WENG Project

To practice what you have learned in *Web Engineering*, you will do a project. This project is weighed 25% for the final grade.

The goal

Create an Issue Tracker with:

- Navigation
- Logo
- Priority
- jQuery Datepicker
- Layout with Twitter Bootstrap
- Persistent Issues (localStorage)
- Load from and save to server

Milestones

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

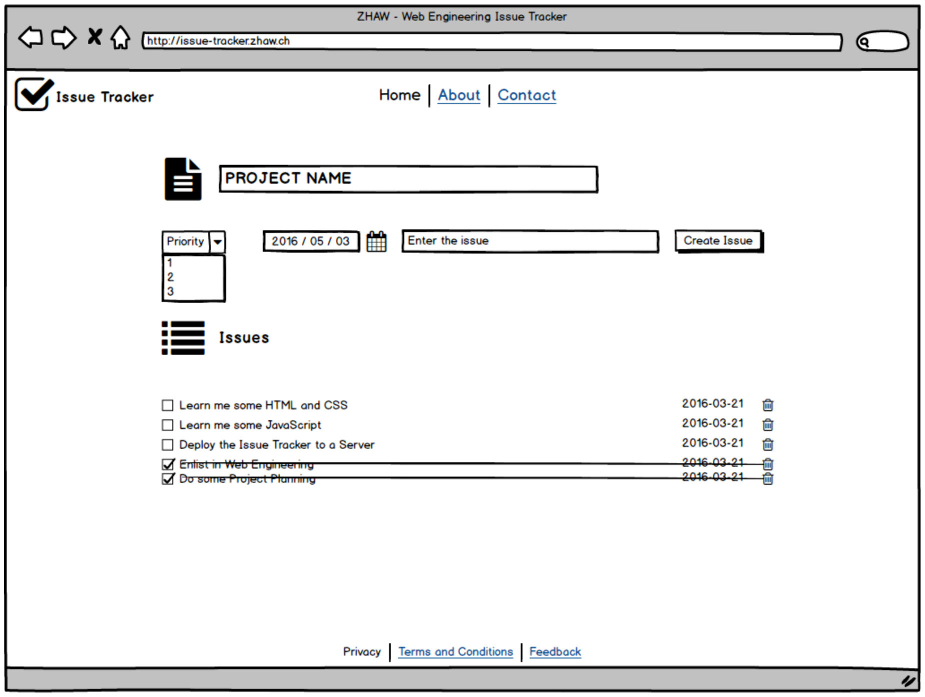
- Be aligned with the theory in the lectures
- Make the scope of the respective technologies explicit to facilitate a better understanding
- Represent a natural succession of steps for similar projects

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

1. **HTML/CSS Prototype**
   (due: 05-APR-2016)
2. **Bootstrap / jQuery Datepicker**
   (due: 01-MAY-2016)
3. **Logic, jQuery, localStorage, REST**
   (due: 25/26-MAY-2016)

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.



☐ Learn me some HTML and CSS 2016-03-21
☐ Learn me some JavaScript 2016-03-21
☐ Deploy the Issue Tracker to a Server 2016-03-21
☑ ~~Enlist in Web Engineering 2016-03-21~~
☑ ~~Do some Project Planning 2016-03-21~~

Figure 1:

Milestone 1 - HTML/CSS Prototype

When starting with a new frontend 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 resides in a separate file.

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

Milestone 2 - Bootstrap / jQuery Datepicker

As you probably have realized, 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 was 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.

In the next Milestone, you will re-do your HTML and CSS by implementing the design with Bootstrap. When done with a little care, it is likely that your app now looks better, more coherent and also works on mobile devices.

It was important, though, to work with HTML and CSS manually. Even if you'll use a framework like Bootstrap in a future project, these frameworks are still made up of HTML and CSS. Therefore, you will soon reach a point where you want to make your own adjustments and again require knowledge of the original building blocks.

As a final step 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.

