Crime Management System

Problem statement:

The objective of this case study is to design and implement a Crime Management System (CMS) for a law enforcement agency. The system should facilitate efficient management, tracking, and analysis of criminal activities This system caters to both individual and business customers, offering services such as account management, funds transfer, loans, investments, and financial reporting.

Scope:

1. **Incident Management:** Users can create and track the incidents.
2. **Officer and user management:** Enable the station head to add and remove officers and view the list of officers, incidents and users.
3. **Officer assignment:** Station Head can assign incident to an officer.
4. **Download incident Report:** Generate incident report

Technologies:

- Frontend: React.js / Angular Js.
- Backend: Java, Spring Boot/C#, .Net / Python Djnago for API development.
- Database: MySql / Sql Server.
- Authentication: JSON Web Tokens (JWT) for secure user authentication.

Reference Link: https://www.sbi.com/

Use case Diagram:

Actor: User

- Use Case: Register Account
- Use Case: Log In
- Use Case: Manage Incidents
    - Create new Incident (murder/theft/missing person/report abuse)
    - View incident history reported by him/her.
    - View Incident Details of a particular incident (incident id for tracking/officer assigned/ progress of investigation)
- Use Case: Check status and Download incident status card (Contains details of incident reported, officer
- Use Case: Log Out

Actor: Officer

- Use Case: Log In
- Use Case: Manage Incidents
    - View assigned incidents with status highlighted (initiated/active/closed/verified.
    - Close the incident by changing status to closed.
      View Incident Details

- Use Case: Log Out

Actor: Station Head:

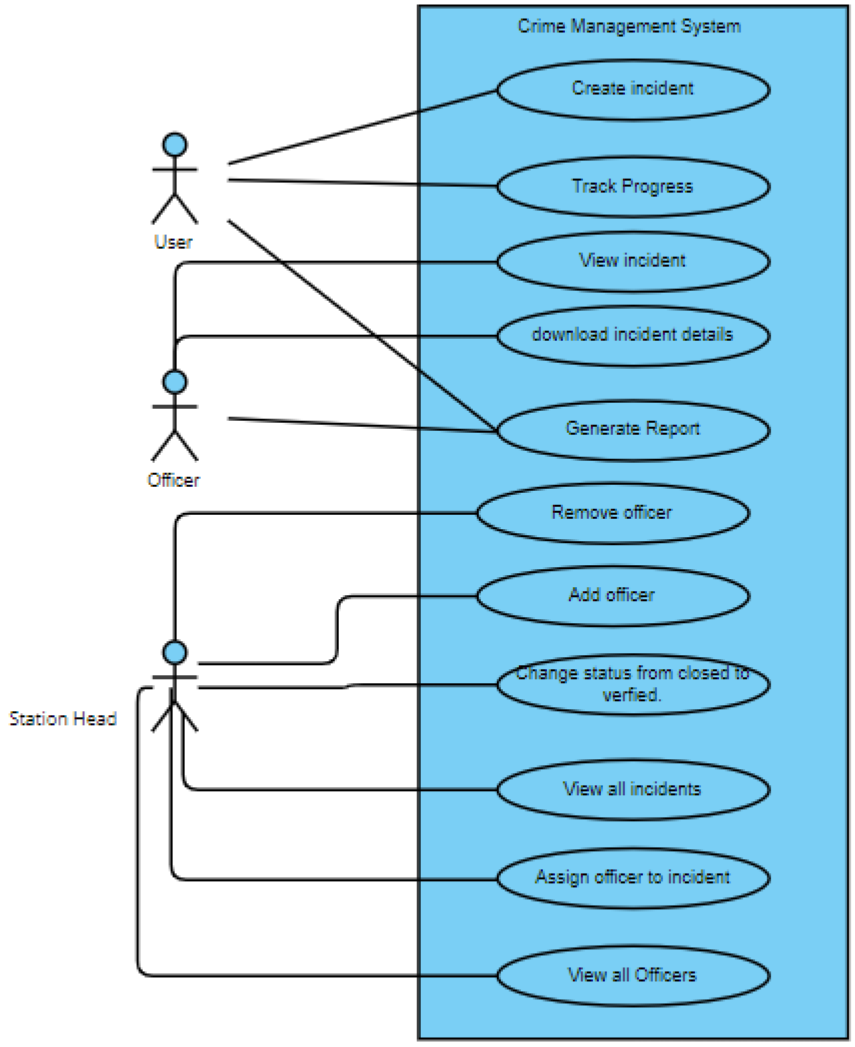
- Use Case: Manage User Accounts and bank employees

