Producer Consumer

Table of contents

Introduction

This project is a simulation of a production line where machines process products in a queue-based system. It leverages several design patterns to manage the system's complexity, including **Observer**, **Concurrency**, and **Snapshot** patterns. The core functionality allows machines to process products concurrently while notifying other components about state changes. The system also supports saving and restoring its state at any given point, ensuring fault tolerance. Additionally, WebSocket integration enables real-time communication with the frontend, providing an interactive and dynamic user experience. This project is designed to model an efficient and scalable production environment using Java and Spring Boot.

Technology Stack

- Frontend: React , Konva
- Backend: Spring Boot
- Real-Time Communication: WebSocket

- How to Run the Application

Prerequisites

Before starting, ensure the following tools are installed and configured:

1. **Backend Requirements:**
   - Java Development Kit (JDK) 11 or later
   - Maven
2. **Frontend Requirements:**
   - Node.js (version 16.x or later)
   - npm (comes with Node.js)

1. **Clone the Repository**

1. Open a terminal or command prompt.
2. Clone the repository:
   `git clone https://github.com/abdonaware/Producer-Consumer-Simulation`
3. Navigate to the project folder:
   `cd Producer-Consumer-Simulation`

2. **Start Backend Server(Spring Boot)**
   1. Navigate to the backend directory:
      `cd backend`
   2. Run java application from run button

3. **Set Up the FrontEnd (React)**
   1. Navigate to the frontend directory:
      `cd frontend`
   2. Run frontend server
      `npm install`
      `npm start`

- Used Design Patterns

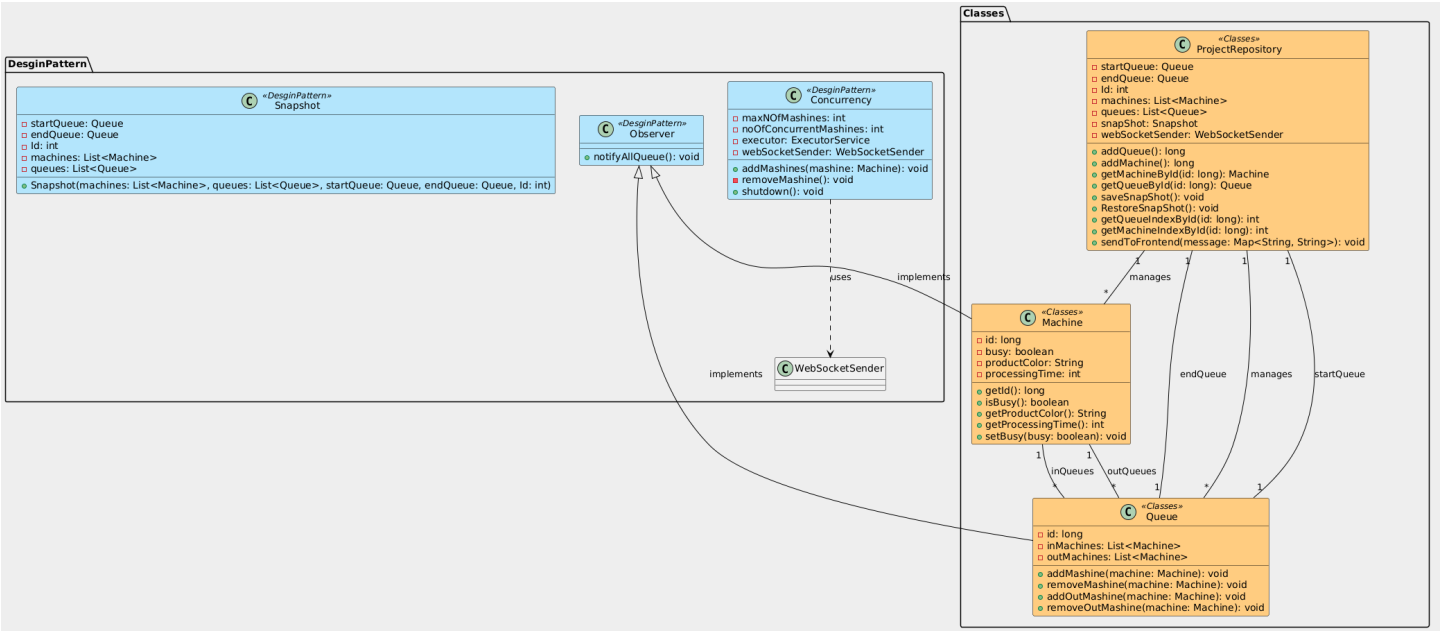
1. **Singleton Pattern**: The Singleton pattern prevents the existence of multiple objects in memory of objects that must only be created once in the system like projectRepository

