UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Written Examination

DIT342 – Web Development

Tuesday, January 3rd, 2023, 14:00 - 18:00

Examiner:
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)

Complete the code snippet of a wiki Express app on the next page of the exam paper (see also Figure 2 for inspiration). Implement two endpoints:

1. One endpoint that allows to retrieve all wiki pages. The endpoint shall implement HATEOAS principles - return for each page the title and a relative path that allows the caller to retrieve more details about the page (see also the second endpoint, described below). Further, the endpoint shall support sorting (ascending and descending, alphabetically by title), which shall be implemented through a query parameter `sort`. The query parameter shall be optional and shall default to ascending sorting, otherwise the values `asc` (ascending) and `desc` (descending) shall be supported.
2. One endpoint that allows to retrieve all details for a single wiki page. This endpoint shall return only a single page, but shall include all page details (title, content, and attachments).

You do not need to handle any error conditions in your endpoints.

> 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 12 – 14 and lines 17 – 20). 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 bodyParser = require('body-parser');
var app = express();
app.use(bodyParser.json());

var pages = [
  { title: 'Bar', content: 'Potent potables',
    attachments: ['picture.gif'] },
  { title: 'Bbq', content: 'Charcoal etc.',
    attachments: ['coal.gif'] }];

app.___(___, function(req, res) {


});

app.___(___, function(req, res) {


});

app.listen(3000);

function sortByTitle(arr, order) {
  var copy = [...arr];
  copy.sort((a, b) => a.title.localeCompare(b.title));
  if (order === 'desc') copy.reverse();
  return copy;
}
```

Figure 1: Complete the Blanks (Express)

2 REST APIs (15P)

Imaging an API for a simple wiki. Assume it adheres to the REST architectural style, and could be used from multiple front-ends:

```
GET /wiki/pages/:page
POST /wiki/pages
PATCH /wiki/pages/:page/attachments
PUT /wiki/pages/:page
DELETE /wiki/pages/:page
```

Figure 2: Excerpt of an Imaginary Wiki API

Answer the following questions about this API, assuming that it has been designed following the REST principles introduced in the lecture.

Q 2.1: (2P) Describe what functionality and behavior you would expect from the following API endpoint. Also discuss what HTTP status codes you would expect, and what the endpoint would typically return.

```
DELETE /wiki/pages/:page
```

Q 2.2: (4P) Consider the following intended addition to the API. This method would be called on page load and return the number of unique visits to the page. The visit count would be incremented by one for each request.

```
GET /wiki/visits
```

Discuss potential problems with this design. Describe also how the API could be improved.

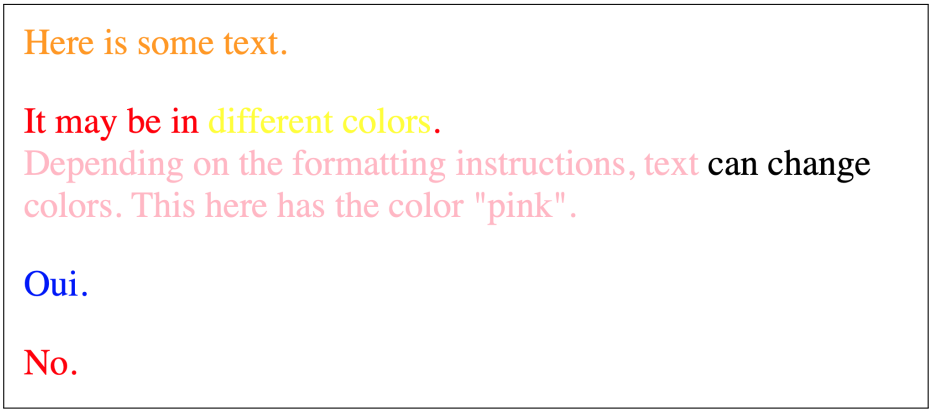
Q 2.3: (5P) Compare the POST and PATCH HTTP methods using the example of the provided wiki API. What are the differences between the two methods? What are the similarities? When would you use which? Particularly contrast them in terms of idempotency and safety.

Q 2.4: (4P) Provide an example HTTP request (as it would look like on the wire, not the JavaScript code you would use to send an HTTP request!) for the following API method. Assume that the request is sent to the server at the URL `http://localhost:3000/`, and that a session cookie "session: 1234" has been set by the server before. Include appropriate headers to tell the server that the client expects a JSON response.

