Western Norway University of Applied Sciences

~~EXAM~~

Solution

Exam code: DAT152

Course name: Advanced Web Applications

Date: December 11, 2023

Type of examination: Written exam

Time: 4 hours (0900-1300)

Number of questions: 6

Number of pages: 25 (including this page and appendices):

Appendices: The last 3 pages

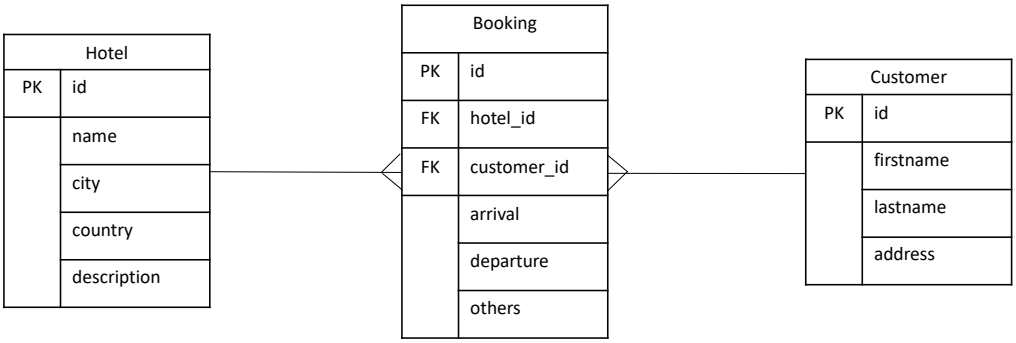
Exams aids: Bilingual dictionary

Academic coordinator: Bjarte Kileng (909 97 348), Tosin Daniel Oyetoyan (405 70 403)

Notes: None

All tasks/subtasks are equally weighted.

Question 1 – Web Frameworks and Services (30% ~ 72minutes)



A 3rd party company (HotelOnline) provides online hotel booking service for customers worldwide. The above model shows the customer hotel-booking model.

a) Build a REST API endpoints for this service. The REST API should provide the following services:
   - Produce a list of all the hotels.
   - Provide details of a hotel.
   - Provide support for creating, updating, and deleting a hotel.
   - Produce a list of all the customers.
   - Provide details of each customer.
   - Provide the support for creating, updating, and deleting a customer.
   - Produce a list of all bookings.
   - Provide details of a specific booking.
   - Provide support for customers to book a hotel, update a booking and cancel their booking.
   - Produce a list of all bookings made by a customer.

   Use the table below to structure your answers (example in the first row).

Service	**API Method**	**Endpoint (URI)**	**HTTP Method**
List all hotels	getHotels	/hotels	GET
…	…	…	…

Solution

Service	**API Method**	**Resource Endpoint (URI Path)**	**HTTP Method**
HotelController			
List all hotels	getHotels	/hotels	GET
Details of each hotel	getHotel	/hotels/{id}	GET
Create a new hotel	createHotel	/hotels	POST
Update a hotel	updateHotel	/hotels/{id}	PUT
Remove an hotel	deleteHotel	/hotels/{id}	DELETE

BookingController			
List all bookings	getBookings	/bookings	GET
Details of a booking	getBookingsById	/bookings/{id}	GET
Update a booking	updateBooking	/bookings/{id}	PUT
Cancel a booking	cancelBooking	/bookings/{id}	DELETE
CustomerController			
List all customers	getCustomers	/customers	GET
Details of a customer	getCustomer	/customers/{id}	GET
Create a customer	createCustomer	/customers	POST
Update a customer	updateCustomer	/customers/{id}	PUT
Delete a customer	deleteCustomer	/customers/{id}	DELETE
Book a hotel by a customer	createCustomerBooking	/customers/{id}/bookings	POST
List bookings by a customer	getBookingsByCustomerId	/customers/{id}/bookings	GET

Note: resource/{id}/resource/{id} is valid if appropriately formed.

b) You are provided with the Customer model class that aggregates the Booking objects. Also, you are provided with the repository class for Customer (CustomerRepository) containing the CRUD methods required to write the service and controller classes. The implementation is using Spring Framework.

```
@Entity
public class Customer {
        ...
    private Long id;

        private Set<Booking> bookings = new HashSet<>();
        public Set<Booking> getBookings() {
          return bookings;
        }
        public void addBooking(Booking booking) {
          bookings.add(booking);
        }
        public void removeBooking(Booking booking) {
          bookings.remove(booking);
        }
    ...
}
```

```
public interface CustomerRepository extends CrudRepository<Customer, Long> {

        List<Customer> findAll();
        Optional<Customer> findById(Long id);
```

```
        Customer save(Customer customer);
        void delete(Customer customer);
}
```

I). Implement the controller class and the methods for all the **Customer** rest api endpoints you identified in a) (example of a template for CustomerController is given below). Note that your method must return the appropriate HttpStatus code. You can also assume that you have two exceptions – CustomerNotFoundException and BookingNotFoundException that you can use in your code.

```
@RestController
@RequestMapping("/hotelonline/api/v1")
public class CustomerController {

        // Write your controller methods here

}
```

