# Assignment 01

# SOFTWARE REENGINEERING

SE-4001

Course Instructor:

Dr. Isma Ul Hassan


**Group Members:**

| | |
|---|---|
| Zeeshan Ali | 20i-2465 |
| Ans Zeshan | 20i-0543 |
| Saad Shafiq | 20i-1793 |
| Hammad Aslam | 20i-1777 |
| Dayyan Shahzad | 20i-2393 |


Due Date:

Sep 28, 2023

# Doctor Appointment Booking System - eDoc

## Project Overview:

Edoc is a web based project. This project helps a certain medical establishment such as a clinic or a hospital client /patients to request an appointment with a doctor online. This project can also help doctors to manage the schedules of their appointments with their patients. This doctor's appointment system will organize the schedules of each patient's appointment, which will be submitted as a request to the doctor they have selected. The system has 3 sides which are the administrator, the doctor, and the patient. The system admin will populate the list of the doctors with their specialties and along with the doctor's details and system credentials. The patients will browse the doctor's appointment system website to find a doctor that has the specialty of their needs. The patient can check the doctor's weekly schedule to help them to choose the day and time which they can comply for the appointment and they will submit their request for an appointment. After that, the doctors can view all their appointments and the appointment request of the patients for their availability.

## Project Features

### Admin:

- Admin can add doctors, edit doctors, delete doctors
- Schedule new doctor’s sessions, remove sessions
- View patient’s details
- View booking of patients

### Doctors:

- View their Appointment
- View their scheduled sessions
- View details of patients
- Delete account
- Edit account settings

### Patients (Clients):

- Make appointment online
- Create accounts themselves
- View their old booking
- Delete account
- Edit account settings

## Techniques to Analyze Legacy Systems

1. **Stakeholder Interviews:**
    - ❖ Conduct interviews with users who have interacted with the system, including administrators, doctors, and patients.
    - ❖ Gather feedback on their experiences and pain points. Identify any specific issues they have encountered, such as usability concerns or missing features.
2. **Business Impact Analysis:**
    - ❖ Assess how the legacy system contributes to the healthcare organization's daily operations, including appointment scheduling and patient management.
    - ❖ Identify potential risks associated with system disruptions, such as data loss or appointment scheduling failures.
3. **Risk Assessment and Mitigation:**
    - ❖ Identify risks associated with maintaining the legacy system, such as security vulnerabilities or scalability limitations.
4. **Legacy Data Assessment:**
    - ❖ Analyze the data stored within the system's database tables. Identify data models and relationships between different tables.
    - ❖ Determine the quality of the data and identify any data that is no longer relevant.
5. **User Experience (UX) Assessment:**
    - ❖ Evaluate the user interface (UI) of the legacy system and Identify usability issues, such as outdated design elements or non-responsive layouts.
6. **Functional Analysis:**
    - ❖ Document the system's current functionality, including features related to doctor and patient management, appointment scheduling, and user authentication.
7. **Security Assessment:**
    - ❖ Perform a security audit of the code to identify potential vulnerabilities, such as weak authentication mechanisms.
8. **Dependency Mapping:**
    - ❖ Identify external dependencies, libraries, and third-party components used in the project.

9. **Reverse Engineering:**
    - ❖ Visualize the relationships between different modules and functions within the code.
10. **Documentation Review:**
    - ❖ Gather and review existing documentation, including requirements, design documents, and user manuals.
    - ❖ Identify gaps in documentation that need to be filled to improve system understanding.

# Strengths & Weakness:

## Strengths:

1. **Security Measures:**

   The code includes some security measures like session management and checks for user types (admin/non-admin) to restrict access to certain pages. SQL injection appears to be prevented by using prepared statements.

2. **Readability and Structure:**

   The code is relatively well-structured and organized into separate files for different functionalities, making it easier to maintain and understand.

3. **Basic Error Handling:**

   The code has some basic error handling in place, such as checking for duplicate email addresses and ensuring that passwords match during doctor registration.

4. **Usage of CSS for Styling:**

   CSS is used to style the web pages, which separates the presentation layer from the code logic.

## Weaknesses:

1. **SQL Injection Risk:**

   While the code appears to use prepared statements for database queries, the use of raw $_GET and $_POST data directly in SQL queries ($_GET["id"], $_POST["name"], etc.) may still pose a risk of SQL injection if not handled properly in all instances. It's important to consistently use prepared statements for all database interactions.

2. **Lack of Input Validation:**

   The code doesn't seem to perform thorough input validation on user-submitted data, which can lead to unexpected behavior or security vulnerabilities. For example, it doesn't validate the format of email addresses or ensure that input data adheres to expected patterns (e.g., name, NIC, etc.).

