing the terminology to talk about different aspects of the program with other programmers. Is there a better way?
- Isn't there a better way to document REST APIs than using prose?
- How does that REST API back-end work, anyway?

During Web3, we will cover all of these (and more) questions. Think of building a second Issue Tracker as a Code Kata - you'll practice the same task, but with more knowledge (Web3) and experience (your last project). This is something that we programmers rarely do.

Usually we learn something and then apply the things directly on the job - and make mistakes there. It's good to separate practice from the actual profession. In the web domain, an Issue tracker is a wonderful Kata. It has enough complexity and usability requirements to resemble a non-trivial real-world project, yet it can be solved in all kinds of different ways.

Mockup

This is a mockup to show the visible feature set of the Issue Tracker. It is not meant to be a screen design, you are free to design the application as it suits you. For example you are free to display the priorities of the issues in a way that you think is good user interaction - you could color them differently, you could put them in different 'priority groups' or do something else completely.

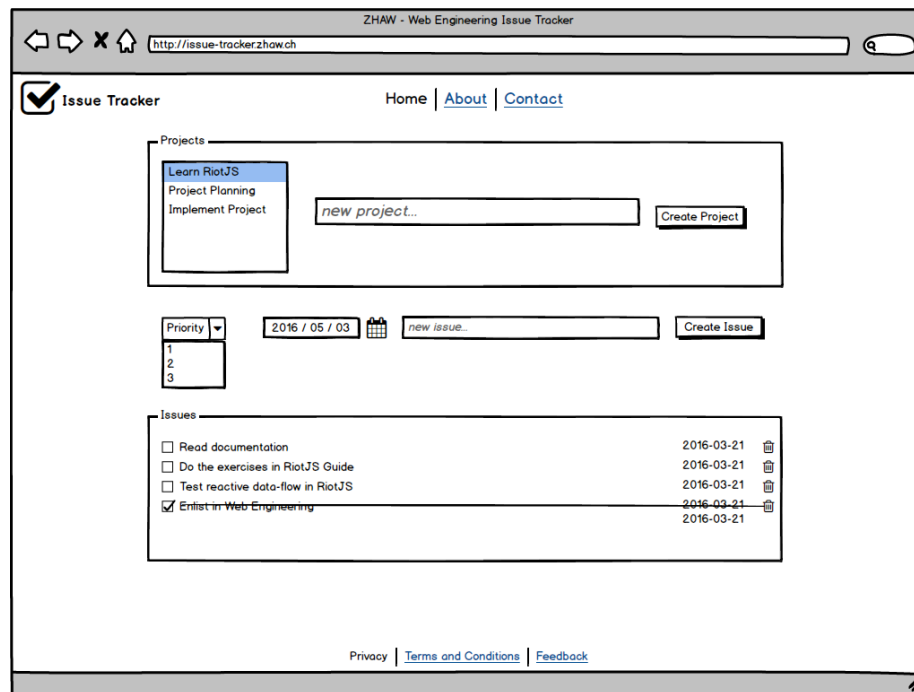


Figure 1: Mockup with feature set

Milestone 1 - HTML/CSS Prototype with Web Server


When starting with a new front end application, it is sensible to create the Markup first and style it before you start adding any client-side logic.

Looking at the Mockup and adding your own ideas of how the Issue Tracker should be structured, write the Markup and style it using a CSS style-sheet which should reside in a separate file.

In this milestone, there is no functional logic and little writing of JavaScript required.


As you probably know from Web1/Web2, it is quite a lot of work to create a web-site from scratch which looks good. The default styling of the browsers is very basic and not necessarily pleasing. Therefore, manual styling of every component would be required which is hard to do properly, because you would first need to figure out a style-guide to make the components look related. Now, if you wanted your web-site to also be responsive and be displayed properly on mobile devices with different screen sizes, the matter would only get more complicated.

Once upon a time, only a couple years ago, this process had to be undergone for every single web-site. Realizing this pattern and tediousness, lots of developers have started to create Frameworks to make this kind of development easier.

If you do not want to come up with your own responsive design, you can implement your HTML with  Bootstrap. When done with a little care, it is likely that your app already looks good, is coherent and also works on mobile devices.

As the next step of this Milestone you will implement the first feature of the application. This is the user story for it:

As a user, when I create a new issue and I click into the date field, I want to select from a date-picker instead of entering the date by hand.

Fortunately, there is a vast set of finished software out there for us to use directly. You're not going to implement your own date-picker (unless you absolutely want to), but use a  jQuery UI Widget called Datepicker. On the jQuery UI web-site, you will find very good examples and documentation.

Finally, you will need to provide a web server that allows you to serve your static mockup and all the resources needed (images, stylesheets, javascript libraries). You can build on the one that was discussed in lecture 3 or 4.

