# TUCKOS- TUCKSHOP MANAGEMENT SYSTEM

**-Lakshya Kapoor (IMT2023509)**

**-Parv Sharma (IMT2023514)**

**-Jayesh Pandit (IMT2023111)**

**-Paritosh Tiwari (IMT2023123)**

**-Shivam Pandya (IMT2023091)**

**-T Shantanu (IMT2023056)**

GITHUB REPO LINK: https://github.com/generator45/Tuckos

## Overview

Tuckos is a web app that streamlines food ordering at the Tuck Shop, reducing crowding. It enables easy online orders, ensuring convenience, faster service, and a smoother user experience.

## Goals:

1) Simplify Food Ordering: Digitize the Tuck Shop ordering process using a user-friendly web app built with Java and C++ for seamless functionality.
2) Enhance Efficiency: Utilize Object-Oriented Programming principles to design robust, modular, and scalable features for order management.

3) Enable Real-Time Communication: Implement REST APIs with Spring Boot for smooth back and front-end interaction, ensuring quick data exchange.
4) Ensure Accessibility: Host the app on Postman to facilitate easy API testing and integration for consistent performance.
5) Promote Modern Solutions: Leverage advanced technologies like Spring Boot to create a secure, reliable, responsive ordering platform.
6) Provide Scalability: Use OOP principles to ensure the app can handle increasing user demands without performance issues.
7) Optimize Workflow: Streamline kitchen operations with real-time order notifications, reducing preparation delays and improving service speed.

## DATABASE SCHEMA:

**Student Table**

| Column | Data Type | Constraints |
|---|---|---|
| rollNo | VARCHAR(15) | `PRIMARY KEY` |
| password | VARCHAR(255) | |
| studentName | VARCHAR(255) | |

**Item Table**

| Column | Data Type | Constraints |
|---|---|---|
| itemId | INT | `PRIMARY KEY` |
| itemName | VARCHAR(255) | |
| price | INT | |
| itemDesc | VARCHAR(255) | |
| imgUrl | VARCHAR(255) | |

3

.

### Inventory Table

| Column | Data Type | Constraints |
|---|---|---|
| itemId | INT | `PRIMARY KEY`, `FOREIGN KEY` REFERENCES **Item(itemId)** |
| quantity | INT | |

### OrderHistory Table

| Column | Data Type | Constraints |
|---|---|---|
| orderId | INT | `PRIMARY KEY` |
| rollNo | VARCHAR(15) | `FOREIGN KEY` REFERENCES **Student(rollNo)** |
| order_date | DATE | |
| order_status | BOOLEAN | |

### OrderItem Table

| Column | Data Type | Constraints |
|---|---|---|
| orderId | INT | `PRIMARY KEY` (orderId, itemId), `FOREIGN KEY` REFERENCES **OrderHistory(orderId)** |
| itemId | INT | `PRIMARY KEY` (orderId, itemId), `FOREIGN KEY` REFERENCES **Item(itemId)** |
| quantity | INT | |
| price | INT | |

Resources:

This project relies on several key resources to ensure successful implementation and understanding of concepts:

1. Rest API in Spring Boot: https://medium.com/javajams/creating-a-rest-api-in-spring-boot-68ce785f652f

2. Java Persistence API (JPA): https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html

3. SQL Tutorial: *https://www.w3schools.com/sql/default.asp.*

## Project Setup:

Clone the repo link :
https://github.com/generator45/Tuckos

This project requires setting up a development environment with the following tools and configurations:

