**Brew To The Future - Project Proposal**

**Date**: 18/09/2024

**Team Name**: EMA Tech

**Team Members**: Matan Hayon, Adi Kapuri, Eyal Shpater

## 1. Problem Identification

When homebrewers do not use a smart electric brewery, they face various challenges that could impact the quality and consistency of their brewing process. The absence of automation means relying on manual tasks, which can result in errors and inefficiencies. Additionally, the lack of data logging features makes it challenging to troubleshoot problems and fine-tune recipes. Furthermore, without remote monitoring capabilities, brewers are limited in their ability to adjust and oversee the brewing process, reducing their option to re-create their own favorite beer recipes.

## 2. Solution Overview

The Smart Brewery Project aims to modernize home brewing through IoT and real-time analytics. By utilizing smart sensors for brewing temperature, fermentation, and ingredient control, it enhances brew quality and flavor consistency. Home brewers can automatically manage their brewing operations, receive alerts, access insights about their brewing recipe history files, and receive AI recipe suggestions. This project represents a fusion of traditional craft with data-driven efficiency, promising a transformative impact on the home brewing industry by improving the brewing process and overall experience.

## 3. Existing Alternatives

3.1. Manual cooking: Using basic tools like a temperature sensor and gas heat to regulate temperature. Brewers monitor and adjust temperature manually, requiring constant attention for consistency and great effort to maintain the wanted temperature.

3.2. Cooler Brewing Method: This method involves using a cooler, such as a large insulated container, to maintain specific temperatures during the mashing process. Brewers can heat water to a precise temperature and then transfer it to the cooler along with the grains, allowing them to mash at a consistent temperature over time.

3.3. Immersion Circulator: An immersion circulator, commonly used in sous vide cooking, can also be employed for precise temperature control during brewing. By circulating water at a precise temperature around the brew vessel, brewers can achieve consistent temperatures throughout the mashing and fermentation processes.

3.4. Electric Heating Elements: Instead of gas, brewers can utilize electric heating elements coupled with temperature sensor control to regulate the brewing temperature. This approach provides more precise temperature control compared to traditional gas heating methods.

## 4. User Demographics

The expected user base of the Smart Brewery application can be diverse, catering to individuals with varying levels of brewing experience and interests in both traditional craft brewing and modern technological advancements. The target audience includes:

- Individuals who are new to homebrewing and are looking for an accessible and user-friendly platform to start their brewing journey.
- Home brewers with some experience who seek to enhance their brewing skills and experiment with different recipes using smart technology.
- Those passionate about craft beer who appreciate the fusion of traditional brewing techniques with modern technology for improved quality and consistency.
- Enthusiasts who are interested in incorporating IoT and real-time analytics into their brewing process, leveraging smart sensors and automation for precise control.
- Individuals who enjoy sharing their brewing experiences, recipes, and tips within a community, fostering a collaborative environment for knowledge exchange.
- Those who are open to exploring AI-generated recipe suggestions, adding a layer of innovation to their brewing experience and encouraging experimentation with new flavors.

The Smart Brewery application aims to appeal to a broad range of users by offering a seamless blend of traditional brewing practices and advanced technology, making home brewing more accessible, enjoyable, and efficient for a diverse audience.

# 5. Dependencies

## 5.1. Libraries:

### 5.1.1. Front-End:

5.1.1.1. react native useRoute
5.1.1.2. react native useNavigation
5.1.1.3. Firebase

### 5.1.2. Back -End:

5.1.2.1. Spring Boot Starter Web
5.1.2.2. Spring Boot Starter Data JPA
5.1.2.3. Spring Boot DevTools
5.1.2.4. PostgreSQL Driver
5.1.2.5. Spring Boot Starter Test
5.1.2.6. Spring Boot Starter JDBC
5.1.2.7. Jakarta Persistence API
5.1.2.8. OpenCSV
5.1.2.9. Spring Boot Starter OAuth2 Client
5.1.2.10. Spring Boot Starter Security

