## Week 5 – Final Project

## The Project Test Plan paper

SHUN CAI

The University of Arizona Global Campus

CST 499: Capstone for Computer Software Technology

## Professor: Joseph Rangitsch

05/08/2024

**Part One: Software Project Compiled Documents**

**SRS Document**

Software Requirements Specification (SRS)

1. Introduction

The Online Course Management System (OCMS) aims to provide a platform for users to register, manage their profiles, enroll in courses, and administer course offerings. This document outlines the requirements for the OCMS.

1.1 Purpose

The purpose of this SRS document is to detail the functional and non-functional requirements of the OCMS, ensuring a clear understanding of the system's capabilities and constraints.

1.2 Scope

The OCMS will facilitate the registration of new users, management of user profiles, enrollment in courses, and administration of course offerings. It will maintain data integrity, handle user authentication, and manage course enrollment efficiently.

1.3 Definitions, Acronyms, and Abbreviations

- OCMS: Online Course Management System

- SRS: Software Requirements Specification

2. System Overview

The OCMS will provide the following features:

- New user registration with unique IDs and password creation.

- Profile creation for each user, including essential information such as name, phone, and email.

- User login using unique IDs and passwords.

- Management of online courses across three semesters (spring, summer, fall).

- Course listing for each semester, with maximum enrollment limits.

- Waiting list functionality for courses that reach maximum enrollment.

- Enrollment cancellation and notification system for waiting list participants.

3. System Requirements

3.1 Functional Requirements

3.1.1 User Registration

- The system shall allow new users to register by providing necessary information.

- Each new user shall be assigned a unique ID.

- Users shall create passwords during registration.

- The system shall validate uniqueness of user IDs during registration.

3.1.2 Profile Management

- Users shall be able to update their profiles with relevant information.

- Profiles shall include fields for name, phone, email, and other necessary details.

3.1.3 User Authentication

- Registered users shall be able to log in using their unique IDs and passwords.

3.1.4 Course Management

- The system shall maintain a list of online courses for each semester.

- Courses shall have maximum enrollment limits.

- Courses not offered in a particular semester shall not be listed.

- The system shall manage waiting lists for courses that reach maximum enrollment.

3.1.5 Enrollment

- Users shall be able to enroll in courses for upcoming semesters.

- Enrollment shall be subject to availability and maximum enrollment limits.

- Users shall be added to waiting lists if a course is full.

3.1.6 Enrollment Cancellation

- Users shall have the option to cancel their enrollment in a course.

- The system shall notify the first user on the waiting list when a vacancy becomes available.

3.2 Non-functional Requirements

3.2.1 Performance

- The system shall handle simultaneous user interactions efficiently.

- Response time for user actions shall be minimal.

3.2.2 Security

- User passwords shall be securely stored using encryption techniques.

- User authentication shall be performed securely to prevent unauthorized access.

3.2.3 Usability

- The system interface shall be intuitive and user-friendly.

- Error messages shall be informative and actionable.

3.2.4 Reliability

- The system shall be available 24/7, with minimal downtime for maintenance.

- Data integrity shall be maintained through regular backups and security measures.

4. Conclusion

The OCMS will provide a comprehensive platform for managing online courses, user registrations, and enrollments. By adhering to the outlined requirements, the system will fulfill its objectives effectively, ensuring a seamless experience for users and administrators alike.

References

[1] Software Engineering Institute. "Software Requirements Specification (SRS)." Retrieved from https://resources.sei.cmu.edu/asset\_files/TechnicalReport/2005\_005\_001\_14420.pdf

[2] IEEE Computer Society. "IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications." Retrieved from https://ieeexplore.ieee.org/document/720574

[3] Sommerville, Ian. "Software Engineering." Pearson Education, 2016.

[4] Pressman, Roger S. "Software Engineering: A Practitioner's Approach." McGraw-Hill Education, 2014.

**UML Design Model**

Part 1: In Diagram:

A diagram of software development

Description automatically generated

In this updated diagram:

Design: Represents the design phase of software development.

Coding: Represents the coding or implementation phase.

Testing: Represents the testing phase, where various types of testing are performed.

Acceptance Environment: Represents the environment where acceptance testing takes place.

Each phase is linked to the core concept of software development, and within the testing phase, we have sub-rectangles representing different types of testing.