Solution

I & c(II)

```
@RestController
@RequestMapping("/hotelonline/api/v1")
public class CustomerController {

        @Autowired
        private CustomerService customerService;

        @GetMapping("/customers")
        @PreAuthorize("hasAuthority('ADMIN')")
        public ResponseEntity<Object> getCustomers(){

                List<Customer> customers = customerService.findAllCustomers();

                if(customers.isEmpty())

                        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
                else
                        return new ResponseEntity<>(customers, HttpStatus.OK);
        }

        @GetMapping(value = "/customers/{id}")
        @PreAuthorize("hasAuthority('USER')")
        public ResponseEntity<Object> getCustomer(@PathVariable("id") Long id) {

                Customer customer = null;
                try {
                        customer = customerService.findCustomer(id);

                        return new ResponseEntity<>(customer, HttpStatus.OK);

                }catch(CustomerNotFoundException e) {

                        return new ResponseEntity<>(e.getMessage(), HttpStatus.NOT_FOUND);
                }
        }

        @PostMapping("/customers")
        @PreAuthorize("hasAuthority('USER')")
        public ResponseEntity<Object> createCustomer(@RequestBody Customer customer){
```

```
                customer = customerService.saveCustomer(customer);

                return new ResponseEntity<>(customer, HttpStatus.CREATED);
        }

        @PutMapping("/customers/{id}")
        @PreAuthorize("hasAuthority('USER')")
        public ResponseEntity<Object> updateCustomer(@RequestBody Customer customer,
                        @PathVariable("id") Long id) throws CustomerNotFoundException{

                customer = customerService.updateCustomer(customer, id);

                return new ResponseEntity<>(customer, HttpStatus.OK);
        }


        @DeleteMapping("/customers/{id}")
        @PreAuthorize("hasAuthority('USER')")
        public ResponseEntity<Object> deleteCustomer(@PathVariable("id") Long id)
          throws CustomerNotFoundException{

                customerService.deleteCustomer(id);

                return new ResponseEntity<>(HttpStatus.OK);
        }

        @GetMapping("/customers/{id}/bookings")
        @PreAuthorize("hasAuthority('USER')")
        public ResponseEntity<Object> getCustomerBookings(@PathVariable("id") Long id)
                        throws CustomerNotFoundException, BookingNotFoundException {

                Set<Booking> bookings = customerService.findBookingsForCustomer(id);

                return new ResponseEntity<>(bookings, HttpStatus.OK);
        }

        @PostMapping("/customers/{id}/bookings")
        @PreAuthorize("hasAuthority('USER')")
        public ResponseEntity<Object> createCustomerBookings(@PathVariable("id") Long id,
@RequestBody Booking booking)
                        throws CustomerNotFoundException {

                Customer customer = customerService.createBookingsForCustomer(id, booking);

                return new ResponseEntity<>(customer, HttpStatus.CREATED);
        }

}
```

II). Write the service class and the methods for the corresponding controller class in I). An example of a template for CustomerService class is given below.

```
@Service
public class CustomerService {

        // Write your service methods here
}
```

Solution

Note: Using the repository class directly in the controller class is valid if the extra logic for performing the task is implemented in the controller. In this case, no need to write the service class below.

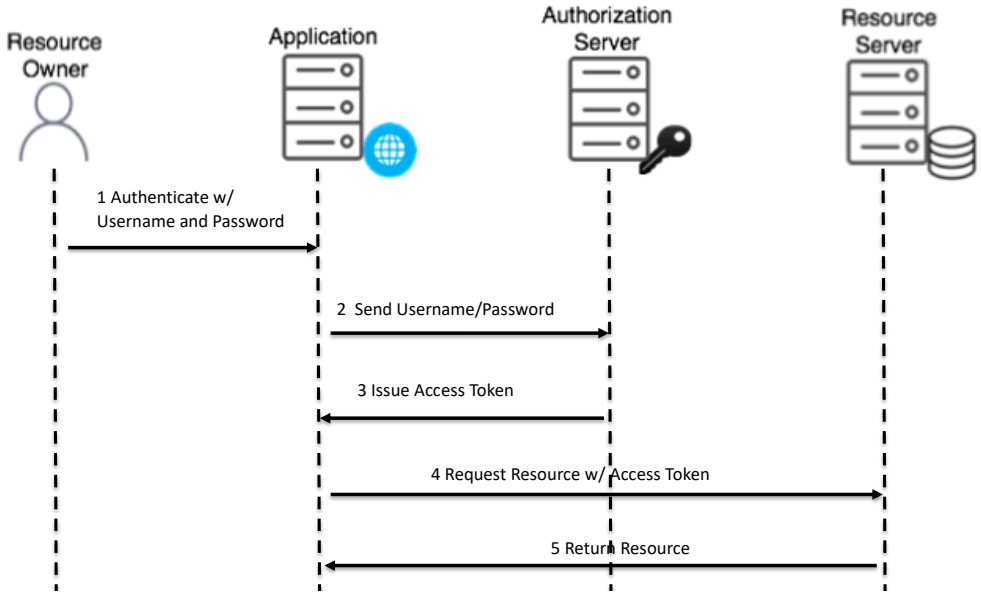
```
@Service
public class CustomerService {

	@Autowired
	private CustomerRepository customerRepository;

	public Customer saveCustomer(Customer customer) {

		customer = customerRepository.save(user);

		return customer;
	}

	public void deleteCustomer(Long id) throws CustomerNotFoundException {

		customer = findCustomer(id);

		customerRepository.delete(customer);
	}

	public Customer updateCustomer(Customer customer, Long id) throws   CustomerNotFoundException {

		findCustomer(id); // trigger exception if not found

		return customerRepository.save(customer);

	}

	public List<Customer> findAllCustomers(){

		List<Customer> allCustomers = (List<Customer>) customerRepository.findAll();

		return allCustomers;
	}

	public Customer findCustomer(Long id) throws CustomerNotFoundException {

		Customer customer = customerRepository.findById(id)
				.orElseThrow(()-> new CustomerNotFoundException("Customer with id: "+id+" not
found"));

		return customer;
	}

	public Set<Booking> findBookingsForCustomer(Long id) throws CustomerNotFoundException{

		Customer customer = findCustomer(id);

		return customer.getBookings();
	}

	public Customer createBookingsForCustomer(Long id, Booking booking) throws CustomerNotFoundException{

		Customer customer = findCustomer(id);

		customer.addBooking(booking);

		return customerRepository.save(customer);
	}
}
```

