FoodDeliveryBackend API Documentation

Last revision: 16/03/2025

Overview

FoodDeliveryBackend is a REST API designed to manage food delivery operations, including authentication, company listings, order processing, delivery regions, weather data, and more. This documentation provides details on the controllers, models, and services used in the backend system.

Configuration (appsettings.json)

Logging

- `Default` and `Microsoft.AspNetCore`: Different levels of logging. Can be set to **Debug, Information, Warning and Error**.

Allowed Hosts

- `AllowedHosts`: Used for host filtering to bind the program to specific hostnames.

Connection Strings

- `DefaultConnection`: Used to connect to different databases with different values. See https://learn.microsoft.com/en-us/ef/core/miscellaneous/connection-strings for more.

WeatherFetcher

- `CronSchedule`: A cron schedule that runs the weather fetching code.
  Set to `15 * * * *`, meaning it runs at minute 15 of every hour.

ServerSettings

- `HostAddress`: The address you wish to host your program on. Defaulted to **localhost**.
- `Port`: The port used by the program to listen in from. Defaulted to 5000

Secrets

- `AdminKey`: Stores an administrative secret key (should be changed and kept secure).

Controllers / Endpoints

1. AuthController [/api/auth]

Handles authentication-related API calls.

- **`Login(string key)`**: Validates the given key and stores it in a cookie if valid.
  If API endpoints are tested using curl, you can't use cookies but instead can add an extra header value to request that require authentication named **"X-API-Key"**. The key used is the same secret/admin key used in the **appsettings.json** file.

  Action: **POST** Endpoint: **/api/auth/login**

- **`Logout()`**: Logs out the user by deleting the authentication cookie.

  Action: **DELETE** Endpoint: **/api/auth/logout**

2. CompaniesController [/api/companies]

Handles company-related API calls.

- **`GetCompanies()`**: Retrieves the list of registered companies in the database.

  Action: **GET** Endpoint: **/api/companies**

3. DeliveryRegionController [/api/region]

Manages delivery region rules. **Authorization required.**

- **`GetRegionRuleById(int id)`**: Fetches a region rule using an integer identifier.

  Action: **GET** Endpoint: **/api/region?id=X**

- **`UpdateRegionRule(DeliveryRegionRule rule)`**: Updates a region rule based on the `DeliveryRegionRule` model.

  Action: **PUT** Endpoint: **/api/region/update**

- **`AddRegionRule(DeliveryRegionRule rule)`**: Adds a new region rule.

  Action: **POST** Endpoint: **/api/region/add**

- **`RemoveRegionRule(int id)`**: Deletes a region rule by ID.

  Action: **DELETE** Endpoint: **/api/region/remove?id=X**

4. OrdersController [/api/orders]

Manages order processing.

- **`CalculateOrderCost(OrderDTO order)`**: Calculates the delivery cost based on user input.

  Action: **POST** Endpoint: **/api/orders/calculate**

5. WeatherController [/api/weather]

Handles weather-related API calls. **Authorization required.**

- **`GetLatestObservations()`**: Retrieves the latest weather observations from the database.

  Action: **GET** Endpoint: **/api/weather/latest**

- **`GetObservationsByTime(int unixTime)`**: Fetches the closest weather observation based on a given Unix timestamp.

  Action: **GET** Endpoint: **/api/weather?time=X**

Data Models

1. Company

Represents a company in the system.

- `Id`: Primary key.
- `Name`: Company name.
- `Description`: Brief description of the company.

JSON Schema:

```
[
  {
    "id": 0,
    "name": "string",
    "description": "string"
  }
]
```

2. DeliveryRegionRule

Represents delivery cost rules based on regions.

- `Id`: Primary key.
- `RegionName`: Name of the region.
- `BaseCarCost`, `BaseBikeCost`, `BaseScooterCost`: Base delivery costs for different transportation methods.
- `SnowyWeatherCost`, `RainyWeatherCost`, `MaxLowTemperatureCost`, `MinLowTemperatureCost`, `HighWindsCost`: Additional costs based on weather conditions.