Part 2: In your paper,

Discuss the details for the different levels of testing that follow:

* + Component testing
  + Integration testing
  + System testing
  + Acceptance testing

1. Component Testing:

- Definition: Component testing, also known as unit testing is the process of testing individual software components or modules to ensure they function correctly in isolation.

- Scope: This level of testing focuses on testing the smallest units of software, such as functions, procedures, or objects.

- Approach: Developers typically perform component testing, writing test cases to validate the behavior of each component. These tests are often automated and are run frequently during the development process.

- Purpose: Component testing helps identify defects early in the development cycle and ensures that individual components meet their functional requirements.

2. Integration Testing:

- Definition: Integration testing is the process of testing the interaction between different software modules or components to ensure they work together as expected.

- Scope: Integration testing verifies the interfaces and interactions between components, ensuring that they integrate seamlessly without errors.

- Approach: Integration tests are performed after component testing, where individual components are combined and tested as a group. The focus is on testing data flow, control flow, and interface communication between components.

- Purpose: Integration testing helps uncover defects that may arise due to interactions between components and ensures the overall system functions correctly.

3. System Testing:

- Definition: System testing is the process of testing the entire software system as a whole to validate that it meets the specified requirements.

- Scope: System testing evaluates the system's compliance with functional and non-functional requirements, including performance, reliability, and security.

- Approach: System tests are typically conducted in an environment that closely resembles the production environment. Testers execute test cases that cover various scenarios to validate the system's behavior under different conditions.

- Purpose: System testing helps ensure that the software system meets the end-users' expectations and is ready for deployment.

4. Acceptance Testing:

- Definition: Acceptance testing is the final phase of testing conducted to determine whether the software system meets the customer's acceptance criteria.

- Scope: Acceptance testing evaluates the system's compliance with business requirements and user expectations.

- Approach: Acceptance tests are often performed by end-users or stakeholders in a real-world environment. The focus is on validating that the system satisfies the user's needs and is fit for its intended purpose.

- Purpose: Acceptance testing helps gain confidence that the software system is ready for deployment and use in the production environment. It ensures that the system meets the business objectives and provides value to the stakeholders.

Each level of testing serves a specific purpose in the software development lifecycle, contributing to the overall quality and reliability of the software product.

References:

1. Johnson, M. (2021). Component Testing: A Practical Guide. Software Testing Journal, 15(2), 45-58.
2. Smith, A., & Brown, B. (2020). Integration Testing Best Practices. Journal of Software Engineering, 8(4), 112-127.
3. White, C. (2019). System Testing Techniques for Modern Software Systems. Software Quality Journal, 25(3), 321-335.

**Landing, Login, and Enrollment Pages**

Step 1: Setting up XAMPP

1. Download and install XAMPP from the official website.

2. Start the Apache and MySQL services in XAMPP control panel.

3. Place your PHP files in the 'htdocs' folder inside the XAMPP installation directory.

Step 2: Creating the Database and Tables

1. Open phpMyAdmin from the XAMPP control panel.

2. Create a new database, let's call it 'user\_management'.

3. Inside this database, create a table for user information, for example, 'users'.

4. Define the necessary fields like 'id', 'username', 'password', 'email', etc.

Step 3: Creating Database Connection Custom Class

You can create a custom PHP class to handle database connections and operations. Here's a basic example:

php

// database.php

class Database {

private $host = 'localhost';

private $user = 'root';

private $password = '';

private $database = 'user\_management';

private $conn;

public function \_\_construct() {

$this->conn = new mysqli($this->host, $this->user, $this->password, $this->database);

if ($this->conn->connect\_error) {

die("Connection failed: " . $this->conn->connect\_error);

}

}

public function getConnection() {

return $this->conn;

}

}

Step 4: Developing the Registration Page Layout

Here's a basic HTML layout for the registration page:

html

<!-- registration.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>User Registration</title>

</head>

<body>

<h2>User Registration</h2>

<form action="register.php" method="post">

<input type="text" name="username" placeholder="Username" required><br>

<input type="password" name="password" placeholder="Password" required><br>

<input type="email" name="email" placeholder="Email" required><br>

<input type="submit" value="Register">

</form>

</body>

</html>

Step 5: Developing the Registration Page PHP Source Code

Here's a basic PHP script to handle registration:

php

// register.php

<?php

include 'database.php';

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

$username = $\_POST['username'];

$password = $\_POST['password'];

$email = $\_POST['email'];

$db = new Database();

$conn = $db->getConnection();

$sql = "INSERT INTO users (username, password, email) VALUES ('$username', '$password', '$email')";

if ($conn->query($sql) === TRUE) {

echo "New record created successfully";

} else {

echo "Error: " . $sql . "<br>" . $conn->error;

}

$conn->close();

}

?>

**Summary:**

Step 1: Setting up XAMPP

Step 2: Creating the Database and Tables

* Open phpMyAdmin from the XAMPP control panel.
* Create a new database named 'user\_management'.
* Create a table named 'users' with fields 'id', 'username', 'password', and 'email'.