c) Assume HotelOnline is using a 3rd party identity provider (IdP) for its customers based on OAuth2 protocol.

I). Briefly explain the authorization flow using OAuth2 "resource owner credentials/password grant" scheme between the resource owner and the IdP.

Solution

Resource owner credentials/password grant: The resource owner (customer or staff) will send their credentials (username&password) via an application agent. The application will forward the credentials to the OAuth authorization server (IdP). The auth server will verify the credentials and issue an access token to the resource owner via the application. The application then uses the access token to request for the protected resources from HotelOnline endpoints.



II). Assume that there are two roles 'ADMIN' and 'USER' in the JWT access token where the ADMIN and USER roles are assigned to the HotelOnline staff, and only the USER role is assigned to customers. Protect the REST API endpoints in the CustomerController class using Spring relevant annotations.

```
@RestController
@RequestMapping("/hotelonline/api/v1")
public class CustomerController {

        // update your controller methods in (b) with the appropriate roles and Spring
        authorization notations

}
```

Solution

See solution in b (I) where the annotations below are used.

@PreAuthorize("hasAuthority('ADMIN')")

@PreAuthorize("hasAuthority('USER')")

Strictly for ADMIN: Customers should not be able to see other customers bookings.

Question 2 – Globalization (10% ~ 24minutes)

a) In your own words, briefly explain what are globalization, internationalization, and localization?

Solution

Simple definitions below (Explanations should describe each of the concepts clearly without ambiguity)

Globalization: the process to make an application available globally and it includes internationalization and localization.

Internationalization: the process to enable an application from technical perspective for multiple language support without having to make changes in the design when implementing a new language.

Localization: the process to adapt application for specific language and regional cultural context, called locale and it also includes translation of text into the target language.

b) Given the following jsp code below, internationalize the page and localize it to English and one other language of your choice. Your solution must include the properties files for English and the other language you chose.

```
<body>
  <h1>DAT152 Exam info</h1>
  <div> This course has between 50 and 100 registered students</div>
  <table border="1">
    <tr>
      <td><b>Date:</b>11.12.2023</td>
    </tr>
    <tr>
      <td><b>Total duration:</b>4 hours</td>
    </tr>
    <tr>
      <td><b>Type:</b>Written exam</td>
    </tr>
    <tr>
      <td><b>Pass mark:</b>You need more than 39.9% correct answers to pass</td>
    </tr>
  </table>
<body>
```

Solution

Property files (Resource bundles)

Message_en_EN.properties

```
info=DAT152 Exam info
stat=This course has between {0} and {1} registered students
date=<b>Date:</b> {0}
duration=<b>Total duration:</b> {0} hours
exam=<b>Type:</b> Written exam
pass=<b>Pass mark:</b>You need more than {0} correct answers to pass
```

Message_no_NO.properties

```
info=DAT152 Eksamensinfo
stat=Emnet har mellom {0} og {1} registrerte studenter
```

```
date=<b>Tid:</b> {0}
duration=<b>Total varighet:</b> {0} timer
exam=<b>Type:</b> Skriftlig eksamen
pass=<b>Bestått merke:</b>Du må oppnå mer enn {0} riktig for å stå
```

