

The background is a dark navy blue. On the left side, there are two overlapping geometric shapes: a blue parallelogram and a light green parallelogram, both slanted downwards from left to right. The text 'ONE STOP.' is written in a white, casual script font and is underlined with a solid white horizontal line.

ONE STOP.



# INTRODUCTION

*One Stop is an online restaurant food ordering system for a particular restaurant. This project works on a dual module where through the user module one can order food items online and through the admin module one can manage all performance related to order.*



# PROJECT OVERVIEW

*In this project we will be moving forward with the principle of RESTful API where the project will completely operate on Swagger API Documentation where Customers and Admin can perform their task.*



# FEATURES

- One stop let its admin create a food menu for a particular restaurant and also let them to alter/ update it accordingly.
- One Stop will also let its admin create a database for the location they will be delivering to and also let them to alter/update it.
- One Stop will have condition that will let users to order food only during a particular time window.
- Users will be able to create multiple orders at a single time by just providing name, contact number, address, pin code and dish id of the food item they want to order.
- Once the order is created user will be able to see their order status, amount, and expected time of arrival.
- Further admin can update the order-status through order-status controller.
- Admin and user can access the project through Swagger API.



# One Stop Terms And Conditions

1. One Stop will give service from 10:00 am to 9:00 pm (21:00) after that it will not provide any service.
2. After ordering Food You can Cancel food within 3 minutes of time after that it won't be cancel.
3. After Ordering food It will take minimum of 25 min to prepare food as it will reflect in expected time.
4. After ordering they can see there order-status which will be updated by admin.



# PROJECT REQUIREMENTS

## pom.xml

SPRING DATA JPA– It allows us to access and persist data between Java object/ class and relational database. JPA follows Object-Relational Mapping (ORM). It is a set of interfaces

SPRING WEB–The spring-web dependency contains common web specific utilities for both Servlet and Portlet environments.

MYSQL CONNECTOR–The mysql-connector-java dependency is for the MySQL database driver.

SPRING VALIDATION– It is a standard validation specification that allows us to easily validate domain objects by using a set of constraints declared in the form of annotations.

SPRING TEST–It pulls all the dependencies related to test.

SPRINGDOC-OPENAPI-UI–This dependency is used for swagger documentation for performing all Restful Services..

LOMBOK–It is used to add all the properties of getters and setters.

SPRING DEVTOOLS–It is used to restart the server automatically after saving the changes.



# Software Requirements

#	Item	Specification/Version
1	Eclipse IDE	Oxygen/2022-06
2	Maven	3.x
3	SpringTools 4 suite(SpringBoot)	4.5.3
5	JDK	11
7	MYSQL	MySQL 8.0 command line



# CONFIGURATION

## Application.properties

### 1. Database Connection

spring.datasource.name=test

spring.datasource.url=jdbc:mysql://localhost:3306/pro

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=update

### 2. Swagger Path

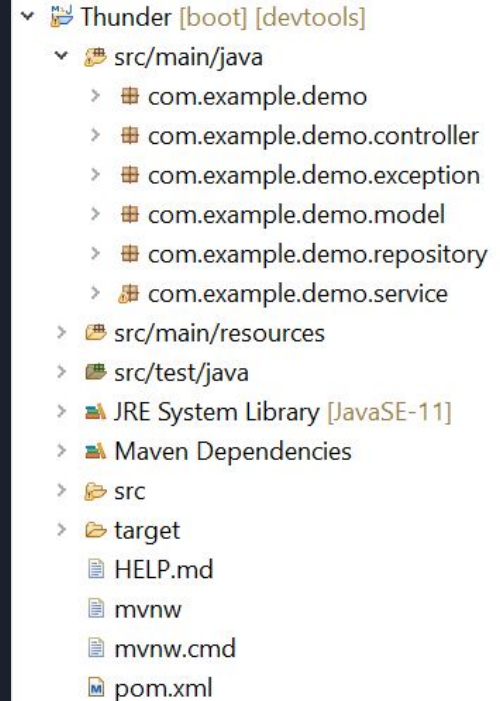
springdoc.api-docs.path=/javainuse-openapi

### 3. server.port=9090 (for changing port number)



# Project Structure

1. **Com.example.demo** — It is used to start spring boot application.
2. **Com.example.demo.controller** —It is responsible for processing incoming REST API requests and sending requests too.
3. **Com.example.demo.exception** —This package is used to handle various exceptions which comes during execution of project.
4. **Com.example.demo.model** — The model package is used to represent the java object carrying data and making the structure of tables in database.
5. **com.example.demo.repository**—This package is to extends the properties of JPArepository.In this package we used interfaces.
6. **Com.example.demo.service** —A service package is used to provide various services and wrapping the provided services in one package.



