SCHOOL OF COMPUTING

DIPLOMA IN INFORMATION TECHNOLOGY
DIPLOMA IN APPLIED AI AND ANALYTICS
DIPLOMA IN CYBERSECURITY & DIGITAL FORENSICS

ST0503 BACKEND WEB DEVELOPMENT

2025/2026 SEMESTER 1
PRACTICAL ASSIGNMENT

Objective of Assignment

To allow students to practice what they have learnt in the module by creating a simplified Bug Bounty System for penetration testers to report and manage discovered vulnerabilities.

Instructions and Guidelines:

1. The assignment should be done individually and will account for **30%** of your final grade.

2. The assignment should be submitted by **Monday, 30 Jun, 2025 08:00 am**.

3. You are **REQUIRED** to do the following for your submission

   - Use the **Github Classroom repository** provided to commit all your progress and changes. It will be reviewed for marking.
   - Complete the CA1 Individual Report. The form is available in BrightSpace under Assignments > Forms > CA1 Individual Report.
   - Prepare the **Declaration of Academic Integrity (SOC) form**. Your Assignment will **NOT** be marked if the form is not submitted. The form is available in BrightSpace under Assignments > Forms > Declaration of Academic Integrity (SOC) form.

- You are also required to **zip up your source code** along and submit it along with your files to Brightspace. You are to remove the **node_module** folder when zipping the files.
- Below is example of what your submission should contain

```
├─ BED-CA1-PYYYYYY-CLASS.zip
├─ Declaration of Academic Integrity (SOC).docx
└─ CA1 Individual Report.docx
```

4. You are required to develop a Backend Server in NodeJS with MySQL Database.

5. The interview will be conducted during the practical lessons on **week 11**. You are expected to explain the program logic and modify the program during the interview. If you are absent from the interview, **you will be awarded zero mark for the assignment.**

6. **No marks will be awarded**, if the work is copied or you have allowed others to copy your work.

7. **50%** of the marks will be deducted for assignments that are received within **ONE (1) calendar** day after the submission deadline. No marks will be given thereafter.

Warning: Plagiarism means passing off as one's own the ideas, works, writings, etc., which belong to another person. In accordance with this definition, you are committing plagiarism if you copy the work of another person and turning it in as your own, even if you would have the permission of that person.

Plagiarism is a serious offence, and if you are found to have **committed, aided, and/or abetted** the offence of plagiarism, disciplinary action will be taken against you. If you are guilty of plagiarism, you may **fail all modules** in the semester, or even be **liable for expulsion**.

Section A: Bug Bounty System

In this section, we will be working on creating a simplified Bug Bounty System for penetration testers to report and manage discovered vulnerabilities. Let's dive into the detailed specifications for the initial database setup and the endpoint requirements.

ERD (Entity-Relationship Diagram):