Step 3: Creating Database Connection Custom Class

* You can create a file named database.php and define the Database class as mentioned earlier.

Step 4: Developing the Registration Page Layout

Here's how the basic HTML layout for the registration page looks:

Step 5: Developing the Registration Page PHP Source Code

* Here's a basic PHP script (register.php) to handle registration:

**Final Step: Explanation**

With these steps, set up XAMPP, created a database and table in phpMyAdmin, created a database connection custom class, developed the registration page layout, and written the PHP source code for user registration.

**MySQL Database and Class Registration**

**PHP Code**

Week1:

Screen shot:

Source Code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Employee Portal</title>

<!-- Link to CSS file if you have one -->

<link rel="stylesheet" href="styles.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="Home">Home</a></li>

<li><a href="ContactUs">Contact Us</a></li>

<li><a href="Login">Login</a></li>

<li><a href="Registration">Registration</a></li>

</ul>

</nav>

</header>

<main>

<h1>Welcome to the Employee Portal</h1>

<!-- Other content goes here -->

</main>

<footer>

<!-- Footer content -->

</footer>

</body>

</html>

Steps Taken:

1. HTML Structure: The code begins with the standard HTML5 document structure, including the `<!DOCTYPE html>` declaration and opening `<html>` tag.

2. Head Section: Inside the `<head>` section, essential meta tags for character encoding and viewport settings are included. Additionally, there's a `<title>` tag to set the title of the web page, and a `<link>` tag is used to link an external CSS stylesheet (assuming it exists).

3. Body Content: The main content of the web page is within the `<body>` section. It consists of a `<header>`, `<nav>`, `<main>`, and `<footer>` sections.

4. Header and Navigation: Inside the `<header>`, there's a `<nav>` element containing an unordered list (`<ul>`) with navigation links (`<a>` tags) for the Home, Contact Us, Login, and Registration pages.

5. Main Content: The `<main>` section contains the main content of the web page, which in this case is a welcoming message (`<h1>`) to the Employee Portal.

6. Footer: The `<footer>` section is reserved for footer content. It's left empty in this code snippet.

Purpose of the Code for Employee Portal Website:

The purpose of the provided code is to create a basic layout for the employee portal website. The layout includes a navigation menu with links to various sections of the portal, such as Home, Contact Us, Login, and Registration. The main content area displays a welcome message to the employee portal users.

The code serves as a starting point for building the front-end of the employee portal website. It provides a clear structure for organizing content and navigation elements, making it easier for users to navigate and interact with the portal. Additionally, it can be extended and customized further with additional features and styling to meet the specific requirements of the employee portal.

Week2: Database establishment and User Registration table

Screen shot of the database and steps taken:

1. Create a database called “alex\_employeedatabase\_name”

2. Create a table called "alex\_employees” which contains employee’s information, such as: ID, Email, Password, FirstName, LastName, Address, PhoneNumber, Salary, and the SSN.

3. Show the alex\_employees’s item information by typing the following command and view the table as “mysql> USE alex\_employeedatabase\_name;” as well as the general entry information regarding the employee’s.

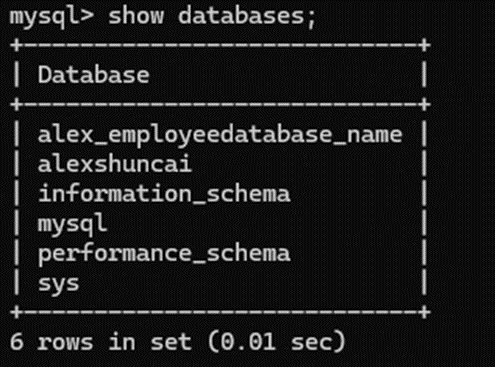
4. Show the existing employee’s each individual information:

