



2025

SMARTMDINA

Full Stack Web Application



[securityinshadows](#)

[HTTPS://WWW.LINKEDIN.COM/IN/AYOUBWAHMANE/](https://www.linkedin.com/in/ayoubwahmane/)

Foreword:

This application was developed for a university assignment. As an MVP, it still contains some bugs, errors, and unfinished work, this will be addressed over time as I revise this and other similar projects.

Table of Contents :

Introduction:	5
Scope:	5
Target Audience:	6
Development Methodology:	7
Approach:	7
Technologies Used:	7
Development Plan:	8
Design Phase:	8
UML Diagrams:	8
Wireframes	9
Web App Architecture:	10
MSCW Analysis & Basic Database schema:	10
Development Timeline:	15
Week 1: Design and Initial Development:	15
Week 2: Refinement and Testing:	15
Frontend Development Plan:	16
Framework and Tools	16
Pages and Features	16
Backend Development Plan:	16
Framework and Tools	16
API Endpoints	17
Security:	17
Database Design Plan:	17
Tables:	17
Testing and Validation:	18
Methods	18

Bug Fixes.....	18
System Architecture:	19
Overview:	19
User Roles:	19
Module Interactions:	19
Developed UML Diagrams:	20
Class Diagram:	20
The main classes include:.....	20
Use Case Diagram:	25
Entity-Relationship Diagram:	26
Implementation Details:	28
Backend Logic:	28
User Management	28
Tourism/Educational Services Management	28
News Management	28
Frontend Logic:	29
Database Design:	29
Testing and Validation:	32
Testing Methodology:.....	32
Test Cases:	32
User Authentication	32
Posting Tourism Data	32
Bug Fixes:	32
Conclusion:	34
Summary:	34
Future Enhancements:	Erreur ! Signet non défini.
Reflection and Future Prospects:.....	34

References:	35
--------------------------	-----------

Introduction:

As technology engulfs our daily lives further, navigating through cities using various applications has become a common occurrence, thus comes the idea of centralizing all relevant resources of one city into one web application instead of having to resort to multiple ones, this is where the smart city web application idea was born.

This report will document the design, development, and implementation of “casamobile”, a project dedicated to creating a smart city web application for Casablanca.

Objective of the Project “casamobile/smartmdina”:

This project aims to create a Smart City web application that serves as a comprehensive digital platform for city residents, tourists, job seekers, students, and administrators. The application provides functionalities for managing city services, accessing vital resources, and facilitating engagement between users and the system through efficient, modular, and scalable software architecture.

Casablanca lacks any comprehensive platform to help its inhabitants navigate around the city, therefore the need for this application is clear and apparent.

“casamobile” refers to the backend of the project, while “smartmdina” refers to the frontend.

Scope:

The smart city web application is made around modules, central categories which serve as the skeleton of the program.

The application includes the following modules:

- **Administration:** Handles user and content management (user deletion, job listings).
- **Tourism:** Displays landmarks, accommodations, and cultural sites.
- **Student:** Offers educational resources and deals for students.
- **Job Applicant:** Provides job listings, application tracking, and recruitment tools.
- **Business:** Shares business news and opportunities.

Target Audience:

The application is designed for regular users such as tourists, students, and job seekers, it is targeted towards inhabitants of Casablanca, but the modularity of the web application allows it to be versatile and flexible, it can be used for any other city with no issue.

The web application's user base can be split into two:

- **Regular Users:** Tourists, students, and job seekers.
- **Admins:** Responsible for managing users, content, and services.

Administrators could be contributors, government workers or contractors, the administration module is concise, efficient and easy to understand as will be highlighted below.

Development Methodology:

Software Development Methodology is the process by which an application is designed, developed and tested, development methodologies follow specific philosophies and standardize the design and development process, while simultaneously simplifying tasks across the SDLC (*GeeksforGeeks, 2024*).

Approach:

An Agile methodology was followed, Agile is a SDLC methodology that relies on separating big tasks into small increments instead of a big launch allowing for iterative development and integration of feedback during the development lifecycle (*Project Management Institute, 2022*). Due to current experience level in dealing with technologies, the modularity of the web application's design as well as the project structure, Agile was chosen as the development methodology. Tasks were tracked through a progress logging docx file to identify completed and pending functionalities systematically.

Login & regis done, need to separate admin from users, but not now.

- Program the card posting & fetching logic.
- Create student & tourism pages
- Create admin module page.
- Admin role dropdown
- Connect business to backend.
- Separate admin from user.
- **I unified admin & contributor for simplicity's sake.**
- **Admin can now remove users just fine 😊**
- Program logic for admin to create users.
- Thoroughly test the site and refine **css**.
- TADA you have a functional site that you can upload, but it's not over yet.
- Integrate Google Maps API to the website.
- Integrate other APIs to make the website more varied.
- Make the code more efficient and reliable.
- Remove log popups.