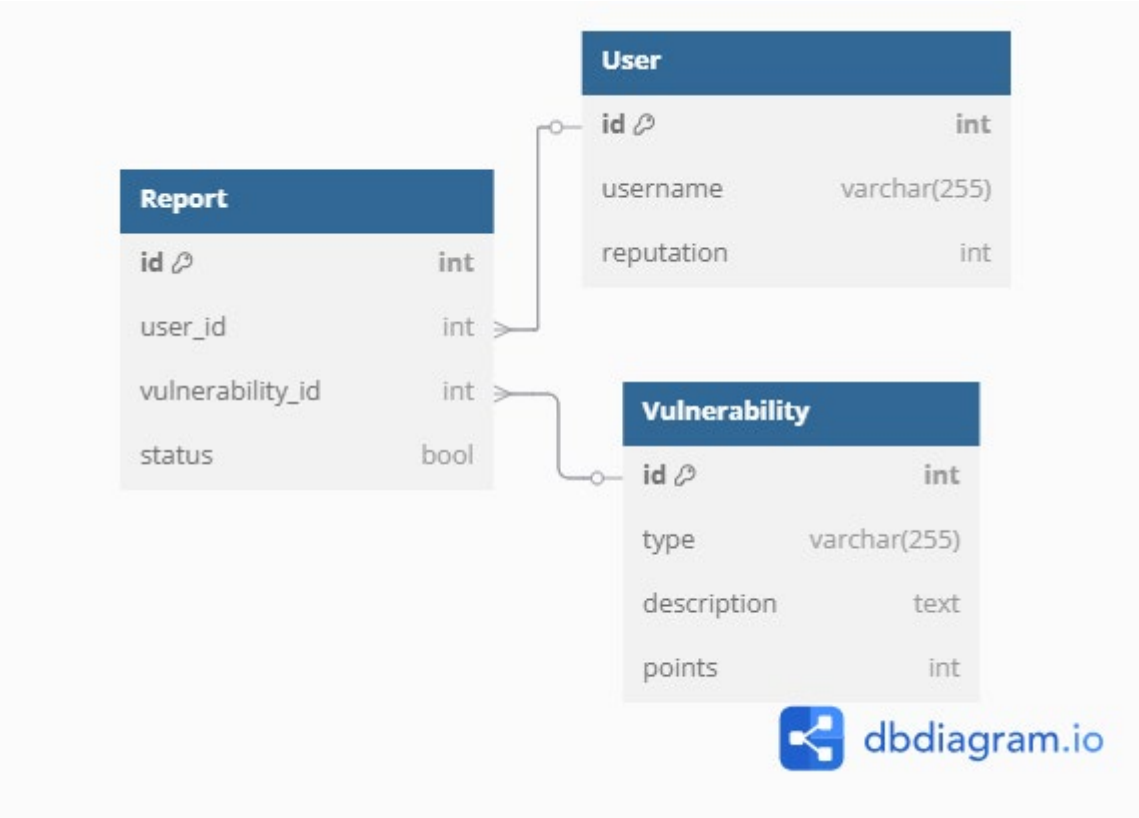


Diagram A

Note: You may add additional columns to the tables. However, it should not affect or changes the response requirement listed in Section A.

MySQL Tables:

```
CREATE TABLE User (
   id INT AUTO_INCREMENT PRIMARY KEY,
   username TEXT NOT NULL,
   reputation INT DEFAULT 0
);

CREATE TABLE Vulnerability (
   id INT AUTO_INCREMENT PRIMARY KEY,
   type TEXT NOT NULL,
   description TEXT NOT NULL,
   points INT NOT NULL
);

CREATE TABLE Report (
   id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT NOT NULL,
   vulnerability_id INT NOT NULL,
   status BOOLEAN NOT NULL DEFAULT 0, -- 0 for Open, 1 for Closed
);
```

Here's the sample data for the Vulnerability table:

id	type	description	points
1	XSS	Cross-Site Scripting (XSS) allows attackers to inject malicious scripts into web pages	50
2	SQL Injection	SQL Injection allows attackers to manipulate a web application's database by injecting malicious SQL code through input fields	100
3	CSRF	Cross-Site Request Forgery (CSRF) security vulnerability tricks authenticated users into performing unwanted actions on a web application.	80

4	Open Redirect	Open Redirect occurs when a web application allows users to control the URL to which they are redirected, potentially leading to phishing attacks or malware distribution	20

Endpoints Requirements for Section A:

1. **POST /users**

   Create a new user by providing their username in the request body. Upon successful creation, the response should include the newly generated id, username, and initial reputation.

   Example Request Body:

```
{
 "username": " pentester123"
}
```

   Example Response:

```
{
 "id": 1,
 "username": " pentester123",
 "reputation": 0
}
```

   Status Code: 201 Created

   Error Handling:

   - If the provided username is already associated with another user, return 409 Conflict.
   - If the request body is missing username, return 400 Bad Request.

2. **GET /users**

   Retrieve a list of all users with their respective id, username, and reputation.

   Example Response:

```
[
 {
  "id": 1,
  "username": " pentester123",
  "reputation": 0
 },
 {
  "id": 2,
  "username": " hackerNoMore",
  "reputation": 0
 }
]
```

   Status Code: 200 OK

3. **GET /users/{id}**

   Retrieve details of a specific user by providing their id. The response should include id, username, and reputation.

   Example Response:

```
{
 " id": 1,
 "username": " pentester123",
 "reputation": 0
}
```

   Status Code: 200 OK

   Error Handling:

   - If the requested user id does not exist, return 404 Not Found.

4. PUT /users/{id}

Update user details by providing id in the URL and updating user in the request body.

Example Request Body:

```
{
 "username": "turnWhiteHat",
 "reputation": 100
}
```

Example Response:

```
{
 "id": 1,
 "username": "turnWhiteHat",
 "reputation": 100
}
```

Status Code: 200 OK

Error Handling:

- If the requested id does not exist, return 404 Not Found.
- If the provided username is already associated with another user, return 409 Conflict.

