**1. Functional Requirements**

**1.1 User Management**

**Registration & Login**

* A visitor may **register** with a unique **username** and **password**.
* Passwords are stored as BCrypt hashes.
* Upon successful registration, the user is redirected to the login page.
* A registered user may **log in** with their username + password.
* Spring Security enforces that only authenticated users can access protected endpoints.

**Roles & Authorization**

* **USER** (the only role in this version):
  + Can upload recipes.
  + Can rate recipes (1–5 stars).
  + Can view a personalized “My Recipes” list and delete only their own recipes.
* **Guests** (not logged in):
  + Can browse, search, and view recipe details.
  + Cannot upload, rate, or delete.

**1.2 Recipe Management**

**Upload Recipe**

* A logged-in user fills a form that requires:
  + **Title** (max 100 chars)
  + **Ingredients** (multi-line text)
  + **Instructions** (multi-line text)
  + **Prep time** (integer minutes)
  + **Cook time** (integer minutes)
  + **Photo** (JPEG/PNG file → saved under static/images/ or configured uploads folder)
  + **Category** (dropdown of pre-populated categories)
  + **Dietary Tags** (zero or more checkboxes)
* All fields are server-validated; missing required inputs redisplay the form with errors.
* If the user has already uploaded an *identical* recipe (same title, ingredients, instructions, times, category, tags), the form re-renders with a “duplicate” warning—no new record is saved.

**Delete Recipe**

* On the “My Recipes” page, each of a user’s own recipes has a **Delete** button.
* Clicking Delete removes the recipe record, all its associated ratings, and its tag-associations from the database.

**Browse Recent Recipes**

* The home page (“Recent”) lists **all** recipes sorted by newest upload first.
* Each recipe card shows: thumbnail, title, average rating (★ / ☆), category.

**Browse Top-Rated Recipes**

* A separate page shows all recipes sorted by descending average rating.

**Search Recipes**

* By Title: enter a keyword → returns all recipes whose titles contain that keyword (case-insensitive).
* The search form lives at /search; results show the same card layout.

**View Recipe Details**

* Click any recipe card → detail page shows:
  + Full-size image, title, uploader username, category, tags
  + Prep & cook times, ingredients & instructions
  + **Average rating** (computed on each page load)
  + If logged in: a 1–5 star **rating form** (pre-populated with the user’s prior vote, if any).

**Rate Recipe**

* A logged-in user may assign 1–5 stars.
* If they have rated before, the new submission **replaces** the old one.
* No separate “average” field is stored; it’s always recomputed from the ratings table.

**1.3 Category & Tag Management**

* Categories and tags are loaded from the database at runtime.
* Currently only Admin-level data edits are done directly in the database or via a separate admin UI (not yet implemented).

**2. Non-Functional Requirements**

**2.1 Performance**

* Typical page loads (list, details, rating) respond in under 500 ms under normal load.
* Database uses simple indexed lookups (by PKs and by title LIKE), so scales to a few thousand recipes.

**2.2 Security**

* All services run over HTTPS in production.
* Passwords are BCrypt-hashed.
* Spring Security enforces authentication on protected endpoints.
* CSRF protection enabled (except where explicitly disabled for simplicity in dev).
* Server-side validation prevents missing fields and rejects bad data.

**2.3 Usability**

* Desktop-first layout using Bootstrap.
* Accessible forms with required fields, logical tab order, and alt text on images.
* Thymeleaf templates render server-side; minimal JavaScript for star hover effects and navbar highlighting.

**2.4 Maintainability & Extensibility**

* Three-layer architecture:
  1. **Model** (JPA entities)
  2. **Service** (business logic, e.g. duplicate check, rating replace)
  3. **Controller** (HTTP endpoints, form binding, redirects)
* Packages are organized by function: model, repository, service, controller, config.
* SLF4J + Logback captures all key actions (login, register, upload, delete, rate) to console + rolling file.
* Future hooks in place for adding moderators, admin UIs, pagination, meal planning, and grocery-list generation.

**3. Key Objects & Relationships (Current)**

1. **User**
   * id, username, password
   * 1 ↔ \* Recipe (owner)
   * 1 ↔ \* Rating (rater)
2. **Recipe**
   * id, title, ingredients, instructions, prepTimeMinutes, cookTimeMinutes, imagePath, uploadDate, username (owner’s username)
   * Many ↔ 1 User (owner)
   * Many ↔ 1 Category
   * Many ↔ Many Tag (join table recipe\_tags)
   * 1 ↔ \* Rating (mapped by recipe)
3. **Rating**
   * id, stars
   * Many ↔ 1 User (rater)
   * Many ↔ 1 Recipe
4. **Category**
   * id, name
   * 1 ↔ \* Recipe
5. **Tag**
   * id, name
   * Many ↔ Many Recipe