BiteBunny Application Documentation

1. General Information

The "BiteBunny" application is a web-based platform designed to connect customers, restaurants, delivery personnel and administrators in a unified, secure and intuitive food ordering and delivery system. The main goal is to automate the process from food selection to delivery while ensuring excellent user and administrative support.

Technology Stack:

- Back-end: Java(Spring Boot)
- Front-end: Angular
- Database: MySQL
- Containers: Docker, Docker Compose

Project Startup:

- Full startup: `docker-compose up --build`
- Local frontend: `npm install && ng serve`

2. User Roles and Features

2.1. Customer

The customer is the central role in the application. They have full access to restaurants, menus and orders.

Key Features:

- Registration and login (including Google/Facebook)
- View and favorite restaurants
- Filter by categories and search by name
- Browse menus and items (description, size, price)

- Order and payment (cash on delivery or card)
- Real-time order tracking
- Profile and order history
- Ratings and reviews for food and restaurants

2.2. Restaurant Employee

Manages the content and operations of a specific restaurant.

Features:

- Create and edit restaurant profile
- Manage menus and items
- Accept or reject orders
- Notify couriers of ready orders
- Access reports and statistics

2.3. Driver

Responsible for logistics and final delivery of orders.

Features:

- Accept orders (first-come-first-serve)
- Manage active and completed deliveries
- Update order status (in-progress / delivered)
- Calculate bonuses for revenue milestones
- View order data (address, customer, restaurant)

2.4. Admin

Has full rights over the system and all its modules.

Features:

- Manage all restaurants and users
- Create and moderate content
- Change statuses and permissions
- Generate reports and statistics
- Maintain infrastructure

3. Security and Access

- JWT-based authentication for sensitive API requests
- Role-based access control (RBAC)
- Password encryption and brute-force protection
- OAuth integration (Google / Facebook)

4. Support and Scalability

- Microservice architecture via modular controllers
- Docker containers for easy scaling
- RESTful API ready for mobile integration
- Expandable with new roles (e.g. operator, marketing)

API Documentation

Technological Architecture:

- Back-end: Java(Spring Boot)
- Front-end: Angular
- Database: MySQL
- Containerization: Docker (startup via docker-compose)
    - Full startup: `docker-compose up –build`
    - Frontend only: `npm install && ng serve`

API Controllers

AdminFoodController

Base URL: /api/admin/food

Method	URL	Description	Headers / Body	Responce
POST	/	Create food item	Authorization, CreateFoodRequest	201 Created
PUT	/{foodId}	Update food availability	Authorization	200 OK
DELETE	/{foodId}	Delete food item	Authorization	200 OK

AdminRestaurantController

Base URL: /api/admin/restaurants

Method	URL	Description	Headers / Body	Responce
POST	/	Create a restaurant	Authorization, CreateRestaurantRequest	201 Created

PUT	/{id}	Update a restaurant	Authorization, CreateRestaurantRequest	200 OK
PUT	/{id}/status	Change restaurant status	Authorization	200 OK
DELETE	/{id}	Delete a restaurant	Authorization	200 OK

AuthController

Base URL: /auth

Method	URL	Description	Request Body / Headers	Responce
POST	/register	Register new user	RegisterRequest	201 Created / 400 / 500
POST	/login	User login	LoginRequest	200 OK / 401 / 500
POST	/refresh	Refresh JWT	RefreshTokenRequest	200 OK / 401

CartController

Base URL: /api/cart

Method	URL	Description	Headers / Body	Responce
GET	/	Get user's cart	Authorization	200 OK / 404
POST	/items	Add item to cart	Authorization, CartItemRequest	200 OK
PUT	/items/{itemId}	Update item quantity	Authorization, CartItemRequest	200 OK
DELETE	/items/{itemId}	Remove item from cart	Authorization	200 OK
DELETE	/	Clear cart	Authorization	204 No Content

FoodController

Base URL: /api/food

Method	URL	Description	Headers / Params	Responce
GET	/search	Search food by name	Authorization, foodName=...	200 OK
GET	/menu/{menuId}	Get food by menu and category	Authorization, foodCategory	200 OK

HomeController

Method	URL	Description	Headers / Params	Responce
GET	/api/home	Home page data	—	200 OK

MenuController

Base URL: /api/menus

Method	URL	Description	Headers / Body	Responce
GET	/restaurant/{restaurantId}	Get restaurant menu		200 OK
POST	/restaurant/{restaurantId}	Create restaurant menu	Menu	200 OK
PUT	/{menuId}	Update menu	Menu	200 OK
DELETE	/{menuId}	Delete menu		200 OK
POST	/{menuId}/foods/{foodId}	Add food to menu		200 OK
DELETE	/{menuId}/foods/{foodId}	Remove food from menu		200 OK

OrderController

Base URL: /api/orders

Method	URL	Description	Headers / Body	Responce
POST	/	Create new order	Authorization, CreateOrderRequest	201 Created / 500
GET	/user	Get current user orders	Authorization, page, size	200 OK / 500
GET	/{orderId}	Get specific order	Authorization	200 OK / 404
PUT	/{orderId}/status	Update order status	Authorization, status param	200 OK / 400

PaymentController

Base URL: /api/payments

Method	URL	Description	Headers / Body	Responce
POST	/	Process payment	Authorization, PaymentRequest	201 Created / 400
GET	/{paymentId}	Get payment by ID	Authorization	200 OK / 404
GET	/order/{orderId}	Get payment by order ID	Authorization	200 OK / 404
POST	/refund/{paymentId}	Issue refund	Authorization	200 OK / 400

RestaurantController

Base URL: /api/restaurants

Method	URL	Description	Headers / Params	Responce
GET	/	Get all restaurants	Authorization	200 OK
GET	/{id}	Get restaurant by ID	Authorization	200 OK / 404

GET	/search?keyword=...	Search restaurants by keyword	Authorization, keyword	200 OK
PUT	/{id}/add-favourites	Add restaurant to favorites	Authorization, id	200 OK

ReviewController

Base URL: /api/reviews

Method	URL	Description	Headers / Body	Responce
POST	/	Create new review	Authorization, CreateReviewRequest	201 Created
GET	/restaurant/{restaurantId}	Get restaurant reviews	Query: page, size	200 OK
GET	/food/{foodId}	Get food reviews	Query: page, size	200 OK
PUT	/{reviewId}	Update review	Authorization, JSON тяло: CreateReviewRequest	200 OK
DELETE	/{reviewId}	Delete review	Authorization	204 No Content

TestController

Метод	URL	Description	Headers / Params	Responce
GET	/api/test	Check if app is running	—	200 OK ("Application is running!")

UserController

Base URL: /api/users

Method	URL	Description	Headers / Params	Responce
GET	/profile	Get user profile via JWT token	Authorization: Bearer <token>	200 OK / 401 Unauthorized

UserProfileController

Base URL: /api/user/profile

Method	URL	Description	Headers / Body	Responce
GET	/	Get current user profile	Authorization: Bearer <token>	200 OK / 404 Not Found
PUT	/	Update user profile	Authorization: Bearer <token>, JSON: UpdateProfileRequest	200 OK / 400 Bad Request
PUT	/password	Change password	Authorization: Bearer <token>, JSON: UpdatePasswordRequest	200 OK / 400 Bad Request
DELETE	/	Delete user profile	Authorization: Bearer <token>	204 No Content / 500

Security

All endpoints except /auth/* require a valid JWT in the Authorization header:

Authorization: Bearer <jwt>

The JWT (JSON Web Token) is issued upon login and used to authenticate requests to protected resources. When the accessToken expires, it can be refreshed via:

POST /auth/refresh