UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Written Examination

DIT342 – Web Development

Monday, October 28, 2024, 08:30 - 12:30

Examiner: Philipp Leitner

Contact Persons During Exam:
Magnus Ågren (+46 733 35 22 92)
Backup:
Philipp Leitner (+46 733 05 69 14)

Allowed Aides:
None except English dictionary (non-electronic), pen/pencil, ruler, and eraser.

Results:
Exam results will be made available no later than 15 working days after the exam date through Ladok.

Total Points: 100

Grade Limits: 0 – 49 points: **U**, 50 – 69 points: **3**, 70 – 89 points: **4**, $\geq$ 90 points: **5**

Review:
The exam review will take place latest three weeks after the exam results have been published in Ladok. It will be announced on Canvas at least one week in advance.

1 Backend Development (18P)

The code snippet (Figure 1) on the next page shows parts of a milestone management Express app, (see also Figure 2 for inspiration). Implement two endpoints:

1. One endpoint that allows editing existing milestones. Milestone updates are passed in the body of the request as a JSON document containing any combination of; a new milestone name, an updated list of tasks, and/or an updated deadline. By matching milestone ids with the id given as a path parameter, the endpoint should find the correct milestone to update in the array `milestones` and overwrite existing properties with those received in the request body. Return the correct status code for a successful operation and the newly updated milestone in JSON format.

2. One endpoint that allows appending tasks to an existing milestone. A task is passed in the body of the request as a JSON document of the form
   {`tasks: new task`}
   Append the new task to the existing ones of the milestone. Return the correct status code to indicate that a new resource has successfully been created and a JSON document of the form {`tasks: full list of tasks`}

You do not need to implement any error handling.

Write on an answer sheet of paper all code that should be inserted into the template below to realize this behavior. Do not edit or remove existing code outside of the “blanks” (the missing code on lines 16 and 26, lines 17 – 23, and lines 27 – 33). Use line numbers to clarify where your code shall be inserted. Available space is *not* necessarily indicative of how much code is required.

```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
app.use(bodyParser.json());

var milestones = [
  {'id': '709a5e', 'name': 'Project presentation',
    'deadline': '2024-10-22',
    'tasks': ["Pep talk", "Polish code"]},
  {'id': '2d37ed', 'name': 'Prepare examination',
    'deadline': '2024-10-21',
    'tasks': ["Come up with new tasks", "Latex formatting",
      "Printing"]}
];

app.______(_______________, function(req, res) {








});

app.______(________________, function(req, res) {








});

app.listen(3000);
```

Figure 1: Complete the blanks (Express)

2 REST APIs (18P)

Figure 2 shows parts of a simplified API for a milestone management app. Assume it adheres to the REST architectural style. You can also assume that milestone ids and task numbers function as unique identifiers.

```
GET /milestones/:milestone
GET /milestones/:milestone/tasks/:task
POST /milestones
POST /milestones/:milestone/tasks
DELETE /milestones/:milestone/tasks/:task
```

Figure 2: Milestone Management API Excerpt

Q 2.1: (4P) What functionality and behavior would you expect from the following two API methods.

```
POST /milestones
POST /milestones/:milestone/tasks
```

Q 2.2: (4P) Consider the following two HTTP requests to the API.

```
GET /milestones/67f60a2/tasks/001
GET /milestones/67f60a2/tasks
```

What does a response code of 404 indicate in these two cases? What is the difference?

Q 2.3: (3P) The status of a task can be unassigned, assigned, or completed. Assume that the API in Figure 2 is extended to allow filtering tasks by their status. What request do you have to send to get all tasks of milestone `d828fd8` that are *completed*?

Q 2.4: (4P) Describe the meaning of each of the four main HTTP response status code groups (2xx, 3xx, 4xx, and 5xx).

Q 2.5: (3P) Which HTTP methods are expected to be idempotent?