3. **Limited Error Handling:**

   While basic error handling is in place, there's room for improvement. It would be beneficial to provide more informative error messages to users and log errors for debugging purposes.

4. **Code Reusability:**

   Code redundancy can be observed in various places, such as SQL queries. Utilizing functions or classes to encapsulate common functionalities can improve code reusability and maintainability.

5. **External Dependencies:**

   The code relies on external factors like the server environment, PHP version, and MySQL. Changes or updates to these external factors may impact the application's functionality.

6. **User Feedback:**

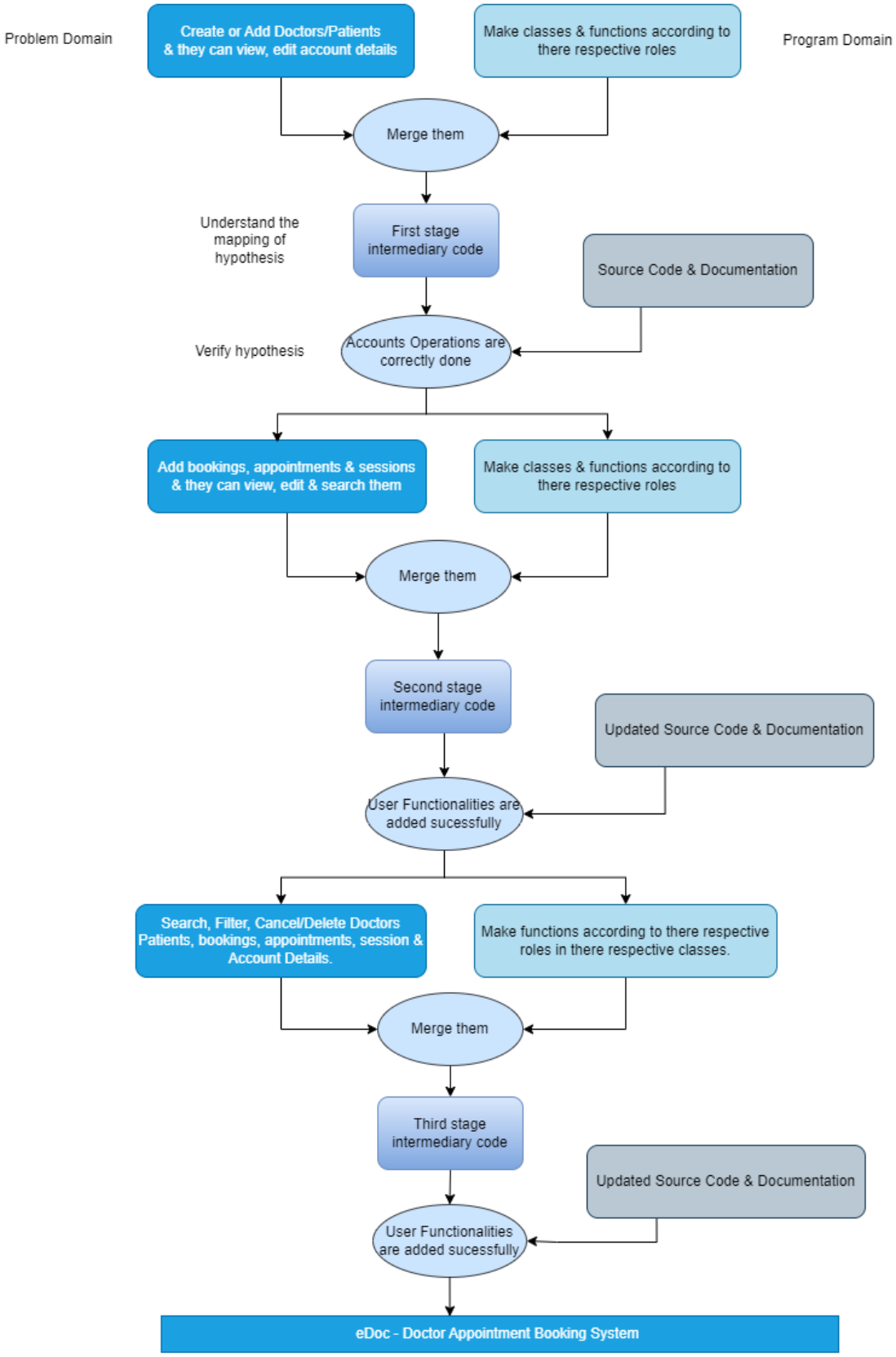
   There is limited provision for providing feedback or error messages to users, which may lead to confusion if something goes wrong.

