API Reference

RecipEase.Server

API Version: 1.0

INDEX

API

1. OIDCCONFIGURATION

1.1 GET /_configuration/{clientId}

Authentication endpoint.

This endpoint is necessary for .NET Identity to work.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*clientId	string	

RESPONSE

STATUS CODE - 200: Success

2. RECIPE

2.1 GET /api/Recipe/{id}

Get a recipe.

Returns the recipe with the given `id`, or gives a 404 status code if it doesn't exist in the database. The recipe will include it's average rating if it has been rated.

This endpoint interacts with all attributes from the `recipe` table, and with the `RecipeId` and `Rating` attributes from the `reciperating` table.

The endpoint performs a `select *` query with a `where` clause to find the specified recipe. If the recipe was found, the `reciperating` table is queried for rows with `RecipeId` matching the found id, and the `Rating` attribute is collected.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*id	int32	The id of the recipe to return.

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
  id             integer
  name*          string
  steps          string
  cholesterol    number
  fat            number
  sodium         number
  protein        number
  carbs          number
  calories       number
  authorId*      string
  averageRating  number
}
```

STATUS CODE - 404: Not Found

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

2.2 PUT /api/Recipe/{id}

Edit a recipe.

Updates the given recipe in the database. The `id` in the url must match the `id` in the recipe.

The customer specified by `authorId` must be the authenticated user making this request.

This endpoint interacts with the `recipe` and `customer` tables. The `UserId` attribute on the `customer` table will be checked against the `authorId`.

The endpoint will perform an `update` command on the `recipe` table to update the recipe, and foreign key constraints will be relied upon to validate the `authorId`.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*id	int32	The id of the recipe to update.

REQUEST BODY - application/json

```
{
  id             integer
  name*          string
  steps          string
  cholesterol    number
  fat            number
  sodium         number
  protein        number
  carbs          number
  calories       number
  authorId*      string
  averageRating  number
}
```

RESPONSE

STATUS CODE - 204: Success

STATUS CODE - 400: Bad Request

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - 404: Not Found

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

2.3 DELETE /api/Recipe/{id}

Delete a recipe.

Deletes the given recipe in the database.

If the recipe does not exist in the database, a 404 status code is returned. The customer specified by `AuthorId` in the found recipe must be the authenticated user making this request.

This endpoint interacts with the `recipe` and `customer` tables. The `UserId` attribute on the `customer` table will be checked against the `AuthorId` from the `recipe` table.

The endpoint will perform a `select` query on the `customer` table to validate the `authorId`, and a `delete` command on the `recipe` table to delete the recipe.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*id	int32	The id of the recipe to update.

RESPONSE

STATUS CODE - 200: Success

STATUS CODE - 400: Bad Request

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - 404: Not Found

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

2.4 GET /api/Recipe

Get a list of recipes.

Returns a list of recipes from the database, optionally filtered by title, category, and/or user id.

This endpoint interacts with all attributes in the `recipe` and `recipeincategory` tables, and the `UserId` attribute in the `Customer` table.

The endpoint performs a `select *` query, with a `where` clause included when necessary to apply the specified filters. The `recipe` table is optionally joined with `recipeincategory` to filter by category. The `recipe` table is optionally joined with the `customer` table to filter by customer.

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
titleMatch	string	String to filter recipes by title. If provided, only recipes with the filter string in their title (case insensitive) will be returned.
categoryName	string	String to filter recipes by category. If provided, only recipes in the given category will be returned. If there is no category with the given name in the database, no recipes will be returned.
userId	string	Get only recipes authored by the customer with this id.

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
[{
Array of object:
  id              integer
  name*           string
  steps           string
  cholesterol     number
  fat             number
  sodium          number
  protein         number
  carbs           number
  calories        number
  authorId*       string
  averageRating   number
}]
```

2.5 POST /api/Recipe

Create a recipe.

Adds the given recipe to the database, and returns it on success. If the `id` is specified for the recipe, the endpoint will attempt to add the recipe to the database with that `id`. If a recipe with the given `id` already exists, an error code is returned. If the `id` is not specified, the `id` is automatically generated for the given recipe.

The customer specified by `authorId` must be the authenticated user making this request.

This endpoint interacts with the `recipe` and `customer` tables. The `UserId` attribute on the `customer` table will be checked against the `authorId`.

The endpoint will perform an `insert` command on the `recipe` table to add the recipe, and foreign key constraints will be relied upon to validate the `authorId`.