Fig. 1: A snippet from 'Progress Logging' docx/Word file.

Technologies Used:

- **Backend:** Java Spring Boot for RESTful API development.

My previous experience with Java made Spring Boot an easy choice for casamobile/backend programming language, it is easy to grasp why it is popular worldwide (*Vermeer, 2020*), the backend is easy to temper with, modify, and my project file structure ensured modularity by separating entities, repositories, controllers etc... and grouping them into different files.

- **Frontend:** React.js for dynamic and responsive user interfaces.

Similar to Spring Boot, my experience with HTML/CSS and JavaScript made React a better candidate than other languages because of its simplicity and modularity, by creating separate components, the programmer is free to re-use them as he sees fit and this is a very valuable tool, I relied on Vite due to its hot reload function/hot module replacement (HMR) (*Vite, n.d.*) which allows for quicker development.

React, much like Spring Boot, is very popular (*Stack Overflow, n.d.*), thus it has more documentation and forums on errors.

- **Database:** MySQL with a relational schema, wampserver.
- **Security:** JWT-based authentication for role management.

Both technologies are relatively simple to use and concise in their documentation.

Overall, my development stack focuses on leveraging documentation, speed, and efficiency while reflecting smartcity/casamobile's modularity and minimalism.

Development Plan:

Casamobile and smartmdina were developed over a period of two week. The web application required thorough planning and depended heavily on adherence to deadlines to finish each task organized as an increment. The plan below outlines the design, implementation, and testing phases, emphasizing scalability, maintainability, and performance.

Design Phase:

UML Diagrams

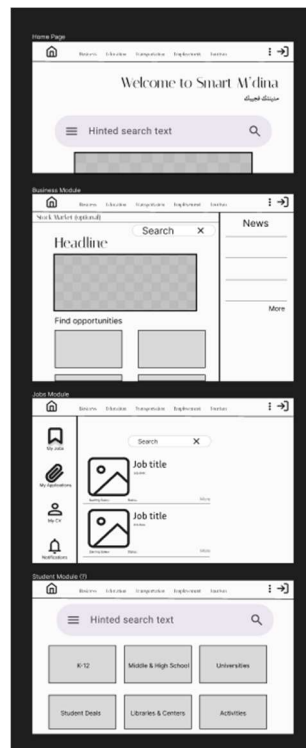
- **Class Diagrams:** Designed to represent entities like User, Job, Tourism, and News. These diagrams illustrate attributes, methods, and relationships.

- **Use Case Diagrams:** Defined user interactions, such as admins managing users and content, and users interacting with city services and jobs.
- **Entity-Relationship Diagrams (ERD):** Depicted normalized database schemas linking users to roles, jobs, applications, and services.

Wireframes

The wireframing was just a guiding principle, straight to the point, outlining where to put items and giving inspiration to how the application will look like. The design was conducted in Figma, with the Project title of “Vision Blueprint”.

- **Home Page:** Wireframes included a dynamic header, trending services section, login/register popups, and navigation links to all major modules.
- **Admin Dashboard:** Designed for user and content management, including lists and forms for creating, deleting, and updating entries.
- **Modules:** Each module (Tourism, Job, Student, Business) included search and filtering capabilities with visually structured layouts for listings and details.



Web App Architecture:

The initial web application architecture revolved around three types of users:

- Regular user that visits the site
- Privileged user that acts as different roles in relevant modules (journalist in news module, investor in business module, guide in tourism module)
- Admin user that assigns privileges, manages content and users.

MSCW Analysis & Basic Database schema:

During the first days of the design process, a MSCW analysis was performed to determine the prioritization of functionalities, coupled with a basic database schema:

Index (Home/Landing Page)

- Central hub of the website.

MUST HAVE:

- A header linking to all major modules (Admin, Tourism, Student, Job, Business).
- A well-designed and visually appealing UI.
- Dynamic display of trending news and services.
- Login and registration forms.

COULD HAVE:

- A global search bar indexing the entire website, with filtering options by category (e.g., jobs, news, services).
- A chatbot assistant for navigation and FAQs.

Admin Dashboard

MUST HAVE:

- Manage authentication and deletion of regular users.
- Create and remove privileged users.
- Approve and reject contributor operations.

- View and respond to user reports.

COULD HAVE:

- Manage chatbot response scripts.
- Create additional admin accounts.
- Require two-factor authentication (2FA) for added security.

