UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Written Examination

DIT342 – Web Development

Monday, August 14th, 2023, 08:30 - 12:30

Examiner: Philipp Leitner

Contact Persons During Exam:
Philipp Leitner (+46733056914)

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 – 74 points: **3**, 75 – 89 points: **4**, $\geq$ 90 points: **5**

Review:
Exams can be inspected at student office after the results have been published. Contact Philipp Leitner in case you want to discuss your exam or grading.

1 Backend Development (30P)

Inspect the Express application on the next page of the exam paper. Answer the following questions:

1. Critique the REST API design of this implementation. In which ways does this implementation deviate from the REST architectural style conventions discussed in the course?

2. Implement three new endpoints that actually follow the REST style, re-implementing the functionality currently implemented in lines 11 to 27 of the bad application below. Write all code on an answer sheet. You do not need to copy the header, the endpoints alone (e.g., `app.get(...)`) are sufficient. You do not need to introduce error handling or other features in addition to what the application currently has.

3. Add a fourth, new, endpoint that allows to delete pages by their ID. You again do not need to implement error handling.

```
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: 'A sunny day', content: 'I spent a full day in the
       sun.'},
8    { title: 'Hungry', content: 'I am so hungry, I could eat a
      horse.'}
9  ]
10
11 app.post('/wiki/pages', function(req, res) {
12     var method = req.query.method
13     if (method === 'get') {
14         res.json(pages)
15     } else if (method === 'add') {
16         var body = req.body
17         pages.push(body)
18         res.status(201).json({id: pages.length-1})
19     } else if (method === 'update') {
20         var body = req.body
21         var id = req.query.id
22         pages[id] = body
23         res.json({id: id})
24     } else {
25         res.status(500).send('Unsupported operation ' +
               method)
26     }
27 })
28
29 app.listen(3000)
```

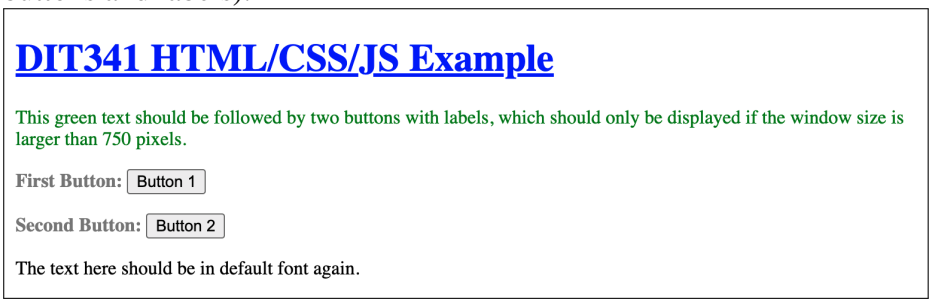
Figure 1: A Poorly Designed REST Application in Express

2 HTML, CSS, and Javascript (22P)

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 following behavior is realized:

- The header should be printed in blue and underlined.
- The first paragraph of text should be printed in green.
- This should be followed by two buttons with labels. The label should be printed in bold and grey. If the buttons are pressed, the function `buttonPressed` should be called with the correct parameter (1 for the first button, 2 for the second).
- The last line of text should be printed in default font.
- The buttons and labels should only be rendered if the window size is at least 750 pixels. Otherwise this content should be hidden.

Please see the screenshot below for an example of what the page should look like for a window size with > 750 pixels (for smaller screens it's the same, but without the buttons and labels).



