# User

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| name\* | String, min 3 max 30 |
| email\* | String, unique mail |
| password\* | String, min 3 max 20 |
| Role\* | Enum [“Admin”, “User”] |
| Avatar | String, Optional |
| Age | Number, Optional |
| Phone Number | Number, Mobile Phone, Length 11(Just EG Numbers), Optional |
| Address | String, Optional |
| Active | Boolean, Enum [False, True] |
| Verification Code | String |
| gender | String, Enum [“male”, “female”] |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(create/ update/ delete/ Get All/ Get One)**

### Business Operation: User **(delete/ Get One)**

# Category

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| name\* | String, min 3 max 30 |
| image | String, URL |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(create/ update/ delete/ Get All/ Get One)**

### Business Operation: User **(Get)**

# Sub Category

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| name\* | String, min 3 max 20 |
| category‎\* | Object ID **Ref**: “Categories” |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(create/ update/ delete/ Get All/ Get One)**

### Business Operation: User **(Get)**

# Review

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| reviewText‎ | String, min 3 |
| Rating\* | Number, Min 1, Max 5 |
| user‎\* | Object ID **Ref**: “User” |
| Product\* | Object ID **Ref**: “Product” |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(Get All by single product/ Get Reviews on Single User)**

### Business Operation: User **(create/ update/ delete/ Get All)**

# Brand

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| name‎\* | String, Min 3, Max 100 |
| image | String, URL |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(create/ update/ delete/ Get All/ Get One)**

### Business Operation: User **(Get)**

# Cart

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| Cart Items‎\* | List Of Product (Object ID Ref:”Product”),[{productId, quantity:1, color:””}] |
| Total Price | Number |
| totalPriceAfterDiscount | Number |
| Coupons | List Of Coupons (Object ID Ref:” Coupon”), [{name:couponName,couponId}] |
| User | Object ID Ref:”User” |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(Get All/ Get One)**

### Business Operation: User **(create/ update/delete/ Get One /** Coupons**)**

# Coupon

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| name‎\* | String |
| Expirdate\* | Date |
| Discount\* | Number, Min 0 |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(create/ update/ delete/ Get All/ Get One)**

# Product

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| title\* | String, min 3 |
| description‎\* | String, min 20 |
| quantity‎\* | Number, min 1, max 500, default‎: 1 |
| Image Cover\* | String, URL |
| Images | List Of String, URL |
| sold | Number, default‎: 0 |
| Price\* | Number, Min1, Max 20000 |
| Price After Discount | Number, Max 20000, default‎: 0 |
| color | List Of String |
| Category\* | Object ID **Ref**: “Categories” |
| Sub Category | Object ID **Ref**: “Sub Categories” |
| brand‎ | Object ID **Ref**: “Brand” |
| Ratings Average | Number, default: 0 |
| Ratings Quantity | Number, default: 0 |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(create/ update/ delete/ Get All/ Get One)**

### Business Operation: User **(Get)**

# Request Product

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| titleNeed‎\* | String |
| details\* | String, Min 5 |
| qauntity\* | Number, Min 1 |
| Category | String |
| User | Object ID Ref:”User” |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(Get All/ Get One)**

### Business Operation: User **(create/ update/ delete/ Get One)**

# 10- Suppliers

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| Name‎\* | String |
| website\* | String, URL |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(create/ update/ delete/ Get All/ Get One)**

### Business Operation: User **(Get)**

# 11- Tax

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| texPrice‎‎ | Number |
| shippingPrice | Number |
| timestamps\* | Date, auto-increment |

## Business Operations

### Business Operation: For Admin **(create/ update/ delete/ Get)**

# 12- Order

## attributes

|  |  |
| --- | --- |
| **Attribute name** | **Notes** |
| \_Id \* | Number, read only, auto-increment |
| User\*‎‎ | Object ID Ref:”User” |
| cartItems\* | List Of Product (Object ID Ref:”Product”), (\_Id, quantity: 1, color, price) |
| texPrice | Number, default: 0 |
| shippingPrice | Number, default:0 |
| totalOrderPrice\* | Number |
| paymentMethodType\* | String, Enum:[“cash”, “card”] |
| isPaid | Boolean, default: false |
| paidAt | Date |
| isDeliverd | Boolean, default: false |
| deliverdAt | Date |
| shippingAddress\* | {alias: String, details: String, phone: String, city: String, postalCode: String, } |
| timestamps\* | Date, auto-increment |

## Business Operations

### **Business Operation: For Admin (Get All/ Get One/ update Paid Cash)**

### **Business Operation: User (create/ Get)**

### **Business Operation: Stripe (update Paid Card)**

**Project Plan for LinkedIn Clone using MERN Stack**

**1. Objectives**

* Create a professional networking platform with functionalities similar to LinkedIn.
* Implement advanced MERN stack features: microservices, GraphQL, and Socket.IO.
* Showcase technical expertise to enhance CV value.