Tourism Module

MUST HAVE:

- Categorized display of services (e.g., landmarks, restaurants, hotels, ATMs, public transport).
- A search bar for filtering services.
- Rating and review system.
- Overview of each service with a short description and current ratings.

COULD HAVE:

- A feature to request and message guides directly.
- Map integration for locating tourist attractions.

Student Module

MUST HAVE:

- Categorized display of services related to students (e.g., libraries, coaching centres).
- Search functionality.
- Rating system for services.
- Overview of services with short descriptions and ratings.

COULD HAVE:

- Application forms for schools and student deals.
- Map integration for locating student facilities.

Job Module

MUST HAVE:

- Search bar for filtering job listings by field, pay, or location.
- Application mechanism with notifications for acceptance or rejection.
- Bookmarking functionality for applicants.

COULD HAVE:

- Map integration for job locations.

Business Module

MUST HAVE:

- Dynamic trending page for news and business opportunities.
- Rating system for businesses.
- Posting mechanism for news articles (admin and privileged users only).
- Search bar for finding business-related content.

COULD HAVE:

- Map overview of business locations.
- Stock market or financial updates integration.

DATABASE:

This is a basic database pseudo-schema.

- **Regular User Table:** username, email, type, password.
- **Admin Table:** username, email, type, password.
- **Priv. User Table:** username, email, type, password.
- **Jobs Table:** title, data, description, pay, tags, status.
- **Applications Table:** username, email, phone number, (cv pdf file), status.
- **News Table:** title, image, body, author, date, tags, status.
- **Tourism Table:** title, image, type, body, address, tags, rating.

- **Educational Institutions:** title, image, type, body, address, tags, rating.
- **Deals Table:** title, image, link, rating.
- **City Maps:** overview maps, general map (for home page).

This design required the creation of a separate Roles table in the database schema, as well as different wireframing and backend structure, the idea was scrapped mid-way through due to inefficiency and maintenance problems, it was decided to have a standard admin/user layout.

The overhaul of the architecture was thorough and started from the backend, below is the database schema in MySQL:

```
1 CREATE TABLE CityMaps (
2   map_id INT AUTO_INCREMENT PRIMARY KEY,
3   map_name VARCHAR(100) NOT NULL,
4   map_data LONGBLOB NOT NULL, -- To store map files or images
5   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
6   updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
7 );
```

```
1 CREATE TABLE EducationalServices (
2   service_id INT AUTO_INCREMENT PRIMARY KEY,
3   title VARCHAR(100) NOT NULL,
4   type ENUM('library', 'coaching center', 'school') NOT NULL,
5   image VARCHAR(255) DEFAULT NULL,
6   body TEXT NOT NULL,
7   address VARCHAR(255) NOT NULL,
8   tags VARCHAR(255) DEFAULT NULL,
9   rating DECIMAL(3, 2) DEFAULT NULL,
10  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
11  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
12 );
```

```
1 CREATE TABLE Tourism (
2   service_id INT AUTO_INCREMENT PRIMARY KEY,
3   title VARCHAR(100) NOT NULL,
4   type ENUM('landmark', 'restaurant', 'hotel', 'atm', 'public transport') NOT NULL,
5   image VARCHAR(255) DEFAULT NULL,
6   body TEXT NOT NULL,
7   address VARCHAR(255) NOT NULL,
8   tags VARCHAR(255) DEFAULT NULL,
9   rating DECIMAL(3, 2) DEFAULT NULL,
10  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
11  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
12 );
```

```
1 CREATE TABLE News (
2   news_id INT AUTO_INCREMENT PRIMARY KEY,
3   title VARCHAR(200) NOT NULL,
4   image VARCHAR(255) DEFAULT NULL,
5   body TEXT NOT NULL,
6   author_id INT NOT NULL,
7   date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
8   tags VARCHAR(255) DEFAULT NULL,
9   status ENUM('pending', 'approved', 'rejected') DEFAULT 'pending',
10  FOREIGN KEY (author_id) REFERENCES Users(user_id) ON DELETE SET NULL
11 );
```

```

1 CREATE TABLE Applications (
2   application_id INT AUTO_INCREMENT PRIMARY KEY,
3   user_id INT NOT NULL,
4   job_id INT NOT NULL,
5   phone_number VARCHAR(15) NOT NULL,
6   cv LONGBLOB NOT NULL,
7   status ENUM('pending', 'accepted', 'rejected') DEFAULT 'pending',
8   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
9   FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE,
10  FOREIGN KEY (job_id) REFERENCES Jobs(job_id) ON DELETE CASCADE
11 );