REQUEST

REQUEST BODY - application/json

```
{
  id             integer
  name*          string
  steps          string
  cholesterol    number
  fat            number
  sodium         number
  protein        number
  carbs          number
  calories       number
  authorId*      string
  averageRating  number
}
```

RESPONSE

STATUS CODE - 201: Success

RESPONSE MODEL - application/json

```
{
  id             integer
  name*          string
  steps          string
  cholesterol    number
  fat            number
  sodium         number
  protein        number
  carbs          number
  calories       number
  authorId*      string
  averageRating  number
}
```

STATUS CODE - 400: Bad Request

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

3. RECIPECOLLECTION

3.1 GET /api/RecipeCollection

Get recipe collections.

Returns the recipe collections of the user with given `userId`.

This endpoint interacts with the `recipecollection` table.

The endpoint performs a `select *` query with a `where` clause to find the specified recipe collections.

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
`userId`	`string`	The id of the customer whose recipe collections should be returned.

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
[{
Array of object:
  userId*      string
  title*       string
  description  string
  visibility*  enum    ALLOWED:0, 1
}]
```

STATUS CODE - 404: Not Found

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

3.2 POST /api/RecipeCollection

Create a recipe collection.

Adds the given recipe collection to the database, and returns it on success. The `title` must be unique across all recipe collections for the given user; if it isn't an error code will be returned.

The customer specified by `userId` must be the authenticated user making this request.

This endpoint interacts with the `recipecollection` and `customer` tables. The `UserId` attribute on the `customer` table will be checked against the `userId`.

The endpoint will perform an `insert` command on the `recipecollection` table to add the recipe collection, and foreign key constraints will be relied upon to validate the `userId`.

REQUEST

REQUEST BODY - application/json

```
{
  userId*      string
  title*       string
  description  string
  visibility*  enum    ALLOWED:0, 1
}
```

RESPONSE

STATUS CODE - 201: Success

RESPONSE MODEL - application/json

```
{
  userId*      string
  title*       string
  description  string
  visibility*  enum    ALLOWED:0, 1
}
```

STATUS CODE - 400: Bad Request

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
```

```
    title     string
    status    integer
    detail    string
    instance  string
}
```

3.3 DELETE /api/RecipeCollection/{title}

Delete a recipe collection.

Deletes the given recipe collection in the database.

If the recipe collection does not exist in the database, a 404 status code is returned. The customer specified by `UserId` in the found recipe collection must be the authenticated user making this request.

This endpoint interacts with the `recipecollection` and `customer` tables. The `UserId` attribute on the `customer` table will be checked against the `UserId` from the `recipecollection` table.

The endpoint will perform a `select` query on the `customer` table to validate the `UserId`, and a `delete` command on the `recipecollection` table to delete the recipe collection.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*title	string	The title of the recipe collection to delete.

RESPONSE

STATUS CODE - 200: Success

4. RECIPEINCOLLECTION

4.1 GET /api/RecipeInCollection

Get a list of all recipes in a given recipe collection.

Returns a list of recipes from the database which are in the given recipe collection.

This endpoint interacts with all attributes in the `recipeincollection` table.

The endpoint performs a `select *` query, with a `where` clause included to filter for the given recipe collection.

REQUEST

REQUEST BODY - application/json

```
{
   userId*      string
   title*       string
   description  string
   visibility*  enum    ALLOWED:0, 1
}
```

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
[{
Array of object:
   recipeId*          integer
   collectionUserId*  string
   collectionTitle*   string
}]
```

4.2 POST /api/RecipeInCollection

Add a recipe to a collection.

Adds the given recipe to the given recipe collection in the database, and returns it on success. If the given recipe or recipe collection do not exist, an error code is returned.

The authenticated user making this request must be the owner of the collection.

This endpoint interacts with the `recipe`, `customer`, `recipeincollection`, and `recipecollection` tables. The keys in the payload will be checked against the rows in `recipe` and `recipeincollection`.

The endpoint will perform an `insert` command on the `recipeincollection` table to add the recipe to the collection, and

foreign key constraints will be relied upon to validate recipe collection and recipe keys.

REQUEST

REQUEST BODY - application/json

```
{
  recipeId*          integer
  collectionUserId*  string
  collectionTitle*   string
}
```

REQUEST BODY - text/json

```
{
  recipeId*          integer
  collectionUserId*  string
  collectionTitle*   string
}
```

REQUEST BODY - application/*+json

```
{
  recipeId*          integer
  collectionUserId*  string
  collectionTitle*   string
}
```

RESPONSE

STATUS CODE - 201: Success

RESPONSE MODEL - application/json