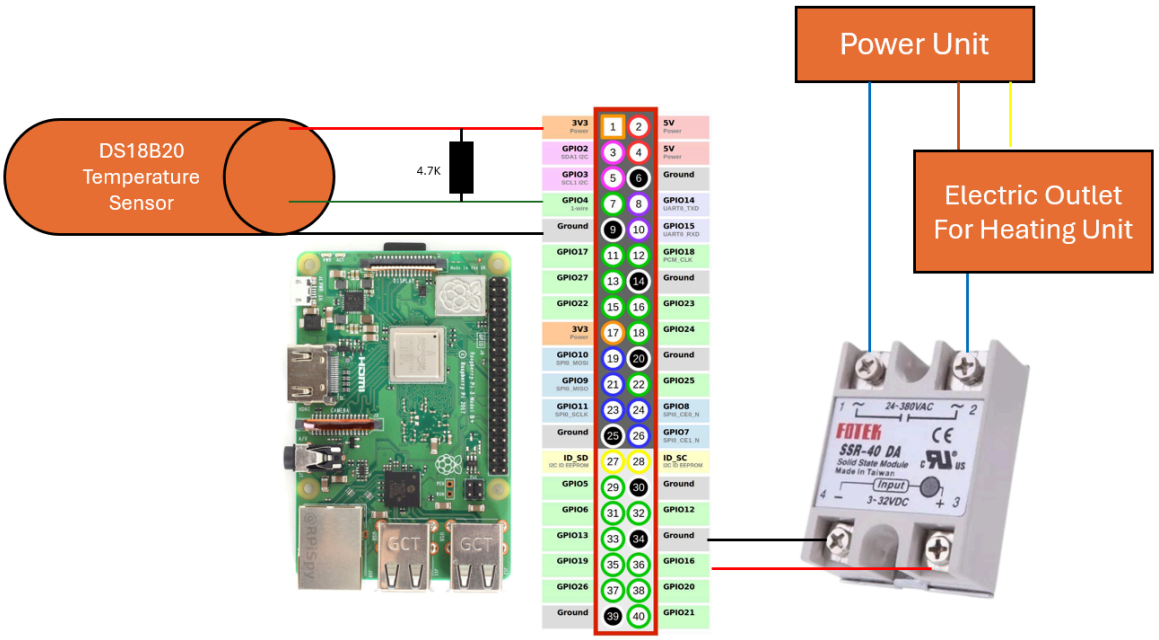
### 5.1.3. Embedded:

5.1.3.1. RaspberryPI 4

## 5.2. Hardware:

**5.2.1.** Temperature sensor (DB18B20)
**5.2.2.** Heating unit

## 5.3. Brewing Box Wiring:



## 6. Features and User Flows:

### 6.1. Share & Download Recipes Online:

**6.1.1.** **User Profiles:** Enable user profiles for individuals to contribute and share their recipes to their followers.

**6.1.2.** **User Contribution:** Allow users to contribute, rate, and review recipes, fostering a collaborative community.

### 6.2. Brew a Recipe:

**6.2.1.** **Recipe Input:** A user-friendly interface for users to input and customize brewing recipes.

Functions:

1. Create new recipe
2. Delete recipe

**6.2.2.** **Automated System:** Integration of an iot automated brewing system monitoring temperature sensor and controlling heat unit, capable of executing recipes with a single click.

**6.2.3.** **Control Interface:** An intuitive control interface to provide users with easy control over the brewing process.

Functions:

1. Show the recipe stages
2. Show the current temperature
3. Show the time passed from the start of the current stage
4. Show temperature change chart