System: Security and Authentication

- Use Case: Authenticate User

Associations:

- User creates incident (Create incident, Check status).
- Station Head assigns incidents to existing officers based on less number of active cases.
- Station Head can add and remove officers.
- Station Head can verify the submitted close case and change the status to verified
- Officer can view his assigned cases and details upon login
- Officer can update the status to closed which is send to approval of station head who has to verify the case.



Development Process:

1. **User Registration and Login:**
    - Civilians can create accounts, providing personal details (Name, Address, Date of birth, aadhaar number, age should be calculated from DOB, PAN number)
    - The system validates the information and creates user profiles.
    - Users log in using their credentials (username/email and password).

2. **User Dashboard and Incident Reporting:**

   Users can log in to profile and access the incident management section which has the following functionalities.

    - **Create new Incident** (murder/theft/missing person/report abuse)
        - Each incident should be assigned a unique id and should be emailed to user upon successful incident creation. Downloadable pdf to be generated upon successful incident creation. Each incident will should be automatically assigned with a status **initiated** upon successful creation.

- **View all incidents** reported by this particular user with incident status highlighted.(**initiated/active/closed/verified**)
- **View Incident Details** (incident id for tracking/officer assigned/ status of incident)
- Types of incidents that can be reported online
    - **Lost property** -property which cannot be located; does not include property that you think was taken from your possession. Example: forgetting a package on the subway
    - **Petit larceny** -property that was taken without permission valued at Rs.1000 or less. Does not include property that was forcibly taken from you.
      Example: Bike being stolen in front of a store
    - **Criminal mischief**- intentional damage to property by a person Example: Intentionally breaking a car window
    - **Graffiti**- intentionally drawing, scratching, or etching on property. Example: spray painting a store front. If reporting graffiti, upload a picture of graffiti.
- Appropriate forms to be displayed for collecting extra information based of each incident type based on the above criteria.

3. **Station Head Dashboard:**
    - Add new officer with all necessary details.
    - View all officers with number of incidents active incidents highlighted.
    - View individual officers incident details(active/closed/verified)
    - Assign incident to a particular officer. Incident status should be changed to active.
    - Verify once officer has closed a case. Change the status to verified.
    - Email should be sent to corresponding user who reported the incident mentioning the closure of incident and number of active cases should be updated.

Security and Compliance:

- User authentication and authorization are enforced to ensure data privacy.

1. JWT Authentication:

JWT authentication involves generating a token upon successful user login and sending it to the client. The client includes this token in subsequent requests to authenticate the user.

- User Login: Upon successful login (using valid credentials), generate a JWT token on the server.
- Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
- Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.
- Token Transmission: Send the signed token back to the client as a response to the login request.

- Client Storage: Store the token securely on the client side (e.g., in browser storage or cookies).

2. JWT Authorization:

JWT authorization involves checking the token on protected routes to ensure that the user has the required permissions.

- Protected Routes: Define routes that require authentication and authorization.
- Token Verification:
    1. Extract the token from the request header.
    2. Verify the token's signature using the server's secret key.
- Payload Verification:
    1. Decode the token and extract user information.
    2. Check user roles or permissions to determine access rights.
- Access Control: Grant or deny access based on the user's roles and permissions.

Logout:

- Logging out involves invalidating the JWT token on both the client and the server to prevent further unauthorized requests.

2. **User Profile:**
    - Each actor of the application must have a user profile page.
    - Profile page must allow the user to upload picture, edit/delete information such as name, phone number.
    - It is recommended to have change password feature in profile page.

Project Development Guidelines

The project to be developed based on the below design considerations.