Summary Milestone 3

- [ ] Re-do HTML and CSS with Bootstrap
- [ ] Use jQuery Datepicker for the issue date field

Milestone 3 - Logic, jQuery, localStorage, REST

In the final step of the project, 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. If however, you see a clear path to writing the logic of an issue tracker with persistence to localStorage and REST, you can also just read the Uses Cases and implement those as you see fit.

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 checkbox 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.

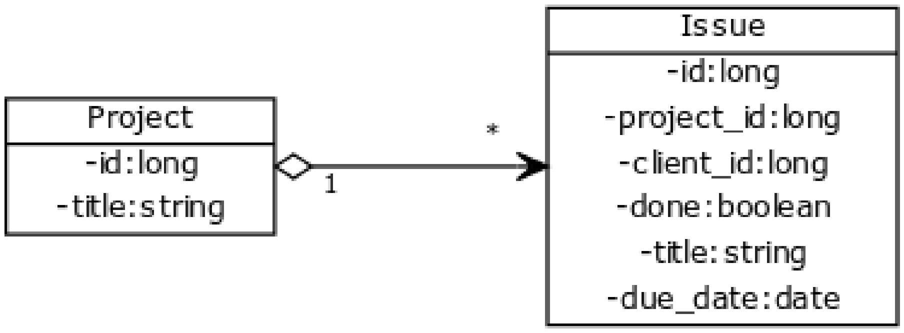
Use Case: Persisting data to the Browser

- 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 a tab), I want all previously entered data to be loaded from localStorage.
- 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 backend. In the RESTful API (see below), you will see this client_id coming up again.

Use Case: Persisting data to a Server

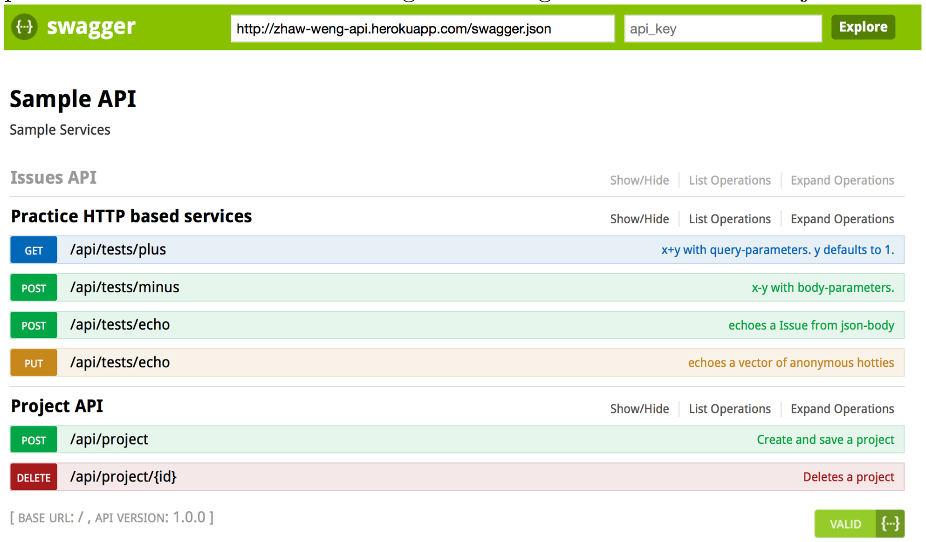
Many web applications offer persistence to a backend. The Issue Tracker will be no exception. However, you will not have to create a backend service with a database for yourself. We will use a modern approach and give you an API with with you can persist the applications data in a RESTful manner.