## 7. Technology Stack

### Embedded:

Development languages: Python, C++

FrameWorks: RaspberryPi Os, Arduino IDE

Internet Connectivity: wifi

Api Communication: Rest API

### Back-End:

Development language: Java

FrameWork: Maven, Spring Boot

Data Base: PostgreSQL

Api Communication: Rest API

### Front-End:

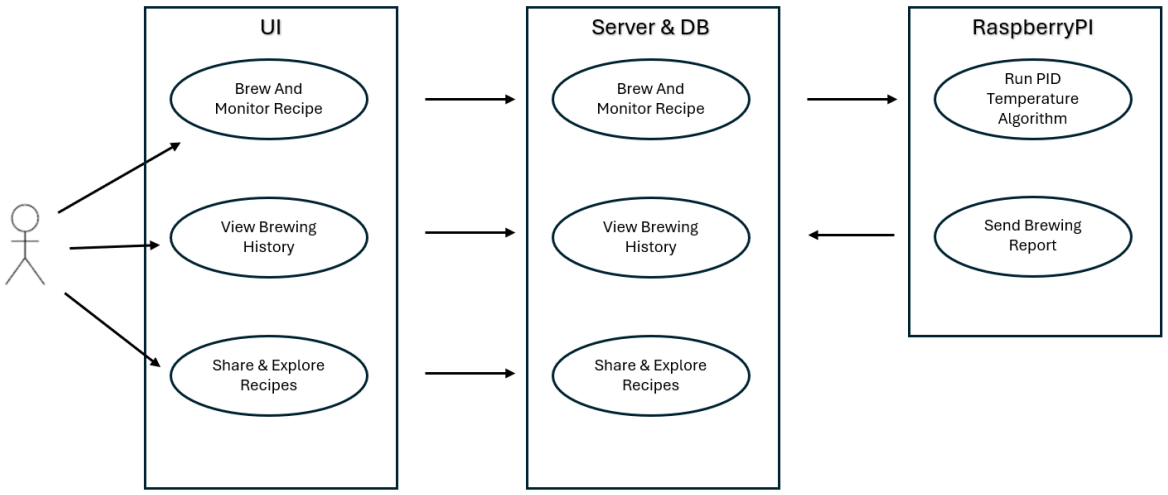
Development language: JS

FrameWork: React Native

Application: Mobile

Api Communication: Rest API

**Use-Case Diagram**



# 8. Schemes

## 8.1. Json Objects

### 8.1.1. Embedded Recipe DTO

- **Purpose**: Represents a brew recipe for the embedded system.
- **Fields**:
  - `brew_id`: Unique identifier for the brew.
  - `recipe_id`: Unique identifier for the recipe.
  - `recipe_name`: Name of the recipe.
  - `user_id`: ID of the user who created the recipe.
  - `step`: An array of steps in the brewing process.
    - `step_id`: Unique identifier for the step.
    - `temperature_celsius`: Target temperature for the step.
    - `duration_minutes`: Duration of the step in minutes.
    - `approval_required`: Indicates if the step requires user approval.
- **Json Example:**

```
{
  "brew_id": 1234,
  "recipe_id": 5678,
  "recipe_name": "My Favorite IPA",
  "user_id": "987654321",
  "step": [
    {
      "step_id": 1,
      "temperature_celsius": 68.0,
      "duration_minutes": 60,
      "approval_required": true
    },
    {
      "step_id": 2,
      "temperature_celsius": 75.0,
      "duration_minutes": 120,
      "approval_required": false
    }
  ]
}
```

### 8.1.2. Brewing Report DTO

- **Purpose**: Provides real-time information about the brewing process.
- **Fields**:
  - `brew_id`: Unique identifier for the brew.
  - `user_id`: ID of the user who started the brew.
  - `recipe_id`: ID of the recipe being used.
  - `timestamp`: Timestamp of the report.
  - `temperature_celsius`: Current temperature of the brew.
  - `current_step`: Current step in the brewing process.
  - `step_start_time`: Timestamp when the current step started.
  - `status_code`: code indicating the success or failure of the sensor. 200 for success, 500 for error.
  - `error_message`: Error message if the sensor encountered an issue.
- **Json Example:**

```
{
  "brew_id": 1234,
  "user_id": "987654321",
  "recipe_id": 5678,
  "timestamp": 1699904000000,
  "temperature_celsius": 68.5,
  "current_step": 2,
  "step_start_time": 1699903400000,
  "status_code": 200,
  "error_message": null
}
```

### 8.1.3. Fermentation Report DTO

- **Purpose**: Provides real-time information about the fermentation process.
- **Fields**:
  - `brew_id`: Unique identifier for the brew.
  - `user_id`: ID of the user who started the brew.
  - `recipe_id`: ID of the recipe being used.
  - `timestamp`: Timestamp of the report.
  - `temperature_celsius`: Current temperature of the fermentation chamber.
  - `relative_humidity`: Relative humidity of the fermentation chamber.
  - `status_code`: code indicating the success or failure of the sensor.. 200 for success, 500 for error.
  - `error_message`: Error message if the sensor encountered an issue.
- **Json Example:**

```
{
  "brew_id": 1234,
  "user_id": "987654321",
  "timestamp": 1699904000000,
  "temperature_celsius": 20.0,
  "relative_humidity": 75.0,
  "status_code": 200,
  "error_message": null
}
```

#### 8.1.4. Recipe DTO

- **Purpose**: Represents a complete recipe for brewing.
- **Fields**:
  - `user_id`: ID of the user who created the recipe.
  - `recipe_name`: Name of the recipe.
  - `recipe`: An array of steps in the brewing process.
    - `step_id`: Unique identifier for the step.
    - `temperature_celsius`: Target temperature for the step.
    - `duration_minutes`: Duration of the step in minutes.
    - `notify_on_completion`: Indicates if a notification should be sent when the step is complete.
    - `message`: Message associated with the step.
  - `notifications`: An array of notifications to be sent after the brewing process.
    - `message`: Message of the notification.
    - `send_after_days`: Number of days after the brew ends when the notification should be sent.
  - `method`: Brewing method (e.g., All Grain, Extract).
  - `style`: Beer style.
  - `abv`: Alcohol by volume.
  - `ibu`: International bitterness units.
  - `original_gravity`: Original gravity of the wort.
  - `final_gravity`: Final gravity of the beer.
  - `color`: Color of the beer (in SRM).
  - `batch_size_liter`: Batch size in liters.
  - `fermentables`: An array of fermentable ingredients.
    - `id`: Unique identifier for the fermentable.

        - `amount_kg`: Amount of fermentable in kilograms.
    - `hops`: An array of hop additions.
        - `id`: Unique identifier for the hop.
        - `amount_g`: Amount of hop in grams.
        - `time_minutes`: Boil time in minutes.
    - `yeast`: An array of yeast strains used in the recipe.
        - `id`: Unique identifier for the yeast.
        - `temperature_celsius`: Target fermentation temperature.
- **Json Example:**

```
{
  "user_id": "987654321",
  "recipe_name": "My Favorite IPA",
  "recipe": [
    {
      "step_id": 1,
      "temperature_celsius": 68.0,
      "duration_minutes": 60,
      "notify_on_completion": true,
      "message": "Mash in grains"
    }
  ],
  "notifications": [
    {
      "message": "Add hops for dry hopping",
      "send_after_days": 7
    }
  ],
  "method": "All Grain",
  "style": "India Pale Ale",
  "abv": 6.5,
  "ibu": 70,
  "original_gravity": 1.060,
  "final_gravity": 1.012,
  "color": 15,
  "batch_size_liter": 20,
  "fermentables": [
    {
      "id": 1,
```

```
      "amount_kg": 4.0
    }
  ],
  "hops": [
    {
      "id": 3,
      "amount_g": 20,
      "time_minutes": 60
    }
  ],
  "yeast": [
    {
      "id": 5,
      "temperature_celsius": 20
    }
  ]
}
```