**2. Core Features**

**User Management**

* Registration, Login, Logout
* Profile creation and customization
* Role-based access (Admin, User)
* User authentication using JWT
* Password reset and email verification

**Networking**

* Add/remove connections
* Endorsements and recommendations
* Messaging (real-time with Socket.IO)

**Content Sharing**

* Posts with text, images, and videos
* Comments, likes, and shares
* Content recommendations

**Job Module**

* Job postings by companies
* Job search with filters
* Application tracking

**Notifications**

* Real-time and batch notifications
* Activity updates (e.g., connection requests, job invites)

**Search**

* Global search across users, posts, jobs, etc.
* Advanced filters (skills, location, industry)

**3. Technical Requirements**

**Frontend**

* React.js for SPA
* Redux or Context API for state management
* Tailwind CSS or Material UI for UI components

**Backend**

* Node.js with Express for APIs
* GraphQL for flexible queries
* MongoDB for database management

**Real-time Features**

* Socket.IO for real-time messaging
* Notification updates

**Deployment**

* Docker containers for microservices
* Kubernetes for orchestration
* CI/CD pipelines with GitHub Actions or Jenkins

**4. Database Design**

**User Schema**

* Fields: name, email, password, role, avatar, skills, experience, education, connections, etc.

**Post Schema**

* Fields: title, content, images, author, likes, comments, shares, timestamps

**Job Schema**

* Fields: title, company, location, description, requirements, salary, applications, timestamps

**Message Schema**

* Fields: sender, receiver, content, status, timestamps

**5. Microservices Architecture**

**Services**

1. **User Service**: Handles user-related functionalities.
2. **Post Service**: Manages content creation and interactions.
3. **Job Service**: Handles job postings and applications.
4. **Notification Service**: Manages alerts and updates.
5. **Messaging Service**: Ensures real-time communication.

**Communication**

* RESTful APIs and GraphQL
* RabbitMQ or Kafka for inter-service communication

**6. Advanced Integrations**

**GraphQL API**

* Enables efficient data fetching for nested resources (e.g., user with posts and connections).

**Authentication and Security**

* OAuth 2.0 for third-party logins (Google, LinkedIn)
* Secure password storage with bcrypt
* Input validation to prevent SQL injection and XSS

**Testing**

* Jest for unit tests
* Postman or Newman for API testing
* Cypress for end-to-end testing

**Performance Optimization**

* Load balancing with NGINX
* Caching with Redis
* CDN for static assets

**7. Development Timeline**

**Phase 1: Planning (2 Weeks)**

* Finalize requirements
* Design database schema
* Define microservices

**Phase 2: Development (8 Weeks)**

1. **Week 1-2**: User management and authentication
2. **Week 3-4**: Post and job modules
3. **Week 5**: Messaging and notifications
4. **Week 6**: Advanced search and filtering
5. **Week 7-8**: Microservices integration

**Phase 3: Testing and Deployment (2 Weeks)**

* Functional and performance testing
* Deploy on cloud (AWS, Azure, or GCP)

**8. Tools and Technologies**

* **Frontend**: React.js, Redux, Tailwind CSS
* **Backend**: Node.js, Express, GraphQL
* **Database**: MongoDB, Redis
* **Deployment**: Docker, Kubernetes, AWS/GCP
* **Testing**: Jest, Cypress, Postman
* **Real-time Communication**: Socket.IO

**9. Deliverables**

* Fully functional LinkedIn clone
* Deployed application with production-ready code
* Documentation for APIs, microservices, and architecture

**Project Plan: LinkedIn-like System**

**Introduction**

The project aims to develop a LinkedIn-like system using the MERN stack. The system will include advanced functionalities such as microservices, GraphQL, and Socket.IO to showcase expertise in modern web development practices.

**Functional Requirements**

**User Management**

* **Registration and Login:** Email/password authentication, Google OAuth.
* **Profile Management:**
  + Update profile information.
  + Upload profile pictures.
  + Add work experience, education, skills, and achievements.
* **Account Settings:**
  + Change password.
  + Manage privacy settings.

**Networking Features**

* **Connections:**
  + Send, accept, and decline connection requests.
  + View mutual connections.
* **Messaging:**
  + Real-time messaging using Socket.IO.
  + Support for file sharing, emojis, and read receipts.

**Content and Interaction**

* **Posts and Articles:**
  + Create, edit, and delete posts.
  + Like, comment, and share posts.
  + Upload multimedia content.
* **News Feed:**
  + Display posts from connections.
  + Personalized feed based on interests and activity.
* **Notifications:**
  + Real-time notifications for new messages, connection requests, and post interactions.

**Search and Filtering**

* **Search Functionality:**
  + Search users, posts, and companies.
  + Advanced filters for skills, location, and experience.

**Job Features**

* **Job Listings:**
  + Employers can post jobs.
  + Users can apply for jobs with their profiles.
  + Bookmark and save jobs.
* **Application Management:**
  + Employers can review applications.
  + Users can track application status.