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:



RESTful API

The RESTful API is described in the standardized Open API Initiative format. You can find it here. On this page, you can not only find the definition of the API, but 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.



swagger http://zhaw-weng-api.herokuapp.com/swagger.json api_key Explore

Sample API

Sample Services

Issues API — Show/Hide | List Operations | Expand Operations

Practice HTTP based services — Show/Hide | List Operations | Expand Operations

GET	/api/tests/plus	x+y with query-parameters. y defaults to 1.
POST	/api/tests/minus	x-y with body-parameters.
POST	/api/tests/echo	echoes a Issue from json-body
PUT	/api/tests/echo	echoes a vector of anonymous hotties

Project API — Show/Hide | List Operations | Expand Operations

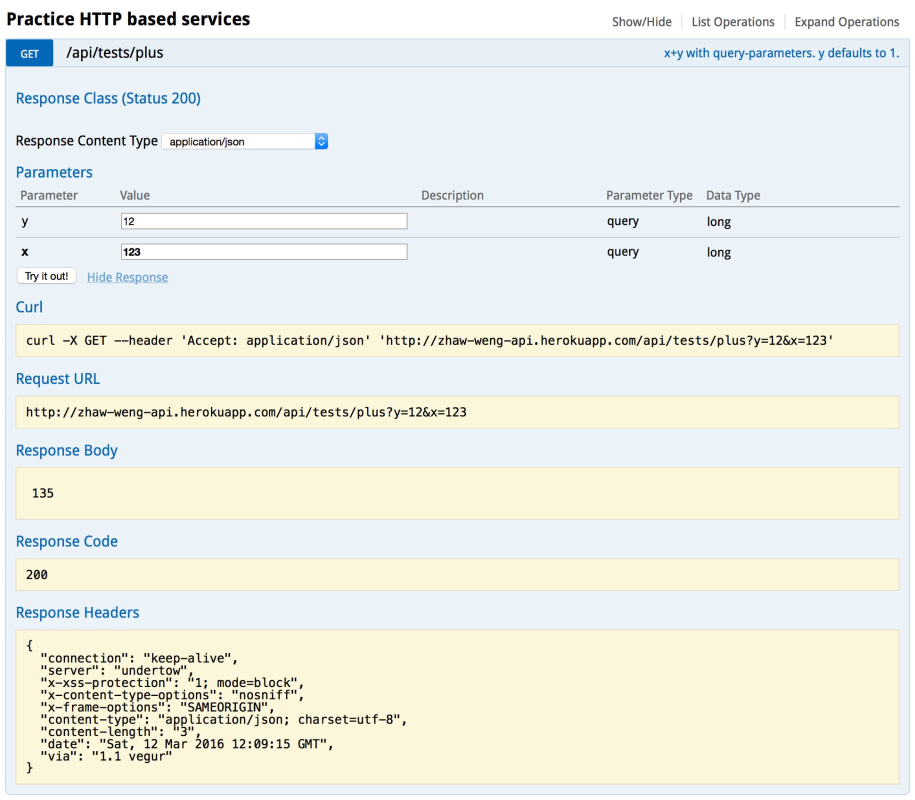
POST	/api/project	Create and save a project
DELETE	/api/project/{id}	Deletes a project

[BASE URL: / , API VERSION: 1.0.0]

VALID {...}

.

This is an example screenshot of how through this test API you can add two numbers:



Practice HTTP based services

GET /api/tests/plus — x+y with query-parameters. y defaults to 1.

Response Class (Status 200)

Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
y	12		query	long
x	123		query	long

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"
}
```

.

Project User Stories

- As a user, when entering a Project title, I want a new Project to be created through the RESTful API and then the resulting `id` to be saved into the existing entry in localStorage.

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 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 backend.

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: srv-lab-t-968.zhaw.ch
- Port: 22
- User: weng1
- Password: weng1pw
- Remote Directory: /var/www/html/weng

After having logged in, you will see folders for the respective milestones `project_milestone_1`, `project_milestone_2`, and `project_milestone_3`. For the respective milestone, change into the correct directory and create a sub-directory for your submission. For your teams' submission, please create a subdirectory using your studends abbreviations spaced with underscores. If for example Gerrit Burkert (bkrt) and Alain M. Lafon (lafo) would form a team, the subdirectory would be called `bkrt_lafo`.

If you need more instructions on how to connect to and use an sFTP server, please refer to the `instructions.pdf` of lecture 3.