

**Dizajni I Sistemit te Softuerit**

Viti akademik 2024 – 2025

**Emri I Projektit: Luxury Step**

Aid Aliu

Aldin Shabani

Blend Kqiku

Profesori: PhD. Cand. Greta Ahma

GitHub Link: https://github.com/AidAliu/LuxuryStep

Prill 2025

**Permbajtja**

[Luxury Step 2](#_Toc187702145)

[1 Hyrje 2](#_Toc187702146)

[2 Conceptual Model 3](#_Toc187702147)

[2.1 Conceptual Model-Presentation Layer 3](#_Toc187702147)

[2.2 Conceptual Model-Business Logic Layer 3](#_Toc187702147)

[2.3 Conceptual Model- Data Access Layer 3](#_Toc187702147)

[3.Utility Tree 5](#_Toc187702151)

[3.1 Usage (Përdorimi) 6](#_Toc187702152)

[3.2 Performance (Performancë) 6](#_Toc187702153)

[3.3 Reliability (Besueshmëri) 6](#_Toc187702154)

[3.4 Modifiability (Modifikueshmëri) 6](#_Toc187702155)

[3.5 Reusability(Ripërdorimi) 7](#_Toc187702156)

[3.6 Adaptability(Përshtatja) 7](#_Toc187702156)

[3.7 Security(Siguria) 7](#_Toc187702156)

[4. Product Overview 7](#_Toc187702157)

[4.1 Product Vision 7](#_Toc187702158)

[4.2. Key Features 7](#_Toc187702159)

[4.3 Stakeholders 8](#_Toc187702160)

[4.4 Target Market 8](#_Toc187702161)

[5. Architectural Models 10](#_Toc187702164)

[LuxuryStep Layer Overview 10](#_Toc187702165)

[5.1 Layer Responsibilities 11](#_Toc187702166)

[5.2 Presentation Layer Decomposition 11](#_Toc187702167)

[5.2.1 Presentation Layer Responsibilities 12](#_Toc187702168)

[5.2.2 Presentation Layer Design Rationale 14](#_Toc187702170)

[5.3 Bussines Logic Layer Decomposition 14](#_Toc187702171)

[5.3.1 Bussines Logic Layer Responsibilities 16](#_Toc187702172)

[5.3.2 Bussines Logic Layer Design Rationale 19](#_Toc187702174)

[5.4Data Access Layer Decomposition 20](#_Toc187702175)

[5.4.1 Data Access Layer Responsibilities 21](#_Toc187702176)

[5.4.2 Data Access Layer Specifications 22](#_Toc187702177)

[5.4.3 Data Access Layer Design Rationale 23](#_Toc187702178)

[5.5 Runtime Components 28](#_Toc187702183)

[5.6 Mapping Between Models 29](#_Toc187702184)

[5.7 Architectural Design Rationale 29](#_Toc187702185)

[6.LuxuryStep Detailed Desing Document 31](#_Toc187702186)

[6.1 Mid Level Desing Models 31](#_Toc187702187)

[6.2 Low-Level Desing 25](#_Toc187702197)

[6.2.1 User Module Responsibilities/specifications](#_Toc187702198) 26  
6.2.2 Product/Shoe Module Responsibilities/specifications 27  
6.2.3 Payment Module Responsibilities/specifications 28  
6.2.4 Review Module Responsibilities/specifications 29  
6.2.5 Wishlist Module Responsibilities/specifications 30  
6.2.6 Order Module Responsibilities/specifications 32

[6.3 Mapping Beetwen Models 44](#_Toc187702199)

[6.4 Detailed Desing Rationale 45](#_Toc187702200)

[7 Activity Diagrams 46](#_Toc187702201)

[7.1.1 User Authentication 46](#_Toc187702203)

[7.1.2 Client shoe-purchase process 48](#_Toc187702204)

[7.2 Sequence Diagrams 51](#_Toc187702205)

[7.2.1 User Authentication 51](#_Toc187702206)

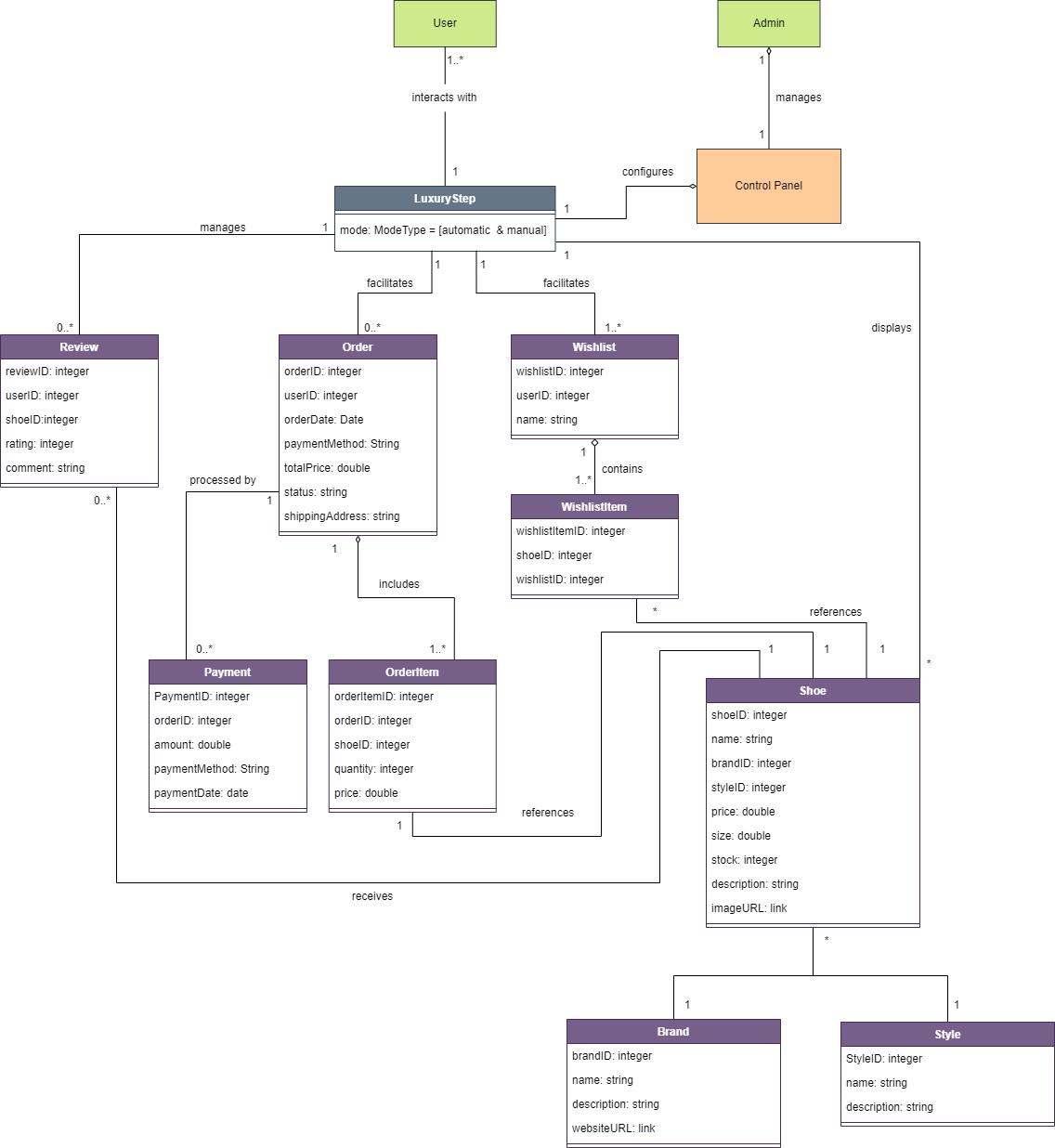
[7.2.2 Taxi Reservation 52](#_Toc187702207)