JSON Schema:

```
{
  "id": 0,
  "regionName": "string",
  "baseCarCost": 0,
  "baseBikeCost": 0,
  "baseScooterCost": 0,
  "snowyWeatherCost": 0,
  "rainyWeatherCost": 0,
  "maxLowTemperatureCost": 0,
  "minLowTemperatureCost": 0,
  "highWindsCost": 0
}
```

3. OrderDTO (Data Transfer Object for Orders)

Represents a simplified order containing only values a user can request.

- `CompanyId`: Identifier for the company related to the order.
- `DestinationRegionId`: Identifier for the applicable delivery region rule.
- `DeliveryMethod`: Specifies the mode of transport.
- `DeliveryTimePeriod`: Specifies the time period of delivery.

JSON Schema:

```json
{
  "companyId": 0,
  "destinationRegionId": 0,
  "deliveryMethod": 0,
  "deliveryTimePeriod": 0
}
```

4. Order

Represents an order in the system. Currenly left unused in any requests.

- `Id`: Primary key.
- `CompanyId`: Identifier for the company related to the order.
- `DeliveryMethod`: Mode of transport.
- `DeliveryRuleId`: Identifier for applicable delivery rules.
- `TotalDeliveryCost`: Final cost of the delivery.

JSON Schema:

```json
{
  {
    "id": 0,
    "companyId": 0,
    "company": {
      "id": 0,
      "name": "string",
      "description": "string"
   },
    "deliveryMethod": 0,
    "deliveryRule": {
      "id": 0,
      "regionName": "string",
      "baseCarCost": 0,
      "baseBikeCost": 0,
      "baseScooterCost": 0,
      "snowyWeatherCost": 0,
      "rainyWeatherCost": 0,
      "maxLowTemperatureCost": 0,
      "minLowTemperatureCost": 0,
      "highWindsCost": 0
    },
    "deliveryRuleId": 0,
    "totalDeliveryCost": 0
  }
}
```

5. WeatherObservation

Represents weather observations stored in the database.

- `Id`: Primary key.
- `StationName`: Name and location of the weather station.
- `WmoCode`: Station code.
- `AirTemperature`: Temperature reading.
- `WindSpeed`: Wind speed measurement.
- `WeatherPhenomenon`: Description of current weather conditions.
- `WeatherTimestampId`: Reference to the corresponding timestamp.

JSON Schema:

```json
[
  {
    "id": 0,
    "stationName": "string",
    "wmoCode": "string",
    "airTemperature": 0,
    "windSpeed": 0,
    "weatherPhenomenon": "string",
    "weatherTimestampId": 0,
    "weatherTimestamp": {
      "id": 0,
      "observationTime": 0
    }
  }
]
```

6. WeatherTimestamp

Stores timestamps for weather observations.

- `Id`: Primary key.
- `ObservationTime`: Time of the last observation.

JSON Schema:

```json
{
  "id": 0,
  "observationTime": 0
}
```

Services & Helpers

1. SecureApiAttribute

An attribute enforcing authentication via cookies.

- **`OnAuthorization(AuthorizationFilterContext context)`**: Validates the authentication cookie containing the secret key.

2. ServiceLocator

A utility class for fetching services.

- **`Provider`**: Stores the service provider initialized at startup.
- **`GetService<T>()`**: Retrieves a requested service by type.

3. WeatherCronJob

A scheduled job to fetch weather data.

- **`ExecuteAsync(CancellationToken token)`**: Runs a loop that checks every 5 seconds if weather data should be fetched.

4. WeatherDataFetcher

Fetches weather data from a fixed XML source.

- **`FetchAndStoreWeatherDataAsync()`**: Fetches weather data and stores it in the database.

Database Context (AppDbContext)

Manages database operations for the application.

- **Entities**:
  - `Companies`
  - `WeatherObservations`
  - `WeatherTimestamps`
  - `DeliveryRegionRules`
- **Methods**:
  - `OnModelCreating(ModelBuilder modelBuilder)`: Configures the database with initial values and entity mappings.

Enumerations

DeliveryMethod

Enumerates the available delivery transportation modes:

- `Car`: A motor vehicle. Integer value of 0.
- `Bike`: A bicycle. Integer value of 1.
- `Scooter`: A motorbike. Integer value of 2.