jsp

```
<body>
  <jsp:useBean id="now" class="java.util.Date"></jsp:useBean>
  <fmt:setLocale value="no_NO"/>
  <fmt:bundle basename="no.hvl.dat152.i18n.Message">

    <h1><fmt:message key="info" /> </h1>
     <fmt:message key="stat">
              <fmt:param>
                <fmt:formatNumber value="50" type="number"/>
              </fmt:param>
              <fmt:param>
                <fmt:formatNumber value="100" type="number"/>
              </fmt:param>
     </fmt:message>

    <table border="1">
     <tr>
            <td>
              <fmt:message key="date">
                <fmt:param>
                  <fmt:formatDate value="${now}" pattern="dd.MM.yyyy"/>
                </fmt:param>
              </fmt:message>
            </td>
     </tr>
     <tr>
            <td>
              <fmt:message key="duration">
                <fmt:param>
                        <fmt:formatNumber value="4" type="number" />
                         </fmt:param>
              </fmt:message>
            </td>
     </tr>
     <tr>
            <td>
              <fmt:message key="exam" />
            </td>
     </tr>
     <tr>
            <td>
              <fmt:message key="pass">
                <fmt:param>
                        <fmt:formatNumber value="0.399" type="percent" maxIntegerDigits="2" minFractionDigits="1"/>
                </fmt:param>
              </fmt:message>
            </td>
     </tr>
    </table>
  </fmt:bundle>
</body>
```

Note: For the date, this answer can also be accepted(<%=new Date(123, 12, 11)%>)

Question 3 – Custom tags (10% ~ 24minutes)

The tasks require to create a custom tag to calculate the exponent of a given number. As shown below, the tag will take a number and an exponent.

```
<body>
   <b>Math.pow(5, 3) = </b><dat152:expFunction number="5" exponent="3"/>
</body>
```

The above tag will produce the result below on a jsp page where it is used:

Math.pow(5, 3) = 125.0

The Math.pow function below can be used for the questions:

double java.lang.Math.pow(**double** number, **double** exponent);

a) Implement the "expFunction" tag using SimpleTagSupport class. You need to override the doTag method and implement your solution. Note that you do not need to write the TLD xml file.

```
@Override
public void doTag() throws JspException, IOException {...}
```

Solution

```
public class ExponentFunction extends SimpleTagSupport {

	private double number;

	private double exponent;

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

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

		double result = Math.pow(number, exponent);

		out.println(result);
	}

	public void setNumber(double number) {
		this.number = number;
	}

	public void setExponent(double exponent) {
		this.exponent = exponent;
	}
}
```

b) Implement the "expFunction" tag using a tag-file. A tag file starts with the line below:

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

Solution

```
<%@ tag language="java" pageEncoding="UTF-8"%>
<%@ tag import="java.lang.Math" %>

<%@ attribute name="number" type="Double" %>
<%@ attribute name="exponent" type="Double" %>


<%
  double result = Math.pow(number, exponent);
%>
<%=result %>
```

Question 4 – Universal Design (5% ~ 12minutes)

Simple definitions below (Explanations should describe the concepts clearly without ambiguity) **See: https://www.w3.org/TR/WCAG21/**

a) What is Web Content Accessibility Guidelines (WCAG)?
Solution
WCAG is a standard that defines how to make web pages and content more accessible to a wider range of people with **disabilities**.

b) Briefly discuss WCAG principle #2 – "Operable" and the guidelines for this principle.
Solution

Operable means that user interface components and navigation must be operable. Users must be able to operate the interface and its controls. The interface cannot require actions that users cannot perform.

Operable has 5 guidelines (WCAG 2.1):

Keyboard Accessible: All functionalities should be available from a keyboard.

Enough Time: Provide users enough time to read and use content.

Seizures and Physical Reactions: Do not design content in a way that is known to cause seizures or physical reactions.

Navigable: Provide ways to help users navigate, find content, and determine where they are.

Input Modalities: Make it easier for users to operate functionality through various inputs beyond keyboard.

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

a) A software development lifecycle consists of requirement, design, implementation, testing, and deployment/release phases. Mention one approach to identify security defects in each phase of the software development lifecycle.

Solution
Requirement: Misuse case, Abuse stories, Protection poker game

Design: Threat Modeling

Implementation: Code review, Static code analysis

System testing: Penetration testing

Deployment: Configuration reviews

b) Given the SQL query below where the name and email are user supplied data:

```
String sqlQuery = "INSERT INTO USERS (id, name, email) VALUES ('"+id+"','"+name+"','"+email+"')";
PreparedStatement stmt = conn.prepareStatement(sqlQuery);
stmt.executeUpdate();
```

I). Briefly explain why this SQL query can be vulnerable to injection.

Solution

Untrusted data (name and email) are bound to the sql query before passing it to the PreparedStatement. The PreparedStatement is ineffective in this instance.

II). Mitigate the SQL injection vulnerability by writing the mitigation code.

Solution

```
String sqlQuery = "INSERT INTO USERS (id, name, email) VALUES (?, ?, ?)";
PreparedStatement stmt = conn.prepareStatement(sqlQuery);
stmt.setInt(1, id);
stmt.setString(2, name);
stmt.setString(3, email);
stmt.executeUpdate();
```

c) Briefly explain the difference between XSS and CSRF

Solution

XSS is an attacker-driven code running within client browsers or other JavaScript engines and can be used for staging different attacks, e.g., steal credentials, deface website, or perform unauthorized activities.