1. Install maven for build automation and dependency management.

2. Install postgreSQL for database connection.

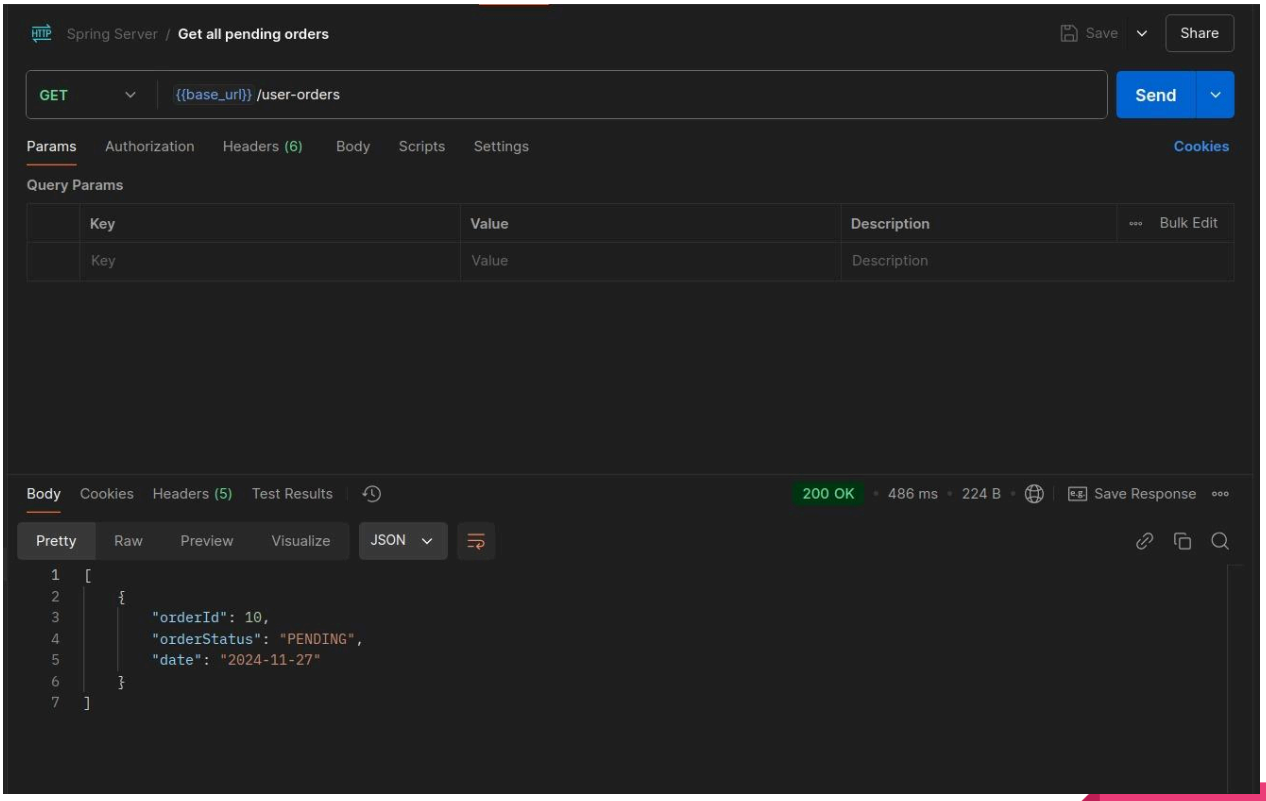
3. Setup application.properties file by adding the variables:

    - **`spring.datasource.url`**: postgres db connection url
    - **`spring.datasource.username`**: postgres username
    - **`spring.datasource.password`**: password for the user

4. To run project locally:

    mvn spring-boot:run -Dspring-boot.run.fork=true

## Now you can test this using PostMan.



## Future integrations:

- Authentication:

  We plan to integrate user authentication in our maven project using “Pepper Method” to hash password. The PEPPER hashing method enhances password security by adding a secret "pepper" value to the password before hashing. Unlike a salt, the pepper is a fixed, secret value stored separately from the database. I implemented this method in my project to strengthen authentication and protect against brute-force attacks.

- JWT tokens:

  We are also planning to add JWT tokens in our project to track login session of a user. **JSON Web Tokens (JWT)** are compact, URL-safe tokens used for secure information exchange between parties. A JWT consists of three parts:

1. **Header**: Specifies the token type (JWT) and signing algorithm (e.g., HS256).
2. **Payload**: Contains claims or data, such as user ID or roles, encoded in Base64.
3. **Signature**: A hashed combination of the header, payload, and a secret key to ensure authenticity and integrity.

JWTs are widely used for authentication and authorization. Once authenticated, a server issues a JWT to the client, which is sent with subsequent requests to verify the user without rechecking credentials.