UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Marian Iosif Andrei
Grupa: 30232

Restaurant

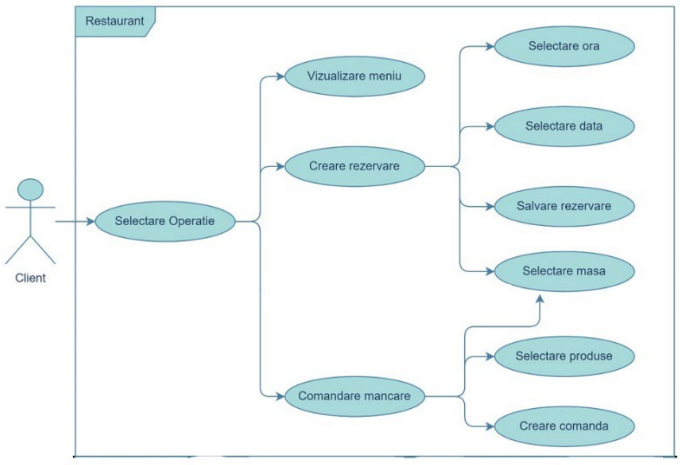
-proiect final Inginerie Software-

1. Descriere:

Platforma pe care am dezvoltat-o reprezintă o soluție comprehensivă pentru digitalizarea restaurantelor, oferind utilizatorilor o experiență fluidă și eficientă. La accesarea platformei, utilizatorii sunt întâmpinați de o pagină de start informativă, care nu doar prezintă programul de funcționare al restaurantului, ci și furnizează detalii esențiale despre locație. Navigând cu ușurință prin interfața noastră prietenoasă, clienții pot explora detaliat meniul oferit, având acces la informații despre fiecare preparat în parte. Pagina de rezervare facilitează procesul de programare a meselor, oferind opțiuni personalizate pentru diverse preferințe. De asemenea, am integrat o pagină de comandă intuitivă, permitând utilizatorilor să plaseze comenzi direct de pe platformă. Cu un design modern și funcționalități integrate, platforma noastră are ca scop îmbunătățirea interacțiunii dintre clienți și restaurante, contribuind la eficientizarea proceselor și creșterea calității experienței digitale în industria culinară.

2. Prezentare use cases:

- Vizualizarea Meniului: Utilizatorii pot explora cu ușurință paginile de meniu online pentru a vizualiza ofertele culinare și a obține informații detaliate despre preparate.
- Gestionarea Rezervărilor Simplificată: Pagina de rezervare permite clienților să programeze mese în avans, optimizând procesul pentru personal și îmbunătățind experiența clienților.
- Comenzi Online Eficiente: Utilizatorii pot plasa rapid comenzi direct de pe dispozitivele lor, reducând timpul de așteptare și asigurând o experiență eficientă.
- Vizualizarea Programului și Datelor Restaurantului: Informații esențiale despre programul de funcționare și datele restaurantului sunt disponibile pe pagina de start, asigurând o informare completă pentru clienți.



