Guess

Question	Question title	Marks	Question type
i	Fremside DAT310		Document
1	DAT310 Info 2019	0	Essay
2	CSS color	2	Multiple Choice
3	jQuery guess 1	2	Multiple Choice
4	CSS guess	2	Multiple Choice
5	JS guess	2	Multiple Choice
6	Validate event	2	Multiple Choice
7	JQuery color	2	Multiple Choice
8	JS event	4	Programming
9	CSS Query	4	Programming

Coding

Question	Question title	Marks	Question type
i	Info coding		Document
10	Flask and templating	24	Programming
11	jQuery coding	24	Programming
12	AJAX	26	Programming
13	Upload	0	Upload Assignment

1

DAT310 Info 2019

The exam contains 11 questions/exercises. You can collect 94 points in total.
You may use all available online and offline resources.
But you have to solve the exam yourself, so **it is not allowed to ask others, chat, ...**
Remember, if you spend a lot of time searching for answers online, you may not have time to complete the exam.

The exam contains two parts.
Part 1 contains multiple choice and text questions.
Answers should be given using the available forms.
The scoring of multiple choice questions is as follows

- The correct solution is 2 points
- Wrong answer is -1 point

Part 2 contains several programming exercises related to example applications.
Starter files for all exercises are provided as a zip archive.
Download the starter files and use your IDE to solve the exercises.

The different exercises include forms.
If you copy any code from online you must include a reference in the form.

To submit your solutions to part 2, submit a single zip file, containing the starter files and your added code.
This zip file can be uploaded on the last page of the exam.
Code added in the individual solution forms will not be considered if a zip file is available.
This should only be a last resort.

If you have comments to any of the exercises, you may write them here

---

Maximum marks: 0

8 JS event

```
function bar() {}

function foo() {}

mydiv = document.getElementById("thediv");
```

Consider the JavaScript snippet above.

How can we attach both the *foo* and *bar* function to *mydiv*, ensuring that they are both run if the element is clicked.

Can you do this such that they can be removed again independently?

Fill in your answer here

Maximum marks: 4

9 # CSS Query

Write a CSS rule that results in the first line of every paragraph <p> having bold font.
Fill in your answer here

Maximum marks: 4

10 Flask and templating

In this exercise you have to complete the **projectQuestion** application. *(24 points)*

You can find the start files in the **projectQuestion** folder.

This exercise has three parts.
1. Complete the template *templates/form.html* for the main page.
2. Add the route and a function that receives and processes the form.
3. Create a third template that shows the information that was sent.

The CSS file for Bootstrap 3 is included in the layout. Bootstrap classes should be used to style the pages.

Part 1. Complete the template *templates/form.html* as shown in Image 1 here.