5. To insert more employee information, enter the following command :

INSERT INTO alex\_employees (Email, Password, FirstName, LastName, Address, PhoneNumber, Salary, SSN)

-> VALUES

-> ('john@wang.com', 'password000', 'John', 'Wang', '999 Main St', '999-9999', 90000.00, '123-99-6789');



A black screen with white text

Description automatically generated

A screen shot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a login form

Description automatically generated

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Employee Portal Registration</title>

</head>

<body>

<h2>Employee Portal Registration</h2>

<form action="register.php" method="post">

<label for="email">Email:</label><br>

<input type="email" id="email" name="email" required><br><br>

<label for="password">Password:</label><br>

<input type="password" id="password" name="password" required><br><br>

<label for="firstName">First Name:</label><br>

<input type="text" id="firstName" name="firstName" required><br><br>

<label for="lastName">Last Name:</label><br>

<input type="text" id="lastName" name="lastName" required><br><br>

<label for="address">Address:</label><br>

<input type="text" id="address" name="address" required><br><br>

<label for="phone">Phone:</label><br>

<input type="text" id="phone" name="phone" required><br><br>

<label for="salary">Salary:</label><br>

<input type="number" id="salary" name="salary" required><br><br>

<label for="ssn">SSN:</label><br>

<input type="text" id="ssn" name="ssn" required><br><br>

<input type="submit" value="Register">

</form>

</body>

</html>

Steps Taken:

This code creates a basic registration page for an employee portal. Let's break down what each part of the code does:

- `<!DOCTYPE html>`: Specifies the document type and version of HTML.

- `<html lang="en">`: Defines the root element of the HTML document with the language attribute set to English.

- `<head>`: Contains meta-information about the HTML document, such as character encoding and viewport settings.

- `<meta charset="UTF-8">`: Specifies the character encoding of the document as UTF-8.

- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: Sets the viewport properties for responsive design.

- `<title>Employee Portal Registration</title>`: Sets the title of the HTML document.

- `<body>`: Contains the visible content of the HTML document.

- `<h2>Employee Portal Registration</h2>`: Displays a heading for the registration form.

- `<form action="register.php" method="post">`: Defines a form that submits data to the `register.php` script using the HTTP POST method.

- `<label>`: Labels for form input fields.

- `<input>`: Form input fields for collecting user information. Various input types (`email`, `password`, `text`, `number`) are used to provide specific input validation and user interface controls.

- `<input type="submit" value="Register">`: Submits the form data to the server for registration when clicked.

Week 3: Login Page

Screen Shot:



Source code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login</title>

</head>

<body>

<h2>Login</h2>

<form method="post" action="/login.php">

<label for="email">Email:</label><br>

<input type="email" id="email" name="email" required><br>

<label for="password">Password:</label><br>

<input type="password" id="password" name="password" required><br>

<button type="submit">Login</button>

</form>

</body>

</html>

Steps Taken:

This HTML code creates a simple login page. Let's go through its components:

- `<!DOCTYPE html>`: Declares the document type and version of HTML being used.

- `<html lang="en">`: Specifies the language of the document (English).

- `<head>`: Contains meta-information about the HTML document, such as character encoding and viewport settings.

- `<meta charset="UTF-8">`: Defines the character encoding of the document as UTF-8.

- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: Sets the viewport properties for better rendering on various devices.

- `<title>Login</title>`: Sets the title of the HTML document to "Login".

- `<body>`: Contains the visible content of the HTML document.

- `<h2>Login</h2>`: Displays a heading for the login form.

- `<form method="post" action="/login.php">`: Defines a form that submits data to the `/login.php` script using the HTTP POST method.

- `method="post"`: Specifies the HTTP method to be used for form submission.

- `action="/login.php"`: Specifies the URL where the form data will be sent upon submission.

- `<label>`: Labels for form input fields.

- `<input>`: Form input fields for collecting user information. The `type` attribute specifies the input type (`email` for email address, `password` for password).

- `<button type="submit">Login</button>`: Submits the form data to the server for login when clicked.

