Webmail Server Development Report

Assignment Overview

The objective of this assignment was to reproduce, understand, and enhance the server-side code from Chapter 8 of the textbook. This chapter involves creating the backend for a webmail system using Node.js and TypeScript. The enhanced implementation includes exclusive features that improve the webmail system's functionality and user experience.

---

1. Environment Setup

- **Operating System**: Ubuntu 22.04 LTS
- **Computer Architecture**: x86_64
- **Node.js Version**: 18.17.0
- **TypeScript Version**: 5.1.6
- **Installed Packages**:
    - `express`: For creating the server and handling HTTP requests.
    - `body-parser`: For parsing request bodies.
    - `jsonwebtoken`: For authentication using JSON Web Tokens (JWT).
    - `mongoose`: For connecting to MongoDB.
    - `nodemailer`: For sending emails.
    - `dotenv`: For managing environment variables.
    - `bcryptjs`: For hashing passwords.
- **Database**: MongoDB (local instance or cloud-based MongoDB Atlas).

---

2. Steps to Reproduce the Code

Install Dependencies:

npm install express body-parser jsonwebtoken mongoose nodemailer dotenv bcryptjs

1. **Install Dependencies**:

   ```
   npm install express body-parser jsonwebtoken mongoose nodemailer
   dotenv bcryptjs
   ```

2. **Initialize Project**:

   npm init -y

tsc --init

3. **Server Code Structure**:

   - `src/index.ts`: Entry point for the server.
   - `src/models/User.ts`: Mongoose schema for user accounts.
   - `src/routes/auth.ts`: Routes for user authentication (login, registration).
   - `src/routes/mail.ts`: Routes for managing emails (send, receive, delete).
   - `src/middleware/authMiddleware.ts`: Middleware for verifying JWT tokens.

4. Run the Server:

tsc && node dist/index.js

3. Added Features

- **Spam Detection**: Implemented a basic keyword-based spam filter that flags incoming emails containing specific spam-related words.
- **Two-Factor Authentication (2FA)**: Integrated a one-time password (OTP) system for user login to enhance security.
- **Email Search and Filters**: Users can search emails by subject, sender, or date, and apply custom filters.
- **Priority Labeling**: Emails can be labeled as "High Priority" for better organization.

4. How to Test the Code

- **Step 1: User Registration**:
  - Use the `/auth/register` endpoint to create a new user account.
  - Send a POST request with `username`, `email`, and `password`.
- **Step 2: User Login**:
  - Use the `/auth/login` endpoint to authenticate the user and receive a JWT token.
  - Include the token in the Authorization header for subsequent requests.

- **Step 3: Sending an Email**:
    - Use the `/mail/send` endpoint with recipient email, subject, and body.

Example:
```
 {
  "to": "recipient@example.com",
  "subject": "Test Email",
  "body": "This is a test email."
}
```

    -
- **Step 4: Viewing Emails**:
    - Use the `/mail/inbox` endpoint to fetch all received emails.
- **Step 5: Flagging Spam**:
    - Use the `/mail/spam` endpoint to list flagged emails.

5. REST and Its Importance

REST (Representational State Transfer) is a design principle that helps in building scalable and stateless web applications. Its relevance to this webmail system includes:

- **Scalability**: REST enables stateless communication between the client and server, allowing the system to handle a large number of simultaneous requests.
- **Ease of Integration**: REST APIs can be consumed by various clients, such as web browsers, mobile apps, or third-party applications.
- **Modularity**: With REST, the webmail system's functionalities are divided into discrete routes (e.g., `/auth/register`, `/mail/send`), making the code easier to maintain and extend.
- **Interoperability**: REST APIs use standard HTTP methods (GET, POST, PUT, DELETE), which are widely understood and supported across platforms.