CSRF is a **session-riding** vulnerability that allows attacker to use a victim's **authenticated session** to perform attacker-initiated action and tricks the browser into making unauthorized requests on behalf of the victim. It can be launched using XSS, but it is not required. However, the main difference to XSS is that CSRF requires the victim to be authenticated.

I). Describe how output encoding works to mitigate XSS.

Solution

Output encoding works by using special encoding (e.g., decimal, hex,) to encode html tags that turn such tags (code) into harmless non-executable data when displayed on the web browser.

II). Describe the synchronizer token pattern for mitigating CSRF vulnerability.

Solution

The main goal is to include cryptographically secured Anti-CRSF randomly generated token on sensitive pages which is unknown to an attacker.

– The token is created at the backend when a session is established

– The token is included in a hidden field on subsequent form submits

– When request is processed on the server, application checks that the value submitted with the form matches the value that is stored in the user's session

d) Given the code snippet below: What major **session** vulnerability can you see in the authentication controller?

```
1.  protected void doPost(HttpServletRequest request,
2.  HttpServletResponse response) throws ServletException, IOException {
3.     boolean successfulLogin = false;
4.     String username = validate(request.getParameter("username"));
5.     String password = validate(request.getParameter("password"));
6.     if (username != null && password != null) {
7.              AppUser authUser = getAuthenticatedUser(username, password);
8.              if (authUser != null) {
9.                successfulLogin = true;
10.               request.getSession().setAttribute("user", authUser);
11.             }
12.    }
13.    // do authenticated stuff...
14. }
```

Solution

Anonymous session is not invalidated which can result into a session-fixation attack.

e) A Json Web Token (JWT) consists of three parts: Header.Claims.Signature

I). Describe the two different ways to derive the signature part.

Solution

The signature can be derived using 1) HMAC (symmetric) where a secretkey is hashed with the base64 encoding of header and claims. Or 2) Public key cryptography (asymmetric) by using the private key to encrypt the base64 encoding of header and claims.

- HMAC: Signature = HMAC(secretKey, base64(header), base64(claims))
- Asymmetric: Signature = PrivateKey(base64(header), base64(claims))

II). Assume the JWT token is signed with the secret key "victory". Discuss the major threat to this token.

Solution

The secret key “victory” used for signing the token is weak. Brute force or dictionary attacks can either be used to crack the key.

f) The jjwt-api library that we have used in Oblig4 contains two methods for parsing the JWT token. Method #1 – parse(token) and Method #2 – parseClaimsJws(token). Given that the modified JWTs below:

I). JWT #1

```
HEADER:
{
    "alg": "none"
}
PAYLOAD:
{
    "iat": 1572038943”,
    "admin": "true",
    "user": "Jerry”
}
Encoded JWT =
eyJhbGciOiJub25lIiwia2lkIjoicEp3W.eyJleHAiOjE2OTg5MTA4OTQsImlhdCI6MTY5ODg3NDg5NC.
```

II). JWT #2

```
HEADER:
{
    "alg": "RS256"
}
PAYLOAD:
{
    "iat": 1572038943”,
```

```
    "admin": "true",
    "user": "Jerry"
}
Encoded JWT =
eyJhbGciOiJSUzI1NiIsImtpZCI6InBKd.eyJleHAiOjE2OTg5MTA4OTQsImlhdCI6MTY5ODg3NDg5NC.
```

The results of the code below for parsing JWT #1 and JWT #2 can be any of:
(1) *throw an InvalidTokenException* in line 12.
(2) *invoke the method "removeAllUsers()"* in line 7.
(3) *logs the error ("You are not an admin user")* in line 9.

Using the above information and your knowledge from Oblig4, discuss the results the code will produce for:

I) JWT #1

Solution

throw an InvalidTokenException in line 12. Since the method parseClaimsJws will verify the signature part and the signature was removed in the encoded JWT.

II) JWT #2.

Solution

throw an InvalidTokenException in line 12. Since the method parseClaimsJws will verify the signature part and the signature was removed in the encoded JWT.

```
1.  try {
2.      Jwt jwt = Jwts.parser().setSigningKey(KEY).parseClaimsJws(accessToken);
3.      Claims claims = (Claims) jwt.getBody();
4.      String user = (String) claims.get("user");
5.      boolean isAdmin = Boolean.valueOf((String) claims.get("admin"));
6.      if (isAdmin) {
7.          removeAllUsers();
8.      } else {
9.          log.error("You are not an admin user");
10.     }
11. } catch (JwtException e) {
12.       throw new InvalidTokenException(e);
13. }
```

g) Password can be stored securely by using 1) cryptographically secure hash, 2) slow hash algorithm, 3) unique per user salt, and 4) pepper. Discuss the security benefits of using these four password storage features.

Solution

Cryptographically strong hash will ensure password is only verifiable and not recoverable.

Slow hash slows down attackers by making it longer to generate the hash of a password and thereby increasing the length of time it takes to crack a given password.