```

```

1 CREATE TABLE Jobs (
2   job_id INT AUTO_INCREMENT PRIMARY KEY,
3   title VARCHAR(100) NOT NULL,
4   description TEXT NOT NULL,
5   pay DECIMAL(10, 2) DEFAULT NULL,
6   tags VARCHAR(255) DEFAULT NULL,
7   status ENUM('active', 'inactive', 'closed') DEFAULT 'active',
8   created_by INT NOT NULL,
9   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
10  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
11  FOREIGN KEY (created_by) REFERENCES Users(user_id) ON DELETE CASCADE
12 );

```

```

1 CREATE TABLE Users (
2   user_id INT AUTO_INCREMENT PRIMARY KEY,
3   username VARCHAR(50) NOT NULL UNIQUE,
4   email VARCHAR(100) NOT NULL UNIQUE,
5   password VARCHAR(255) NOT NULL,
6   role_id INT NOT NULL,
7   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
8   updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
9   FOREIGN KEY (role_id) REFERENCES Roles(role_id) ON DELETE CASCADE
10 );

```

The backend was based on a client-server model with a RESTful API for communication between React.js frontend and Spring Boot backend.

- **Security:** Integrated JWT authentication for secure session management and access control.

Development Timeline:

Week 1: Design and Initial Development:

1. Day 1-2:

- Set up the project repositories and environments.
- Designed UML diagrams and wireframes.
- Established database schema in MySQL.

2. Day 3-5:

- Developed basic backend functionality for user authentication and management.
- Created initial React components for the home page and admin dashboard.
- Implemented basic API endpoints for user registration, login, and data retrieval.

3. Day 6-7:

- Built skeletons for Tourism, Job, Student, and Business modules.
- Connected the frontend to backend APIs using Axios.
- Integrated JWT authentication for secure access control.

Week 2: Refinement and Testing:

1. Day 8-10:

- Completed CRUD functionality for modules (e.g., posting and deleting services, jobs, and news).
- Implemented frontend search and filter components for Tourism and Job modules.
- Enhanced UI responsiveness and design consistency.

2. Day 11-12:

- Conducted unit testing for backend APIs using JUnit.
- Performed integration testing with Postman to ensure API reliability.
- Fixed bugs related to routing and authentication.

3. Day 13-14:

- Added role-based access for admin and user functionalities.
- Performed end-to-end testing using React Testing Library and Cypress.

- Finalized project documentation, diagrams, and deliverables.

Frontend Development Plan:

Framework and Tools

- **React.js:** Used to build dynamic and responsive user interfaces.
- **Axios:** For making asynchronous API requests to the backend.
- **Styling:** CSS and styled components were employed for consistent, responsive design.

Pages and Features

1. **Home Page:**
 - Displays trending news and services fetched via API.
 - Includes login/register popups and navigation links to other modules.
2. **Admin Dashboard:**
 - Allows admins to manage users (create, delete) and content (post, edit, delete).
 - Provides an interface for responding to user reports.
3. **Tourism Module:**
 - Categorized service display with filtering and search functionalities.
 - Integration of a rating and review system.
4. **Job Module:**
 - Searchable job listings categorized by field, pay, or location.
 - Features for bookmarking jobs and submitting applications.
5. **Student and Business Modules:**
 - Displays resources for students and trending business opportunities.

Backend Development Plan:

Framework and Tools

- **Spring Boot:** Used for building RESTful APIs.
- **Spring Security:** Handles JWT-based authentication and role-based access.

API Endpoints

1. User Management:

- POST /api/users/register: Register users (role defaults to "user").
- POST /api/users/login: Authenticate users and generate JWT tokens.
- GET /api/users/allusers: Retrieve all users (admin access only).
- DELETE /api/users/{id}: Remove users by ID.

2. Tourism Module:

- GET /api/tourism: Fetch all tourism services.
- POST /api/tourism: Add a new service (admin access only).
- PUT /api/tourism/{id}: Update an existing service.
- DELETE /api/tourism/{id}: Remove a service.

3. Job Module:

- GET /api/jobs: Fetch job listings.
- POST /api/jobs: Post a new job (admin access only).
- DELETE /api/jobs/{id}: Remove a job listing.

4. News Module:

- GET /api/news: Fetch news articles.
- POST /api/news: Add new articles (admin access only).

Security

- JWT-based authentication ensures secure user sessions.
- Protected endpoints restrict unauthorized access.

Database Design Plan:

Tables

1. Users Table:

- Fields: user_id, username, email, password, role, created_at, updated_at.

2. Jobs Table:

- Fields: job_id, title, description, pay, status, created_by.

3. Tourism Table:

- Fields: service_id, title, type, address, tags, rating.

4. **Applications Table:**

- Fields: application_id, user_id, job_id, phone_number, cv, status.

5. **News Table:**

- Fields: news_id, title, body, author_id, tags.