## Improvement Areas:

1. **Input Validation:**
   - Validate user inputs for email addresses, NICs, names, and other data fields to ensure they conform to expected patterns and formats.
2. **Database Interaction:**
   - Ensure that all database queries use prepared statements consistently to prevent SQL injection.
3. **Error Handling:**
   - Improve error handling by providing informative error messages to users when something goes wrong.
   - Implement robust error logging mechanisms to capture and log errors for debugging purposes.
4. **Code Modularization:**
   - Refactor the code to improve modularity and separation of concerns. Encapsulate related functionalities into functions or classes to enhance code reusability and maintainability.
5. **User Interface (UI/UX):**
   - Enhance the user interface and user experience by implementing responsive design for better usability on various devices.

6. **Security Best Practices:**
    - Implement additional security best practices, such as password hashing and salting for user authentication.
7. **External Dependencies:**
    - Ensure that the codebase remains compatible with the required PHP version and MySQL version.
    - Regularly update and patch external dependencies to address security vulnerabilities.
8. **Code Comments and Documentation:**
    - Provide clear and comprehensive code comments and documentation to make the codebase more understandable for developers who may work on it in the future.
9. **User Feedback:**
    - Implement a user-friendly feedback mechanism to inform users about the outcome of their actions and guide them in case of errors or issues.
10. **Performance Optimization:**
    - Profile the application to identify and address performance bottlenecks, if any, to ensure optimal response times.
11. **Testing and Quality Assurance:**
    - Implement thorough testing, including unit tests, integration tests, and user acceptance testing, to identify and fix issues before they reach production.
12. **Scalability Considerations:**
    - Design the application with scalability in mind to accommodate future growth in terms of users and data.

## Mental Model:



# Abstract Syntax Tree:

## Doctors Module:


```
- program  {
    kind: "program"
  + loc: {start, end}
  - children:  [
    + inline {kind, loc, value, raw}
    + expressionstatement {kind, leadingComments, loc, expression}
    + if {kind, loc, test, body, alternate, ... +1}
    + expressionstatement {kind, leadingComments, loc, expression}
    - expressionstatement  = $node {
        kind: "expressionstatement"
      + loc: {start, end}
      + expression: assign {kind, loc, left, right, operator}
      }
    + expressionstatement {kind, loc, expression}
    + expressionstatement {kind, loc, expression}
    + expressionstatement {kind, loc, expression}
    + inline {kind, leadingComments, loc, value, raw}
    + echo {kind, loc, shortForm, expressions}
    + inline {kind, loc, value, raw}
    + echo {kind, loc, shortForm, expressions}
    + inline {kind, loc, value, raw}
    + expressionstatement {kind, loc, expression}
    + expressionstatement {kind, loc, expression}
    + echo {kind, loc, shortForm, expressions}
    + expressionstatement {kind, loc, expression}
```

```
    + if {kind, loc, test, body, alternate, ... +1}
    + inline {kind, loc, value, raw}
    - if  {
        kind: "if"
      + loc: {start, end}
      + test: variable {kind, loc, name, curly}
      - body: block  {
          kind: "block"
        + loc: {start, end}
        - children:  [
          + expressionstatement {kind, loc, expression}
          + expressionstatement {kind, loc, expression}
          - if  {
              kind: "if"
            + loc: {start, end}
            + test: bin {kind, loc, type, left, right}
            - body: block  {
                kind: "block"
              + loc: {start, end}
              + children: [4 elements]
              }
            - alternate: if  {
                kind: "if"
              + loc: {start, end}
              + test: bin {kind, loc, type, left, right}
              - body: block  {
```

```
    kind: "block"
  + loc: {start, end}
  + children: [2 elements]
  }
- alternate: if  {
    kind: "if"
  + loc: {start, end}
  + test: bin {kind, loc, type, left, right}
  - body: block  {
      kind: "block"
    + loc: {start, end}
    + children: [4 elements]
    }
  - alternate: if  {
      kind: "if"
    + loc: {start, end}
    + test: bin {kind, loc, type, left, right}
    - body: block  {
        kind: "block"
      + loc: {start, end}
      - children:  [
          + expressionstatement {kind, loc, expression}
          + expressionstatement {kind, loc, expression}
          + expressionstatement {kind, loc, expression}
          + expressionstatement {kind, loc, expression}
          + expressionstatement {kind, loc, expression}
```

```
  + expressionstatement {kind, loc, expression}
  + expressionstatement {kind, loc, expression}
  - echo  {
      kind: "echo"
    + loc: {start, end}
      shortForm: false
    - expressions:  [
      - bin  {
          kind: "bin"
        + loc: {start, end}
          type: "."
        - left: bin  {
            kind: "bin"
          + loc: {start, end}
            type: "."
          - left: bin  {
              kind: "bin"
            + loc: {start, end}
              type: "."
            - left: bin  {
                kind: "bin"
              + loc: {start, end}
                type: "."
              - left: bin  {
                  kind: "bin"
                + loc: {start, end}
                  type: "."
```