1. **Hyrje**

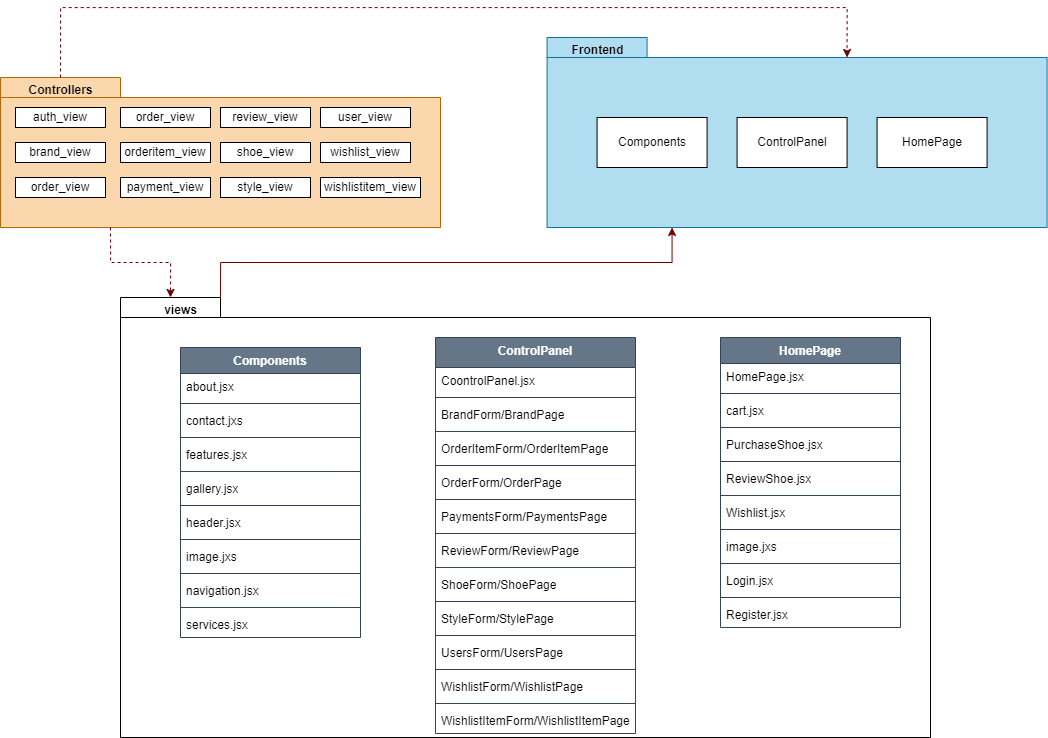
**LuxuryStep** është një platformë inovative që kombinon fuqinë e teknologjisë me pasionin për këpucët luksoze, duke krijuar një hapësirë ku përdoruesit mund të eksplorojnë, rishikojnë dhe blejnë produkte me cilësi të lartë. E ndërtuar me **Django** për backend dhe **React** për frontend, **LuxuryStep** synon të ofrojë një eksperiencë të shpejtë, të sigurt dhe të personalizuar për të gjithë adhuruesit e modës dhe koleksionistët e këpucëve. Platforma shërben si një vendtakim për entuziastët e këpucëve, duke ofruar mundësinë për të ndarë mendimet e tyre për marka dhe modele të ndryshme, si dhe për të bërë blerje të sigurta nga shitës të verifikuar.

**LuxuryStep** është konceptuar si një ekosistem i plotë për komunitetin e këpucëve luksoze, duke pasur synime të qarta si:

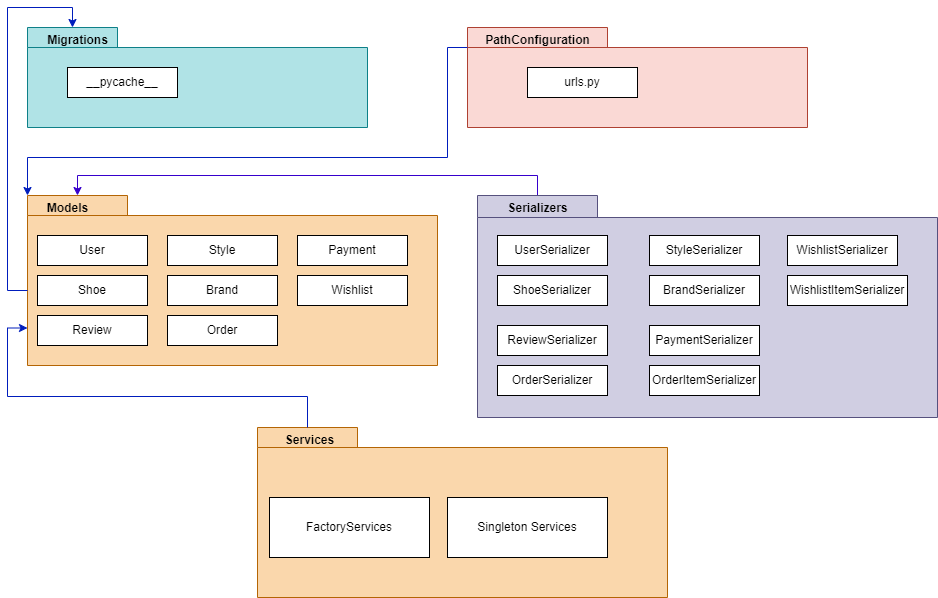
1. **Lehtësimi i Zgjedhjes për Blerësit** – Përdoruesit mund të lexojnë rishikime autentike dhe të bazuara në eksperiencën e konsumatorëve të tjerë para se të marrin një vendim për blerje.
2. **Rritja e Besimit përmes Transparencës** – Çdo produkt ka një vlerësim të detajuar dhe rishikime të verifikuara për të garantuar autenticitetin dhe cilësinë.
3. **Një Hapësirë për Adhuruesit e Këpucëve** – LuxuryStep krijon një komunitet ku koleksionistët dhe të apasionuarit pas këpucëve mund të ndajnë mendimet e tyre.
4. **Menaxhimi i Preferencave Personale** – Përdoruesit mund të krijojnë një listë dëshirash me këpucët që pëlqejnë dhe të ndjekin uljet e çmimeve.
5. **Sigurimi i Transaksioneve të Besueshme** – Platforma mundëson blerjen dhe shitjen e këpucëve përmes një sistemi të sigurt pagese dhe verifikimi.
6. **Conceptual Model**