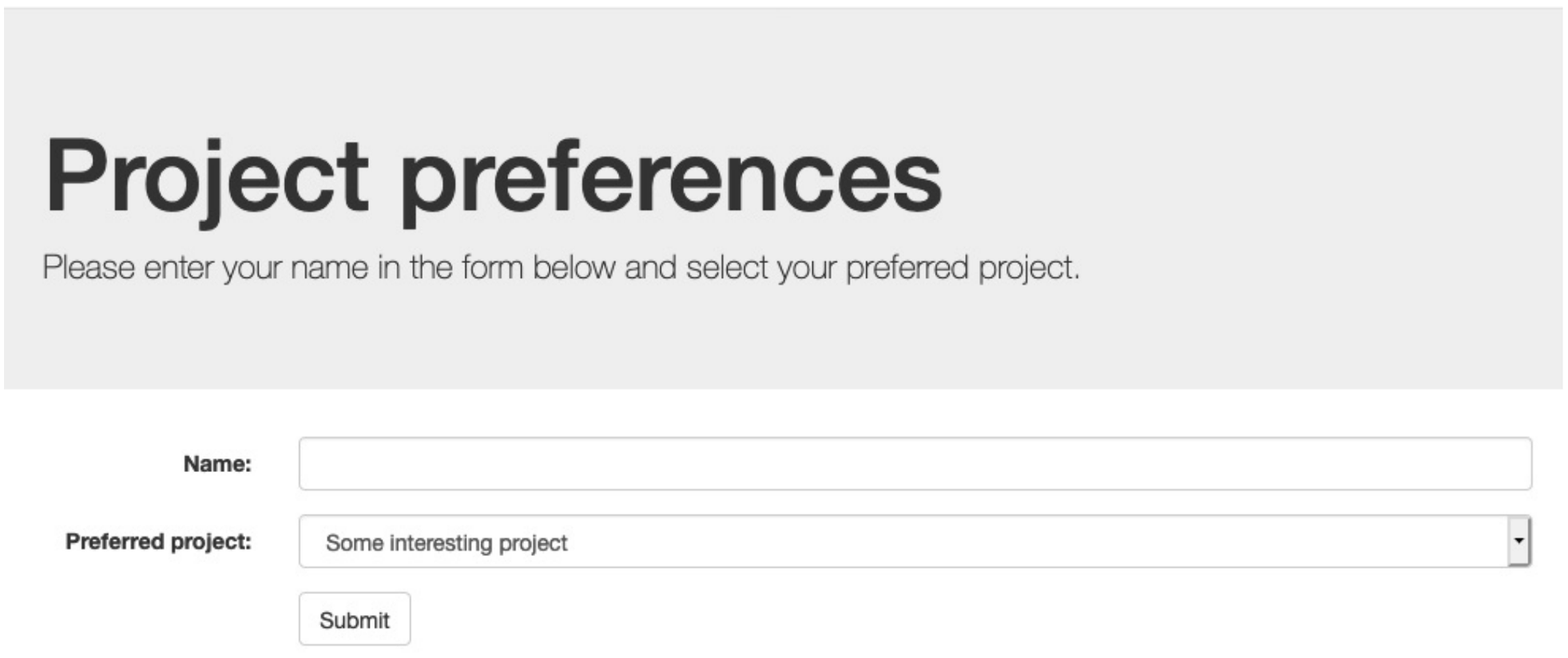

Image 1: The form

Be sure to include the following:

- The heading (Project preferences ...) is on a gray box that spans the entire width of the page.
- The form on the page has the following fields:
  - **Name** where the user can enter their name (input).
  - **Preferred project** where the user can select a project (select).
    - The options for this select are given as the variable *projects* in the *render_template* function.
    - The options should be added to the template using a *for* loop.
- A **Submit** button where the form can be submitted.
  - The form must be sent in a **post-request** to a new *route*.

Part 2. Add a new route to **app.py.**
Add a new route that processes the **post request** from the form you have implemented.
The function should check if the **name** field is filled. If not, the user should be redirected to "/" to refill the form.
Otherwise, a new page will be displayed, implemented through the template *thanks.html*.

Part 3. Implement template *templates/thanks.html* as shown in Image 2 here.

Image 2: Show information sent in the form.skjemaet.

Be sure to include the following:

- The template extends the main template *layout.html.*
- The heading (Project preferences ...) is on a gray box that spans the entire width of the page.
- Names that were submitted in the **Name** field appear in the header.
- The name and preferred project are shown in a table.

List source here

Maximum marks: 24

11 jQuery coding

Additional functionality and styles for the jQuery countdown application. *(24 points)*
In this exercise, you have to implement some additional functionality for a jQuery application.
You find the countdown application in the zip archive in the folder **timer/**.
You need to add some *css rules* and implement some functionality in the *script.js* file.

Style: To implement styles, you can use the html file **index-mockup.html**.
Add new rules to *style.css* so that they appear in the application. *See picture 1 below.*

The page is divided into an *aside* element and a *main* element.
<aside> should always be 300px wide. It should extend from the top to the bottom of the page and a solid border (2px wide) with color # 808080 on the right.
<main> should be to the right of <aside>. Check that the timer is centered in main.

Functions:
1. Implement the ***newTimeDisplay****(timer){ }* function in *script.js*.
This function adds a small version of the timer to <aside>.
For a one minute timer titled "The title" it should add the following html:

```
<div class = "smalltimer">
    <span class = "timeLeft"> 01:00 </span>
    <span> The title </span>
</div>
```

- Make sure you add the classes.
- When *newTimeDisplay* is called repeatedly, several timers should be displayed.

2. In the *newTimeDisplay* function, **add a listener** to the div element so that, when clicked, *timer.showAsMain* is invoked.
Tip: timer.showAsMain must be called by another "anonymous" function.

3. (Difficult) Implement the ***updateTimeDisplay*** function.
timer.updateTimeDisplay is called every second. It should update the time displayed in the small timer created by *newTimeDisplay*.
Tip: One possible way to implement this function is to store the DOM element created in newTimeDisplay in the Timer object and use it in updateTimeDisplay.

4. Implement the ***removeDisplay*** function.
This function is called when the user hits the **Cancel** button on the main display.
It should remove the timer from <aside>, i.e. removing the element added in *newTimeDisplay.*
Tip: One possible way to implement this function is to store the DOM element created in newTimeDisplay in the Timer object and use it in removeDisplay.