#### 8.1.5. Hop DTO

- **Purpose**: Represents a hop variety.
- **Fields**:
  - `id`: Unique identifier for the hop.
  - `name`: Name of the hop.
  - `averageAA`: Average alpha acid content.
  - `use`: Intended use (e.g., Boil, Aroma).
  - `type`: Type of hop (e.g., Pellet, Whole).
- **Json Example:**

```
{
  "id": 1,
  "name": "Citra",
  "averageAA": 14.5,
  "use": "Boil",
  "type": "Pellet"
}
```

#### 8.1.6. Yeast DTO

- **Purpose:** Represents a yeast strain.
- **Fields:**
  - `id`: Unique identifier for the yeast.
  - `name`: Name of the yeast strain.
  - `laboratory`: Laboratory that produced the yeast.

- `type`: Type of yeast (e.g., Ales, Lagers).
- `alcoholTolerance`: Alcohol tolerance.
- `flocculation`: Flocculation characteristics.
- `attenuation`: Attenuation level.
- `minTemp`: Minimum fermentation temperature.
- `maxTemp`: Maximum fermentation temperature.

- **Json Example:**

```
{
  "id": 1,
  "name": "Wyeast 1056 American Ale",
  "laboratory": "Wyeast",
  "type": "Ales",
  "alcoholTolerance": "Medium",
  "flocculation": "Medium",
  "attenuation": "75%",
  "minTemp": "60°F",
  "maxTemp": "75°F"
}
```

#### 8.1.7. Fermentable DTO

- **Purpose:** Represents a fermentable ingredient.
- **Fields:**
  - `id`: Unique identifier for the fermentable.
  - `name`: Name of the fermentable.
  - `country`: Country of origin.
  - `category`: Category of fermentable (e.g., Grain, Sugar).
  - `type`: Type of fermentable (e.g., Malt, Adjunct).
  - `color`: Color of the fermentable (in °L).
  - `ppg`: Points per gallon (a measure of fermentability).
- **Json Example:**

```
{
  "id": 1,
  "name": "Maris Otter",
  "country": "United Kingdom",
  "category": "Grain",
  "type": "Base Malt",
  "color": 2.0,
  "ppg": 125
}
```

### 8.1.8. Notification DTO

- **Purpose:** Represents a notification sent to the user.
- **Fields:**
  - `message`: Message of the notification.
  - `timestamp`: Timestamp when the notification was sent.
  - `requires_approval`: Indicates if the notification requires user approval.
  - `status`: Status of the notification (200 for success, 500 for error notification).
- **Json Example:**

```
{
  "message": "Mash in grains",
  "timestamp": 1699904000000,
  "requires_approval": false,
  "status": 200
}
```

### 8.1.9. Rating DTO

- **Purpose:** Represents a rating and review for a recipe.
- **Fields:**
  - `rating`: Numerical rating (e.g., 1-5).
  - `review`: Text review.
  - `userId`: ID of the user who left the rating and review.
  - `reviewDate`: Timestamp of the review.
  - `recipeId`: ID of the recipe being rated.
- **Json Example:**

```
{
  "rating": 4.5,
  "review": "Great recipe!",
  "userId": "987654321",
  "reviewDate": 1699904000000,
  "recipeId": 5678
}
```

## 8.2. Data Base

### 8.2.1. Device Table

```
CREATE TABLE device (
serial_number VARCHAR(255) PRIMARY KEY,
type VARCHAR(255),
user_id VARCHAR(255) NOT NULL
);
```

### 8.2.2. Fermentable Table

```
CREATE TABLE fermentables (
id BIGSERIAL PRIMARY KEY,
category VARCHAR(255),
color VARCHAR(255),
country VARCHAR(255),
name VARCHAR(255),
ppg DOUBLE PRECISION NOT NULL,
type VARCHAR(255)
);
```

### 8.2.3. Hop Table

```
CREATE TABLE hops (
id BIGSERIAL PRIMARY KEY,
type VARCHAR(255),
average_aa DOUBLE PRECISION,
name VARCHAR(255),
use VARCHAR(255)
);
```

### 8.2.4. Recipe Table

```
CREATE TABLE recipe (
recipe_id INT PRIMARY KEY,
abv DOUBLE PRECISION NOT NULL,
batch_size_liter INT NOT NULL,
color DOUBLE PRECISION NOT NULL,
final_gravity DOUBLE PRECISION NOT NULL,
ibu DOUBLE PRECISION NOT NULL,
method VARCHAR(255),
original_gravity DOUBLE PRECISION NOT NULL,
recipe_json TEXT,
recipe_name VARCHAR(255) NOT NULL,
style VARCHAR(255),
time_created BIGINT NOT NULL,
```

```
user_id VARCHAR(255) NOT NULL
);
```