3 CSS Formatting (9P)

Consider the HTML document in Figure 3. Describe in which colors the browser will render the text. Assume that the default browser color is black. You can refer to line numbers when providing your answers.

```
 1  <!doctype html>
 2  <html>
 3  <head>
 4    <style>
 5      div, a { color: purple; }
 6      .priority { color: red; }
 7      #priority { color: orange; }
 8      .taskTitle { color: blue; }
 9      a[target="_blank"] { color: green; }
10    </style>
11  </head>
12  <body>
13    <div>
14      <p class="taskTitle">Prepare exam</p>
15      This is an
16      <span id="priority">urgent task!</span>
17      Take a look at the related
18      <a target="_blank" href="milestone.html">milestone</a>
19      or check the
20      <a href="project.html">project submission.</a>
21      <div style="color: pink;">
22        This task must be completed before the exam date.
23      </div>
24    </div>
25    Someone needs to take care of this task!
26  </body>
27  </html>
```

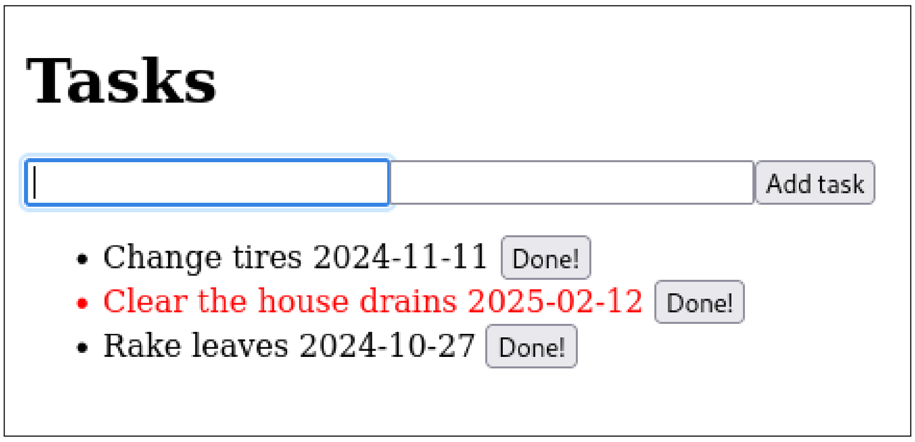
Figure 3: HTML and CSS

4 Frontend Development (18P)

Complete the code snippet of a Vue.js app in Figure 5. Implement the following behavior:

- When the *Add task* button is clicked, the `addTask` function shall be called. This function shall add the new task and its due date to the list of tasks. For simplicity, assume that the provided values are correct; you don't need to implement any error handling.
- Render the tasks and their due dates in an unordered list. Place a button next to each task. When clicked, this button should invoke the `completeTask` function, which shall remove the task and its due date from the list of tasks.
- Tasks that are overdue shall be printed in red, using the predefined CSS style. Other tasks shall be printed in black. A task is overdue if its due date is later (greater) than the defined `overdueDate`. Implement this check as the function `overdue`.
  Hint: you can use the `v-bind` directive to dynamically bind attributes in Vue.js, with the syntax `v-bind:<attribute>="<JS code>"`.

Refer to the screenshot in Figure 4 for what the generated site shall look like.


Tasks

- Change tires 2024-11-11
- Clear the house drains 2025-02-12
- Rake leaves 2024-10-27

Figure 4: Vue.js example

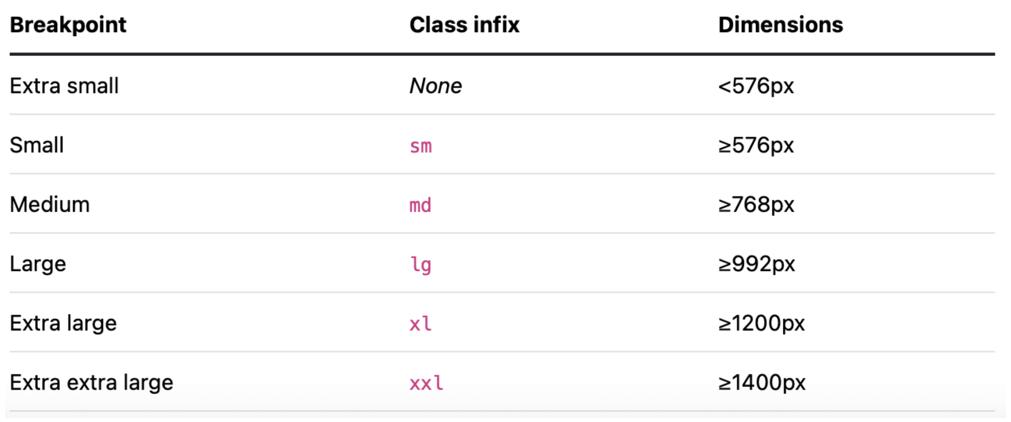
Write on an answer sheet of paper all code that should be inserted into the template below to realize this behavior. Do not edit or remove existing code outside of the ”blanks” (the missing code on lines 9 – 11, lines 13 – 15, and the implementation of the functions `addTask`, `completeTask`, and `overdue`). Use line numbers to clarify where your code shall be inserted. Available space is *not* necessarily indicative of how much code is required.

```
<html><head><!-- assume vue.js is imported -->
<style>
  .overdue { color: red };
</style></head>
<body>
  <div id="vueapp">
    <h1>Tasks</h1>
    <p>
      <input type="text" _________=_________ />
      <input type="text" _________=_________ />
      <button @click="_________">Add task</button>
    </p>
    <p><ul><li __________________>


    </li></ul></p>
  </div>
  <script>
    Vue.createApp({
      data() { return {
          overdueDate: Date.parse('2025-01-01'),
          tasks: [],
          task: "",
          due: "",
      }},
      methods: {
        addTask: function() {

        },
        completeTask: function(task) {

        },
        overdue: function(due) {

        }}}).mount('#vueapp')
</script></body></html>
```