3. Prezentare pattern utilizat:

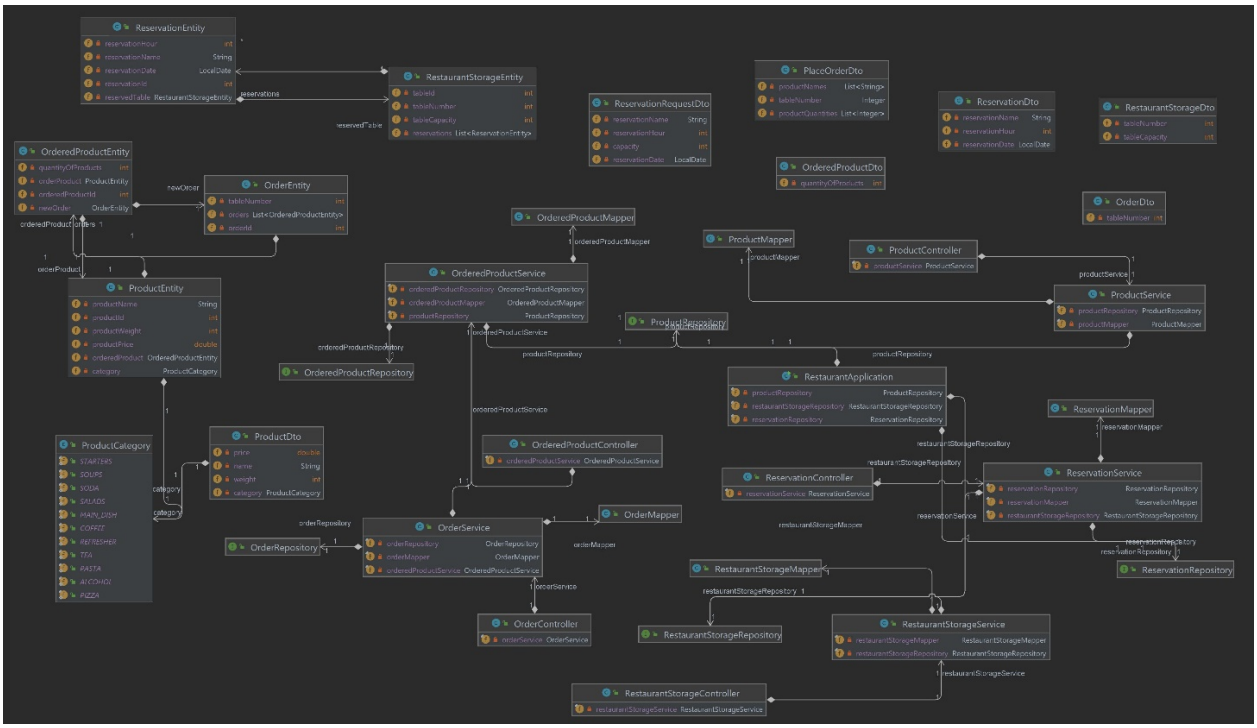
Layered Architecture Pattern este un model de proiectare care organizează aplicațiile software în straturi sau nivele distincte, fiecare având o responsabilitate specifică și comunicând doar cu straturile adiacente. Acest pattern promovează o separare clară a preocupărilor, îmbunătățind astfel modularitatea și întreținerea codului. În contextul aplicațiilor Java Spring Boot, Layered Architecture Pattern este adesea folosit pentru a structura proiectele, asigurând o dezvoltare scalabilă și ușor de înțeles.

- Entities:
  - Rol: Această parte a arhitecturii conține clasele care modelează entitățile de bază ale aplicației, de exemplu, obiectele persistente în baza de date.
  - Clasele JPA (Java Persistence API) care definesc modelele de date și relațiile dintre ele.
- DTOs (Data Transfer Objects):
  - Rol: Folosite pentru a transfera date între straturi și a evita expunerea directă a entităților de bază. DTO-urile pot fi personalizate pentru cerințele specifice ale interfeței utilizator sau ale altor componente.
  - Clase Java simple care reflectă structura datelor transferate între frontend și backend.
- Repositories:
  - Rol: Această parte gestionează interacțiunea cu baza de date. Folosește operațiuni specifice (CRUD) pentru a accesa și manipula datele stocate.
  - Interfețe sau clase care extind JpaRepository și definesc metode pentru interogări în baza de date.
- Mappers:
  - Rol: Realizează conversiile între entități și DTO-uri, asigurând transferul eficient și corect al datelor între straturi.
  - Clase care implementează logică de mapare între entități JPA și DTO-uri.
- Services:
  - Rol: Gestionarea logicii de afaceri și implementarea funcționalităților specifice aplicației. Comunică cu straturile de repoziții și mappers.
  - Servicii care conțin logica de afaceri, apelând metodele din straturile de repoziții și mappers.
- Controllers:
  - Rol: Interfața cu utilizatorii și gestionarea cererilor HTTP. Acestea consumă servicii pentru a obține date și returnează răspunsuri.
  - Clasele care sunt anotate cu @RestController și expun metodele ca puncte finale (endpoints) accesibile prin HTTP.