+ loc: {start, end}
type: "."
+ left: bin {kind, loc, type, left, right}
- right: string {
kind: "string"
+ loc: {start, end}
value: "<br><br>\n </td>\n
</tr>\n <tr>\n
<td class=\"label-td\" colspan=\"2\">\n
<label for=\"nic\" class=\"form-label\">NIC: </label>\n
</td>\n </tr>\n
<tr>\n <td class=\"label-td\"
colspan=\"2\">\n "
raw: "'<br><br>\n </td>\n
</tr>\n <tr>\n
<td class=\"label-td\" colspan=\"2\">\n
<label for=\"nic\" class=\"form-label\">NIC: </label>\n
</td>\n </tr>\n
<tr>\n <td class=\"label-td\"
colspan=\"2\">\n '"
unicode: false
isDoubleQuote: false
}
}
+ right: variable {kind, loc, name, curly}
}
+ right: string {kind, loc, value, raw, unicode, ... +1}
}
+ right: variable {kind, loc, name, curly}
}


}
+ right: string {kind, loc, value, raw, unicode, ... +1
}
+ right: variable {kind, loc, name, curly}
}
+ right: string {kind, loc, value, raw, unicode, ... +1}
}
]
}
]
}
shortForm: false
}
shortForm: false
}
shortForm: false
}
shortForm: false
}
]
}
shortForm: false
}
+ inline {kind, loc, value, raw}
]
errors: [ ]
+ comments: [5 elements]
}

## Patient Module:


## Admin Module:


```
- program  {
    kind: "program"
  + loc: {start, end}
  - children:  [
    + inline {kind, loc, value, raw}
    + expressionstatement {kind, leadingComments, loc, expression}
    + if {kind, loc, test, body, alternate, ... +1}
    + expressionstatement {kind, leadingComments, loc, expression}
    - inline  = $node {
        kind: "inline"
      + loc: {start, end}
        value: "    <div class=\"container\">\n        <div class=\"menu\">\n            <table
        border=\"0\">\n                <tr>\n                    <td style=\"padding:10px\" cols
        <table border=\"0\" class=\"profile-container\">\n                            <tr>\n
        width=\"30%\" style=\"padding-left:20px\" >\n                                <img sr
        width=\"100%\" style=\"border-radius:50%\">\n                            </td>\n
        style=\"padding:0px;margin:0px;\">\n                                <p class=\"profi
        <p class=\"profile-subtitle\">admin@edoc.com</p>\n                            </td>\
        </tr>\n                            <tr>\n                                <td colspan=\"2
        <a href=\"../logout.php\" ><input type=\"button\" value=\"Log out\" class=\"logout-btn b
        </td>\n                            </tr>\n                        </table>\n
        </tr>\n                <tr class=\"menu-row\" >\n                    <td class=\"menu-bt
        <a href=\"index.php\" class=\"non-style-link-menu\"><div><p class=\"menu-text\">Dashboar
        </td>\n                </tr>\n                <tr class=\"menu-row\">\n
        icon-doctor \">\n                    <a href=\"doctors.php\" class=\"non-style-link-
        text\">Doctors</p></a></div>\n                </td>\n                </tr>\n
        >\n                <td class=\"menu-btn menu-icon-schedule \">\n
        class=\"non-style-link-menu\"><div><p class=\"menu-text\">Schedule</p></div></a>\n
        </tr>\n                <tr class=\"menu-row\">\n                    <td class=\"menu-btn
        active menu-icon-appoinment-active\">\n                        <a href=\"appointment.php
        non-style-link-menu-active\"><div><p class=\"menu-text\">Appointment</p></a></div>\n
        </tr>\n                <tr class=\"menu-row\" >\n                    <td class=\"menu-bt
      }
    + expressionstatement {kind, loc, expression}
    + expressionstatement {kind, loc, expression}
    + echo {kind, loc, shortForm, expressions}
    + expressionstatement {kind, loc, expression}
    + inline {kind, loc, value, raw}
    + echo {kind, loc, shortForm, expressions}
    + inline {kind, loc, value, raw}
    + expressionstatement {kind, loc, expression}
    + for {kind, loc, init, test, increment, ... +2}
    + inline {kind, loc, value, raw}
    + if {kind, loc, test, body, alternate, ... +1}
    + inline {kind, loc, value, raw}
    + expressionstatement {kind, loc, expression}
    + if {kind, loc, test, body, alternate, ... +1}
    + inline {kind, loc, value, raw}
    + if {kind, loc, test, body, shortForm}
    - inline  {
        kind: "inline"
      + loc: {start, end}
        value: "    </div>\n\n</body>\n</html>"
        raw: "\n    </div>\n\n</body>\n</html>"
      }
  ]
  errors: [ ]
```