```
{
  recipeId*          integer
  collectionUserId*  string
  collectionTitle*   string
}
```

STATUS CODE - 400: Bad Request

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

4.3 DELETE /api/RecipeInCollection

Remove a recipe from a collection.

Adds the given recipe to the given recipe collection in the database, and returns it on success. If the given recipe or recipe collection do not exist, an error code is returned.

The authenticated user making this request must be the owner of the collection.

This endpoint interacts with the `recipe`, `customer`, `recipeincollection`, and `recipecollection` tables. The keys in the payload will be checked against the rows in `recipe` and `recipeincollection`.

The endpoint will perform a `delete` command on the `recipeincollection` table to remove the recipe from the collection, and foreign key constraints will be relied upon to validate recipe collection and recipe keys.

REQUEST

REQUEST BODY - application/json

```
{
   recipeId*          integer
   collectionUserId*  string
   collectionTitle*   string
}
```

REQUEST BODY - text/json

```
{
   recipeId*          integer
   collectionUserId*  string
   collectionTitle*   string
}
```

REQUEST BODY - application/*+json

```
{
   recipeId*          integer
   collectionUserId*  string
   collectionTitle*   string
}
```

RESPONSE

STATUS CODE - 200: Success

5. RECIPERATING

5.1 POST /api/RecipeRating

Create a recipe rating.

Adds the given recipe rating to the database, and returns it on success.

The customer specified by `userId` must be the authenticated user making this request. The recipe specified by `recipeId` must exist in the database.

This endpoint interacts with the `reciperating`, `recipe`, and `customer` tables. The `UserId` attribute on the `customer` table will be checked against the `userId`. The `Id` attribute on the `recipe` table will be checked against the `recipeId`.

The endpoint will perform an `insert` command on the `reciperating` table to add the recipe rating, and the foreign key constraints will be relied upon to validate the `recipeId` and `userId`.

REQUEST

REQUEST BODY - application/json

```
{
  userId*    string
  recipeId*  integer
  rating     integer  between 1 and  5
}
```

RESPONSE

STATUS CODE - 201: Success

RESPONSE MODEL - application/json

```
{
  userId*    string
  recipeId*  integer
  rating     integer  between 1 and  5
}
```

STATUS CODE - 400: Bad Request

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

---

6. WEATHERFORECAST

6.1 GET /api/WeatherForecast

Returns all WeatherForecasts.

Retrieves all items and all attributes from the `weatherforecasts` table.

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
[{
Array of object:
  id           integer
  date*        string
  temperatureC integer
  summary      string   max:20 chars
  temperatureF integer  READ-ONLY
  humidity     number
}]
```

6.2 POST /api/WeatherForecast

Adds the new forecast to the database.

Leave `id` out or specify a value of 0 to have it be automatically generated.

REQUEST

REQUEST BODY - application/json

```
{
  id           integer
  date*        string
  temperatureC integer
  summary      string   max:20 chars
  temperatureF integer  READ-ONLY
  humidity     number
}
```

RESPONSE

STATUS CODE - 201: Success

RESPONSE MODEL - application/json

```
{
  id           integer
  date*        string
  temperatureC integer
  summary      string   max:20 chars
```

```
  temperatureF  integer  READ-ONLY
  humidity      number
}
```

STATUS CODE - 400: Bad Request

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

6.3 GET /api/WeatherForecast/{id}

Returns the WeatherForecast with the given id.

Retrieves the item with the given id value in the ID column from the `weatherforecasts` table.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*id	int32	The ID of the WeatherForecast to retrieve.

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
  id            integer
  date*         string
  temperatureC  integer
  summary       string   max:20 chars
  temperatureF  integer  READ-ONLY
  humidity      number
}
```

STATUS CODE - 404: Not Found

RESPONSE MODEL - application/json

```
{
```

```
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

6.4 PUT /api/WeatherForecast/{id}

Replaces the forecast in the database.

Ensure the `id` in the body matches the id of the request URL,
otherwise a 400 response will be returned.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*id	int32	The ID of the WeatherForecast to update.

REQUEST BODY - application/json

```
{
  id           integer
  date*        string
  temperatureC integer
  summary      string   max:20 chars
  temperatureF integer  READ-ONLY
  humidity     number
}
```

RESPONSE

STATUS CODE - 204: Success

STATUS CODE - 400: Bad Request

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - 404: Not Found

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - **default:** Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

6.5 DELETE /api/WeatherForecast/{id}

Removes the forecast from the database.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*id	int32	The ID of the WeatherForecast to delete.

RESPONSE

STATUS CODE - **200:** Success

STATUS CODE - **400:** Bad Request

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - **404:** Not Found

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  type      string
  title     string
  status    integer
  detail    string
  instance  string
}
```