Summary Milestone 1


- ☐ Implement HTML and CSS with Bootstrap or using custom code
- ☐ Use jQuery Datepicker for the issue date field
- ☐ Build a webserver in node that allows you to serve the prototype

Grading

Achieving the above tasks will be awarded with 2 points each. For details regarding grading, see the chapter on grading.

Milestone 2 - Logic, RiotJS, localStorage

In this Milestone, you will add the missing logic to make the Issue Tracker a functional product. To convey what is to be done, this Milestone is broken down into specific *Use Cases* that in turn are broken down into *User Stories*. The User Stories should help you guide along the requirements.

Implement this Milestone as a reactive Single Page Application using  RiotJS.

Use Case: Adding and mutating Issues within the project

- As a user, when I have selected a priority, given a date, entered an Issue title and have clicked “Create Issue”, I want the new issue to appear in the issue list.
- As a user, after I have created a new Issue, I want the input fields to be cleared.
- As a user, when clicking the check-box of an Issue, I want the ‘completed’ state of the Issue to be toggled.
- As a user, when clicking the ‘trash’ icon, I want the Issue to be deleted.

Use Case: Persisting data to the Browser

- As a user, I want the projects data to be saved to localStorage.
- As a user, when I refresh the browser (or close and re-open the tab), I want all previously entered data (that is projects and issues) to be loaded from localStorage for instant feedback.
- As developer, when creating a new Issue, I want a new UUID to be created and saved as `client_id`. It will later guarantee consistency between issues in the browser and back end. In the RESTful API (see below), you will see this `client_id` coming up again.

Grading

Achieving the above user stories will be awarded with 2 points each. For details regarding grading, see the chapter on grading.

Milestone 2B - Tests

This Milestone is optional!

When writing software and designing an architecture, using TDD can be immensely helpful, because it gives you an explicit moment in time to think about how your interfaces should behave.

Also, when writing non trivial software, complexity can be an issue. There's uncertainty about breaking existing functionality and there's an increasing fear of refactoring.

When writing software in a team, it can be hard to communicate about existing features and how specific parts of software behave. Some people try to address this by writing documentation. However, documentation has a tendency to be either incomplete, wrong or outdated. Having readable tests will clearly document your current state of functionality on from a UX or architecture standpoint.

Having tests (not necessarily all written in pure TDD fashion), are a great way to deal with these issues. By having a good test-suite you have turned liabilities into benefits. Therefore, we encourage you to write tests, but having them is not explicitly required as part of this project.

This is an example for some model and interface specs. Of course, with a project of this scope, it also makes total sense to write acceptance tests for your functionality from a UI or end-user perspective.

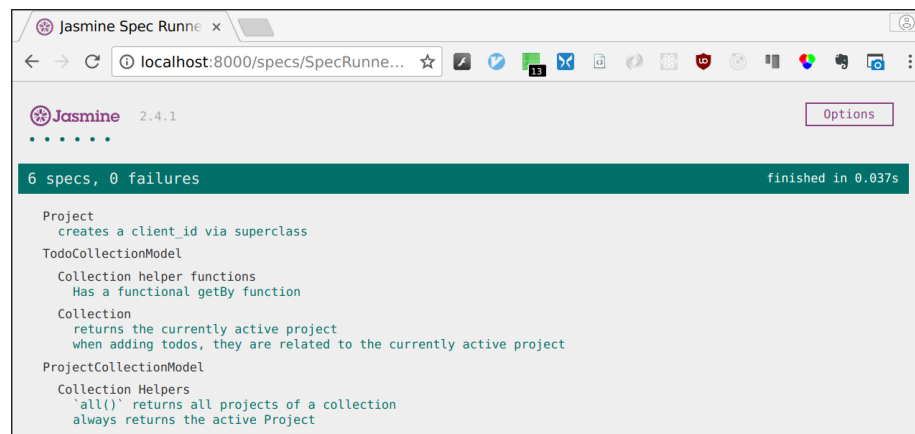


Figure 2: Model and interface specs

Grading

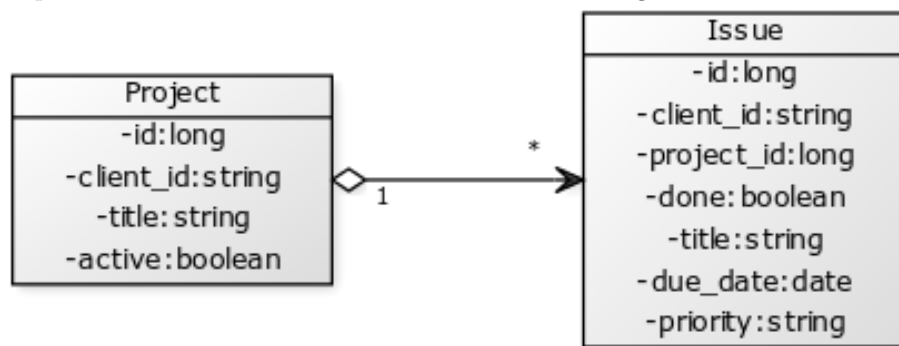
Writing a comprehensive test-suite (a reasonable amount of tests that check for reasonable things) will be awarded with 7 points total. For details regarding grading, see the chapter on grading.