****

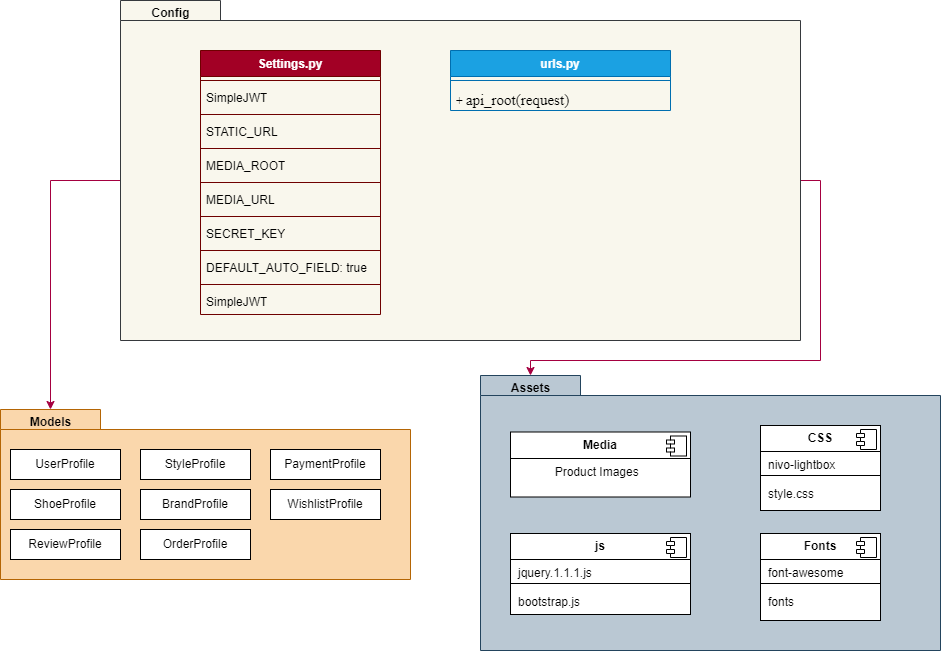
* 1. **Conceptual Model-Presentation Layer**

****

* 1. **Conceptual Model-Business Logic Layer**

****

* 1. **Conceptual Model- Data Access Layer**

****

1. **Utility Tree**

**3.1 Usage**

* **Shkruaj Rishikime**: Lejo përdoruesit të japin vlerësime për këpucët e blera.
* **Blerja e Këpucëve**: Siguro një proces të lehtë dhe të sigurt për blerjen dhe pagesën.
* **Menaxhimi i Listës së Dëshirave**: Mundëso shtimin ose heqjen e këpucëve nga lista e dëshirave.

**3.2 Performance**

* **Ngarkim i Shpejtë i Faqeve: Optimizimi i kohës së ngarkimit për kërkimet dhe faqet e produkteve.**

**3.3 Reliability**

* **Transaksione të Sigurta**: Sigurimi i metodave të pagesës të koduara dhe të mbrojtura.

**3.4 Modifiability**

* **Personalizimi i Ndërfaqes**: Lejo administratorët të modifikojnë ndërfaqen për promovime.

**3.5 Reusability**

* **API për Produktet**: Përdorimi i API-ve për të integruar sistemet e jashtme për përditësimin e inventarit ose rekomandime.
* **Analiza e Rishikimeve**: Ofrimi i përmbledhjeve nga vlerësimet e përdoruesve për qëllime marketingu.

**3.6 Adaptability**

* **Mbështetje për Monedhën**: Lejo platformën të trajtojë pagesa me kartë (VISA/MASTER), PayPal ose para në dorë me dollarë.
* **Infrastrukturë e Shkallëzueshme**: Sigurimi që sistemi të përballojë rritjen e përdoruesve dhe inventarit.

**3.7 Security**

* **Sistem i Autentifikimit**: Siguro hyrje të mbrojtura me enkriptim të fjalëkalimeve dhe role të përdoruesve.
* **Ndjekja e Porosive**: Menaxhimi dhe shfaqja e historikut të porosive në mënyrë të sigurt.

1. **Product Overview**

**4.1 Product Vision**

LuxuryStep është një platformë e specializuar e-commerce e krijuar për të ofruar një përvojë të lehtë dhe të përshtatshme për adhuruesit dhe blerësit e këpucëve. Ajo kombinon funksionalitetin e një platforme tradicionale të tregtisë elektronike me një sistem vlerësimi dhe rishikimi, duke i lejuar përdoruesit të shfletojnë, blejnë dhe rishikojnë lloje të ndryshme të këpucëve. Platforma gjithashtu mbështet menaxhimin e listave të dëshirave dhe ofron kontrolle administrative për moderimin e rishikimeve të këpucëve.

**4.2 Key Features:**

 **Katalog i Gjithanshëm i Këpucëve:**

* Ofron një gamë të gjerë këpucësh të kategorizuara sipas markës, stilit, madhësisë dhe çmimit.
* Përshkrime të detajuara, informacione mbi stokun dhe imazhe me cilësi të lartë për çdo produkt.

 **Menaxhimi i Listës së Dëshirave:**

* Përdoruesit mund të krijojnë dhe menaxhojnë lista dëshirash për këpucët e preferuara.
* Mundësia për të ruajtur këpucët për blerje ose referencë të mëvonshme.

 **Porositja dhe Gjurmimi i Porosive:**

* Lejon përdoruesit të shtojnë këpucët në shportë, të vendosin porosi dhe të ndjekin statusin e tyre.
* Përfshin azhurnime në kohë reale mbi stokun dhe llogaritje dinamike të çmimeve.

 **Rishikimet e Përdoruesve:**

* Përdoruesit mund të vlerësojnë dhe të shkruajnë rishikime për këpucët e blera.
* Ndihmon përdoruesit e tjerë të marrin vendime të informuara bazuar në vlerësimet dhe komentet.

 **Paneli Administrativ:**

* Lejon administratorët të moderojnë rishikimet e këpucëve dhe të ruajnë integritetin e sistemit të rishikimeve.
* Përfshin funksionalitete për menaxhimin e produkteve dhe komenteve të përdoruesve.

**4.3 Stakeholders:**

 **Përdoruesit Finalë:**

* Përdoruesit kryesorë që shfletojnë, blejnë dhe rishikojnë këpucët.
* Krijojnë lista dëshirash të personalizuara dhe vendosin porosi.

 **Markat e Këpucëve:**

* Partnerë që promovojnë produktet e tyre në platformë.
* Përfitojnë nga rishikimet e përdoruesve dhe rritja e dukshmërisë.

 **Administratorët e Platformës:**

* Mbikëqyrin dhe menaxhojnë ndërveprimet e përdoruesve, rishikimet dhe produktet.

 **Zhvilluesit:**

* Mirëmbajnë platformën dhe përmirësojnë funksionalitetet e saj për shkallëzueshmëri dhe përdorshmëri më të mirë.

**4.4 Target Market:**

 **Dizajn Dinamik i Bazës së të Dhënave:**

* Lidhjet midis përdoruesve, rishikimeve, porosive dhe këpucëve sigurojnë një menaxhim të qëndrueshëm dhe të besueshëm të të dhënave.
* Arkitekturë modulare që lejon shkallëzueshmëri dhe mirëmbajtje të lehtë.

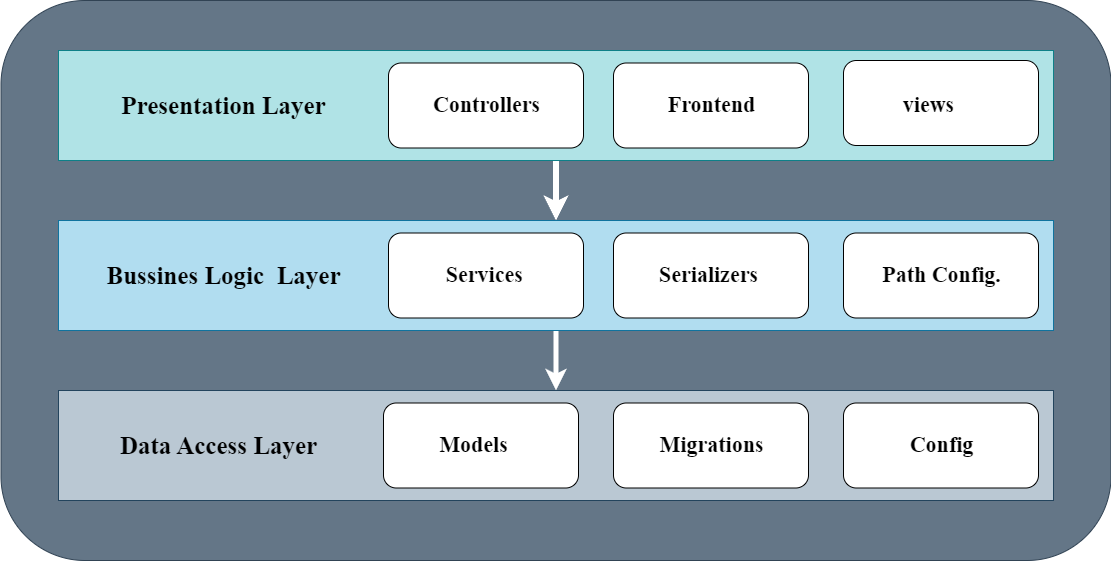
 **Ndërfaqe Interaktive për Përdoruesit:**