Testing and Validation:

Methods

- **Unit Testing:** Validated backend APIs using JUnit.
- **Integration Testing:** Tested API reliability and response times with Postman.
- **Frontend Testing:** Used React Testing Library and Cypress for end-to-end testing.

Bug Fixes

- Fixed routing issues in the frontend.
- Resolved inconsistencies in role-based access control.
- Addressed errors in form validation for user registration and login.
- Fixed CSS styling issues and inconsistencies.
- Wrapped main.tsx in User Context to maintain authentication.

This comprehensive plan details the design, development, and testing efforts for the Smart City web application, ensuring its readiness for deployment and scalability for future enhancements.

System Architecture:

Overview:

The system adopts a client-server architecture:

- **Client:** React.js frontend for user interaction.
- **Server:** Spring Boot backend for business logic.
- **Database:** Relational database for persistent storage.

User Roles:

- **Admin:**
 - Manages users, content, and listings.
 - Posts and deletes services and job listings.
- **User:**
 - Views city services.
 - Applies for job listings.
 - Rates and comments on services. (Not required).

Module Interactions:

- **Admin Module:** Manages backend data.
- **Tourism Module:** Displays and categorizes city attractions.
- **Job Module:** Lists jobs and manages applications.
- **Business Module:** Shares business-related news and trends.

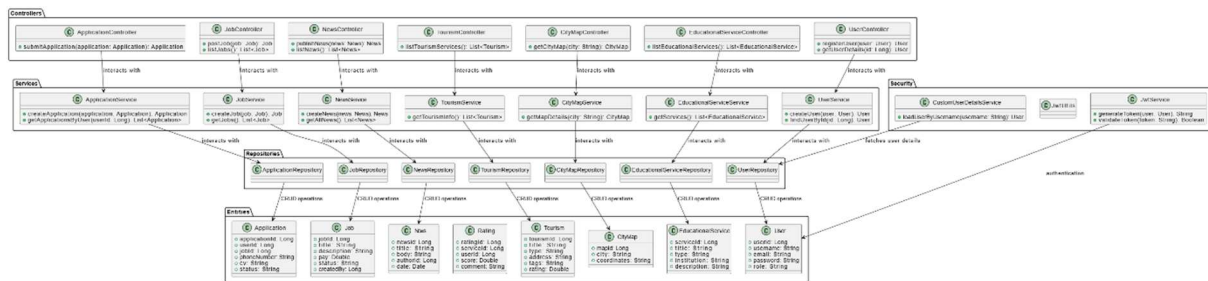
Developed UML Diagrams:

Class Diagram:

Definition:

A class diagram in the Unified Modeling Language (UML) is a static structure diagram that outlines the structure of a system by highlighting its classes, their attributes, operations (or methods), and the relationships among objects (*Visual Paradigm, n.d.*).

(Check attached files).



The main classes include:

- **User:** Attributes like userId, username, email, password, and role.
- **Tourism:** Attributes like serviceId, title, type, and rating.
- **News:** Attributes like newsId, title, body, and author.
- **Job:** Attributes like jobId, title, description, and pay.
- **Application:** Attributes like applicationId, userId, jobId, and status.

1. User:

- Attributes:

- 'userId': A unique identifier for the user.
- 'username': The chosen username for login and identification.
- 'email': The email address used for communication.
- 'password': A hashed and secured password for authentication.

- `role`: The role assigned to the user, such as admin or regular user.

2. Application:

- Attributes:
 - `applicationId`: The unique ID for a job application.
 - `userId`: Links the application to a specific user.
 - `jobId`: Links the application to a specific job posting.
 - `phoneNumber`: Contact number provided by the applicant.
 - `cv`: The curriculum vitae (resume) uploaded by the applicant.
 - `status`: Indicates the state of the application (e.g., pending, approved).

3. Job:

- Attributes:
 - `jobId`: A unique identifier for the job posting.
 - `title`: The title of the job.
 - `description`: Details about the job responsibilities and requirements.
 - `pay`: The remuneration for the job.
 - `status`: Indicates whether the job is open or closed.
 - `createdBy`: The user ID of the individual who created the job posting.

4. News:

- Attributes:
 - `newsId`: The unique identifier for a news item.

- `title`: The headline or title of the news article.
- `body`: The content of the article.
- `authorId`: The ID of the user who authored the news.
- `date`: The publication date of the news.

5. Rating:

- Attributes:
- `ratingId`: The unique ID for a rating.
- `serviceId`: Links the rating to a specific service.
- `userId`: The ID of the user who provided the rating.
- `score`: The numerical score of the rating.
- `comment`: Additional feedback provided by the user.