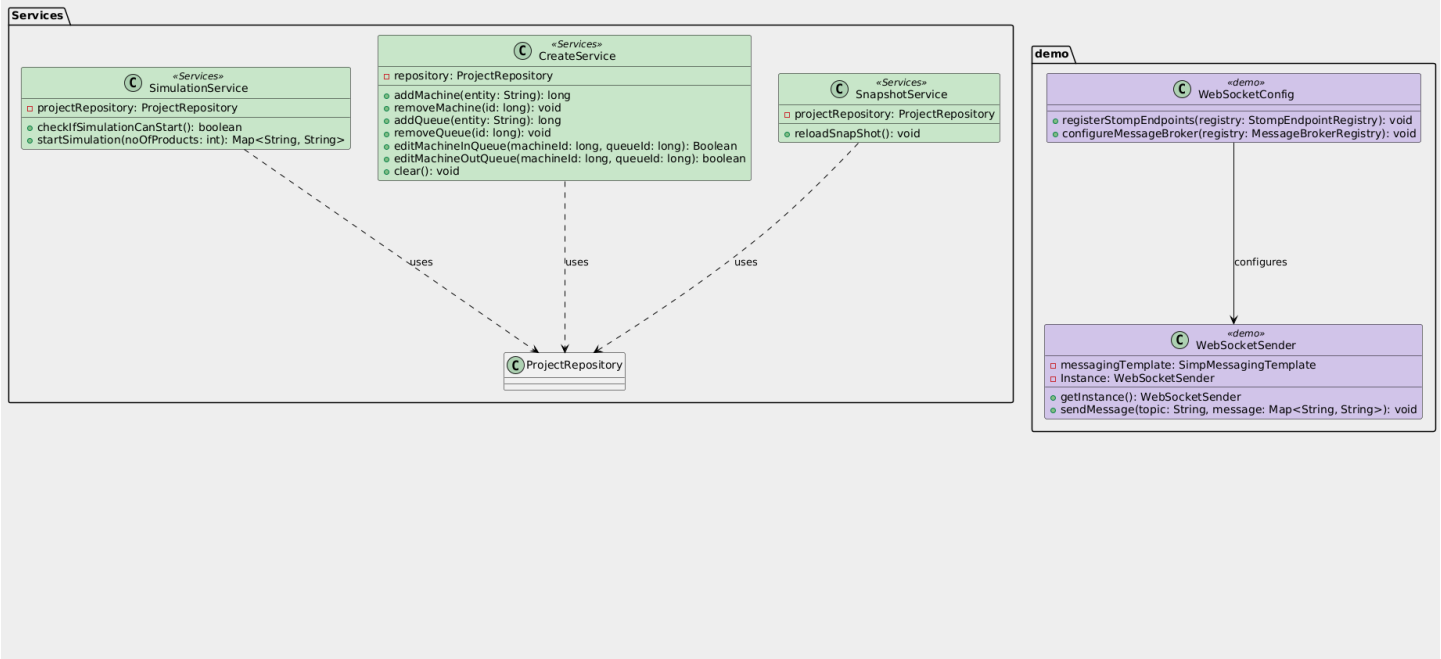
2. **Observer Pattern**: The Observer pattern allows queues to automatically be notified when a machine's state changes, ensuring real-time updates without direct interaction.

3. **Concurrency Pattern**: The Concurrency pattern ensures that multiple machines can process tasks simultaneously without causing data inconsistencies. It enhances the system's efficiency by allowing parallel task execution.

4. **SnapShot Pattern**: The Snapshot pattern saves the system's state at a start point, allowing it to be restored on restart.

- UML Design





Full UML:
https://ibb.co/bHRDSPY

- How to Use

- **Create Your System**
  - On entering the simulation you will find start and end queue fixed on the stage which represent the start and end queue.
  - Add machines and queues in the stage and connect them with arrows to initialize your system.
  - Define number of products you want to simulate.
- **Run Simulation**
  - Click on run button to start the simulation.
  - If you system valid the simulation starts, else you get error message.
- **Simulation**
  - Simulation starts from “Start” queue passing products to machines and machines pass products to next queue and so on till product rich “End” queue.
  - Every product have a color the machine witch operate on this product take this color in order to easy track the process
- **End Of Simulation**
  - When process end the simulations stops automatically then you can edit the system , number of products or restart the previous one

- Screen Shots from application