* E dizajnuar për të qenë e përgjegjshme dhe intuitive, duke ofruar një përvojë të qetë të shfletimit dhe blerjes.
* Integrimi i teknologjive moderne të uebit për performancë të përmirësuar.

 **Transaksione të Sigurta:**

* Përdor porta të sigurta pagese dhe ruajtje të dhënash të koduara për të mbrojtur informacionin e përdoruesve.

1. **Architectural Models**



Platforma **LuxuryStep** adopton arkitekturën **Model-View-Template (MVT)**, e lehtësuar nga **Django REST Framework**. Kjo arkitekturë siguron ndarjen e përgjegjësive, duke mundësuar modularitet dhe mirëmbajtje më të mirë:

* **Modeli**: Përfaqëson skemën e bazës së të dhënave dhe trajton logjikën e biznesit, duke përfshirë entitete si **Shoe, Review, Order** dhe **Wishlist**.
* **View (Pamja)**: Proceson kërkesat hyrëse, ndërvepron me modelet dhe përgatit përgjigjet.
* **Template (Shablloni)**: Menaxhon **Presantation Layer**, duke ofruar përmbajtje dinamike të renderuar në frontend.

Kjo arkitekturë krijon kufij të qartë midis menaxhimit të të dhënave, logjikës së biznesit dhe ndërfaqes së përdoruesit, duke ofruar fleksibilitet dhe shkallëzueshmëri.

**MVT** u zgjodh sepse:

* Integrohet pa probleme me **Django REST Framework** për ndërtimin e API-ve të fuqishme.
* Mundëson zhvillim të shpejtë duke përdorur komponentë të ripërdorshëm dhe konvencione.

## **5.1** **Layer Responsibilities**

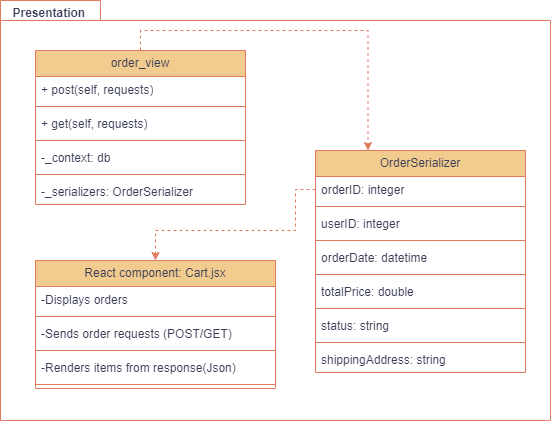
|  |  |
| --- | --- |
| **Layer** | **Përshkrimi** |

|  |  |
| --- | --- |
| **Presentation Layer** | Shtresa që menaxhon ndërfaqen e përdoruesit dhe ndërveprimin me aplikacionin. Në **LuxuryStep**, kjo shtresë përfshin **frontend-in** (React), **views** (Django REST Framework) dhe **controllers** që përpunojnë kërkesat e përdoruesit. |

|  |  |
| --- | --- |
| **Business Logic Layer** | Përgjegjëse për rregullat e biznesit dhe logjikën e aplikacionit. Kjo shtresë përmban **services**, **serializers** dhe **path config** (urls.py që menaxhon rruget e API-së). |

|  |  |
| --- | --- |
| **Data Access Layer** | Merret me ruajtjen dhe menaxhimin e të dhënave në bazën e të dhënave. Përfshin **models**, **migrations** dhe **config** (settings.py ku përcaktohen lidhjet me bazën e të dhënave). |

**5.2 Presentation-Layer Decomposition**

****

Diagrami tregon ndërveprimin midis frontend-it (Cart.jsx), backend-it (order\_view), dhe serializimit të të dhënave (OrderSerializer).

**5.2.1 Presentation Layer Responsibilities**

| **Moduli** | **Funksionaliteti** | **Ndërveprimet** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| **Cart.jsx** | Shfaq porositë, dërgon kërkesa për porosi (POST/GET), renderon JSON | Thërret order\_view përmes API |

|  |  |  |
| --- | --- | --- |
| **order\_view** | Menaxhon krijimin dhe marrjen e porosive, përpunon kontekstin e DB | Përdor OrderSerializer, ndërvepron me DB |

|  |  |  |
| --- | --- | --- |
| **OrderSerializer** | Serializon dhe deserializon të dhënat e porosive | Përdoret nga order\_view për kthimin e JSON |

**Controllers**: Përmban funksionet **API** që trajtojnë kërkesat nga klienti dhe ndërveprojnë me shtresën e logjikës së biznesit. Këto janë zakonisht **View** ose **APIViews** në Django REST Framework.

**Frontend**: Aplikacioni **React** që konsumon API-të e Django dhe paraqet UI-në për përdoruesit. Përfshin komponentët **React** dhe ruterin për navigim.

**Views**: Në kontekstin e **Django-s**, përfshin klasat dhe funksionet në **views folder** që trajtojnë logjikën e kërkesave **HTTP** dhe kthejnë përgjigjet. Në **React**, kjo mund të jetë komponentët që menaxhojnë interaktivitetin.

**Controllers**

* **ShoeView →** Menaxhon kërkesat për menaxhimin e këpucëve (CRUD).
* **ReviewView →** Lejon krijimin dhe shfaqjen e rishikimeve për këpucët.
* **OrderView →** Trajton porositë e bëra nga përdoruesit.

**Frontend (React)**

* **Wishlist.jsx →** Shfaq listën e këpucëve që janë në wishlist.
* **Purchase.jsx →** Paraqet faqen ku kryhet pagesa.
* **Cart.jsx →** Menaxhon shportën e blerjeve dhe përfundimin e porosive.

**Views (Django)**

* **ShoeAPIView →** Merr një listë këpucësh ose detajet e një këpuce.

**Django Serializers:**

Në kontekstin e Django-s, Serializers shërbejnë si një formë e ViewModel që përgatit dhe validon të dhënat përpara se të kthehen në klient ose të ruhen në bazën e të dhënave. Ato mundësojnë ndarjen e logjikës së prezantimit nga modelet e të dhënave.