1	**Backend Development**	• Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services. • Use Java/C# latest features. • Use ORM with database. • perform backend data validation. • Use Swagger to invoke APIs. • Implement API Versioning. • Implement security to allow/disallow CRUD operations. • Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant.

		• Any error message or exception should be logged and should be user-readable (not technical). • Database connections and web service URLs should be configurable. • Implement JWT for Security. • Follow Coding Standards with proper project structure.
2	**Error Handling and Exception Management**	• Ensure errors and exceptions are handled with the status code. • Ensure that there are appropriate error-handling mechanisms in place to handle unexpected situations gracefully. Implement custom exceptions. • Ensure robust error logging and reporting mechanisms.
3	**Testing and Debugging:**	• Implement Unit Test Project for testing the API. • Establish what each unit test aims to verify request mappings, status codes, returned views or response bodies. • Focus on business logic and typically mock away data access layers. • Ensure the interaction with the database behaves as expected. ◦ Test Structure ◦ Arrange: Set up the test data and mocks. ◦ Act: Execute the method being tested. ◦ Assert: Verify the output or behavior with assertions.
4	**Logging**	• Define different logging levels (e.g., DEBUG, INFO, WARN, ERROR) based on the severity of the logged message. • Ensure that log messages are descriptive and informative, providing enough context to understand the event or condition being logged. • Include relevant information such as timestamps, user IDs, and error codes in log messages. • Must be logged in a separate file under the source folder of the application.
5	**Email notification**	• send email notification to the user when there is user action performed. • notify the user when user logged in into the profile and when database interaction is performed the notification are send to corresponding email.
6	**File Management**	• PDF formats should be used for bills/invoices and such kinds. • Ensure to format the data properly in the PDF and make it downloadable and sharable via email. • If the application (like quiz tool) has a way to interact with excel/csv (like bulk upload/download) implementation is a must.

Frontend Constraints

1.	**Layout and Structure**	Create a clean and organized layout for your registration and login pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes.
2	**Visual Elements**	**Logo:** Place your application's logo at the top of the page to establish brand identity.
		Form Fields: Include input fields for email/username and password for both registration and login. For registration, include additional fields like name and possibly a password confirmation field.
		Buttons: Design attractive and easily distinguishable buttons for "Register," "Login," and "Forgot Password" (if applicable).
		Error Messages: Provide clear error messages for incorrect login attempts or registration errors.
		Background Image: Consider using a relevant background image to add visual appeal.
		Hover Effects: Change the appearance of buttons and links when users hover over them.
		Focus Styles: Apply focus styles to form fields when they are selected
3.	**Color Scheme and Typography**	Choose a color scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colors for readability. Select a legible and consistent typography for headings and body text.
4.	**Registration Page/Add Bank Employee**	**Form Fields:** Include fields for users to enter their name, email, password, and any other relevant information. Use placeholders and labels to guide users.
		Validation: Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors. **Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission.
		Password Strength: Provide real-time feedback on password strength using indicators or text. **Password Requirements**: Clearly indicate password requirements (e.g., minimum length, special characters) to help users create strong passwords.
		Registration Success: Upon successful registration, redirect users to the login page.
5.	**Login Page: Customer/Bank Employee**	**Form Fields:** Provide fields for users to enter their email and password.
		Password Recovery: Include a "Forgot Password?" link that allows users to reset their password.
6.	**Common to React/Angular**	• Use Angular/React to develop the UI. • Implement Forms, data binding, validations, error message in required pages. • Implement Routing and navigations. • Use JavaScript to enhance functionalities. • Implement External and Custom JavaScript files. • Implement Typescript for Functions Operators.

		• Any error message or exception should be logged and should be user-readable (and not technical). • Follow coding standards. • Follow Standard project structure. • Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets.

Good to have implementation features:

- Generate a SonarQube report and fix the required vulnerability.
- Use the Moq framework as applicable.
- Create a Docker image for the frontend and backend of the application .
- Implement OAuth Security.
- Implement design patterns.
- Deploy the docker image in AWS EC2 or Azure VM.
- Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies.
- Use AWS RDS or Azure SQL DB to store the data.