Salt will ensure that two passwords are different in the database and increase the length of the rainbow table. Salt can also increase the complexity of brute-forcing if stored on a separate database.

Pepper is a secret not stored in the database and that is combined with the password during login. It is meant to defeat offline dictionary and rainbow-table attacks.

Question 6 – JavaScript (20% ~ 48 minutes)

a) Asynchronous code and JavaScript:

i. Give examples on situations where JavaScript code will be run asynchronously. Observe, you are not asked to write any JavaScript code.

To get points on this task, you must clearly explain the asynchronous behaviour of your examples.

Solution:

Examples of situations where JS is run asynchronously:

- A button was clicked.
- Some text was entered in an input element.
- A file has finished to load.
- Server request got a response.

In all the above examples, a signal is sent when an event occurs. Code can be registered to run when the event happens, e.g. using *addEventListener*. The code is run independent of the main program flow.

ii. The browser is running asynchronously a piece of JavaScript code that is blocking. How does this affect the browser and the application? You must explain your answer.

Solution:

Blocking asynchronous code will block the owning thread. The owning thread will complete freeze and become non-responsive. Unless *WebWorkers*, asynchronous code will block the main thread.

Event handlers of GUI elements can only run in the main thread. Therefore, any event handler attached to an HTML element, e.g. an HTML button will block the main thread when running.

When the main thread is blocking, the browser pane will freeze. The browser pane will not respond to user actions, nor do anything else than running the blocking code.

In older browsers, all panes and windows would freeze.

iii. The JavaScript code example below uses the *async* and *await* keywords:

```
async function myFunction() {
    try {
        const answer = await waitforSomething();
        console.log(`Resolved with value "${answer}"`);
    } catch (e) {
        console.log(`Rejected with "${e.message}"`);
    }
}
```

Code snippet 1: JavaScript code with async and await

What kind of ECMA object must the function *waitforSomething* return?

Write code for a possible *waitforSomething* function. The object returned by *waitforSomething* should include some asynchronous code.

Solution:

The *waitforSomething* function must return a Promise.

A *waitforSomething* function with asynchronous code:

```
function waitforSomething() {
    return new Promise(
        (resolve, reject) => {
            /**
              * The event listener callback will be
              * run asynchronously
              **/
            document.addEventListener('DOMContentLoaded',()=> {
                const bt = document.querySelector("button");
                if (bt !== null) {
                    resolve(bt);
                } else {
                    reject(new Error("No HTML button"));
                }
            })
        }
    )
}
```

b) In this task you are asked to write the JavaScript code of a GUI component **ExamManager** that is explained below.

A web application is made up of three GUI components:

- **ExamManager**, with custom tag *exam-manager*,
- **ExamInfo**, with custom tag *exam-info*,
- **ExamList**, with custom tag *exam-list*.

The component **ExamManager** is made up of the components **ExamInfo** and **ExamList** and is based on the following HTML template:

```
const template = document.createElement("template");
template.innerHTML = `
    <h1>Exam manager</h1>
    <div>
        <exam-info></exam-info>
        <exam-list></exam-list>
    </div>`;
```

Code snippet 2: HTML template for component ExamManager

The component **ExamList** can display a list of course exams with exam dates, and **ExamInfo** lets the user add a new course exam or modify the date of an existing exam.

The illustration below shows a possible view of the **ExamManager** component.

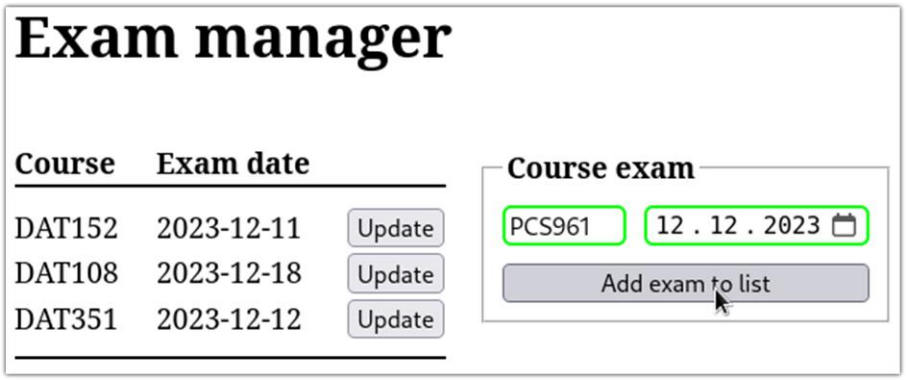


Figure 1: ExamManager *component view*

The right side of Figure 1 shows the **ExamInfo** component, and is also shown below:

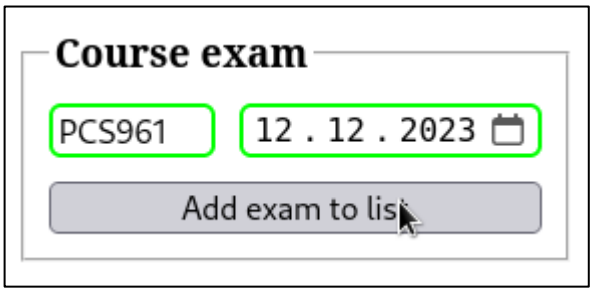


Figure 2: View of ExamInfo component

The left side of Figure 1 shows the **ExamList** component, and is also shown below:

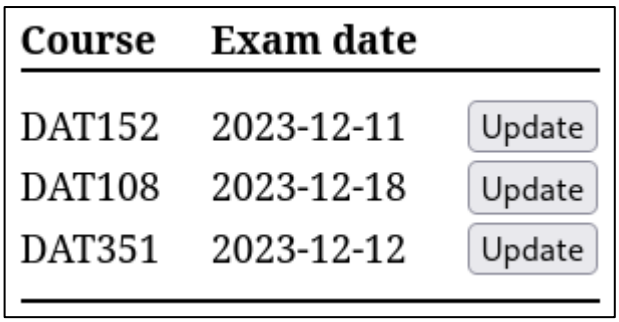


Figure 3: View of ExamList compoenet

Methods of both **ExamInfo** and **ExamList** uses a parameter *examdata*. This is an object with the following properties.

- *code*: The course code
- *date*: Date of the exam

The *date* property of *examdata* is a string formatted as YYYY-MM-DD. You can assume that *code* contains digits and capital letters only, as the **ExamInfo** component will capitalize all letters in the course code and only allow valid course codes.

The HTML element class of **ExamInfo** has two public methods only:

- *setInfo(examdata)*: The method will set exam information that will be displayed by the component.

  Parameters:

  - In parameter: **Object**
  - Return value: None

  With *courseinfo* representing an occurrence of **ExamInfo**, the view show in Figure 2 can be the result of the following use of *setInfo*:

```
courseinfo.setInfo({
   "code": "PCS961",
   "date": "2023-12-12"
});
```

Code snippet 3: Using the method setInfo

- *examinfoCallback (callback)*: The method will add a callback that is run when the user clicks the button *Add exam to list*.

  Parameters:

  - In parameter: Function
  - Return value: Not required

  The method can return a value to identify the callback, but that is not required.

  When *callback* is run on a click at the button, the callback is run with a parameter *examdata* that contains the data of the HTML **INPUT** elements of **ExamInfo**.

The HTML element class of **ExamList** has two public methods only:

- *addExam (examdata)*: The method will add an exam to the list of exams displayed by the **ExamList** component.

  Parameters:

  - In parameter: **Object**
  - Return value: None

  The parameter *examdata* contains the information to be shown by the **ExamList** component.

  If the course code does not exist in the exam list, a new entry with the new exam is added to the end of the exam list.

  If the course code already exists in the exam list, the course is not added to the list, but the exam date is modified according to the new data.

- *examupdateCallback (callback)*: The method will add a callback that is run when the user clicks one of the buttons *Update*.

  Parameters:

  - In parameter: Function
  - Return value: Not required

  The method can return a value to identify the callback, but that is not required.

  When *callback* is run on a click at one of the buttons, the callback is run with a parameter *examdata* that contains the data of the corresponding exam.

The HTML element class **ExamManager** includes the HTML template code of Code snippet 2, and also the following JavaScript code:

```
class ExamManager extends HTMLElement {
    #shadow;

    // Add the necessary private fields

    constructor() {
        super();

        this.#shadow = this.attachShadow({ mode: 'closed' });
        const content = template.content.cloneNode(true);

        // More code

        this.#shadow.append(content);

        // More code
    }

    // Add any additional private methods
 }
```

Code snippet 4:JavaScript code of ExamManager

Task: Fill in the missing code of Code snippet 4 above.

Solution:
The red lines are code added to Code snippet 4 above:

```
class ExamManager extends HTMLElement {
    #shadow;

    constructor() {
        super();

        this.#shadow = this.attachShadow({ mode: 'closed' });
        const content = template.content.cloneNode(true);
        const examinfo = content.querySelector("exam-info");
        const examlist = content.querySelector("exam-list");

        this.#shadow.append(content);

        examinfo.examinfoCallback(
            (examdata) => {examlist.addExam(examdata)}
        );
        examlist.examupdateCallback(
           (examdata) => {examinfo.setInfo(examdata)}
        );
    }
}
```

c) In this task you are asked to write the JavaScript code of **ExamList**. Observe that you are not asked to write the code of **ExamInfo** on this exam.

The component **ExamList** is based on the following HTML templates:

```
const template = document.createElement("template");
template.innerHTML = "<div id="examlist"></div>";
```

*Code snippet 5: Main HTML template for **ExamList***

```
const examtable = document.createElement("template");
examtable.innerHTML = `
    <table>
        <thead><tr><th>Course</th><th>Exam date</th></tr></thead>
        <tbody></tbody>
    </table>`;
```

*Code snippet 6: Template for the HTML Table of **ExamList***

```
const examrow = document.createElement("template");
examrow.innerHTML = `
    <tr>
        <td></td>
        <td></td>
        <td><button type="button">Update</button></td>
    </tr>`;
```

Code snippet 7: Template for the HTML Table row for the display of one exam

See the previous task for the details of the **ExamList** public methods *addExam* and *examupdateCallback*.