# Com.example.demo.controller

**AdminFoodmenuController** : mainly handled by admin for adding new food item in menu list and update the existing food item from menu list.

AdminOrderstatusController handled by admin for updating the status of the particular order.

AdminPincodeController mainly handled by admin for adding new pin for providing services and deleting existing pin details and user get the pindetails where delivery is available or not through this.

GlobalExceptionHandler : This part mainly use for handling the global exception part.

UsersOrderController mainly handle the order creation part where user can create order with auto generated order id.

```
▼ com.example.demo.controller
  > AdminFoodmenuController.java
  > AdminOrderstatusController.java
  > AdminPincodeController.java
  > GlobalExceptionHandler.java
  > UsersOrderController.java
```



## *com.example.demo.exception*

1. **ProductNotFoundException** : This is for handling Food not found in the food records.
2. **PinCodeNotFoundException** : This is for handling PinCode not found in the pincode records.
3. **InvalidPincodeFoundException** : This is for handling if the PinCode is not valid before saving the data into records.
4. **OpeningClosingoneStopException** : This is for handling whether a shop is open or closed at that time for ordering Food.
5. **OrderNotFoundException** : This is for handling whether an order is present or not after ordering food.

```
com.example.demo.exception
  CancelOrderException.java
  OpeningClosingoneStopException.java
  OrderNotFoundException.java
  PinCodeNotFoundException.java
  productNotFoundException.java
```

# GlobalExceptionHandler.java

It is used for handling all global exception with the help of Exceptionhandler annotation.

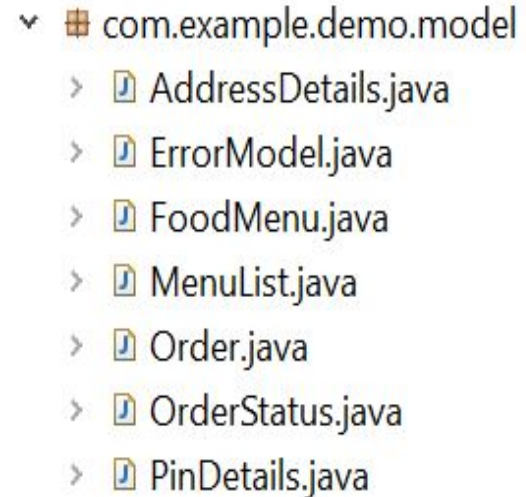
```
@RestControllerAdvice
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {
    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status, WebRequest request) {
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getAllErrors().forEach((error) ->{
            String fieldName = ((FieldError) error).getField();
            String message = error.getDefaultMessage();
            errors.put(fieldName, message);
        });

        return new ResponseEntity<Object>(errors,HttpStatus.BAD_REQUEST);
    }
    @ExceptionHandler(productNotFoundException.class)
    public ResponseEntity<ErrorModel> handleproductNotFoundException(productNotFoundException ex, WebRequest req)
    {
        ErrorModel model = new ErrorModel(new Date(), ex.getMessage(),HttpStatus.NOT_FOUND.value(),req.getDescription(false));

        return new ResponseEntity<ErrorModel>(model,HttpStatus.NOT_FOUND);
    }
    @ExceptionHandler(OpeningClosingoneStopException.class)
    public ResponseEntity<ErrorModel> handleoneStopException(OpeningClosingoneStopException ex, WebRequest req)
    {
        ErrorModel model = new ErrorModel(new Date(), ex.getMessage(),HttpStatus.NOT_FOUND.value(),req.getDescription(false));
```

# Com.example.demo.model

1. **AddressDetails.java** - This class includes all the details regarding address of the customer.
2. **ErrorModel.java**- This class includes all the details regarding exceptions used in the java class.
3. **FoodMenu.java**- This class is used to define the menu list of the restaurant.
4. **MenuList.java**- This class is used if the user wants to order multiple items at a same time in a single order.
5. **Order.java**- This model class includes all the necessary data types used for order creation.
6. **OrderStatus.java**- This model class includes all the details regarding the status of the order created by the user.
7. **PinDetails.java**- This class is used to define the details of the address where the orders are getting delivered by the restaurant.