Milestone 3 - RESTful API

Use Case: Persisting data to a Server

Many web applications offer persistence to a back end. The Issue Tracker will be no exception. However, you will not have to create a back end service with a database for yourself. We will use a modern approach and give you an API with which you can persist the applications data in a RESTful manner.

You will not have to touch the Database yourself, however it might help to know the schema. This is the UML diagram of the Database:



Note: Optionally, you can create your own back end service in NodeJS. See Milestone 3B for details.

RESTful API

Your Single Page Application now looks feature-complete on the client side. You are also persisting issues to the browser. However, you are not persisting issues to a back end, yet.

For this, you can use a reference implementation of the API that you can use described in the standardized Open API Initiative format here: <http://zhaw-issue-tracker-api.herokuapp.com/swagger-ui/index.html>. On this page, you can find the definition of the API and you can also directly test it out in the browser. The definition page is split into the “Issues API”, the “Project API” and an API to “Practice HTTP based services”. The latter is a good starting point to get a feel for how HTTP based APIs work if you want to practice a little bit before diving into using the Issues and Projects API to refresh your knowledge of Web2.

Swagger API Screenshot

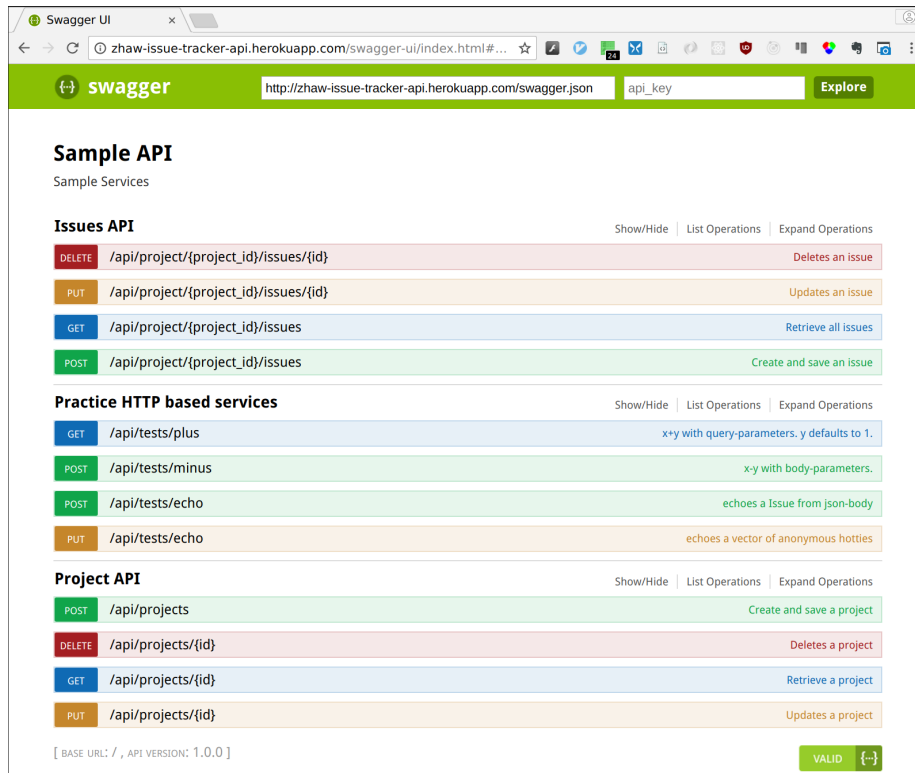


Figure 3: Swagger API

Screenshot: adding two numbers

Practice HTTP based services [Show/Hide](#) | [List Operations](#) | [Expand Operations](#)

GET /api/tests/plus [x+y with query-parameters. y defaults to 1.](#)

Response Class (Status 200)

Response Content Type

Parameters

Parameter	Value	Description	Parameter Type	Data Type
y	<input type="text" value="12"/>		query	long
x	<input type="text" value="123"/>		query	long

[Try it out!](#) [Hide Response](#)

Curl

```
curl -X GET --header 'Accept: application/json' 'http://zhaw-weng-api.herokuapp.com/api/tests/plus?y=12&x=123'
```

Request URL

```
http://zhaw-weng-api.herokuapp.com/api/tests/plus?y=12&x=123
```

Response Body

```
135
```

Response Code

```
200
```

Response Headers

```
{
  "connection": "keep-alive",
  "server": "undertow",
  "x-xss-protection": "1; mode=block",
  "x-content-type-options": "nosniff",
  "x-frame-options": "SAMEORIGIN",
  "content-type": "application/json; charset=utf-8",
  "content-length": "3",
  "date": "Sat, 12 Mar 2016 12:09:15 GMT",
  "via": "1.1 vegur"
}
```

Figure 4: Example request

Project User Stories

- As a user, when entering a Project title, I want a new Project to be created through the RESTful API.
- As a developer, after having created a Project through the RESTful API, I want to save the `id` in `localStorage` to the existing entry with the `UUID` so that in the future I can reference the correct Project.