Task: Write the JavaScript code of **ExamList**. You can assume that course codes are digits and capital letters only.

Solution:

```
class ExamList extends HTMLElement {
    #examlist;
    #callbacks = new Map();

    constructor() {
        super();

        const shadow = this.attachShadow({ mode: 'closed' });
        const content = template.content.cloneNode(true);
        this.#examlist = content.getElementById("examlist");
        shadow.append(content);
    }

    examupdateCallback(method) {
        const callbackid = Symbol("newexamCallbackID");
        this.#callbacks.set(callbackid, method);
        return callbackid;
    }

    addExam(examdata) {
        let table = this.#examlist.querySelector('table');

        if (table == null) {
            const content = examtable.content.cloneNode(true);
            table = content.querySelector("table");
            this.#examlist.append(content);
        }

        const tbody = table.tBodies[0];
        let row = tbody.querySelector(
            `tr[data-identity='${examdata.code}']`
        );
        if (row === null) {
            const content = examrow.content.cloneNode(true);
            row = content.firstElementChild;
            tbody.append(content);
            row.setAttribute("data-identity", examdata.code);
            row.cells[0].textContent = examdata.code;
            const updateBt = row.cells[2].firstElementChild;
            updateBt.addEventListener('click',
                this.#updateExam.bind(this)
            );
        }
        row.cells[1].textContent = examdata.date;
    }

    #updateExam(event) {
        const row = event.target.closest("tr");
```

```
        const code = row.cells[0].textContent;
        const date = row.cells[1].textContent;

        this.#callbacks.forEach(callback => callback(
            {
                code: code,
                date: date
            }
        ));
    }
}
```

Appendix

Help for question 1 (REST API using Spring Framework)

org.springframework.web.bind.annotation.GetMapping
org.springframework.web.bind.annotation.PutMapping
org.springframework.web.bind.annotation.DeleteMapping
org.springframework.web.bind.annotation.PostMapping
org.springframework.http.ResponseEntity(HttpStatusCode status)
org.springframework.http.ResponseEntity(T body, HttpStatusCode status)
org.springframework.web.bind.annotation.PathVariable
org.springframework.web.bind.annotation.RequestBody
org.springframework.http.HttpStatus
HttpStatus.OK
HttpStatus.CREATED
HttpStatus.NO_CONTENT
HttpStatus.NOT_FOUND
org.springframework.web.bind.annotation.RequestParam
org.springframework.security.access.prepost.PreAuthorize
org.springframework.beans.factory.annotation.Autowired

Help for question 2 (JSTL fmt)

Tag Summary	
requestEncoding	Sets the request character encoding
setLocale	Stores the given locale in the locale configuration variable
timeZone	Specifies the time zone for any time formatting or parsing actions nested in its body
setTimeZone	Stores the given time zone in the time zone configuration variable
bundle	Loads a resource bundle to be used by its tag body
setBundle	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable
message	Maps key to localized message and performs parametric replacement
param	Supplies an argument for parametric replacement to a containing <message> tag
formatNumber	Formats a numeric value as a number, currency, or percentage
parseNumber	Parses the string representation of a number, currency, or percentage
formatDate	Formats a date and/or time using the supplied styles and pattern

parseDate	Parses the string representation of a date and/or time

Help for question 6 (JavaScript)

EventTarget: addEventListener() method

The ***addEventListener()*** method of the **EventTarget** interface sets up a function that will be called whenever the specified event is delivered to the target.

Common targets are **Element**, or its children, **Document**, and **Window**.

Syntax:

```
addEventListener(type, listener)
```

Element: append() method

The *Element.append()* method inserts a set of **Node** objects or string objects after the last child of the **Element**. String objects are inserted as equivalent **Text** nodes.

Syntax:

```
append(param1)
append(param1, param2)
append(param1, param2, /* …, */ paramN)
```

Parameters are **Node** or string objects to insert.

Element: setAttribute() method

Sets the value of an attribute on the specified element. If the attribute already exists, the value is updated; otherwise a new attribute is added with the specified name and value.

Syntax:

```
setAttribute(name, value)
```

Parameter *name* is the name of the attribute, and *value* is the value to assign to the attribute.

Element: getAttribute() method

The *getAttribute()* method of the **Element** interface returns the value of a specified attribute on the element.

Syntax:

```
getAttribute(attributeName)
```

Element and Document: querySelector() method

The *querySelector()* method returns the first element that is a descendant of the element on which it is invoked that matches the specified group of CSS selectors. If no matches are found, null is returned.

Syntax:

```
querySelector(selectors)
```

Element: closest() method

The *closest()* method of the **Element** interface traverses the element and its parents (heading toward the document root) until it finds a node that matches the specified CSS selector.

Syntax:

```
closest(selectors)
```

Element: firstElementChild property

The **Element**.*firstElementChild* read-only property returns an element's first child Element, or null if there are no child elements.

Node: textContent property

The *textContent* property of the **Node** interface represents the text content of the node and its descendants.