#### 8.2.5. Brew Table

```
CREATE TABLE brew (
id BIGINT PRIMARY KEY,
start_time BIGINT NOT NULL,
user_id VARCHAR(255) NOT NULL,
recipe_id INT REFERENCES recipe(recipe_id)
);
```

#### 8.2.6. Brewing Report Table

```
CREATE TABLE brewing_report (
id BIGINT PRIMARY KEY,
current_step INT NOT NULL,
error_message VARCHAR(255),
status_code INT NOT NULL,
step_start_time BIGINT NOT NULL,
temperature_celsius DOUBLE PRECISION NOT NULL,
timestamp BIGINT NOT NULL,
user_id VARCHAR(255) NOT NULL,
brew_id BIGINT REFERENCES brew(id)
);
```

#### 8.2.7. Fermentation Report Table

```
CREATE TABLE fermentation_report (
id BIGINT PRIMARY KEY,
error_message VARCHAR(255),
relative_humidity DOUBLE PRECISION NOT NULL,
status_code INT NOT NULL,
temperature_celsius DOUBLE PRECISION NOT NULL,
timestamp BIGINT NOT NULL,
user_id VARCHAR(255) NOT NULL,
brew_id BIGINT REFERENCES brew(id)
);
```

#### 8.2.8. Rating Table

```
CREATE TABLE rating (
id BIGINT PRIMARY KEY,
rating DOUBLE PRECISION NOT NULL,
review VARCHAR(255),
review_date BIGINT,
user_id VARCHAR(255),
```

recipe_id INT REFERENCES recipe(recipe_id)
);

#### 8.2.9. Recipe Fermentable Table

CREATE TABLE recipe_fermentable (
id BIGINT PRIMARY KEY,
amount_kg DOUBLE PRECISION NOT NULL,
fermentable_id BIGINT REFERENCES fermentables(id),
recipe_id INT REFERENCES recipe(recipe_id)
);

#### 8.2.10. Recipe Hop Table

CREATE TABLE recipe_hops (
id BIGINT PRIMARY KEY,
amount_g DOUBLE PRECISION NOT NULL,
time_min INT NOT NULL,
hop_id BIGINT REFERENCES hops(id),
recipe_id INT REFERENCES recipe(recipe_id)
);

#### 8.2.11. Style Table

CREATE TABLE style (
id INT PRIMARY KEY,
name VARCHAR(255) NOT NULL
);

#### 8.2.12. Yeast Table

CREATE TABLE yeasts (
id BIGSERIAL PRIMARY KEY,
alcohol_tolerance VARCHAR(255),
attenuation VARCHAR(255),
flocculation VARCHAR(255),
laboratory VARCHAR(255),
max_temp VARCHAR(255),
min_temp VARCHAR(255),
name VARCHAR(255),
type VARCHAR(255)
);

#### 8.2.13. Recipe Yeast Table

CREATE TABLE recipe_yeast (
id BIGINT PRIMARY KEY,
temperature_celsius DOUBLE PRECISION NOT NULL,
recipe_id INT REFERENCES recipe(recipe_id),

```
    yeast_id BIGINT REFERENCES yeasts(id)
);
```


recipe
abv double precision
batch_size_liter integer
color double precision
final_gravity double precision
ibu double precision
method varchar(255)
original_gravity double precision
recipe_json text
recipe_name varchar(255)
style varchar(255)
time_created bigint
user_id varchar(255)
recipe_id integer
hops
type varchar(255)
average_aa double precision
name varchar(255)
use varchar(255)
id bigint
fermentables
category varchar(255)
color varchar(255)
country varchar(255)
name varchar(255)
ppg double precision
type varchar(255)
id bigint
yeasts
alcohol_tolerance varchar(255)
attenuation varchar(255)
flocculation varchar(255)
laboratory varchar(255)
max_temp varchar(255)
min_temp varchar(255)
name varchar(255)
type varchar(255)
id bigint
recipe_id
recipe_id
recipe_id
recipe_id
recipe_id
yeast_id:id
brew
start_time bigint
user_id varchar(255)
recipe_id integer
id bigint
recipe_yeast
temperature_celsius double precision
recipe_id integer
yeast_id bigint
id bigint
fermentable_id:id
hop_id:id
recipe_fermentable
amount_kg double precision
fermentable_id bigint
recipe_id integer
id bigint
rating
rating double precision
review varchar(255)
review_date bigint
user_id varchar(255)
recipe_id integer
id bigint
recipe_hops
amount_g double precision
time_min integer
hop_id bigint
recipe_id integer
id bigint
brew_id:id
brew_id:id
device
type varchar(255)
user_id varchar(255)
serial_number varchar(255)
style
name varchar(255)
id integer
brewing_report
current_step integer
error_message varchar(255)
status_code integer
step_start_time bigint
temperature_celsius double precision
timestamp bigint
user_id varchar(255)
brew_id bigint
id bigint
fermentation_report
error_message varchar(255)
relative_humidity double precision
status_code integer
temperature_celsius double precision
timestamp bigint
user_id varchar(255)
brew_id bigint
id bigint

# 9. Rest API

## 9.1. Embedded Endpoints:

### 9.1.1. Get Recipe to Brew

Retrieve the brewing recipe for a specific brewing device.