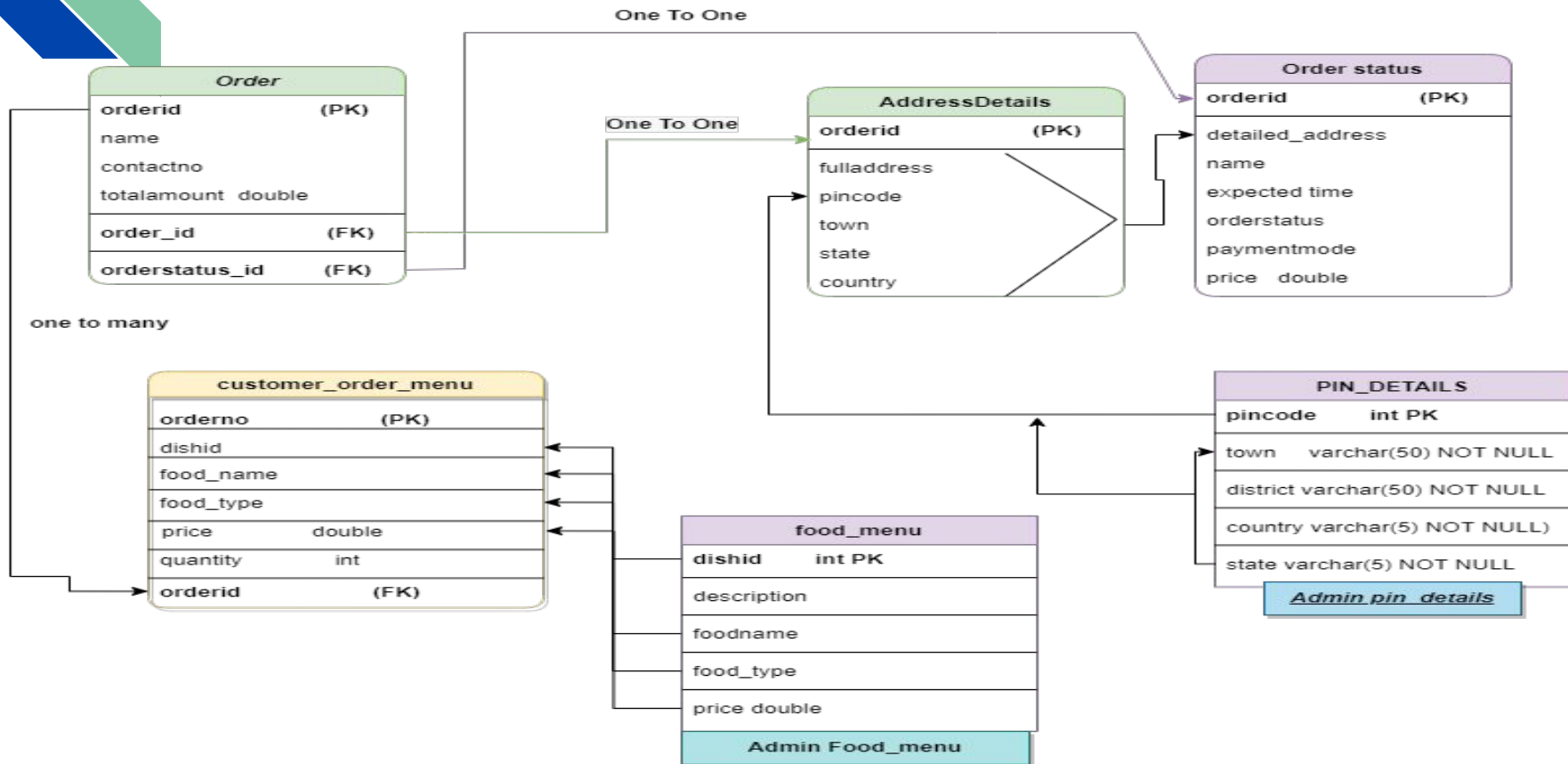
```
▼ com.example.demo.model
  > AddressDetails.java
  > ErrorModel.java
  > FoodMenu.java
  > MenuList.java
  > Order.java
  > OrderStatus.java
  > PinDetails.java
```



# Database Model Overview

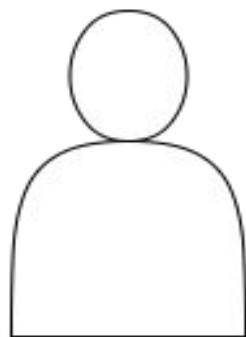
1. **Pin\_details model** : This database model consist all of the pin details where restaurant providing the delivery services. It is used by admin for updating and used by users for checking whether delivery services is available or not in their area.
2. **Food\_menu model** : This database model is consist of all the food available in restaurant. It is used by admin for adding or deleting food from menu and used by users for checking what food is available to order.
3. **Customer\_order\_menu model** : This model is used by admin to check all the orders details regarding which food ordered in how much quantity.
4. **Order\_details** : This model is also used by admin to check the order details regarding details of customer and booking time.
5. **Orderaddress\_details** : This model is used to retrieve the data of the address of users to check where order is delivered.
6. **Order\_status model** : This model is used by admins for updating the status of order whether order is delivered or not and used by users for check the status for same purposes.