5. POST / vulnerabilities

Create a new vulnerability by providing type, description, and points in the request body. Upon successful creation, the response should include the newly generated id.

Example Request Body:

```
{
 "type": "XSS",
 "description": "The search bar is vulnerable to Cross-Site Scripting.”,
 "points": 75
}
```

Example Response:

```
{
 "id": 5
}
```

Status Code: 201 Created

Error Handling:

- If the request body is missing any of the required fields (type, description, or points), return 400 Bad Request.

6. GET /vulnerabilities

Retrieve a list of all vulnerabilities.

Example Response:

```
[
{
 "id": 1,
 "type": "XSS";
 "description": "Cross-Site Scripting (XSS) allows attackers to inject malicious scripts into web pages";
 "points": 50
},
{
 "id": 2,
 "type": "SQL Injection";
 "description": " SQL Injection allows attackers to manipulate a web application's database by injecting malicious SQL code through input fields";
 "points": 100
}
]
```

Status Code: 200 OK

7. POST /reports

Create a new report linking a user to a vulnerability. Provide user_id and vulnerability_id in the request body. The initial status of the report should be set to 0 (Open). Upon successful creation of the report:

- The report record is added to the Report table.
- The system should retrieve the points associated with the vulnerability_id from the Vulnerability table.
- These retrieved points should be added to the reputation of the User identified by the user_id in the Report record.
- The response should include the newly created report details (id, user_id, vulnerability_id, status). You should include the updated reputation of the user in the response for confirmation.

Example Request Body:

```
{
 "user_id": 2,
 "vulnerability_id": 5
}
```

Example Response:

```
{
 "id": 1, // note this is report id
 "user_id": 2, // user who found the bug
 "vulnerability_id": 5, // corresponding id in vulnerability table
 "status": 0, //open – ie found and unresolved
 "user_reputation": 100 // Assuming user 2 had 0 reputation and vulnerability 5 has 100 points
}
```

Status Code: 201 Created

Error Handling:

- If the request body is missing user_id or vulnerability_id,, return 400 Bad Request.
- If the user_id or vulnerability_id does not exist, return 404 Not Found.

8. **PUT /reports/{report_id}**

Update the status of a specific report by providing the report_id (matches the id field in Report table) in the URL and the new status (typically 1 to resolve it) in the request body. If the status is updated to 1 (Closed), the points for the associated vulnerability should be added to the reputation of the user specified by the user_id in the request body. This user_id represents the user closing the report. It will overwrite the user_id of the original bug finder.

Example Request Body:

```
{
 "status": 1,
 "user_id": 7 // The ID of the user closing the report
}
```

Example Response:

```
{
 "id": 1, //assuming report id 1
 "status": 1, // indicating report id 1 is closed or resolved
 "closer_id": 7, // Explicitly show who closed it
 "user_reputation": 150 // Updated reputation of user 7
}
```

Status Code: 200 OK

Error Handling:

- If the requested report_id or provided user_id do not exist, return 404 Not Found.
- If the request body is missing status or user_id, return 400 Bad Request.

Section B: Infuse Gamification Idea

In this section, we invite you to explore your creativity for the gamified bug bounty system. Leveraging the reputation points earned from discovering and resolving vulnerabilities, your goal is to motivate increased participation and engagement within the bug hunting community. Embrace the spirit of Role Playing Games (RPGs), the appeal of digital pets, or any other captivating theme that sparks your imagination. Feel free to draw inspiration from popular games, fantasy worlds, or unique concepts to shape the narrative and enhance the bug hunting experience. This theme will be translated to the front end component for your next assignment. You may need to modify your tables from Section A in order to infuse your gamification idea.

Theme Ideas:

1. **Badge Collection:**
   Introduce a system of badges that users can earn by achieving specific milestones in bug hunting. Badges can represent various accomplishments, such as finding specific types of bugs (e.g., "XSS Hunter"), reporting critical vulnerabilities (e.g., "Critical Catch"), or contributing to bug fixes (e.g., "Bug Fixer"). This theme provides a clear sense of progression and encourages users to explore different aspects of bug hunting.