6. Tourism:

- Attributes:
- `tourismId`: A unique identifier for a tourism service.
- `title`: The name of the tourism spot or service.
- `type`: The category of tourism (e.g., historical, natural).
- `address`: The physical location of the tourism service.
- `tags`: Relevant tags to describe the service.
- `rating`: The overall rating of the tourism service.

7. CityMap:

- Attributes:

- `mapId`: A unique identifier for a city map.
- `city`: The name of the city.
- `coordinates`: Geographical coordinates for mapping.

8. EducationalService:

- Attributes:

- `serviceId`: A unique ID for the educational service.
- `title`: The name of the educational service.
- `type`: The category of service (e.g., school, course).
- `institution`: The institution providing the service.
- `description`: A brief overview of the service.

Services

The Service layer contains the business logic, interacting with repositories to manage data and providing functionality for controllers.

1. UserService:

- Functions: Create users and retrieve user details by ID.

2. ApplicationService:

- Functions: Handle the creation of applications and fetch applications for specific users.

3. JobService:

- Functions: Enable the creation of job postings and fetch a list of all jobs.

4. NewsService:

- Functions: Publish news articles and retrieve all news.

5. TourismService:

- Functions: Provide information about available tourism services.

6. CityMapService:

- Functions: Fetch detailed city maps based on the city name.

7. EducationalServiceService:

- Functions: Retrieve a list of all educational services.

Repositories

Repositories act as the data access layer, interacting directly with the database to perform CRUD (Create, Read, Update, Delete) operations.

- Each entity has a corresponding repository:
 - UserRepository: Manages user data.
 - ApplicationRepository: Handles job application records.
 - JobRepository: Operates on job postings.
 - NewsRepository: Stores and retrieves news articles.
 - TourismRepository: Manages tourism-related data.
 - CityMapRepository: Stores city map details.
 - EducationalServiceRepository: Handles data for educational services.

Controllers

Controllers are the entry point for external interactions, such as API requests. They invoke services to fulfill requests.

1. UserController:

- Functions: Register new users and fetch user details.

2. ApplicationController:

- Functions: Submit job applications.

3. JobController:

- Functions: Post new jobs and list existing job postings.

4. NewsController:

- Functions: Publish news and retrieve all news articles.

5. TourismController:

- Functions: Provide a list of tourism services.

6. CityMapController:

- Functions: Retrieve city maps.

7. EducationalServiceController:

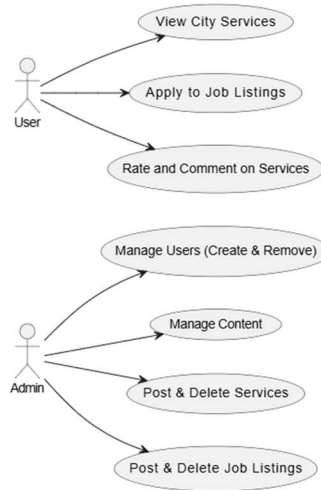
- Functions: Provide a list of educational services.

Use Case Diagram:

Definition:

In Unified Modeling Language (UML), a use case diagram is a graphic depiction of how users (actors) interact with a system. It demonstrates how diverse users interact with distinct use cases, or functionalities, within a system, capturing its functional needs (*GeeksforGeeks, 2025*).

- Admin manages users and content.
- Users interact with services, jobs, and ratings.



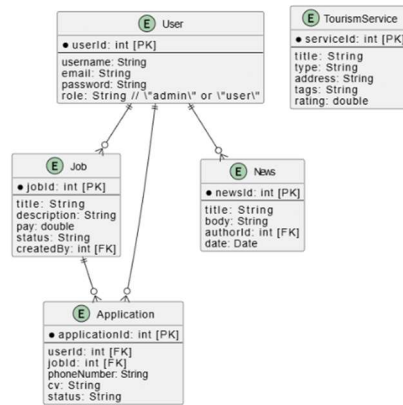
Entity-Relationship Diagram:

Definition:

Lucidchart (n.d.) defines ER diagrams as: “a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research.”

The Entity-Relationship Diagram (ERD) outlines a robust and efficient database structure, emphasizing clear entities and relationships. The primary entity, User, includes attributes such as username, email, password, and role (distinguishing between "admin" and "user" roles). This entity connects to three key modules: Job, Application, and News, establishing one-to-many relationship with User through the createdBy foreign key.

The role-based access system in User and the use of foreign key constraints maintain integrity, this ERP acts as a representative for the rest of the backend interactions as they use the same logic.



Relationships include:

- Users and roles consolidated into the role field.
- Jobs and applications connected by `jobId` and `userId`.
- Tourism and news tables linked indirectly to users for ratings or contributions.