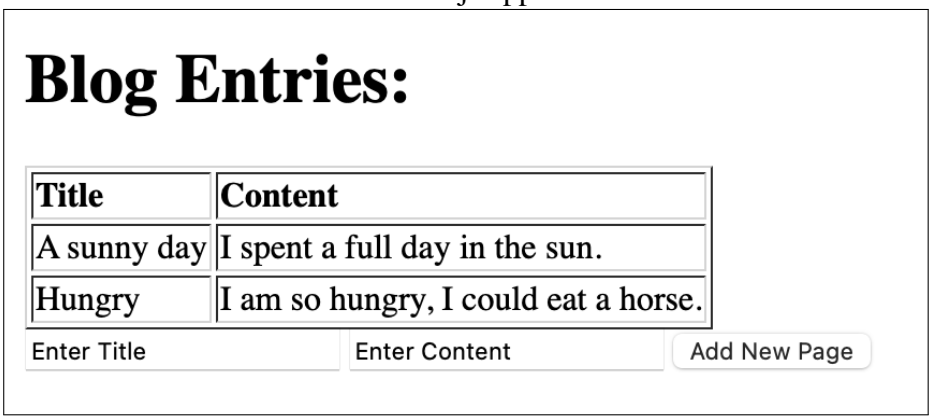
Write on an extra paper all code that should be inserted into the template below to realize this behavior. Do not edit or remove existing code outside of the "blanks" (the missing CSS selectors in lines 3 – 5, lines 6 – 9, and lines 27 – 30). Use line numbers to clarify where your code shall be inserted. Available space is not necessarily indicative of how much code is required.

```
1  <html><head>
2      <style>
3        ___ { color: blue; text-decoration: underline }
4        ___ { color: green; }
5        ___ { color: grey; font-weight: bold; }
6
7
8
9      </style>
10     <script>
11         function buttonPressed(buttonNr) {
12           console.log('Button '+buttonNr+' pressed.');
13         }
14     </script>
15   </head>
16   <body>
17     <h1> DIT342 HTML/CSS/JS Example </h1>
18
19     <div green>
20       <p>
21         This green text should be followed
22         by two buttons with labels,
23         which should only be displayed if the
24         window size is larger than 750 pixels.
25       </p>
26     </div>
27
28
29
30
31     <p> The text here should be in default font again. </p>
32
33   </body>
34 </html>
```

Figure 2: Complete the Blanks (HTML/CSS/JS)

3 Frontend Development (12P)

Complete the code snippet of a Vue.js app on the next page of the exam paper. Complete the code so that (upon loading) the following list is dynamically rendered based on the data model of the Vue.js app:



Clicking the button “Add New Page” should create a new page entry from the values in the two text fields and automatically update the table above. Note that the two text fields shall be initialized with the values `Enter Title` and `Enter Content`, respectively. Your application does not need to handle any errors. The button and input fields shall be simple HTML input elements, do not use Bootstrap or BootstrapVue in this exercise.

Write on an extra 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” (lines 10 – 12, lines 14 – 16, and lines 28 – 30). Use line numbers to clarify where your code shall be inserted. Available space is not necessarily indicative of how much code is required.

```
<!doctype html>
<html>
  <!-- assume Vue.js is imported in Head -->
  <body>
    <div id="vueapp">
        <h1>Blog Entries:</h1>
        <table border="1">
            <tr><td><b>Title</b></td>
              <td><b>Content</b></td></tr>




        </table>



    </div>
    <script>
            var app = new Vue({
                el: '#vueapp', data: {
                    pages: [
{ title: 'A sunny day', content: 'I spent a full day in the sun.'},
{ title: 'Hungry', content: 'I am so hungry, I could eat a horse.'}
                    ],
                    title: 'Enter Title',
                    content: 'Enter Content'
                },



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

Figure 3: Complete the Blanks (Vue.js)

4 Single Page Application (14P)

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.my-blog-space.se
```

Describe the requests and responses exchanged in the following scenario: A user types the URL in their browser, which loads the page. The application shows a login form, which the user fills. The user then clicks a submit button. The application validates the credentials, and then shows a page with a list of blog entries for this user.
Hint: consider which library calls that might be used, which HTTP methods and headers might be used, and what the request and response bodies might contain.

As a second task, write down an example HTTP request and response for the initial call to the URL above. Do not write the Axios code to *generate* an HTTP request, show what the request would actually look like on network level.

5 Accessibility (8P)

Name and briefly describe four principles or techniques that you can use to make your web applications more accessible. Also explicitly discuss which types of users these principles / techniques would support primarily.

6 Networking Protocols (14P)

Depict the 7-layer OSI stack. Briefly describe each layer, and give an example protocol or technology that could be used on that layer.