2. **The Bug Bounty Guild:**

   - Users are members of a guild of elite bug hunters.
   - They earn reputation to climb the guild ranks (e.g., Novice, Tracker, Master Hunter).
   - Ranks unlock special "abilities" or tools (e.g., advanced scanning tools, faster report submission).
   - New tables could manage guild ranks, user ranks, and unlocked abilities.

3. **Pet Guardians:**

   - Users raise and evolve digital pets that have an affinity for finding certain types of bugs.
   - The more bugs a user finds, the stronger and more unique their pet becomes.
   - Pets could have different abilities or appearances.
   - New tables would track pets, their attributes, and their owners.

Remember, the goal of this section is to have fun and unleash your creativity while building a gamified bug bounty system. Your design choices should align with the theme and narrative you choose. Feel free to add elements that make your project unique and engaging.

Database Design:

1. Design additional or modify existing tables using Entity Relationship Diagram.
2. You are required to connect tables you designed with the Diagram A shown in page 3.
3. In your ERD, you are required to include the initial tables in Diagram A.
4. Submit the ERD image in your report.

Screenshot Requirements

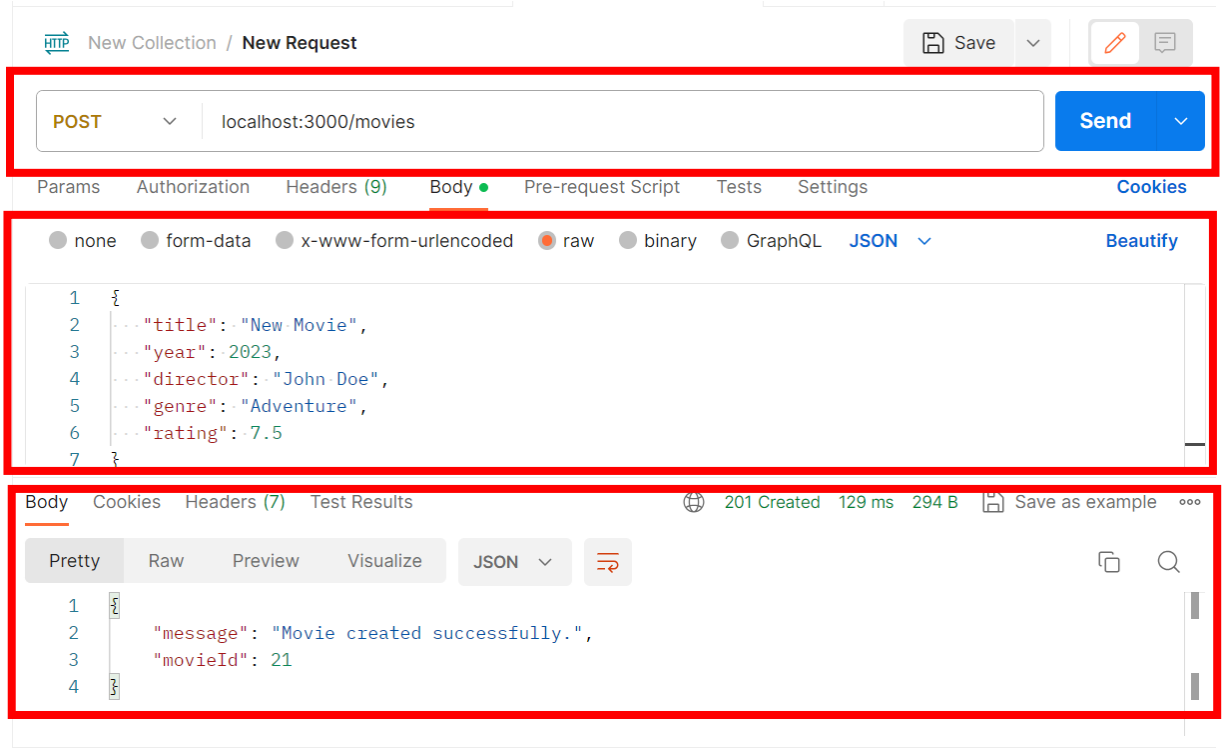
NOTE: Endpoints without screenshots of POSTMAN test result will be

a) Considered as UNTESTED for correctness.
b) Therefore, it will be CONSIDERED as INCOMPLETE.
c) Screenshot(s) for each endpoint must include the following shown in Figure A
  i. Request URL & Type
  ii. Request Body
  iii. Response Body
d) For endpoints with error handling, screenshots for each error handling is required.

i)

ii)

iii)



(Figure B)

Assessment Requirements

We will be evaluating different aspects of your gamified bug bounty system as described in **Section A and B**. These aspects, or competencies, are critical to the overall success and effectiveness of the application. Please take this as an opportunity to showcase your skills by implementing a gamified platform that encourage users to complete the bug bounty system.