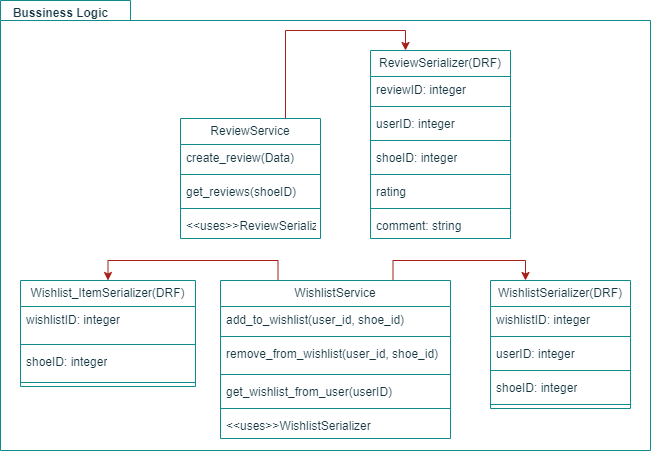
**5.2.2 Presentation Layer Rationale**

Shtresa e **Prezantimit** në këtë sistem përbëhet nga komponentët frontend (React) që ndërveprojnë me API-të e backend-it për të shfaqur të dhëna dhe për të trajtuar veprimet e përdoruesve. Racionalizimi i kësaj shtrese është si më poshtë:

1. **Ndarja e Përgjegjësive**
   * **Komponenti React (Cart.jsx)** është përgjegjës për shfaqjen e elementeve të UI dhe dërgimin e kërkesave për të marrë ose vendosur porosi.
   * Logjika e backend-it, si përpunimi i porosive, qëndron e ndarë në **order\_view** dhe **OrderSerializer**.
2. **Komunikim Efikas me API-në**
   * **Cart.jsx** ndërvepron me backend-in përmes kërkesave **POST/GET** tek **order\_view**.
   * **order\_view** përdor **OrderSerializer** për të strukturuar të dhënat përpara se t’i dërgojë në frontend.
3. **Konsistenca dhe Formësimi i të Dhënave**
   * **OrderSerializer** siguron që të dhënat të jenë të formatuara saktë (p.sh., orderID si integer, totalPrice si double) përpara se të dërgohen në frontend.
   * Kjo garanton që **Cart.jsx** të marrë të dhëna të strukturuara, duke minimizuar përpunimin në frontend.
4. **Shkallëzueshmëria dhe Mirëmbajtja**
   * **Komponenti Cart.jsx** është modular dhe mund të zgjerohet me funksionalitete shtesë si filtrimi i porosive ose modifikimi i artikujve para pagesës.
   * **order\_view** mund të zgjerohet me pika hyrëse shtesë ose logjikë biznesi pa ndikuar në shtresën e UI.
5. **Eksperienca e Përdoruesit dhe Performanca**
   * **Cart.jsx** azhurnon UI-në në mënyrë dinamike bazuar në përgjigjet e API-së, duke siguruar një përvojë të rrjedhshme për përdoruesin.
   * Vetëm të dhënat e nevojshme merren dhe shfaqen, duke optimizuar performancën dhe reduktuar kërkesat e panevojshme në API.

Ky dizajn siguron një ndarje të qartë midis përgjegjësive të frontend-it dhe backend-it, duke promovuar **modularitetin, mirëmbajtjen dhe trajtimin efikas të të dhënave** në sistem.

### **5.3** **Business Logic Layer Decomposition**

****

Logjika e Biznesit për Reviews dhe Wishlist (shembull)

**5.3.1 Business Logic Layer Responsibilities**

| **Service/Serializer** | **Funksionaliteti** |
| --- | --- |

|  |  |
| --- | --- |
| **ReviewService** | Menaxhon rishikimet e këpucëve. |

|  |  |
| --- | --- |
| **create\_review(Data)** | Krijon një rishikim të ri. |

|  |  |
| --- | --- |
| **get\_reviews(shoeID)** | Kthen rishikimet për një këpucë të caktuar. |

|  |  |
| --- | --- |
| **ReviewSerializer (DRF)** | Serializon të dhënat e rishikimit. |

|  |  |
| --- | --- |
| **reviewID: integer** | Identifikues unik i rishikimit. |

|  |  |
| --- | --- |
| **userID: integer** | Identifikuesi i përdoruesit që ka bërë rishikimin. |

|  |  |
| --- | --- |
| **shoeID: integer** | Identifikuesi i këpucës që është rishikuar. |

|  |  |
| --- | --- |
| **rating** | Vlerësimi i dhënë (p.sh., 1-5 yje). |

|  |  |
| --- | --- |
| **comment: string** | Komenti i përdoruesit për produktin. |

| **Service/Serializer** | **Funksionaliteti** |
| --- | --- |

|  |  |
| --- | --- |
| **WishlistService** | Menaxhon listën e dëshirave të përdoruesve. |

|  |  |
| --- | --- |
| **add & remove wishlist(userID, shoeID)** | Shton një këpucë në listën e dëshirave të përdoruesit. |

|  |  |
| --- | --- |
| **get\_wishlist\_from\_user(userID)** | Kthen listën e dëshirave të një përdoruesi. |

|  |  |
| --- | --- |
| **WishlistSerializer (DRF)** | Serializon të dhënat e listës së dëshirave. |

|  |  |
| --- | --- |
| **wishlistID: integer** | Identifikues unik i listës së dëshirave. |

|  |  |
| --- | --- |
| **userID: integer** | Identifikuesi i përdoruesit që ka krijuar listën e dëshirave. |

|  |  |
| --- | --- |
| **shoeID: integer** | Identifikuesi i këpucës së shtuar në listën e dëshirave. |

|  |  |
| --- | --- |
| **Wishlist\_ItemSerializer (DRF)** | Serializon një artikull të listës së dëshirave. |

|  |  |
| --- | --- |
| **wishlistID: integer** | Identifikuesi i listës së dëshirave ku ndodhet artikulli. |

|  |  |
| --- | --- |
| **shoeID: integer** | Identifikuesi i këpucës në listën e dëshirave. |

Services: Përmban logjikën e biznesit për menaxhimin e përdoruesve, porosive dhe listave të dëshirave dhe implementon validimin dhe llogaritjet. Ndërvepron me Modelet në shtresën e aksesit të të dhënave. Përdoret nga Kontrollerat në Shtresën e Prezantimit.

Serializers: Konverton llojet komplekse të të dhënave (p.sh., modelet Django) në format JSON dhe kryen validimin e të dhënave të futura në kërkesat API. Ndërvepron me Modelet për të siguruar që të dhënat të jenë të strukturuara siç duhet.

Path Configuration: Përcakton pikat fundore të API-së dhe rutimin e URL-ve për shërbime të ndryshme. Lidh Shtresën e Prezantimit (Kontrollerat, Frontend) me Logjikën e Biznesit.

**5.3.2 Business Logic Layer Rationale**

Ky dekompozim ndihmon në ndarjen e përgjegjësive dhe përmirëson organizimin e kodit, duke lehtësuar zgjerueshmërinë e aplikacionit në të ardhmen.

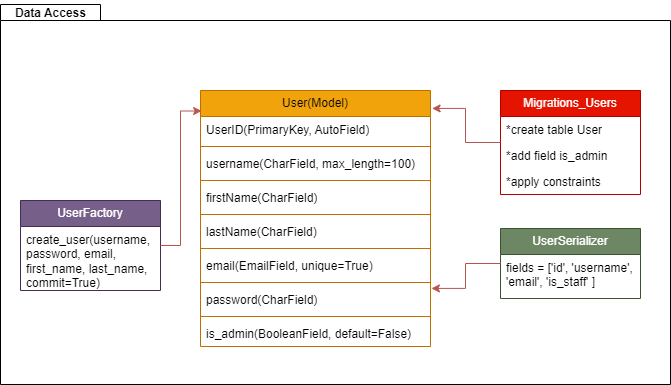
### **Përfitimet e këtij dizajni**

* **Modulariteti**: Funksionalitetet janë të ndara qartë, duke e bërë kodin më të mirëmbajtshëm.
* **Ripërdorimi**: Serializer-at e përcaktuar sigurojnë që të dhënat të jenë të strukturuara në mënyrë të qëndrueshme në të gjithë sistemin.
* **Ndërveprimi me API**: Të dyja shërbimet ekspozojnë operacione që mund të përdoren nga frontend-i për të menaxhuar rishikimet dhe listat e dëshirave në mënyrë efikase.
* **Përputhshmëria me Django REST Framework (DRF)**: Përdorimi i serializer-ëve të DRF siguron integrim të lehtë dhe validim të të dhënave.