Implementation Details:

Backend Logic:

User Management

- `@PostMapping("/register")`
 - `public User createUser(@RequestBody User user) {`
 - `String role = "user";`
 - `if (user.getRole() != null && user.getRole().equals("admin")) {`
 - `role = "admin";`
 - `}`
 - `return userService.createUser(user, role);`
 - `}`
- Admins can create users and assign roles.
 - Authentication uses JwtUtils to generate secure tokens.

Tourism/Educational Services Management

- `@PostMapping`
 - `public Tourism createTourism(@RequestBody Tourism tourism) {`
 - `return tourismService.createTourism(tourism);`
 - `}`
- Tourism entries can be created, updated, and deleted via RESTful APIs.

News Management

- `@PutMapping("/{id}")`
 - `public News updateNews(@PathVariable Long id, @RequestBody News updatedNews) {`
 - `return newsService.updateNews(id, updatedNews);`
 - `}`
- News articles are posted and managed with admin approval.

Frontend Logic:

- **Home Page:** Displays trending news and services.
- **Tourism Module:** Integrated with backend APIs for dynamic data rendering.
- **Job Module:** Features job filtering and application submission.
- **Components:** Navbar header for logged out users, UserNavbar for logged in users, AdminNavbar for admins. Popup forms for admin, applications, and authentication services, UserContext to keep checking who's logged in and fetch userId and other relevant information to send in case it's needed (for example, when submitting a news article).

Screenshots:

```
import React, { useState, useEffect } from 'react';
import { useParams } from 'react-router-dom'; // Hook to get the jobId from the URL
import './css/JobModule.css';
import ApplyToJob from '../components/ApplyToJob';
import UserNavbar from '../components/users/UserNavbar';
import SideNav from '../components/SideNav';

Codeium: Refactor | Explain | Generate JSDoc | X
const JobDetails: React.FC = () => {
  const { jobId } = useParams<{ jobId: string }>(); // Get jobId from the URL parameters
  const [job, setJob] = useState<{
    title: string;
    description: string;
    pay: string;
    status: string;
  } | null>(null); // State to store job details
  const [isApplyOpen, setApplyOpen] = useState(false);
  const [loading, setLoading] = useState(true); // Loading state to handle asynchronous data fetch
```

```
useEffect(() => {
  // Fetch job details from the backend when the component is mounted
  Codeium: Refactor | Explain | X
  const fetchJobDetails = async () => {
    try {
      const response = await fetch(`http://localhost:8080/api/jobs/${jobId}`);
      if (response.ok) {
        const data = await response.json();
        setJob(data); // Set job data in the state
      } else {
        console.error('Job not found');
        setJob(null); // Set job to null if not found
      }
    } catch (error) {
      console.error('Error fetching job details:', error);
      setJob(null);
    } finally {
      setLoading(false); // Set loading to false once the request completes
    }
  };
});
```

```
const SideNav: React.FC = () => {
  return (
    <div className="side-nav">
      <Link to="/user/employment/myjobs" className="side-nav-item">
        <img src={jobIcon} alt="My Jobs" />
        <span>My Jobs</span>
      </Link>

      <Link to="/user/employment/myapplications" className="side-nav-item">
        <img src={applicationIcon} alt="My Applications" />
        <span>My Applications</span>
      </Link>

      <Link to="/user/employment/mycv" className="side-nav-item">
        <img src={cvIcon} alt="My CV" />
        <span>My CV</span>
      </Link>
    </div>
  )
}
```

```
const StudentModule: React.FC = () => {
  Codeium: Refactor | Explain | Generate JSDoc | X

  const categories = [
    {
      title: 'Schools & Universities',
      description: 'Explore educational institutions for your studies.',
      imageUrl: 'https://www.jenkinspeer.com/wp-content/uploads/2014/09/ed-uncw-SOE-7.jpg',
    },
    {
      title: 'Libraries',
      description: 'Find libraries and coaching centers to help with your learning.',
      imageUrl: 'https://external-preview.redd.it/_vDnByo3viBiLFZlKSQFodH_qWD6UUPvwIJJe_JUKD0.jpg?a',
    },
    {
      title: 'Student Deals',
      description: 'Check out the latest student discounts and offers.',
      imageUrl: 'https://i.pinimg.com/originals/c2/dc/cf/c2dccf244a33981319d2d9906b1202d0.jpg',
    },
  ],
}
```

Database Design:

- **Users Table:**
 - Fields: userId, username, email, password, role, created_at, updated_at.
- **Tourism Table:**
 - Fields: serviceId, title, type, body, address, rating, tags.
- **Jobs Table:**
 - Fields: jobId, title, description, pay, status, created_by.