Issues User Stories

- As a user, when creating an Issue, I want a new Issue to be created through the RESTful API in the scope of the current Project.
- As a developer, after having created an Issue through the RESTful API, I want to save the `id` in `localStorage` to the existing entry with the `UUID` so that in the future I can reference the correct Issue.
- As a user, when editing or deleting an existing Issue, I want this mutation to be reflected through the RESTful API in the scope of the current Project.
- As a user, when clicking the check-box of an Issue, I want the ‘completed’ state of the Issue to be toggled through the RESTful API.
- As a user, when clicking the ‘trash’ icon, I want the Issue to be deleted through the RESTful API.
- As a user, when reloading the page, I want the Project `id` to be loaded from `localStorage`, so that a RESTful request can be made to load the Issues from the back end.

Grading

Achieving the above user stories will be awarded with 1 point each. For details regarding grading, see the chapter on grading.

Milestone 3B (Optional): Write your own back end service

This Milestone is optional!

Up until now, you have used the reference implementation of the back end service to persist your data. To conclude building a complete modern SPA, you can now build your own custom NodeJS service with a SQLite or PostgreSQL database. If you want to deploy your service to Heroku, you will have to go for PostgreSQL.

Technical Task

Your task is:

1. To write your own RESTful API in NodeJS to include the same Issues and Projects API as the reference implementation above. Add this functionality to the NodeJS application that you already build in Milestone 1.
2. Make your SPA persist to this API.

Grading

Writing your own back-end service will be awarded with 7 points total. For details regarding grading, see the chapter on grading.

Milestone 4: Deployment

Your front end application is now feature-complete. It is time to put it on the Web, so that everyone can start using it!

Use-Case: Deployment

The goal is to deploy the current application on Heroku and have access to it on the Web.

Tasks

1. Read the Getting Started documentation of Heroku.
2. Use the NodeJS server from Milestone 1 to to serve your front end application.
3. Follow the Heroku guide and install the application to Heroku.

Note: Check out how port binding is done in the example application. During your local development, you will usually access the web application through a manually configured port. In most productive web applications that would be port 80. Heroku uses a special configuration which does this for you.

Expected result

- You have integrated your SPA into a NodeJS web server which you have deployed to Heroku.
- Everyone can use your application by accessing it via a URL.

Grading

Having deployed your web server with your front-end on Heroku, you will be awarded with 5 points total. For details regarding grading, see the chapter on grading.

How to hand in the project

For every reached Milestone, you will hand in your current version of the project. To hand in the project, please upload it to a sFTP server with the following credentials:

- Server: dublin.zhaw.ch
- Port: 22
- User: your_zhaw_user_name
- Password: your_zhaw_password

You can then test your upload in the browser: http://dublin.zhaw.ch/~your_zhaw_user_name/

For the respective milestone, please create a directory with the milestones as structure as such: **project_milestone_1** and **project_milestone_2**. Change into the correct directory and upload your submission.

For the milestones 3B and 4, create a ZIP file of the code and submit it in the same way. Please make this a standard PKZIP compatible file¹ and don't use any other software that sounds similar like 7-zip. Also create a **README.md** file that includes the URL to your application on Heroku.

¹On OS X, use the 'archive' feature in finder. On Linux, use the command line utility 'zip'. On Windows, use 'WinZip'.

Grading

- You will hand in the respective milestones as described above.
- Your code will be discussed in one of the following labs between the lecturer and you.
- If a milestone hasn't been handed in on time, but the functionality is implemented in a later milestone, you will receive 25% of the points of said milestone.
- You can implement an 'extra feature' to gain additional points. An 'extra feature' is a feature that makes the application behave more reasonably than is explicitly required the User Stories. At the minimum this could be input validation. It could also be something sensible like a better synchronization scheme between localStorage and REST. If you build an extra feature, please make sure to add a description in your **README.md** file. Also make sure to point it out when showing your project to the lecturer during the lab hours.

You can get the following amount of points per Milestone and requirement. These points will account for 20% of your grade for Web3, there is no separate grade for the project.

Total required	M1	M2	M2B	M3	M3B	M4	Extra Feature
45	6	14	7	8	7	5	5