**User Interface** lidhet me **Bussiness Logic Layer** përmes **API**-ve për:

1. **Fetch Data**: Tërheq informacionin e produkteve, listat e dëshirave të përdoruesve dhe rishikimet.
2. **Submit Actions**: Dërgon veprimet e përdoruesit si shtimi i artikujve në shportë, kryerja e porosive dhe dërgimi i rishikimeve.
3. **Receive Responses**: Shfaq reagime për përdoruesin, si përfundimi me sukses i një porosie ose dërgimi i një rishikimi.

**5.4 Data Access Layer Decomposition**

****

**Shtresa e qasjes se te dhenave (Data Access Layer – DAL)** ne projektin LuxuryStep eshte pergjegjese per ruajtjen dhe menaxhimin e te dhenave ne bazen e te dhenave. Ajo eshte struktur ne tre perberes kryesore:

1. **Models –** Perfaqesojne strukturen dhe marredheniet e te dhenave ne tabelat e bazes se te dhenave.
2. **Data –** Menaxhon komunikimin me bazen e te dhenave perms ORM dhe perpunon logjiken baze te qasjes/CRUD
3. **Migrations –** Kujdesen per historkun e ndryshimeve ne skemen e bazes se te dhenave, duke mundesuar krijimin ose modifikimin e tabelave pa humbur informacion.

Ky model i ndarjes së përgjegjësive **rrit modularitetin** dhe e bën sistemin më të menaxhueshëm, duke lejuar **ndryshime** dhe **përmirësime** të vazhdueshme **pa ndikuar** negativisht në funksionalitetet ekzistuese të aplikacionit.

**5.4.1 Data Access Layer Responsibilities**

|  |  |
| --- | --- |
| **Module** | **Responsibilities** |
| **User(Model)** | * Modelon entitetin kryesor te perdoruesit * Permban fushat themelore si username, email, password, etj * Perdoret per autentifikim dhe ruajtje te te dhenave te perdoruesve |
| **Migrations\_Users** | * Menaxhon ndryshimet ne skemen e bazes se te dhenave per tabelen ‘User’ * Krijon tabelen User, shton kolona te reja(p.sh, is\_admin) * Aplikon kufizimet dhe relacionet |
| **UserSerializer** | * Serializon dhe deserializon te dhenat e modelit User * Percakton se cilat fusha ekspozohen ose pranohen permes API-se * Siguron validimin baze te te dhenave |
| **UserFactory** | * Ofron metoda per krijimin e instancave te User me logjike te personalizuar * Lehteson testimin dhe insrtimet e automatizuara ne baze te te dhenave. |

**5.4.2 Data Access Layer Specifications**

|  |  |  |  |
| --- | --- | --- | --- |
| **Pershkrimi** | **Syntax** | **Pre** | **Post** |
| Krijon tabelen ‘User’ permes mergimeve | python manage.py makemigrations  python manage.py migrate | -Duhet te jete perkufizuar modeli User ne models.py. -Nuk ekziston tabela ‘User’ ne skemen ekzistuese. | -Krijohet tabela ‘User’ me fushat (UserID, username, email, etj). -Ruhet struktura e re ne bazen e te dhenave. |
| Krijon nje perdorues te ri | UserFactory.create\_user(…) | -Te jete aplikuar migrimi (tabela ‘User’ ekziston). -Parametrat e nevojshem p.sh. username, password, etj. | -Nje rresht i ri shtohet ne tabelen ‘User’. -Kthehet instance e User e sapokrijuar. |
| Serializon te dhenat e user | class UserSerializer | -Modeli User duhet te jete I definuar. -DRF i instaluar dhe funksional | -Te dhenat e User konvertohen ne JSON(ose format tjeter) per transmetim. -Mundeson validim te fushave |
| Lexon/krijon user permes ORM | User.objects.create(...) User.objects.get(...) | -Te jete krijuar tabela ‘User’ permes migrimeve. -Parametra korrekte per create ose get. | -Shtohet ose lexohet nje rresht ne/baze te tabeles ‘User’. |
| Perditeson fushat e nje useri | User.objects.update(…) | -Ekziston nje user i identifikuar me UserID ose username. -Te jete inicializuar ORM me settings.py | -Rreshti ne DB perditsohet me vlerat e reja. -Ndryshimet ruhen pergjithmone ne bazen e te dhenave |

* Krijon tabelën ‘User’: Kjo përfshin ekzekutimin e komandës makemigrations dhe migrate për të reflektuar ndryshimet në bazën e të dhënave. Sigurohet që fusha si UserID, username, email etj. të jenë të pranishme në tabelë.
* Krijon një përdorues të ri: UserFactory.create\_user(...) gjeneron një instancë User me të dhënat e nevojshme (p.sh. username, password). Ky veprim lehtëson krijimin e User si në testime ashtu edhe në seed data.
* Serializon të dhënat e user: UserSerializer mundëson shndërrimin e objektit User në JSON (ose një format tjetër) për konsum nga frontend (React) dhe validon fushat hyrëse në rastet e POST/PUT.
* Lexon/krijon user përmes ORM: User.objects.create(...) dhe User.objects.get(...) ofrojnë metodën bazë për CRUD. Krijojnë një rresht të ri ose kthejnë një ekzistues, varësisht nga parametri i përdorur.
* Përditëson fushat e një useri: User.objects.update(...) ndryshon fushat e userit ekzistues në bazën e të dhënave, duke ruajtur çdo ndryshim në mënyrë permanente.

**5.4.3 Data Access Layer Design Rationale**

Per te menaxhuar aksesin ne bazen e te dhenave, nje shtrese e vecante e **quajtur Data Access Layer(DAL)** siguron modularitet, koherence dhe ndarje te pergjegjesive. Edhe pse do te ishte e mundur te integroheshin operacionet e bazes se te dhenave dretperdrejt ne controllers ose ne shtresa te tjera te aplikacionit, ndarja e tyre ne DAL ofron nje dizajn me te strukturuar dhe me te mirembajtshem.