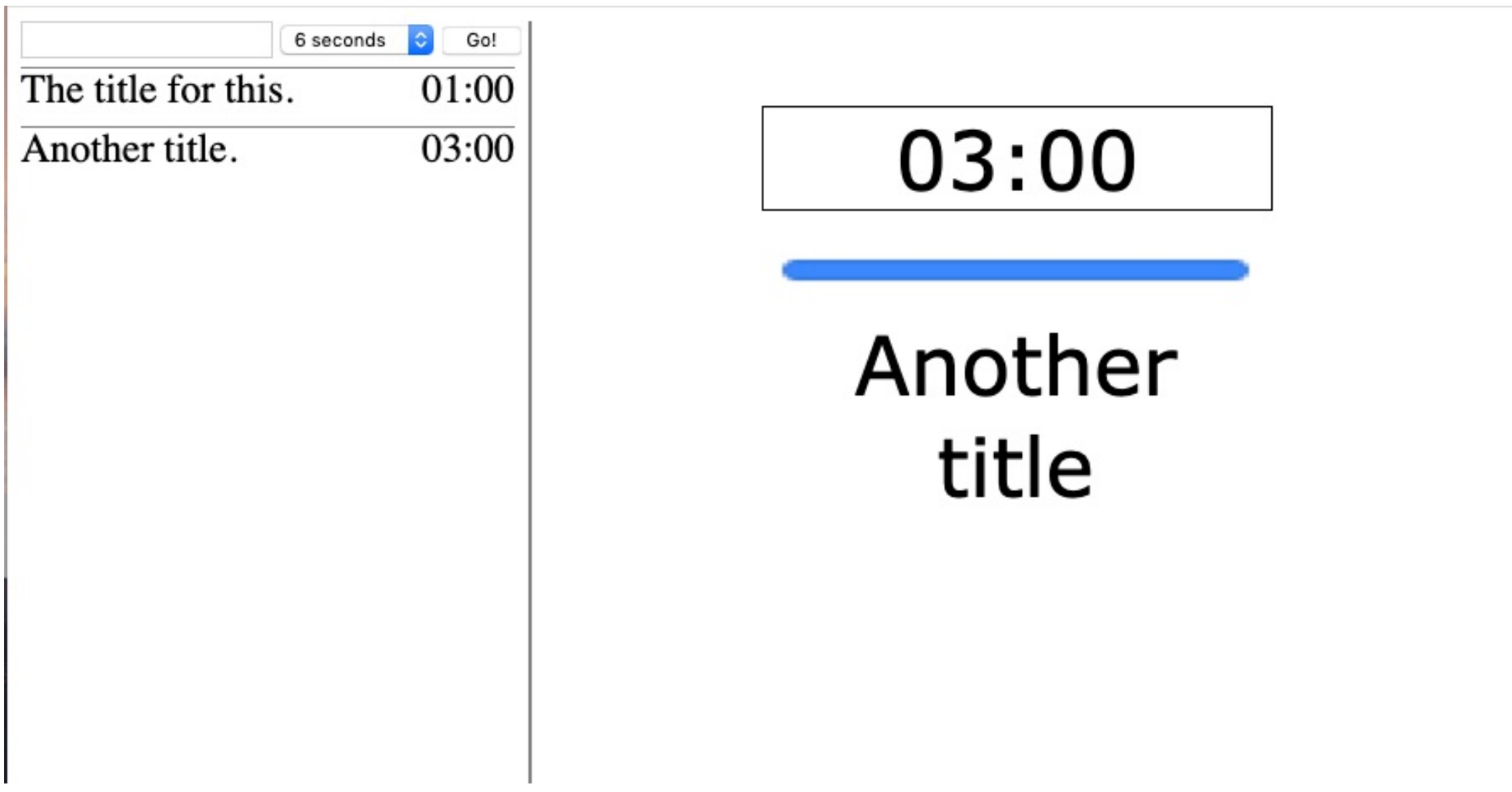

Image 1: *index-mockup.html* with the desired styling.

List kilder her

Maximum marks: 24

12 AJAX

Implementer AJAX kall for en adress book application. *(26 poeng)*

In this exercise you have to implement AJAX call for addresbook JS application from the lecture.
You can find the application in the zip archive in the **addressbook/** directory.

You need to add code to both *static/addressbook.js*, *static/data.js*, and *app.py.*

1. In the current version some addresses are added statically in *static/data.js.* You can find the same addresses in *data.csv.* **Read these addresses from the file and load them via an AJAX request.**

- Implement a function in *app.py* that reads the addresses from file.
- Store the addresses from the file in the session. Only read the file, if no addresses are stored in the session.
- Implement the *getData* function in *static/data.js*. This function should send an AJAX request to retrieve the addresses stored in the session.
- Expand *app.py* to process this request and
- send the addresses in **JSON format.**
- Process the addresses so that they appear in the application.
- Call the *getData* function when the page is loaded.

2. The application allows to update, delete and add new addresses.
Use an AJAX call to store the addresses in the session.

- Implement the *setData* function in *static/data.js.* This function should send an AJAX request to update the address list stored in the session.
- The request must be a **post-request**.
- The address must be sent in **JSON format.**
- Expand *app.py* to store the updated addresses in the session.
- Call the *getData* function in *static/addressbook.js,* when an address is updated, when an address is deleted, or when a new address is created.

For this exercises it is enough if *setData* sends the complete list of addresses whenever one is updated.

List sources here

Maximum marks: 26

13

Upload

Create a zip arcive from the hole *files* folder and upload it here:

Maximum marks: 0