Prin implementarea acestui model de arhitectură în proiectele Java Spring Boot, dezvoltatorii pot menține o structură clară a codului, facilitând colaborarea și extensibilitatea pe termen lung a

aplicației. Separarea componentelor în straturi distincte permite testarea ușoară a fiecărei componente în mod izolat și oferă o abordare modulară în dezvoltarea software.

4. Diagrama de clase:



5. Prezentarea diagramei de componente:

Diagrama de componente în cadrul unui proiect, inclusiv în contextul arhitecturii stratificate descrise mai sus, oferă o perspectivă vizuală asupra relațiilor și interacțiunilor dintre diversele componente software. Această diagrăme este utilă în mai multe moduri și rezolvă diverse probleme:

1. **Vizualizarea Structurii Sistemului:**
    - **Utilitate:** Diagrama de componente oferă o vedere structurală clară a componentelor software și a relațiilor dintre ele. Aceasta facilitează înțelegerea modului în care pachetele și modulele sunt organizate în cadrul aplicației.
2. **Comunicare Eficientă a Arhitecturii:**
    - **Utilitate:** Diagrama ajută la comunicarea eficientă a arhitecturii proiectului între membrii echipei de dezvoltare și alte părți interesate, cum ar fi managerii de proiect sau clienții.
3. **Identificarea Dependențelor între Componente:**

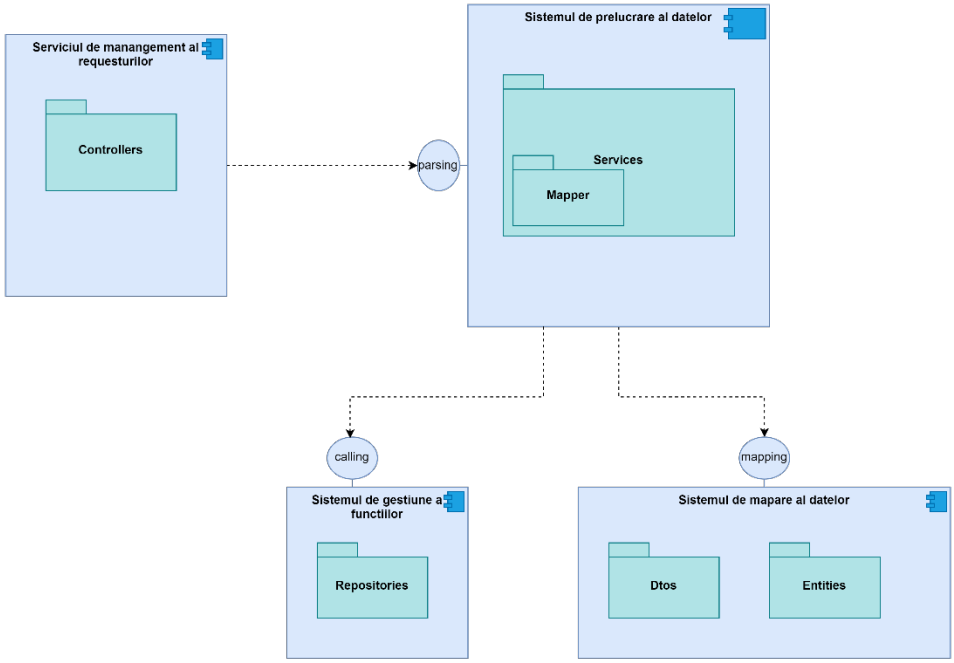
- **Utilitate:** Prin vizualizarea săgeților de dependență între componente, se pot identifica clar relațiile de interdependență dintre diferite părți ale sistemului.

4. **Planificarea și Gestionarea Implementării:**
    - **Utilitate:** Diagrama de componente poate servi ca instrument util în planificarea implementării și gestionarea procesului de dezvoltare. Se pot identifica componente care pot fi dezvoltate și testate în paralel.
5. **Izolarea și Remedierea Problemei:**
    - **Utilitate:** Atunci când apar probleme sau erori, diagrama de componente poate ajuta la localizarea rapidă a zonei afectate. Dezvoltatorii pot urmări fluxul datelor și comunicării pentru a identifica sursa problemei.
6. **Evaluarea Impactului Schimbărilor:**
    - **Utilitate:** Diagrama permite evaluarea rapidă a impactului schimbărilor propuse. Dezvoltatorii pot vizualiza cum modificările într-un anumit pachet sau componentă vor afecta restul sistemului.
7. **Ghid pentru Testare și Validare:**
    - **Utilitate:** Diagrama de componente servește ca ghid util pentru echipele de testare, ajutându-le să identifice și să prioritizeze zonele critice ale sistemului care trebuie supuse testării intensive.