- **URL**: /api/embedded/{deviceSerialNumber}/brew/recipe
- **Method**: GET
- **Path Parameters**:
  - deviceSerialNumber (String): The serial number of the brewing device.
- **Response**:
  - 200 OK: Returns an EmbeddedRecipeDTO object containing the recipe details for the specified device.
- **Example Response:**


```
{
    "brew_id": 602,
    "recipe_id": 1353,
    "recipe_name": "Hopsi Dopsi",
    "user_id": "111740898570066860021",
    "step": [
        {
            "step_id": 1,
            "temperature_celsius": 68.0,
            "duration_minutes": 0,
            "approval_required": true
        },
        {
            "step_id": 2,
            "temperature_celsius": 68.0,
            "duration_minutes": 60,
            "approval_required": true
        }
}
```

### 9.1.2. Start Brewing

Begin the brewing process for a device.

- **URL:** /api/embedded/{deviceSerialNumber}/brew/start
- **Method:** POST
- **Path Parameters:**
  - **deviceSerialNumber(String):** The serial number of the brewing device.
- **Query Parameters:**
  - **interval (long)**: The time interval (in milliseconds) between brewing process updates.
- **Example Request:**
  - POST /api/embedded/12345/brew/start?interval=60000

### 9.1.3. Mark Brewing as Finished

Mark the brewing process as completed.

- **URL**: /api/embedded/{deviceSerialNumber}/brew/recipe/marks_as_completed
- **Method**: PUT
- **Path Parameters**:
  - deviceSerialNumber (String): The serial number of the brewing device.

### 9.1.4. Add Brewing Report

Submit a brewing report to update the current brewing status.

- **URL**: /api/embedded/{deviceSerialNumber}/report/brewing
- **Method**: POST
- **Path Parameters**:
  - deviceSerialNumber (String): The serial number of the brewing device.
- **Request Body**:
  - BrewingReportDTO: A JSON object that contains the brewing report data.
- **Response:** 200 OK: Returns an integer status representing the current brew status of the device. (as described here)

- **Example Request:**
  - POST /api/embedded/12345/report/brewing
    body:

```
{
        "brew_id": 2052,
        "user_id": "111740898570066860021",
        "recipe_id": 1254,
        "timestamp": 17153566609860,
        "temperature_celsius": 24.5,
        "current_step": 1,
        "step_start_time": 1715356660471,
        "status_code": 200,
        "error_message": null
    }
```

### 9.1.5. Add Fermentation Report

Submit a fermentation report to update the fermentation status.

- **URL**: /api/embedded/{deviceSerialNumber}/report/fermentation
- **Method**: POST
- **Path Parameters**:
  - deviceSerialNumber (String): The serial number of the brewing device.
- **Request Body**:
  - FermentationReportDTO: A JSON object that contains the fermentation report data.
- **Response**:
  - 200 OK: Returns an integer status representing the current fermentation status of the device.
- **Example Request:**
  - POST /api/embedded/12345/report/fermentation
    body:

```
{
        "brew_id": 2052,
        "user_id": "111740898570066860021",
        "timestamp": 17153566609860,
        "temperature_celsius": 24.5,
        "relative_humidity": 2.6,
        "status_code": 200,
        "error_message": null
    }
```

## 9.2. Frontend Endpoints:

All endpoints are protected by OAuth2. The authenticated user's information (i.e., `userId`) is automatically derived from the OAuth2 token passed in the `Authorization` header of each request. The `userId` is not directly sent in the request body or URL but is retrieved from the token, ensuring secure handling of user-specific actions such as recipe management, brewing data retrieval, and device management.

### 9.2.1. Create a New Recipe

Adds a new brewing recipe and starts brewing it.

- **URL:** /api/brew/recipe
- **Method:** POST
- **Request Body**:
  - RecipeDTO: A JSON object that contains the recipe data.
- **Example Request:**
  - POST api/brew/recipe
    body:

```
{
  "user_id": "111740898570066860021",
  "recipe_name": "Hopsi Dopsi",
  "recipe": [
    {
      "step_id": 1,
      "temperature_celsius": 68,
      "duration_minutes": 0,
      "notify_on_completion": true,
      "message": "Add grains"
    },
    {
      "step_id": 2,
      "temperature_celsius": 68,
      "duration_minutes": 60,
      "notify_on_completion": true,
      "message": "Remove grains"
    }
  ],
  "notifications": [
    {
      "message": "Add Hops to dryhop",
      "send_after_days": 2
    }
```

```
  ],
  "method": "All Grain",
  "style": "India Pale Ale",
  "abv": 9.2,
  "ibu": 4.1,
  "original_gravity": 1.049,
  "final_gravity": 1.01,
  "color": 3.7,
  "batch_size_liter": 34,
  "fermentables": [
    {
      "id": 26,
      "amount_kg": 3.25
    },
    {
      "id": 10,
      "amount_kg": 1.75
    }
  ],
  "hops": [
    {
      "id": 31,
      "amount_g": 20,
      "time_minutes": 15
    }
  ],
  "yeast": [
    {
      "id": 12,
      "temperature_celsius": 20
    },
    {
      "id": 115,
      "temperature_celsius": 20
    }
  ]
}
```


### 9.2.2. Add a Recipe

Adds a new brewing recipe.