Screenshots:













application_id created_at job_id user_id cv phone_number status

rating created_at service_id updated_at address image tags title type body description average_rating

pay created_at job_id updated_at description status tags title

author_id date news_id image status tags title body description

rating created_at service_id updated_at address image tags title type body description

				created_at	updated_at	user_id	email	password	username	role
<input type="checkbox"/>				2025-01-21 00:36:39.407685	2025-01-21 00:36:40.719402	18	hi@hi.com	hi	hi	admin
<input type="checkbox"/>				2025-01-23 10:24:15.963011	2025-01-23 10:24:15.963011	21	example@example.com	example	example	user
<input type="checkbox"/>				2025-01-23 10:57:01.715117	2025-01-23 10:57:01.715117	22	example1@gmail.com	example1	example1	user
<input type="checkbox"/>				2025-01-23 11:01:22.604577	2025-01-23 11:01:24.073554	23	hi2@gmail.com	hi	hi2	user

Testing and Validation:

Testing Methodology:

- **Unit Testing:** Focused on backend APIs for user management and content posting.
- **Integration Testing:** Validated frontend-backend communication using Postman.

Test Cases:

User Authentication

- **Input:** Username and password.
- **Expected Output:** JWT token for valid credentials, error message for invalid inputs.
- **Output:** Error due to database structure.

Posting Tourism Data

- **Input:** Valid JSON object for a new tourism entry.
- **Expected Output:** 201 Created response with saved object.
- **Output:** 500 internal server error, api fetch link in frontend was “/api/tourism” instead of <http://localhost:8080/api/tourism>.

Bug Fixes:

- Fixed an issue where admin-created users had incorrect default roles.
- Fixed routing issues.
- Corrected routing errors on the frontend for job module navigation.
- Corrected api link errors by pasting the entire api link.
- Fixed an issue where news article can't be submitted due to authorId field not being filled (by using User Context to automatically send userId based on the currently logged user).
- Fixed issue where Admin ContTable component didn't accept database information.
- Corrected code error that caused the api to fetch from the frontend instead of the backend, resulting in multiple “unexpected data” problems.

- Corrected code error where the onLoginSuccess expected a certain argument in App.tsx but didn't in Navbar.tsx.
- Removed contributor user due to errors and maintenance issues to save time.
- Removed logged out navigation due to errors and maintenance issues
- Etc...

Conclusion:

Summary:

The Smart City application achieves its goal of streamlining access to city services and resources. It provides a modular structure for user management, service listings, and job applications, this report thoroughly outlines its short-lived development life cycle and its potential to grow into something more functional and acceptable. The design has been carefully curated and many microprojects were considered inside this project (Vision Blueprint, casamobile, smartmdina, contributor operations etc...). Still, there are many future enhancements to consider.

Reflection and Future Prospects:

The project reinforced the importance of scalable design and robust testing. By leveraging a modular architecture, the system ensures maintainability and extensibility, with more time this project can be scaled up into a market-ready product. It would require an overhaul of the frontend by re-organizing module pages and components and re-using them more efficiently, a global CORS configuration, securing database interactions, switching to MongoDB, utilizing the principle of microservices by dividing each module into its separate department project, conducting stress and penetration tests, adding 2FA, google earth APIs, a proper chatbot and a sleeker design by changing the CSS.

References:

1. GeeksforGeeks (2024) What are software development methodologies: 15 key methodologies, GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/what-is-software-development-methodology-15-key-methodologies/#what-is-software-development-methodology-> (Accessed: 21 January 2025).
2. GeeksforGeeks (2025) Use case diagram - unified modeling language (UML), GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/use-case-diagram/> (Accessed: 23 January 2025).
3. Lucidchart (no date) What is an entity relationship diagram (ERD)?, Lucidchart. Available at: <https://www.lucidchart.com/pages/er-diagrams> (Accessed: 23 January 2025).
4. Project Management Institute (2022) What is agile?, Project Management Institute. Available at: <https://www.pmi.org/disciplined-agile/agile/whatisagile> (Accessed: 21 January 2025).
5. Stack Overflow (no date) Tag Trends, Stack overflow. Available at: <https://trends.stackoverflow.co/?tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs%2Cvuejs3> (Accessed: 21 January 2025).
6. Vermeer, B. (2020) Spring dominates the Java ecosystem with 60% using it for their main applications, Snyk. Available at: <https://snyk.io/blog/spring-dominates-the-java-ecosystem-with-60-using-it-for-their-main-applications/> (Accessed: 21 January 2025).
7. Visual Paradigm (no date) What is a class diagram?, What is class diagram? Available at: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/> (Accessed: 23 January 2025).
8. Vite (no date) Why vite, vitejs. Available at: <https://vite.dev/guide/why.html> (Accessed: 21 January 2025).