Figure 5: Complete the blanks (Vue.js)

5 Responsive Web Design (10P)

Consider the code snippet in Figure 9 where a website layout is defined using the Bootstrap grid system. Figure 7 sketches a line layout for a browser window larger than 1300px. Figure 8 sketches a line layout for a browser window on a mobile device with 530px. In both figures, the first div-element is already sketched in. On an answer sheet of paper, replicate figures 7 and 8, and sketch all the div-elements for both browser window sizes. Figure 6 provides an overview of the different breakpoints in Bootstrap.



Breakpoint	Class infix	Dimensions
Extra small	*None*	<576px
Small	`sm`	≥576px
Medium	`md`	≥768px
Large	`lg`	≥992px
Extra large	`xl`	≥1200px
Extra extra large	`xxl`	≥1400px

Figure 6: Bootstrap breakpoints

Div 1											

Figure 7: Large browser window – 1300px

Div 1											

Figure 8: Small browser window – 530px

```html
<!doctype html>
<html lang="en">

<head>
  <meta name="viewport" content="width=device-width, initial-
     scale=1 shrink-to-fit=no">

  <!-- Assume Bootstrap is imported -->

  <style>
      div div div { border: 1px solid black; }
  </style>
</head>

<body>
  <div class="container-fluid text-center">
    <div class="row">
      <div class="col-1">Div 1</div>
      <div class="col-6 col-sm-1">Div 2</div>
      <div class="col-6 col-md-3">Div 3</div>
    </div>

    <div class="row">
      <div class="col col-md-1">Div 4</div>
      <div class="col col-md">Div 5</div>
      <div class="col col-lg-2">Div 6</div>
    </div>
  </div>
</body>
</html>
```

Figure 9: Responsive layout example

6 Communication Protocols (8P)

Consider a web application that sends messages over the Internet via TCP/IP, similar to the application in your project. For these messages, what are the respective resposibilities of TCP and IP – which of the protocols handles what?

7 Cookies (9P)

Explain how cookies work, by describing the three messages in Figure 10.

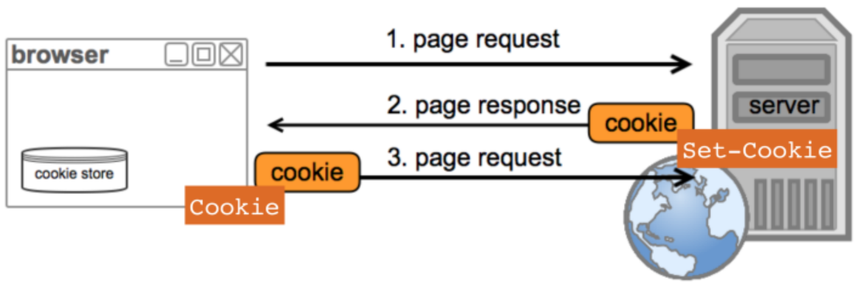


Figure 10: Cookie flow

8 Cloud Computing (10P)

Compare on-premise hosting to cloud hosting; that is, describe these two approaches and contrast their key differences.
Hints: What drives cost in these two approaches? How do these two approaches relate to infrastructure-as-code (IaC)?