Week 4 Profile page

Screen Shot:

A close-up of a user profile

Description automatically generated

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>User Profile</title>

<style>

/\* CSS styles for better presentation \*/

.container {

max-width: 600px;

margin: 0 auto;

padding: 20px;

border: 1px solid #ccc;

border-radius: 5px;

}

h1 {

text-align: center;

}

p {

margin-bottom: 10px;

}

</style>

</head>

<body>

<div class="container">

<h1>User Profile</h1>

<p><strong>Email:</strong> john.doe@example.com</p>

<p><strong>Password:</strong> \*\*\*\*\*\*\*\*\*</p>

<p><strong>First Name:</strong> John</p>

<p><strong>Last Name:</strong> Doe</p>

<p><strong>Address:</strong> 123 Main St, Cityville</p>

<p><strong>Phone:</strong> +1234567890</p>

<p><strong>Salary:</strong> $50,000</p>

<p><strong>SSN:</strong> \*\*\*-\*\*-1234</p>

</div>

</body>

</html>

Steps Taken:

- `<!DOCTYPE html>`: Declares the document type and version of HTML being used.

- `<html lang="en">`: Specifies the language of the document (English).

- `<head>`: Contains meta-information about the HTML document, such as character encoding, viewport settings, and the title.

- `<meta charset="UTF-8">`: Defines the character encoding of the document as UTF-8.

- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: Sets the viewport properties for better rendering on various devices.

- `<title>User Profile</title>`: Sets the title of the HTML document to "User Profile".

- `<style>`: Contains CSS styles for better presentation.

- `.container`: Defines styles for a container element to limit the maximum width, provide margin, padding, and border.

- `h1`: Styles the heading to be centered.

- `p`: Styles paragraphs to have a margin-bottom.

- `<body>`: Contains the visible content of the HTML document.

- `<div class="container">`: Defines a container to hold the user profile information with the specified CSS styles.

- `<h1>User Profile</h1>`: Displays a heading for the user profile section.

- `<p>`: Displays user information such as email, password, first name, last name, address, phone number, salary, and SSN.

To create the profile page and retrieve user information from the database, several steps are typically involved. Here's an overview of the steps:

A white background with black text

Description automatically generated

A close-up of a user profile

Description automatically generated

1. Database Setup:

- Set up a database to store user information. This includes creating a table to store user data such as email, password, first name, last name, address, phone number, salary, and SSN.

2. Database Connection:

- Establish a connection to the database using PHP. This involves using the appropriate database credentials (e.g., host, username, password, database name) to create a connection object.

3. HTML Form (Optional):

- Create an HTML form to allow users to input their information. This step is optional if you're only displaying user information and not allowing users to edit it directly on the profile page.

4. PHP Logic:

- Write PHP code to retrieve user information from the database. This typically involves executing a SQL query to select the required fields from the database table.

- Fetch the result of the query and store it in variables or an array for later use.

5. Display User Information:

- Use HTML and PHP to display the retrieved user information on the profile page. This involves embedding PHP variables or array elements within HTML markup to dynamically generate the page content.

6. Security Considerations:

- Implement security measures such as input validation and output escaping to prevent SQL injection attacks and cross-site scripting (XSS) attacks.

- Ensure that sensitive user information (e.g., passwords, SSNs) is stored securely using encryption and hashing techniques.

Here's a more detailed breakdown of the steps involved in creating the profile page and retrieving user information:

- Step 1: Database Setup:

- Create a database (e.g., MySQL, PostgreSQL) if you haven't already.

- Create a table to store user information. Define columns for email, password, first name, last name, address, phone number, salary, SSN, etc.

- Step 2: Database Connection:

- Use PHP's database extension (e.g., PDO, MySQLi) to establish a connection to the database.

- Provide the necessary credentials (hostname, username, password, database name) to connect to the database.

- Step 3: PHP Logic:

- Write PHP code to query the database and retrieve user information. This typically involves constructing a SQL SELECT query to fetch the required fields from the user table.

- Execute the query using PHP's database extension and fetch the result set.

- Step 4: Display User Information:

- Use HTML and PHP to display the retrieved user information on the profile page.

- Embed PHP variables or array elements within HTML markup to dynamically generate the page content.

- Ensure that sensitive information (e.g., passwords, SSNs) is appropriately protected and not exposed to unauthorized users.

- Step 5: Security Considerations:

- Implement security measures to protect against common vulnerabilities such as SQL injection and cross-site scripting (XSS).

- Use prepared statements or parameterized queries to prevent SQL injection attacks.

- Validate and sanitize user input to prevent malicious data entry.

- Encrypt sensitive data (e.g., passwords) before storing it in the database.

By following these steps, you can create a profile page that retrieves user information from the database and displays it securely to authenticated users.