UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Written Examination

DIT342 – Web Development

Monday, October 24th, 2022, 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)

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 editing existing pages. Page updates are passed in the body of the request, as a JSON document containing either new content, an updated list of attachments, or both. By matching the title with the page name given as path parameter, the endpoint should find the correct page to update in the array `pages` and overwrite existing properties with those received in the request body. Return the correct status code for a successful operation and the newly updated page in JSON format.

2. One endpoint that allows appending attachments to an existing page. Attachments are passed in the body of the request, as a JSON document of the form { `attachments: new attachment` }. Append the new attachment to the existing ones of the page. Return the correct status code to indicate that a new resource has successfully been created, and a JSON document of the form { `attachments: full list of attachments` }.

You do not need to handle any error conditions in your endpoints. Specifically, neither endpoint needs to handle the case of the requested URL not corresponding to an existing page.

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 and 20, lines 13 – 17, and lines 21 – 25). Use line numbers to clarify where your code shall be inserted. Available space is not necessarily indicative of how much code is required.

```
1   var express = require('express');
2   var bodyParser = require('body-parser');
3   var app = express();
4   app.use(bodyParser.json());
5
6   var pages = [
7     { title: 'Bar', content: 'Potent potables',
8       attachments: [] },
9     { title: 'Bbq', content: 'Charcoal etc.',
10      attachments: [] }];
11
12  app.put(page, function(req, res) {
13      var title = req.params.title;
14                  app.get(pages, title),
15                          this.pages = res.params.pages;
16
17          return.status(200).json({this.pages})
18  });
19
20  app.post(pages, function(req, res) {
21      var title = req.params.title;
22        this.attachments = res.params.title
23
24
25          return.status(201).json({"attachments": pages.title.attachments
26  });
27
28  app.listen(3000);
```

Figure 1: Complete the Blanks (Express)

2 REST APIs (21P)

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
POST /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: (4P) Describe what functionality and behavior you would expect from the following two API methods. What is the difference between them, and how would you expect to be able to use them?

```
POST /wiki/pages
POST /wiki/pages/:page/attachments
```

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.

safe: get
unsafe: put, patch
idempotent: delete, post

Q 2.3: (6P) Consider the following suggestion for a developer API towards the wiki, which allows developers access to modify the wiki by passing a database query to MongoDB.

patch - unsafe.

```
PATCH /wiki/dev?modification="Raw MongoDB Query"
```

Discuss potential problems with this design.

Q 2.4: (7P) Define the concepts of safety and idempotency (for HTTP methods in general). Which of the 5 operations in the above API excerpt would you expect to be safe, and which would you expect to be idempotent? Provide an example of an additional method (following the same patterns as the existing methods) that would be idempotent (but not safe).

Safe: GET /wiki/pages/:page

Idempotent: POST /wiki/pages
DELETE /wiki/pages/:page

additional PATCH /wiki/pages/:page/attachments

Safe methods don't modify the state of data.

Idempotent methods modify state of data in expected way regardless of how many times they have been called and executed.

3 Understanding CSS Formatting (14P)

Consider the HTML document below. Describe in which colors the browser will render the text, if the page is viewed on a wide 4K screen. 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 { color: red; }
6        .selector { color: orange; }
7        #highlight { color: yellow; }
8        p[lang="en"] { color: pink; }
9        @media (min-width: 768px) {
10         p { color: purple; }
11       }
12     </style>
13   </head>
14   <body>
15       <p class="selector">A paragraph of text.</p>
16       The specific
17       <span id="highlight"> coloring </span>
18       it is rendered with
19       <div style="color: green;">
20           depends on the formatting instructions.
21       </div>
22       <p lang="fr">Oui.</p>
23       <p lang="en">No.</p>
24   </body>
25 </html>
```

Figure 3: HTML and CSS

15 orange
16 black
17 yellow
18 black
20 green
22 purple
23 pink

4 Frontend Development (10P)

Complete the code snippet of a Vue.js app shown in Figure 4. Implement the following behavior:

- When pressing the button, the method "renamePage" shall be called.
- The new page title shall be taken from the input field. Appropriate API methods shall be invoked to perform the renaming in the backend. Consider the API in Figure 2 for available methods.
- The page title shall be updated.

You can assume that the API calls succeed, you don't need to implement 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 7 and 8, as well as the function implementation on lines 27 – 30). Use line numbers to clarify where your code shall be inserted. Available space is not necessarily indicative of how much code is required.

```
1  <html>
2    <!-- assume Vue.js and Axios are imported in Head -->
3  <body>
4    <div id="vueapp">
5      <h1>{{title}}</h1>
6      <p>
7          <input v-model="___">
8          <button @click="___">Rename the page</button>
9      </p>
10     <div>{{content}}</div>
11   </div>
12   <script>
13         var app = new Vue({
14             el: '#vueapp',
15             data: {
16               title: "Current page title",
17               newTitle: '',
18               content: "Lorem ipsum..."
19             },
20             mounted() {
21               // Assume this.title and this.content
22               // are populated here
23             },
24             methods: {
25                 renamePage: function () {
26                 var newTitle = response.data.title
27                 axios.patch(this.title      , newTitle)
28                   .then(response => {
29                         status(200).json({"Title has been changed
30                 }                                    successfully"});
31             }
32         });
33     </script>
34 </body></html>
```

Figure 4: Complete the Blanks (Vue.js)

7 title
8 renamePage()

5 Single Page Application (10P)

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://dit342-exam.se
```

Describe the requests and responses exchanged in the following scenario: A user types the URL in their browser, which loads the page. The user then clicks a button, which causes a view with a list of camels to be show.

Hint: consider which library calls that might be used, which HTTP methods and headers that might be used, and what the request and response bodies might contain.

6 Responsive Web Design (9P)

Describe the idea behind Responsive Web Design (RWD). Give two examples of how the principles of RWD can be used to implement *accessibility*. Give also two examples of providing web accessibility independently of RWD implementation.

7 Testing Strategies (8P)

Imagine you are providing a web API for a railroad system. The API provides information about, for example, departures, arrivals, and current train locations. Assume that the API already has an established user base. A new feature, providing information on train delays, is to be added to the API. Describe how this new feature could be tested; cover two strategies that could be used before deployment, and two that could be used after deployment.

8 Networking Protocols (10P)

Describe the key differences between TCP and UDP. Relate this to which use cases each protocol is suitable for.

RWD
all users have access to resources of World Wide Web regardless of their physical limitation (e.g. disabilities such as no sight), situational or economical limitations.

Principles:
- create app for different screen sizes
- app can be accessible for special devices (e.g., screen readers, Braille screens)
- have a button on the page for people with limited sight capabilities to hide unimportant content & enlarge and redistribute important one on the page.
- design the page layout in a way that element of the page could be easily available by using the keyboard navigation buttons, but not only mouse / touch pad