NOTE: Section A and B will be assessed as ONE application.

Below are the key competencies that will be assessed:

1. **Architecture** (10 Marks)
   This competency assesses the organization of your codebase and the folder structure of your project. We will be looking for a clear and well-defined architectural design that enables easy maintenance, scalability, and expansion.
2. **Dependency Management** (5 Marks)
   In this aspect, we will evaluate how you manage the package.json, external dependencies and libraries used in your project. Proper dependency management ensures stability, security, and efficient updates.
3. **API Design** (10 Marks)
   API design is crucial for creating a user-friendly and intuitive interface for your task tracker. We'll be examining how well you adhere to RESTful conventions and provide consistent and well-documented endpoints.
4. **Middleware Usage** (10 Marks)
   This competency focuses on the use of middleware to streamline request processing. We will assess how effectively you leverage middleware functions in your application.
5. **Database Design** (5 Marks)
   Database design plays a key role in data management and efficiency. We will evaluate your schema design, normalization, and relationship definitions for effective data storage and retrieval.
6. **SQL Queries** (10 Marks)
   This aspect involves assessing your SQL query skills, including query optimization,

indexing, and efficient data manipulation to ensure smooth and responsive interactions with the database.

7. **Functionality** (10 Marks)
   Functionality refers to how well your task tracker meets the specified requirements and fulfils its intended purpose. We will be looking for a reliable and well-implemented set of core features.
8. **Code Quality** (15 Marks)
   Code quality is essential for maintainability and readability. We will assess the overall cleanliness, organization, and adherence to coding best practices in your codebase. You may want to read up on coding best practices (example: Naming Conventions, Design Patterns)
9. **Modularity** (10 Marks)
   Modularity involves breaking down your code into logical and reusable modules. We will evaluate how well you've organized your project to promote code reusability and maintainability.
10. **Error Handling** (10 Marks)
    Error handling is critical for providing a seamless user experience. We will examine how well you manage errors, provide informative feedback, and handle exceptional situations gracefully.
11. **Documentation** (5 Marks)
    Documentation is key to helping others understand and work with your code. We will assess the clarity and comprehensiveness of your code comments and external documentation.

Please approach each competency with creativity and attention to detail. The focus is on understanding your approach, decision-making, and implementation choices throughout the project.

Version Control Requirement

In this project, we will be using Git for version control and GitHub Classroom for managing code submissions. Version control is a crucial aspect of software development that allows you to track changes, collaborate effectively, and maintain a clean and organized codebase. To ensure smooth progress and efficient evaluation, it is mandatory for all students to commit their code regularly using Git.

Version Control with Git:

1. **Commit Regularly:** It is essential to commit your code regularly (**minimally once a week**) as you make progress on your project. Regular commits demonstrate a well-organized development process, help track changes, and make it easier to revert to previous states if needed.
2. **Commit Rationale:** For each commit, provide a clear and concise rationale in the commit message. A good commit message explains the changes made in the commit and why those changes were made. This helps reviewers understand your thought process and the purpose of each commit.

GitHub Classroom Submissions:

1. **Use GitHub Classroom:** All code must be committed through GitHub Classroom. Follow the provided repository link to create your personal repository and commit your code into it.
2. **Submission README:** Include a detailed README file in your repository. The README should contain clear instructions on how to run your application, any prerequisites or dependencies, and any additional details that will assist the reviewers in evaluating your project.

Penalties for Non-Compliance:

Failure to adhere to the version control requirements and submission guidelines will result in penalties that could impact your final evaluation. Penalties may include:

- **Disorganized Commits:** Poorly organized commits without clear rationale may lead to a deduction of points in the code quality assessment.

- **Incomplete Submission:** If your submission is incomplete or missing critical components, it may be deemed ineligible for evaluation, resulting in a zero score for the affected parts.

Note: The goal of these requirements is to promote good development practices and ensure a fair and transparent evaluation process. By adhering to the version control guidelines and submission protocols, you'll have a well-documented, organized, and properly managed project.

-- End of Assignment Brief --