**Data Access Layer** eshte pergjegjes per:

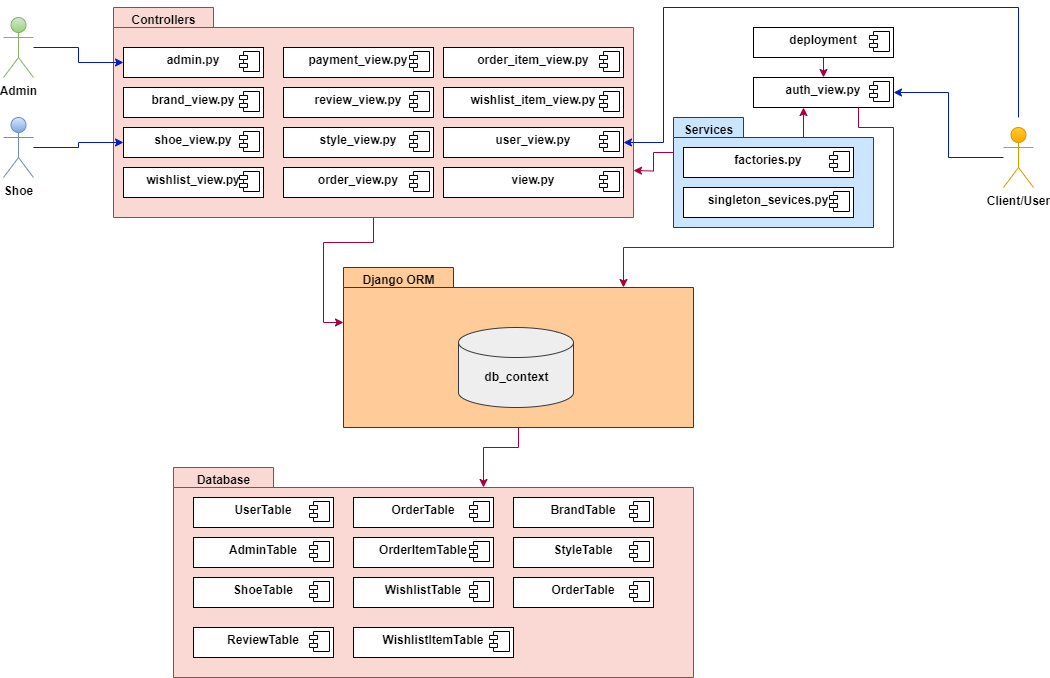
* Menaxhimin e modeleve te te dhenave(Data Models)
* Migrimet(Migrations)
* Filtrat golbale(Global Query Filters)

Kjo siguron që të gjitha operacionet mbi bazën e të dhënave—krijimi, leximi, përditësimi dhe fshirja (**CRUD**)—të kryhen në **mënyrë të standardizuar** dhe të ndarë nga **Presentation Layer** dhe **Business Logic Layer**. Në këtë mënyrë, **të gjitha aspektet** e aksesit në bazën e të dhënave bëhen të **ripërdorshme** dhe lehtësisht **të testueshme**.

* **Data**: Përfaqëson strukturën e tabelave në bazën e të dhënave, duke reduktuar duplikimet dhe duke siguruar koherencë në të gjithë aplikacionin.
* **Migrations**: Sigurojnë që ndryshimet në strukturën e bazës së të dhënave (p.sh., shtimi i fushave të reja apo modifikimi i kufizimeve) të ruajnë integritetin e të dhënave në mjediset e **zhvillimit**, **testimit** dhe **prodhimit**.
* **Filters**: Mundësojnë që të dhënat e fshira ose të padukshme për përdorues të veçantë të përjashtohen automatikisht nga kërkesat, duke ofruar një shtresë shtesë sigurie.

Falë **DAL**, kemi një **arkitekturë modulare**, **të testueshme**, dhe **të shkallëzueshme**, e cila lejon **përmirësime të vazhdueshme** pa ndikuar negativisht në shtresat e tjera të aplikacionit.

**5.5 Runtime Components**

****

Diagrami përshkruan arkitekturën e **LuxuryStep**, ku **Controllers(views)** menaxhojnë kërkesat dhe i përcjellin te **Services**, të cilat përmbajnë **factories.py** dhe **singleton\_services.py** për menaxhimin e krijimit dhe instancimit të shërbimeve. Services komunikojnë drejtpërdrejt me bazën e të dhënave për të kryer operacione **CRUD**, duke siguruar një ndarje të pastër të përgjegjësive dhe një strukturë fleksibile.

**5.6 Mapping between Models**

Ne këtë tabelë tregohen marrëdhëniet ndërmjet shtresave dhe përbërësit e tyre

| **User** | **Wishlist** | **Wishlist Item** | **Orders** | **Services** |
| --- | --- | --- | --- | --- |
| **Startup** | AuthenticationController | WishlistController | OrderController | Hosting Environment |
| **User** | UserController | UserController | UserController | ReviewsController |
| **Wishlist** | WishlistController | WishlistItemController |  |  |
| **Wishlist Item** |  | WishlistItemModel, ShoeModel |  |  |
| **Orders** |  |  | OrderController | OrderService |
| **Services** | Authentication, UserActivity | Wishlist Management | Wishlist Item Management | Order Management |

| **Runtime Components** | **Modulet** |
| --- | --- |
| **Startup** | Konfigurimi i shtresës së nisjes, mjedisi i hostimit |
| **Konfigurimi i Porosive** | Përmban parametra të mjedisit dhe databazës |
| **Control Panel** | Controllers.OrderController, Controllers.UserController |
| **Ndërfaqja e Përdoruesit** | Controllers.AuthenticationController, Controllers.UserController |
| **Menaxhimi i Porosive** | Controllers.OrderController, Views.OrderListView, Views.OrderDetailView, Views.ActiveOrderView, Views.AddToOrderView, Views.RemoveFromOrderView, Views.FinalizeOrderView, Views.PurchaseOrderView |
| **Menaxhimi i Artikujve të Porosisë** | Controllers.OrderItemController, Views.OrderItemListView, Views.OrderItemDetailView, Views.OrderItemListCreateView |
| **Menaxhimi i Produkteve (Këpucëve)** | Controllers.ShoeController |
| **Shërbime dhe Menaxherë** | Services.OrderManagerSingleton, Services.OrderItemManagerSingleton |
| **Aksesi në Databazë** | Models.Order, Models.OrderItem, Models.Shoe, Migrations |
| **Mjedisi i Hostimit** | Web Server, konfigurimi i hostimit dhe ekzekutimi i sistemit |

**5.7 Architectural Design Rationale**

Ky dokument përshkruan arsyetimin pas arkitekturës së projektit **LuxuryStep**, i cili është një sistem për rishikimin dhe shitjen e këpucëve, i ndërtuar duke përdorur **Django Rest Framework për backend** dhe **React për frontend**.

Ne kemi adoptuar arkitekturën **Model-View-Template (MVT)** në backend (Django) dhe kemi ndarë projektin në disa shtresa kryesore:

1. **Presentation Layer** (Frontend - React)
2. **Business Logic Layer** (Backend - Django Services)
3. **Data Access Layer** (Django ORM dhe DB SQLite)

Kjo ndarje siguron **modularitet, mirëmbajtje më të lehtë dhe shkallëzueshmëri të sistemit**.

Django përdor **MVT (Model-View-Template)**, i cili është një variant i Model-View-Controller (MVC), por Django menaxhon automatikisht pjesën e "Controller"-it, duke e bërë zhvillimin më efikas.

| **Shtresa** | **Përshkrimi** |
| --- | --- |
| **Model** | Përfaqëson të dhënat dhe ndërveprimet me bazën e të dhënave. Për shembull, kemi modele për **User, Shoe, Review, Wishlist etj**. |
| **View** | Përpunon kërkesat HTTP dhe bashkëvepron me modelet për të gjeneruar të dhëna për frontend. Në këtë projekt kemi përdorur **Django REST Framework (DRF) për API-të**. |
| **Template** | Përdoret për të paraqitur të dhënat në frontend nëse përdoret Django Templates. Megjithatë, ne kemi një frontend të ndarë (React), kështu që Templates nuk përdoren. |

Në vend të templates, **React** konsumon **API**-të e **Django** për të marrë dhe shfaqur të dhënat.

### **Presentation Layer (Frontend - React)**

* Kjo shtresë është përgjegjëse për ndërfaqen e përdoruesit.
* React konsumon API-të nga Django për të marrë të dhënat dhe për të shfaqur informacionin për përdoruesit.
* Axios përdoret për të bërë kërkesat HTTP.

