Components

Backend Services

All three Spring Boot services use a **layered architecture**:

- **Entities & Repositories**: Each data model (e.g. `User`, `Restaurant`, `Order`) is a JPA entity annotated with `@Entity`. A corresponding Spring Data JPA repository (interface) provides CRUD operations. For example, `UserRepository extends JpaRepository<User, Long>`. The layered design ensures separation of concerns: controllers do not interact with the DB directly, but call service-layer classes.

- **Services (Business Logic)**: Business services (e.g. `UserService`, `RestaurantService`, `OrderService`) contain core logic. They handle tasks like validating input, calculating totals, and orchestrating actions. They call repositories to query or update data. MapStruct is used here to map between entities and DTOs (data transfer objects) to avoid repetitive setter code. For instance, an `OrderMapper` would convert an `Order` entity to an `OrderDto`.

- **Controllers (API Endpoints)**: REST controllers (annotated `@RestController`) define HTTP endpoints. Examples: `UserController` has `/register`, `/login`, `/users/{id}`; `RestaurantController` has `/restaurants` endpoints; `OrderController` has `/orders` endpoints. Controllers parse requests (path variables, JSON bodies) into DTOs or parameters and invoke service methods. They return JSON responses.

- **Security (JWT & Roles)**: Spring Security is configured in each service to secure endpoints. On login, the User Service issues a JWT signed with a secret key. Subsequent requests include the JWT in the `Authorization: Bearer <token>` header. A JWT filter validates the token and sets the user's authentication in the security context. Method-level or path-level security annotations (e.g. `@PreAuthorize("hasRole('ADMIN')")`) enforce that only the right roles can call certain APIs. This achieves stateless auth: "the authorizing server needs to maintain no state; the token itself is all that is needed".

- **Documentation (Swagger/OpenAPI)**: Each service includes `springdoc-openapi` to auto-generate API docs. Annotating controllers and models makes the OpenAPI spec available at `/v3/api-docs` and Swagger UI at `/swagger-ui/index.html`.

Frontend Components

The React app is structured with reusable components and Redux for state:

- **Authentication Flow**: Components for Login and Register pages handle user input and call the backend auth endpoints. Upon successful login, the JWT is saved (in localStorage or Redux state) and Redux stores the current user info. Protected routes check for a valid token. Axios interceptors attach the JWT to each outgoing API request.

- **Customer UI Components**: Includes components such as `RestaurantList`, `RestaurantCard`, `MenuList`, `Cart`, and `OrderStatus`.

    - *RestaurantList* fetches and displays restaurants (with filters by cuisine or rating).

    - *MenuList* shows dishes of a selected restaurant (including image, description, price).

    - *Cart* maintains selected items in Redux and lets the user update quantities before checkout.

    - *OrderStatus* polls the Order Service for the latest status of a placed order.

- **Admin UI Components**: Accessible only to users with `ADMIN` role. It includes:

    - *RestaurantManagement*: lists existing restaurants with edit/delete options, plus a form to add new ones.

    - *DishManagement*: for a selected restaurant, list its dishes and allow adding/editing/deleting menu items.

    - *OrderManagement*: list all orders across users, with details. Admin can click to update an order’s status (e.g. from "Placed" to "Out for Delivery").

- **State Management (Redux)**: The app uses Redux to store global state like the authenticated user (including role) and the shopping cart. Reducers and actions handle login/logout, adding/removing items from cart, and updating the cart total. Components dispatch actions, and selectors read state. This unidirectional data flow makes the UI predictable and testable.

- **UI Library**: A component library like Material-UI can be used for form inputs, tables, buttons, etc., to speed up development and ensure consistent design. The layout includes navigation links (different for admins vs customers) and protected routes that redirect unauthorized users.