**Admin Panel**

* Manage users and content.
* Monitor system performance and reports.
* Control advertisements and promotions.

**Technical Requirements**

**Frontend**

* **React.js:** Create reusable components.
* **State Management:** Use Redux or Context API.
* **Responsive Design:** Ensure mobile-first design with CSS frameworks like Tailwind or Material-UI.

**Backend**

* **Node.js and Express.js:** Build REST and GraphQL APIs.
* **Authentication:**
  + JWT for token-based authentication.
  + OAuth2 for third-party login.
* **Microservices Architecture:**
  + Independent services for user management, messaging, jobs, and notifications.
  + API Gateway for request routing.

**Database**

* **MongoDB:**
  + User profiles, posts, and job data.
  + Indexing for optimized search.
* **Redis:** Cache frequently accessed data.

**Real-Time Communication**

* **Socket.IO:** Implement real-time features such as messaging and notifications.

**API Development**

* **GraphQL:**
  + Query and mutation support for flexible data fetching.
  + Subscriptions for real-time updates.

**DevOps and Deployment**

* **Containerization:** Use Docker for containerized services.
* **CI/CD:** Set up pipelines with GitHub Actions.
* **Hosting:** Deploy microservices to AWS or Azure.

**Testing**

* Unit and integration tests using Jest.
* End-to-end tests using Cypress.

**Advanced Features**

**Machine Learning**

* **Recommendations:**
  + Suggest connections based on mutual connections, skills, or location.
  + Recommend jobs based on user profiles.
* **Search Ranking:** Optimize search results using ML models.

**Analytics and Insights**

* **User Analytics:**
  + Track profile views and post engagement.
  + Display insights to users and employers.
* **System Metrics:** Monitor active users, job applications, and post interactions.

**Payment Integration**

* Stripe for premium subscriptions and job promotions.

**Deliverables**

1. Detailed documentation for each feature.
2. Fully functional system with deployment scripts.
3. Unit and integration tests.
4. Admin panel for monitoring and management.
5. Deployment on cloud services.

**Project Phases**

**Phase 1: Planning and Design**

* Requirement gathering.
* Create wireframes and design prototypes.

**Phase 2: Core Development**

* Develop user management and authentication.
* Implement posts, messaging, and connections.

**Phase 3: Advanced Features**

* Integrate microservices and GraphQL.
* Add job-related features and notifications.

**Phase 4: Deployment and Testing**

* Deploy the system to cloud platforms.
* Conduct extensive testing.

**Tools and Technologies**

* **Frontend:** React.js, Redux, Tailwind CSS.
* **Backend:** Node.js, Express.js, GraphQL.
* **Database:** MongoDB, Redis.
* **DevOps:** Docker, GitHub Actions, AWS.
* **Testing:** Jest, Cypress.
* **Real-time Communication:** Socket.IO.

This project plan ensures a comprehensive system that demonstrates expertise in advanced MERN stack concepts and enhances your CV value.

LinkedIn-like System Project Planning

This document outlines the project planning for a LinkedIn-like system built using the MERN stack. The system will include advanced topics like microservices, GraphQL, and Socket.IO.

**System Overview**

Technology Stack: MERN (MongoDB, Express.js, React.js, Node.js)  
Architecture: Microservices-based approach  
Key Features: User profiles, posts, networking, messaging, notifications, job listings, and content recommendations.  
Advanced Features:  
- GraphQL for optimized querying and performance  
- Socket.IO for real-time messaging and notifications

**1. User**

Attributes:  
\_Id: Auto-increment, read-only  
Name: Min 3, Max 30  
Email: Unique  
Password: Min 3, Max 20  
Role: Enum [“Admin”, “User”]  
Avatar: String (Optional)  
Age: Number (Optional)  
Phone Number: String (Optional)  
Address: String (Optional)  
Active: Boolean  
Gender: Enum [“Male”, “Female”]  
Timestamps: Date (auto-generated)  
  
Business Operations:  
Admin: Create, Update, Delete, Get All, Get One  
User: Get One, Delete

**2. Microservices Architecture**

In a microservices approach, each feature is split into its own service, improving scalability and maintainability.  
Services include User Service, Product Service, Review Service, Order Service, and Notification Service.

**3. GraphQL and Real-Time Features**

- \*\*GraphQL\*\* will optimize data fetching, making it more flexible and efficient.  
- \*\*Socket.IO\*\* will handle real-time features like messaging, notifications, and live updates.

**4. Security & Authentication**

Using JWT-based Authentication for secure login.  
Password will be hashed using bcrypt.  
Role-based Access Control will be implemented for different permissions.

**5. Frontend Development**

Frontend will be built using React.js.  
Features include profile management, posts, messaging, notifications, and search functionality.

**6. Testing**

Unit testing, integration testing, and end-to-end testing will be implemented.

**7. Deployment**

Deployment will use Docker containers and Kubernetes for orchestration.  
Cloud-based deployment on AWS, Azure, or Google Cloud.