1. Functional Requirements

1. **User Management**
    - **Registration & Login**
        - A visitor can register by choosing a unique username, unique email, and password.
        - The system checks uniqueness (no email/username duplicates).
        - Successful registration automatically logs the user in.
        - A registered user can log in using either their username or email along with their password.
    - **Roles & Authorization**
        - **Primary User (USER):**
            - Can upload recipes.
            - Can rate existing recipes (1–5 stars).
            - Can browse/search recipes by title or dietary tag.
            - Can view and update their own profile (username, email, password).
        - **Admin/Moderator (MODERATOR):**
            - Has all USER capabilities.
            - Can delete any recipe.
            - Can create, update, or delete recipe categories.
            - Can create, update, or delete dietary tags.
2. **Recipe Management**
    - **Upload Recipe**
        - A logged-in USER fills out a form requiring:
            - Title (max 100 characters)
            - Ingredients (multi-line text)
            - Instructions (multi-line text)
            - Prep time (minutes)
            - Cook time (minutes)
            - Photo (JPEG/PNG ≤ 5 MB)
            - Category (must select one existing category)
            - Dietary tags (zero or more existing tags)
        - The system validates every field; if valid, it saves the photo, creates a new Recipe record with status = "APPROVED," and records upload time and tag associations.
    - **Delete Recipe**
        - An Admin/Moderator can delete any recipe. After confirmation, the system removes the recipe record, its associated ratings, and any tag-associations, and may delete the stored image.
    - **Browse Recent Recipes**
        - Any user (including guests) can view "Recent Recipes," which shows all approved recipes sorted by newest upload first, 12 per page. Each card displays thumbnail, title, average rating, category, tags, and uploader.
    - **Search Recipes**

        - **By Title:** users enter a keyword; the system returns approved recipes whose titles contain that keyword (case-insensitive), sorted by upload date.
        - **By Tag:** users select one or more dietary tags; the system returns approved recipes that include all selected tags, sorted by upload date. Both searches are paginated (12 per page).
    - **View Recipe Details**
        - Clicking any recipe card shows its detail page: title, large photo, uploader, prep/cook times, ingredients list, instructions, category, dietary tags, and average star rating (computed from its ratings).
        - If a logged-in user has not yet rated, a 5-star selector appears; if they have, their rating is shown and can be updated.
    - **Rate Recipe**
        - A logged-in USER can assign 1–5 stars to any approved recipe.
        - If they have not rated it before, a new Rating record is created; if they have, their existing rating is updated.
        - After each rating, the system recalculates and stores the recipe's new average rating.
3. **Category Management (Admin)**
    - **Add Category:** Admin enters a unique name to create a new category.
    - **Edit Category:** Admin renames an existing category (new name must remain unique).
    - **Delete Category:** Before deletion, any recipes assigned to that category must be reassigned (e.g., to "Uncategorized"); then the category is removed.
4. **Tag Management (Admin)**
    - **Add Tag:** Admin enters a unique tag name (e.g., "Vegetarian") to create a new dietary tag.
    - **Edit Tag:** Admin changes the name of an existing tag (must remain unique).
    - **Delete Tag:** Removes a tag and disassociates it from all recipes ("recipe_tags" join entries deleted).

---

2. Non-Functional Requirements

1. **Performance**
    - Listing or searching recipes (up to ~5,000) must respond in under 2 seconds.
    - Sorting by upload date or average rating should complete within 300 ms under moderate load (≤ 100 concurrent users).
2. **Security**
    - All traffic served over HTTPS.
    - Passwords hashed with BCrypt.
    - Server-side validation for all inputs to prevent SQL injection and XSS.
    - CSRF tokens enabled on every form.
    - Role-based access control:
        - Guests cannot upload or rate.

