-
    - Help and plagiarism

Dates

Date
12.04. start
23.04. report **deadline**
04.06. code **deadline**

Overview:

For the exam project you are required to design and implement a complete web application, frontend and backend.

You are free to chose what kind of application you implement, but restrictions apply to both technology that you may use and there is some required functionality.

Report due date 23.04.21

2 weeks after the start of the project, you have to hand in a report describing the application that you want to implement. The report must contain the following:

1. What does the application do?
2. What data is stored in the database (e.g. table names)?
3. An initial layout for your page. The initial layout can be created as a static HTML page, or simply as an image (e.g. you can use PowerPoint).

The report does not need to include all provided functionality, and you can change specific functionality, data and page layout. However, the overall theme for the page is binding.

Project delivery

On the final delivery you should hand in your complete code, and a README.md markdown file. Your code should also include a SQLite database file including your application's database with example data.

The `README.md` file should contain the following sections:

- How to run: e.g.

    Start the application by running `app.py` in the application folder.

- List of all functionality: List all implemented functionality, to make sure all your work is taken into account. For example
    - Preferences are stored in a cookie and present if the user revisits the page.
    - New categories can be added on the category pages.

        - An Icon and color can be chosen for the category.
    - If the user tries to register a password with less than 5 characters, an Error is displayed.

Technology allowed

You may only use the frameworks listed below. If you are unsure if something is allowed, please ask on Discord.

Backend:

Your web server should be Flask with an SQLite database. You may use plugins like Flask-login but that is not required. If you do so, note it in your README.md.

Frontend:

The frontend should be either pure JS or Vue. You may use plugins like Vuex or Vue-Router in Vue. You may use the Lodash JS library. You may use all buildin JS APIs. You may use use either Vue version 2 or 3.

You are not allowed to use a different JS Framework like React.

Layout:

You can either use plain CSS or Bootstrap. If you copy CSS files from the web, e.g. `reset.css` or `normalize.css`, you must specify this in the README.md. If specified, it will not count as plagiarism.

Note that there are specific functional requirements when using Bootstrap.

Functional requirements

The following are the functional requirements which build the bases for grading your project. You can omit some features, but this will effect grading.

SPA and REST API

Your application should be a Single page application (SPA) that communicates with the Flask backend using AJAX requests.

The application should be implemented using Vue. Additionally, the application should contain multiple routes, using the Vue-Router.

The Web server should serve the initial application as static files and expose application data as a REST API.

For maximum score, your application must be well-organized and separated into components.

Alternatives:

Pure JS: Instead of using Vue you can also create a plain JS application, similar to Assignment 4/Assignment 8. However, your score will be reduced if your app does not use routing. You can implement a router in Plain JS.

Flask Templates: You can choose to implement your application using Flask templates instead of Vue and AJAX calls. This will give you a significant reduction in score. However, if you aim for a passing grade and want to avoid the complexity of Vue and AJAX calls, this might be a good choice.

Login

Your application should allow login and registration of new users. Users should be stored in the database.

Data items

Besides users, the database should contain at least two types of data objects, i.e. two tables. Data should be exposed using a REST API that allows to query, update, insert and delete data. Some of these operations should require specific authentication, e.g., not every user is allowed to delete every item.

Data collection display

The frontend must contain functionality to display a collection of items. There should be functionality to **filter**, **sort** and **change the display** of items. *Preferences on sorting or display should be stored*, to be present when the user revisits the site. (May be stored in cookies or similar.)

Validation

Forms for login, registration, adding and updating data items should be validated. Critical elements, such as passwords and username must be validated on the server-side. If validation fails, a meaningful error message must be displayed.

Layout

The layout of your page should be fluid, i.e. adjust to different browser window sizes.

If you use plain CSS your layout should at least adjust to window sizes between 1800px and 800px.

If you use bootstrap, your layout should be responsive adjusting also to phone sized displays, e.g. 375px.

Your layout should contain some kind of background or banner images.

Additional features

You are expected to implement additional, more complex features.

Examples for more complex features can be:

- Storage and handling images uploaded by the user.
- Storage and display of dates.

Grading

The submitted project code builds the main basis for the grading. However, to which extend you implement what is described in the report will affect your score.

Your application should be well tested and working. All functionality should be error free, including JS code. The presence of **dysfunctional or unused code** will give a penalty. Remove such code before you submit.

For grading Firefox version 87 will be used. To avoid compatibility errors it is recommended to test your application on that browser.

The following table gives a rough overview of grading criteria. Note that in all categories it is possible to receive points for additional or more complex functionality.

Project report	**5%**
Project code	**95%**
Frontend	45%
- Functional Vue application	
- Display and sorting of data	
- Storage of user preferences	
- Form validation and error display	
- Clean separation and reuse of components	
- Use of Vue Router	
- Comments and clean code	
Layout	10%
- Fluid layout	
Database	10%
- Layout and access functions	
Flask backend	30%
- REST API	
- Server side validation and error handling	
- Authentication and access control	
- Comments and clean code	

Help and plagiarism

If we find code with significant similarities in two submissions this will be filed as plagiarism. We will check this using an automated plagirism checker. However, this does not include code copied from examples in the course.

Note that this should not stop you from helping each other. As long as you do not write code for your fellow students, but help them debug or design a certain functionality, it is not considered plagiarism.

We will also continue labs and offer help.