```
GET /wiki/pages/:page
```

3 HTML, CSS, and Javascript (20P)

Complete the code snippet of an HTML / CSS / Javascript document on the next page of the exam paper. Complete the HTML page so that the browser will render the text in the colors shown in the screenshot. You are *not allowed* to modify the CSS definitions in the header. In-line CSS style definitions are *allowed*, but only if the same behavior cannot be achieved using the CSS styles defined in the header.
Further explain textually how conflicting CSS definitions are resolved, and name and explain at least three examples of this from your solution to this code task.



Specification:

- First line: orange.
- Second line: red, but the string ”different colors” in yellow.
- Third line: pink, but with the string ”cam change” in black.
- Fourth line: blue.
- Fifth line: red.

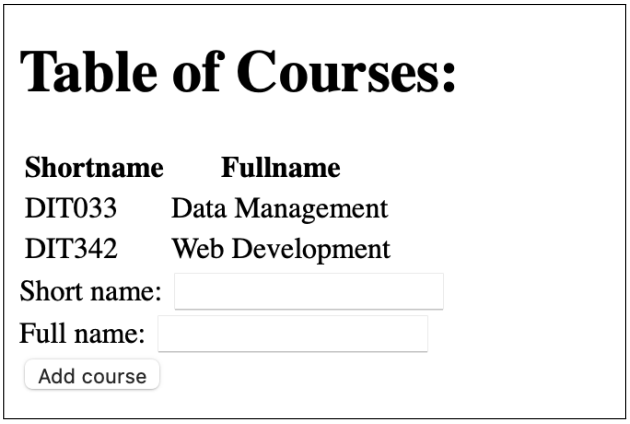
Write **on an extra paper** all code that should be inserted into the template below to realize this behavior. Code written on the exam sheet will not be graded! Do not edit or remove existing code outside of the "blanks" (indicated through "_____"). Use line numbers to clarify where your code shall be inserted. Available space is not necessarily indicative of how much code is required.

```
1   <!doctype html>
2   <html>
3     <head>
4       <style>
5         p { color: grey }
6         .selector { color: orange; }
7         div { color: red; }
8         #highlight { color: yellow; }
9         div > span { color: black; }
10        p[lang="en"] { color: red; }
11        p[lang="fr"] { color: blue; }
12      </style>
13    </head>
14    <body>
15      <div>
16          <p _____>Here is some text.</p>
17          It may be in _____ different colors _____ .
18          <div _____>
19              Depending on the formatting instructions,
20              text _____ can change _____ colors.
21              This here has the color "pink".
22          </div>
23          <p _____ >Oui.</p>
24          <p _____ >No.</p>
25          <p _____ >Should not be displayed at all.</p>
26      </div>
27    </body>
28  </html>
```

Figure 3: Complete the Blanks (HTML/CSS/JS)

4 Frontend Development (20P)

Complete the code snippet of a Vue.js app on the next page of the exam paper. Write an app that renders the contents of the array `courses` as a table. Below the table you should render two text input fields, one for course short names and one for full course names. Finally, there should be a button - if that button is pressed, a new course (in the same format as the existing ones) shall be added to the array, and the table shall be updated accordingly. After pressing the button the input fields shall be cleared. Use Vue.js two-way data binding to achieve this. You do not need to do any input validation. Below is an example of what the app should look like when it is finished.



Write down the missing code on an extra paper and refer to the lines in the template via line numbers. You will need to add code in lines 9 – 11 and 24 – 25. Available space is not necessarily indicative of how much code is required. There is no need to interact with a backend in this task.

```
<!doctype html>
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js
        "></script>
</head>
<body>
  <div id="vueapp">
    <h1>Table of Courses:</h1>



  </div>

  <script>
        var app = new Vue({
            el: '#vueapp',
            data: {
                courses: [{short: "DIT033", full: "Data
                    Management"}, {short: "DIT342", full: "
                    Web Development"}],
                to_add_short: '',
                to_add_full: '',
            },
            methods: {
                addCourse: function () {


                }
            }
        });
    </script>
</body>

</html>
```

Figure 4: Complete the Blanks (Vue.js)

5 Single Page Application (12P)

Imagine a Single Page Application implemented using the same libraries as in the course project, for example, *Express* and *Vue.js*. Assume that this application is available at the URL

```
http://www.mycoolwiki.com
```

Q 5.1: (DNS Resolution) Explain the iterative process of DNS resolution for the URL above. Assume that your local DNS proxy does not have the IP address of any of the relevant domains cached.

Q 5.2: (HTTP Interactions) Assume the following scenario: a user types the URL in their browser, which loads the page. The initial page contains a list of all available wiki pages. Then the user clicks on a specific wiki page, which causes the page to be shown in an editable form. The user edits the page, and clicks a ”Submit Changes” button. Deescribe the typical interactions between client and server (assuming that this is indeed a Single Page Application). Name what HTTP methods would typically be used, what response codes you would expect, and which requests would be sent synchronously /asynchronously.

6 Virtual DOM (4P)

What is the ”virtual DOM” in the context of frontend development? Explain the idea behind it, and also explain what the ussage of a virtual DOM means for a frontend developer.

7 Testing Strategies (8P)

Imagine you are building a wiki application such as the one described in the previous questions. You now need to consider how to test the application. Assume that the system already has a small but established user base.

Name and describe strategies you can use to test the frontend, the backend, and the system as a whole.

8 JavaScript (3P)

Explain the idea of hoisting in JavaScript, and give a minimal example of a program where hoisting would occur.