        - USER cannot delete recipes or manage categories/tags.
        - MODERATOR endpoints require moderator role.
3. **Usability & Accessibility**
    - Desktop/laptop only—no mobile responsiveness required.
    - Layout optimized for ≥ 1280 px screens (e.g., sidebar + main-content).
    - Color contrast ≥ 4.5:1 for text.
    - All buttons and links must be keyboard-navigable.
    - Provide meaningful alt text on recipe images.
4. **Maintainability & Extensibility**
    - Three-layer architecture:

1. **Model/Entities** (database tables and JPA classes)
2. **Services** (business logic)
3. **Controllers** (HTTP endpoints)

    - Packages segmented by function (`model`, `repository`, `service`, `controller`).
    - Follow standard Java naming conventions (e.g., `RecipeService`, `UserController`).
    - JUnit 5 + Mockito unit tests for service classes (≥ 60% coverage).
    - Integration tests (Spring Boot Test) for core flows: registration/login, upload/delete, rating, search.
    - Logging via SLF4J + Logback: record user registrations, recipe uploads, deletions, rating actions, and errors.
    - Database migrations managed by Flyway.
5. **Reliability & Availability**
    - Target uptime ≥ 99.5 %.
    - Custom 404 and 500 error pages with friendly messages.
    - Daily PostgreSQL backups and periodic backup of uploaded images.
6. **Portability & Deployment**
    - Java 17+ runtime.
    - Maven build tool (`pom.xml`).
    - PostgreSQL (v13+) on localhost, database named `recipehub`.
    - Spring Boot's `application.properties` configured for PostgreSQL with `spring.jpa.hibernate.ddl-auto=update` during development.
    - Dockerfile optional but not required.

---

3. Objects, Classes, and Relationships

1. **User**
    - **Attributes:**
        - `id` (primary key)
        - `username` (unique, not null)
        - `email` (unique, not null)
        - `passwordHash` (not null)
        - `role` (USER or MODERATOR)

        - `createdAt` (timestamp)
    - **Relationships:**
        - **One → Many** with **Recipe** (`uploader`). A user uploads multiple recipes.
        - **One → Many** with **Rating** (`rater`). A user can rate multiple recipes.
2. **Recipe**
    - **Attributes:**
        - `id` (primary key)
        - `title` (≤ 100 chars, not null)
        - `ingredients` (text, not null)
        - `instructions` (text, not null)
        - `prepTimeMinutes` (integer, not null)
        - `cookTimeMinutes` (integer, not null)
        - `imagePath` (string, not null)
        - `uploadDate` (timestamp, not null)
        - `status` (string, not null, always "APPROVED")
        - `averageRating` (float, nullable)
        - `category` (string, not null; must match a Category name)
    - **Relationships:**
        - **Many → One** with **User** (`uploader`). Each recipe is uploaded by one user.
        - **One → Many** with **Rating**. A recipe can have multiple ratings.
        - **Many ↔ Many** with **Tag** (via join table). A recipe can have multiple dietary tags.
3. **Rating**
    - **Attributes:**
        - `id` (primary key)
        - `stars` (1–5, not null)
        - `ratedAt` (timestamp, not null)
    - **Relationships:**
        - **Many → One** with **User** (`rater`). Each rating is by one user.
        - **Many → One** with **Recipe** (`recipe`). Each rating belongs to one recipe.
    - **Constraint:**
        - A composite unique constraint on `(raterId, recipeId)` ensures a user rates a given recipe only once.
4. **Category**
    - **Attributes:**
        - `id` (primary key)
        - `name` (unique, not null) – e.g., "Desserts," "Salads"
    - **Relationships:**
        - **One → Many** with **Recipe**. Each recipe's `category` string must match exactly one Category name. Deleting a Category requires reassigning affected recipes first.
5. **Tag**
    - **Attributes:**
        - `id` (primary key)
        - `name` (unique, not null) – e.g., "Vegetarian," "Gluten-Free"

- **Relationships:**
  - **Many ↔ Many** with **Recipe**.
    - Implemented via a join table `recipe_tags` (columns: `recipe_id`, `tag_id`).
    - Each recipe can have zero or more tags; each tag can apply to zero or more recipes.
    - Deleting a tag removes its join-table entries so recipes no longer reference it.