Problemele pe care diagrama de componente le rezolvă includ:

- **Complexitatea Creșterii:**
    - **Soluție:** Vizualizarea relațiilor și interacțiunilor între componente ajută la gestionarea complexității creșterii proiectului. Dezvoltatorii pot să înțeleagă mai bine structura și să ia decizii informate.
- **Dependențe Neclare:**
    - **Soluție:** Diagrama identifică clar depedențele dintre componente, reducând riscul de a introduce neclarități sau confuzii în dezvoltare.
- **Coordonarea și Comunicarea Ineficientă:**
    - **Soluție:** Diagrama de componente servește ca instrument de comunicare eficient, asigurând o coordonare mai bună între membrii echipei și părțile interesate.

În concluzie, diagramele de componente sunt instrumente puternice în proiectele de dezvoltare software, oferind o viziune clară asupra structurii și relațiilor componente, facilitând astfel gestionarea, dezvoltarea și mentenanța sistemelor software.



6. Tehnologii folosite:

Platforma pe care ai descris-o beneficiază de o arhitectură modernă și utilizează diverse tehnologii pentru diferitele sale componente. Iată o scurtă descriere a tehnologiilor folosite:

- Backend cu Java Spring:
  - Limbaj de Programare: Java
  - Framework Backend: Spring Boot
  - Baza de Date: H2 Database (poate fi utilizată pentru dezvoltare și teste, ușor de configurat și integrat cu Spring Boot)
  - ORM (Object-Relational Mapping): Spring Data JPA (pentru interacțiunea cu baza de date)
  - Gestionarea Dependințelor: Gradle
- Frontend cu Angular:
  - Limbaj de Programare: TypeScript
  - Framework Frontend: Angular
  - Stilizare: SCSS
  - Gestionarea Dependințelor și Pachetelor: npm (Node Package Manager)
  - Comunicare cu Backend: HTTP/HTTPS pentru a face solicitări către API-urile backend-ului
  - Gestionarea Versiunilor și Controlul Codului:
- Autentificare și Autorizare:
  - JSON Web Tokens (JWT)

- Documentarea API-urilor:
  - OpenAPI pentru documentarea automată a API-urilor
- Mediu de Dezvoltare și Testare:
  - IDE (Integrated Development Environment): IntelliJ IDEA, Eclipse (pentru Java) și Visual Studio Code (pentru Angular)

Folosirea acestor tehnologii oferă beneficii precum scalabilitate, ușurința în dezvoltare, securitate și gestionare eficientă a dependințelor. Este important să menții toate componentele actualizate și să urmezi cele mai bune practici de dezvoltare pentru a asigura un sistem stabil și eficient.

7. Manual de utilizare:

Platforma pune la dispozitie o pagina de start cu un meniu friendly in care se pot selecta diferite cazuri de utilizare. Clientul poate vizualiza in pagina de start informatii despre restaurant si programul de lucru al acestui restaurant. Apoi in functie de ce acesta isi doreste poate selecta sa vizualizeze meniul restaurantului care va fi afisat intr-un scroll panel. Acesta poate alege o alta optiune cum ar fi crearea unei rezervari unde acesta are cateva informatii de introdus sau de a selecta din interfata si de crea rezervarea. In functe de actiunile utilizatorului v-a aparea un pop-up cu un mesaj de succes sau eroare in functe de actiunile facute de utilizator si datele introduse de acesta. Ultima optiune a utilizatorului este de a crea o comanda, interfata asemanatoare cu cea de meniu, dar unde acesta poate selecta produsele dorite, cantitatea comandata si masa la care se face comanda, dupa care acesta poate plasa comanda respectiva. De pe ficare pagina in parte se poate intoarce la pagina principala cu un buton de back.