# DataBase Full diagram



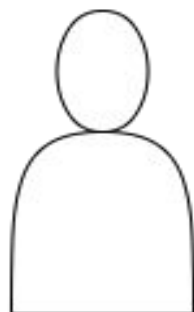
# Working-Model

**Admin**



- Get OrderDetails(GetMapping)
- Update order Status(PutMapping)

**Customer**



**SEND**

**FETCH**

**pincode**

- Create
- Update
- Delete
- Getall

**Food\_menu**

- Create
- Updatebyid
- Deletebyid
- Getall

RestFul API

- Creating Order(PostMapping).
- Cancel Order
- Check pincode by id
- getall foodmenu to order
- Show Order Status (GetMapping)



# Validation

Validation is used to for validating the format of input by user what kind of we need .if it not in the correct way it will throw error.

- A. Pin\_details model have validation part.
- B. Food\_menu model have validation part.
- C. Order model have validation part.

```
public class FoodMenu {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private int dishid;  
    @NotBlank(message="Enter Food Name Can't be empty")  
    private String foodname;  
    @NotBlank(message="Veg Or Non-Veg")  
    private String foodtype;  
    @NotBlank(message="should be in summary for the dish")  
    private String description;  
    private float price;  
}
```

```
public class PinDetails {  
    @Id  
    @NotNull  
    @Min(value=100000,message="Need Greater than 100000")  
    @Max(value=999999,message="Need less than 999999")  
    private int pincode;  
    @NotBlank(message="Enter value town")  
    private String town;  
    @NotBlank(message="Enter value District")  
    private String district;  
    @NotBlank(message="Enter value Country")  
    private String country;  
    @NotBlank(message="Enter value State")  
    private String state;  
}
```

```
public class Order {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private int orderid;  
    @NotBlank(message="Required Name")  
    private String name;  
    @NotBlank(message="Required Contact Number")  
    @Pattern(regexp="[6-9][0-9]{9}",message="Enter Valid Mobile Number")  
    private String contactno;  
    private String bookingtime;  
    private double totalamount;  
  
    // for Adding all address.....  
    //cascade = CascadeType.ALL ( PERSIST, REMOVE, REFRESH, MERGE, DETACH )  
    @OneToOne(cascade = CascadeType.ALL)  
    //order_id work as foreign key in this table and orderid work as primary  
    @JoinColumn(name="order_id",referencedColumnName = "orderid")  
    private AddressDetails address;
```



# *com.example.demo.repository*

It is used to connect the model class with the help of JPA repository and it's an interface which extends jpa repository.

**AddressDetailRepo** : connect with the address details model.

**FoodRepo** : connect with the food menu model.

**MenuListRepo** : connect with the customer order menu model.

**OrderRepo** : connect with order detail model.

**OrderStatusRepo** : connect with model where status of the order are updated.

**PinDetailsRepo** : connect with pin detail model where we find all the details of specific pincode.

```
▼ com.example.demo.repository
  > AddressDetailRepo.java
  > FoodRepo.java
  > MenuListRepo.java
  > OrderRepo.java
  > OrderStatusRepo.java
  > PinDetailsRepo.java
```



# *com.example.demo.service*

A service package is used to provide various services and wrapping the provided services in one package.

FoodService : class is used to provide the services mentioned in the FoodMenu model.


MenuService class is user to provide the services that are provided in the MenuList class of model package.

OrderAddressService : class is used here to provide various services that is mentioned in AddressDetails class of the model package.

OrderService class : is providing the services of Order class in model package.

OrderStatusService class is made here for providing the services of OrderStatus class made in Model package.

PinService class is used in service package to apply all the services in details which is mentioned in PinDetails class of model package.



```
▼ com.example.demo.service
  > FoodService.java
  > MenuService.java
  > OrderAddressService.java
  > OrderService.java
  > OrderStatusService.java
  > PinService.java
```