System - Solution

Course: T101 - Tele Demo Course

Created: June 25, 2025

Total Marks: 100

■ Problem Statement

System that allows users to perform basic CRUD (Create, Read, Update, Delete) operations on student records. The frontend should be developed using React and styled with HTML, CSS, and Bootstrap to ensure a clean and responsive user interface. The backend should be built using Express.js, exposing RESTful API endpoints to handle requests and interact with a MySQL database for persistent storage of student information such as name, email, and course. The system should include input validation, modular code organization, and follow best practices in API design and component-based architecture.

■ Knowledge Topics

- MySQL CRUD Operations
- Form Validation

■ Scoring Rubric

Assessment Aspect	Marks
Database Schema Design	25 marks
API Endpoint Implementation	30 marks
Frontend Component Architecture	25 marks
Input Validation & Error Handling	20 marks

■ Knowledge Pills

■ 1. MySQL CRUD Operations

Overview

MySQL CRUD Operations refer to the fundamental operations that allow a system to create, read, update, and delete records in a MySQL database. These operations form the basis of many applications that interact with backend databases to store and manage data efficiently.

Key Principles and Components

Understanding CRUD operations in MySQL involves recognizing each operation's role. Here are some of the core elements:

- **Create:** Insertion of new records using INSERT statements.
- **Read:** Retrieving existing records via SELECT queries.
- **Update:** Modifying existing database entries using UPDATE statements.
- **Delete:** Removing records with DELETE commands.

Why It Matters in This Context

When building applications, such as a Book Inventory Management System, proper implementation of CRUD operations is vital. It ensures data integrity, efficiency in data processing, and seamless interactions between the user interface and the database. The standardized structure of MySQL commands also facilitates easier maintenance and scalability of applications.

Common Applications or Variations

CRUD operations are commonly used in various software applications. Variations might include:

- Stored procedures for complex operations.
- Use of ORM (Object-Relational Mapping) libraries to abstract SQL queries.
- Transactional controls to manage batches of operations.

How This Topic Helps Solve the Alternate Example

In the alternate scenario, where the task is to develop a Book Inventory Management System, MySQL CRUD operations enable efficient management of book records. Whether adding new books, retrieving details about a specific title, updating book statuses, or deleting outdated records, mastering CRUD operations directly influences the system's reliability and performance.

■ 2. Form Validation

Overview

Form Validation is the process of ensuring that user input meets the required criteria before it is sent to the server for processing. This is a critical feature in applications to safeguard data quality and enhance user experience.

Key Principles and Components

Effective form validation involves both client-side and server-side checks to ensure that the data being collected is accurate and complete. Key components include:

- **Required Fields Validation:** Ensuring mandatory fields are not left empty.
- **Format Validation:** Checking the input against a predefined format (e.g., email, phone number).
- **Range Checks:** Verifying that numerical inputs fall within a specified range.
- **Custom Rules:** Applying business-specific logic to input data.

Why It Matters in This Context

In systems like the alternate Book Inventory Management System, form validation prevents invalid data entry, which in turn stops potential issues during data processing. Enhancing the user experience with accurate error messages and feedback is also critical.

Common Applications or Variations

Form validation is widely used across web applications. Common techniques include:

- Client-side validation using JavaScript to provide instant feedback.
- Server-side validation to ensure security and data integrity.
- Utilizing validation libraries like Joi, Yup, or built-in framework utilities.

How This Topic Helps Solve the Alternate Example

Within the context of the Book Inventory Management System, form validation ensures that the librarians or administrators are entering correct and complete data. For example, when adding a new book, validation is used to confirm that ISBN numbers, titles, and author names are provided correctly. This reduces the chances of errors that might corrupt the database and ensures that readers or users receive accurate information when browsing the catalog.

■ DETAILED SOLUTION

■ 1. Database Schema Design [25 marks]

Database Schema Design

The solution begins by designing a precise MySQL table specifically for student records. The table includes fields for **name**, **email**, and **course** that directly satisfy the problem requirements.

The design includes the following problem-specific steps:

• **Primary Key:** An auto-incrementing unique identifier is used as the primary key so that each record can be individually managed.
• **Field Specifications:** Data types are deliberately chosen (e.g., VARCHAR for name and email, and possibly a TEXT or ENUM for course) to match the expected input.
• **Constraints:** A unique constraint on the email field is implemented to avoid duplicate student entries and ensure data integrity.
• **Indexes and Optimization:** Indexes on fields that are often queried, such as course, improve query performance in CRUD operations.

This structured approach ensures that the database schema is optimized for the intended CRUD operations and fulfills all problem-specific details.

■ 2. API Endpoint Implementation [30 marks]

API Endpoint Implementation

The backend solution is centered around a set of RESTful API endpoints carefully tailored to perform CRUD operations on student records through Express.js.

Key implementation steps include:

• **Endpoint Definitions:** Specific endpoints such as GET /students, GET /students/:id, POST /students, PUT /students/:id, and DELETE /students/:id are defined to map directly to problem requirements.
• **Modular Code Organization:** Each endpoint is implemented in a modular fashion ensuring separation of concerns, which enhances maintainability and scalability.
• **Database Integration:** Each endpoint interacts with the MySQL database using query statements that directly mirror CRUD operations (e.g., INSERT for create, SELECT for read, UPDATE for update, DELETE for delete), ensuring the back end remains synchronized with the front end.
• **RESTful Best Practices:** Status codes and response messages are carefully chosen to communicate the success or failure of each operation, thereby

providing clarity for debugging and client-side interaction.

This section establishes a clear pathway for handling all student record operations with precision, directly addressing the problem’s requirements.

■ 3. Frontend Component Architecture [25 marks]

Frontend Component Architecture

The solution leverages a component-based architecture in React, which directly addresses the need for a clean and responsive user interface.

Key steps in the implementation include:

• **Component Breakdown:** The frontend is designed with discrete components for listing student records, viewing details, creating new entries, editing, and deleting. This makes the UI more manageable and inherently scalable.
• **Responsive Design:** Integration of Bootstrap and custom CSS ensures that the components adjust seamlessly to different screen sizes while maintaining a consistent look and feel.
• **Separation of Concerns:** The logic for fetching data, updating state, and handling UI updates is encapsulated within each component, thereby promoting clarity and less interdependencies.
• **Data Flow:** The React application is structured to handle data in a unidirectional flow, ensuring that the interface updates in sync with the underlying database changes triggered by API calls.

This section fully addresses the requirement for a responsive, component-based front end tailored for student record management.

■ 4. Input Validation & Error Handling [20 marks]

Input Validation & Error Handling

This solution pays special attention to input validation and error handling which are critical for ensuring data integrity throughout the system.

The problem-specific approach includes:

• **Frontend Validation:** Before form submission, each input (name, email, course) is validated using HTML input types and custom event handlers, which immediately alerts the user in case of invalid data.
• **Backend Validation:** In Express.js, each API endpoint incorporates thorough validations for incoming data, ensuring that only valid and sanitized entries reach the MySQL database.
• **Error Messaging:** Comprehensive error handling is implemented so that each potential failure (e.g., invalid email format, missing required field) returns a clear error message, aiding in quick debugging and user feedback.
• **Modular Approach:** Both front end and back end validations are encapsulated in modular functions that can be reused across multiple endpoints and components, ensuring consistency in error responses.

This targeted validation approach not only prevents faulty data from being entered but also supports the overall robustness of the application as per the problem's specifications.

Solution generated on June 26, 2025 at 03:38 PM