- **URL:** /api/brew/recipe/add
- **Method:** POST
- **Request Body:**
  - RecipeDTO: A JSON object that contains the recipe data.

#### 9.2.3. Get All Recipes

Retrieves a list of all brewing recipes.

- **URL:** /api/brew/recipes/all
- **Method:** GET
- **Response:** List of RecipeDTO objects.

#### 9.2.4. Get Recipes by User

Get all recipes associated with the logged-in user.

- **URL:** /api/brew/recipe/user
- **Method:** GET
- **Response:**
  - RecipeDTO: A JSON object that contains the recipe data.

#### 9.2.5. Get Recipe by ID

Retrieves a specific recipe by its ID.

- **URL**: /api/brew/recipe/id/{recipeId}
- **Method**: GET
- **Path Parameters**:
  - **recipeId** (int): The ID of the recipe to retrieve.
- **Response**:
  - **RecipeDTO**: A JSON object containing the recipe data.

#### 9.2.6. Delete Recipe by ID

Deletes a specific recipe by its ID.

- **URL**: /api/brew/recipe/{recipeId}
  **Method**: DELETE
  **Path Variable**:
  - **recipeId** (int): The ID of the recipe to delete.

### 9.2.7. Get Brewing Methods

Retrieve available brewing methods.

- **URL**: /api/brew/methods
- **Method**: GET
- **Response**:
  - **List<String>**: A list of brewing method names.

### 9.2.8. Get Brewing Styles

Retrieve available brewing styles.

- **URL**: /api/brew/styles
- **Method**: GET
- **Response**:
  - **List<String>**: A list of brewing style names.

### 9.2.9. Get Hops

Retrieve all available hop.

- **URL**: /api/brew/ingredients/hops
- **Method**: GET
- **Response**:
  - **List<HopDTO>**: A list of JSON objects representing hop data.
- **Example Response**:

```
[
  {
    "id": 1,
    "name": "Admiral",
    "averageAA": 14.3,
    "use": "Boil",
    "type": "Pellet"
  },
  {
    "id": 2,
    "name": "Ahtanum",
    "averageAA": 5.5,
    "use": "Boil",
    "type": "Pellet"
  }
]
```

### 9.2.10. Get Hop by ID

Retrieves a specific hop by its ID.

- **URL**: /api/brew/ingredients/hops/{id}
- **Method**: GET
- **Path Variable**:
  - **id** (long): The ID of the hop to retrieve.
- **Response**:
  - **HopDTO**: A JSON object representing the hop data.

### 9.2.11. Get Yeasts Ingredients

Retrieves all available yeasts.

- **URL:** /api/brew/ingredients/yeasts
- **Method:** GET
- **Response**:
  - **List<YeastDTO>**: A list of JSON objects representing yeast data.
- **Example Response:**


```
[
  {
    "id": 1,
    "name": "Escarpment Labs - Brut Ale",
    "laboratory": "",
    "type": "Ales",
    "alcoholTolerance": "Medium",
    "flocculation": "Low",
    "attenuation": "85%",
    "minTemp": "57°F",
    "maxTemp": "72°F"
  },
   {
    "id": 72,
    "name": "Bootleg Biology - Chardonnay",
    "laboratory": "Bootleg Biology",
    "type": "S. boulardii",
    "alcoholTolerance": "Medium",
    "flocculation": "Low",
    "attenuation": "70%",
    "minTemp": "65°F",
    "maxTemp": "75°F"
  }
]
```

### 9.2.12. Get Yeast by ID

Retrieves a specific yeast by its ID.

- **URL**: /api/brew/ingredients/yeasts/{id}
- **Method**: GET
- **Path Variable**:
  - **id** (long): The ID of the hop to retrieve.
- **Response**:
  - YeastDTO: A JSON object representing the hop data.

### 9.2.13. Get fermentables Ingredients

Retrieves all available fermentables.

- **URL:** /api/brew/ingredients/fermentables
- **Method:** GET
- **Response**:
  - **List<FermentableDTO>**: A list of JSON objects representing fermentable data.
- **Example Response:**


```
[
  {
    "id": 1,
    "name": "Youngberry",
    "country": "",
    "category": "Fruit",
    "type": "Fruit",
    "color": "0 °L",
    "ppg": 2.5
  },
  {
    "id": 2,
    "name": "Admiral - Yolo Gold Wheat",
    "country": "United States",
    "category": "Grain",
    "type": "Base malt",
    "color": "3 °L",
    "ppg": 39
  }
]
```

#### 9.2.14. Get Fermentable by ID

Retrieves a specific yeast by its ID.

- **URL**: /api/brew/ingredients/fermentables/{id}
- **Method**: GET
- **Path Variable**:
  - **id** (long): The ID of the hop to retrieve.
- **Response**:
  - **FermentableDTO**: A JSON object representing the hop data.

#### 9.2.15. Brew Recipe by ID

Start brewing a specific recipe by its ID.

- **URL**: /api/brew/recipe/{recipeId}
- **Method**: POST
- **Path Variable**:
  - **recipeId** (int): The ID of the recipe to brew.

#### 9.2.16. Get Brewing Data

Retrieve brewing data for the logged-in user.

- **URL**: /api/brew/data
- **Method**: GET
- **Response**:
  - **List<BrewingReportDTO>**: A list of BrewingReportDTO: A JSON object that contains the brewing report data.

#### 9.2.17. Get Latest Brewing Report

Retrieve the latest brewing report for the logged-in user.

- **URL:** /api/brew/data/latest
- **Method:** GET
- **Response:**
  - **BrewingReportDTO:** A JSON object representing the latest brewing report.

#### 9.2.18. Get Fermentation Report

Retrieve fermentation data for the logged-in user.

- **URL: /api/fermentation/data**
- **Method:** GET
- **Response:**
  - **List<FermentationReportDTO>:** A list of fermentation reports for the user.

#### 9.2.19. Get Latest Fermentation Report

Retrieve the latest fermentation report for the logged-in user.

- **URL: /api/fermentation/data/latest**
- **Method:** GET
- **Response:**
  - **FermentationReportDTO:** A JSON object representing the latest fermentation report.

#### 9.2.20. Get Notification

Retrieve notifications for the logged-in user.

- **URL: /api/notification**
- **Method:** GET
- **Response:**
  - **NotificationDTO:** A JSON object representing the user's notification.
- **Example Response**:

```
{
    "message": "Add grains",
    "timestamp": 1725879143582,
    "requires_approval": false,
    "status": 200
}
```

#### 9.2.21. Mark Step as Complete

Marks the current brewing step as complete for the user.

- **URL:** /api/brew/recipe/current_brewing/step/complete
- **Method:** POST

#### 9.2.22. Add Device

Add a brewing or fermentation device to the user's account.

- **URL:** /api/device/add
- **Method:** POST
- **Query Parameters:**
  - **deviceSerialNumber** (String) (required): The required device serial number.
  - **type** (String): (required, must be either "brewing" or "fermentation")

#### 9.2.23. Rate Recipe

Rates a specific recipe.

- **URL:** /api/brew/rate/rating/{recipeId}
- **Method:** POST
- **Path Variable**:
  - **recipeId** (int): The ID of the recipe to rate.
- **Query Parameters**:
  - **rating** (double): The rating value to assign to the recipe.

#### 9.2.24. Add Review to Recipe

Adds a review to a specific recipe.

- **URL:** /api/brew/rate/review/{recipeId}
- **Method:** POST
- **Path Variable**:
  - **recipeId (int):** The ID of the recipe to review.
- **Query Parameters**:
  - **review (String):** The review text.

#### 9.2.25. Get Ratings by Recipe ID

Retrieve all ratings for a specific recipe.

- **URL:** /api/brew/rate/{recipeId}
- **Method:** GET
- **Path Variable**:
  - **recipeId** (int): The ID of the recipe whose ratings are being fetched.
- **Response:**
  - **List<RatingDTO>**: A list of JSON objects representing the ratings for the recipe.
- **Example Response:**


```
[
    {
        "rating": 3.5,
        "review": "sample review",
        "userId": "111740898570066860021",
        "reviewDate": 1726342264332,
        "recipeId": 1252
    }
]
```

#### 9.2.26. Stop Brewing

Stops the current brewing process for the user.

- **URL:** /api/brew/stop
- **Method:** POST

### 9.3. General Endpoints:

#### 9.3.1. Health Check

Returns a random sentence from a predefined list, typically used as a health check or fun response.

- **URL: /, /api/health, /health**
- **Method:** GET

#### 9.3.2. Load Data

Triggers the loading of initial data into the brewing system's database.

- **URL:** /api/init/load_data
- **Method:** GET

### Brewing Status Code on Server

| Brewing Status | Code | Description |
|---|---|---|
| BREWING | 100 | Represents an ongoing brewing process. |
| FERMENTATION | 102 | Reflects a process that is being processed. |
| COMPLETED | 200 | Indicates the brewing process completed successfully. |
| STOPPED | 400 | Indicates the brewing process was stopped, often due to a client-side issue. |
| PENDING | 202 | The brewing process is accepted but not yet completed. |
| ERROR | 500 | Indicates an error in the brewing process. |
| UNKNOWN | 520 | Represents an unknown error in the brewing process. |
| NOT_STARTED | 204 | Indicates that the brewing process has not started, no content. |