### **Business Logic Layer (Backend - Django Services)**

* Kjo shtresë përmban të gjithë logjikën e biznesit të aplikacionit.
* Është përgjegjëse për funksionalitete si:
  + **Menaxhimi i rishikimeve të këpucëve** (ReviewService)
  + **Menaxhimi i listës së dëshirave** (WishlistService)
  + **Menaxhimi i porosive dhe produkteve**
* Shërbimet e biznesit përdorin **Serializers (Django REST Framework - DRF)** për të transformuar të dhënat nga modelet në JSON për frontend.

### **3.3. Data Access Layer**

* Kjo shtresë përmban **modelet** që përfaqësojnë bazën e të dhënave.
* **Django ORM (Object-Relational Mapping)** përdoret për të ndërvepruar me bazën e të dhënave, duke siguruar një abstraktim të lehtë mbi SQL.

**Shembuj të modeleve:**

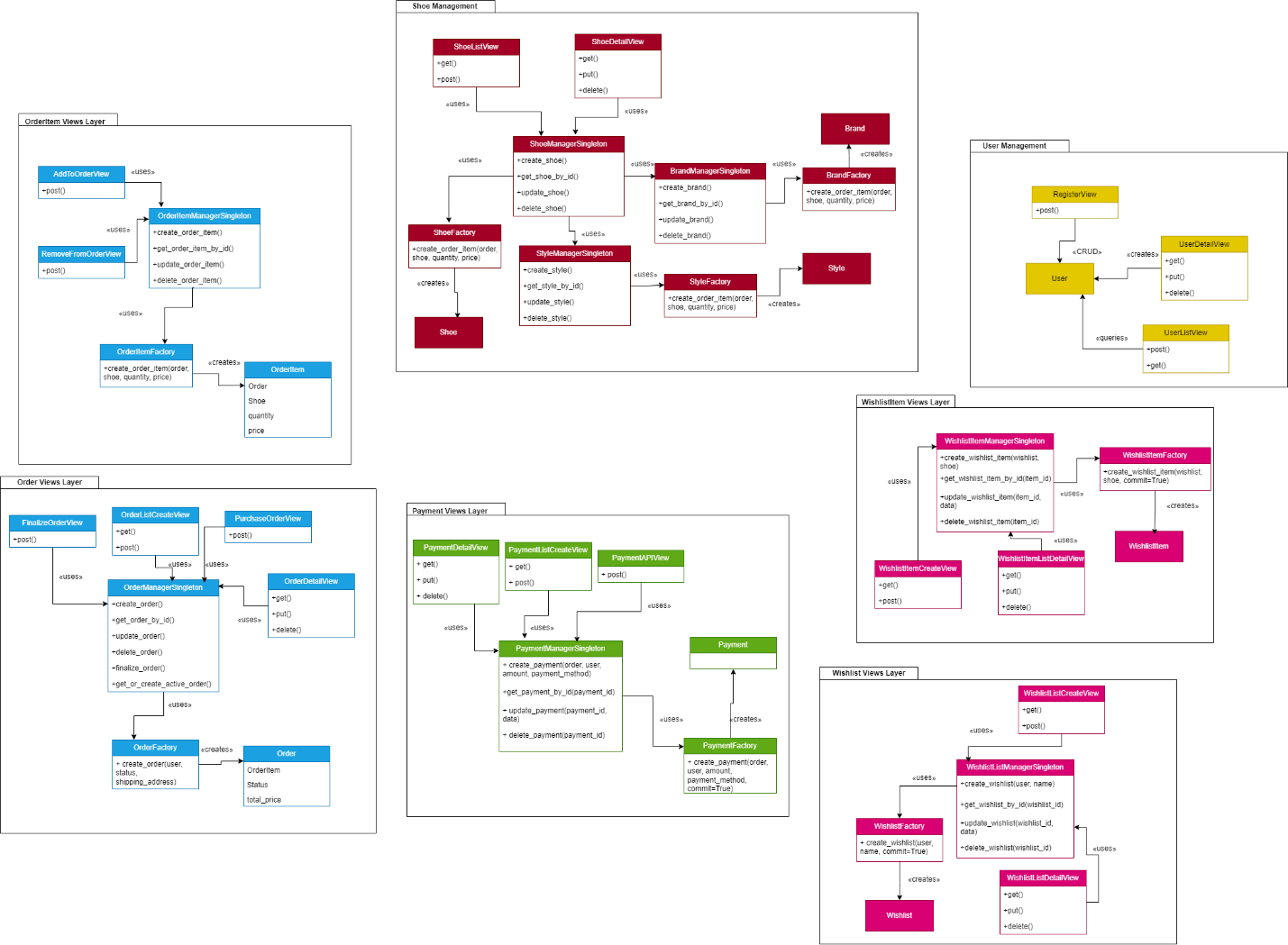
* + **Shoe**
  + **Review**
  + **Wishlist**
  + **Order**
* DB SQLite përdoret si bazë të dhënash për performancë më të mirë dhe shkallëzim.

Kjo arkitekturë na lejon të kemi një sistem të **modularizuar, të strukturuar dhe të lehtë për tu mirëmbajtur**. Duke përdorur **Django REST Framework** për backend dhe **React për frontend**, kemi ndarë qartë përgjegjësitë ndërmjet shtresave të sistemit, duke bërë më të lehtë zhvillimin, testimin dhe zgjerimin e funksionaliteteve të mëtejshme.

## **6.LuxuryStep Detailed Design Document**

## **6.1 Mid Level Design Models**

**Mid-Level Design përfshin module të ndryshme për menaxhimin e porosive, këpucëve, pagesave, listave të dëshirave dhe përdoruesve. Diagrami është i ndarë në disa module të pavarura, secili duke përfaqësuar një pjesë thelbësore të logjikës së biznesit sipas specifikave të projektit tuaj:**

****

• **Order Module** – Menaxhon të gjitha operacionet lidhur me porositë, duke përfshirë krijimin, përditësimin, fshirjen dhe finalizimin e porosive. Në diagram, **OrderManagerSingleton** dhe **OrderFactory** bashkë me entitetin **Order** tregojnë se si përpunohen porositë.

• **Order Item Module** – Administron artikujt që përfshihen në porosi. Ky modul përdor **OrderItemManagerSingleton** dhe **OrderItemFactory** për të krijuar, modifikuar dhe fshirë artikujt e porosive, duke siguruar që çdo porosi përmban detajet e duhura për secilin artikull.

• **Shoe Management Module** – Përfshin menaxhimin e produkteve (këpucëve) dhe ndërveprimin e tyre me entitetet **Brand** dhe **Style**. Ky modul shfaq **ShoeListView** dhe **ShoeDetailView** në shtresën e view-ve, ndërsa **ShoeManagerSingleton** dhe **ShoeFactory** përfaqësojnë logjikën e biznesit për menaxhimin e këpucëve. Entiteti **Shoe** tregon lidhjet me **Brand** dhe **Style**, duke reflektuar marrëdhëniet e domosdoshme për atributet e produktit.

• **Payment Module** – Menaxhon proceset e pagesave për porositë. Ky modul përdor **PaymentManagerSingleton** dhe **PaymentFactory** për të përpunuar pagesat, ndërveprimi i të cilave shfaqet përmes view-ve si **PaymentListCreateView** ose **PaymentDetailView**, duke krijuar entitetin **Payment** që përfaqëson transaksionet financiare.

• **Wishlist Module** – Lejon përdoruesit të krijojnë dhe menaxhojnë lista me produkte të preferuara. Ky modul përfshin krijimin dhe administrimin e entitetit **Wishlist** përmes view-ve të specializuara, të mbështetura nga **WishlistManagerSingleton**.

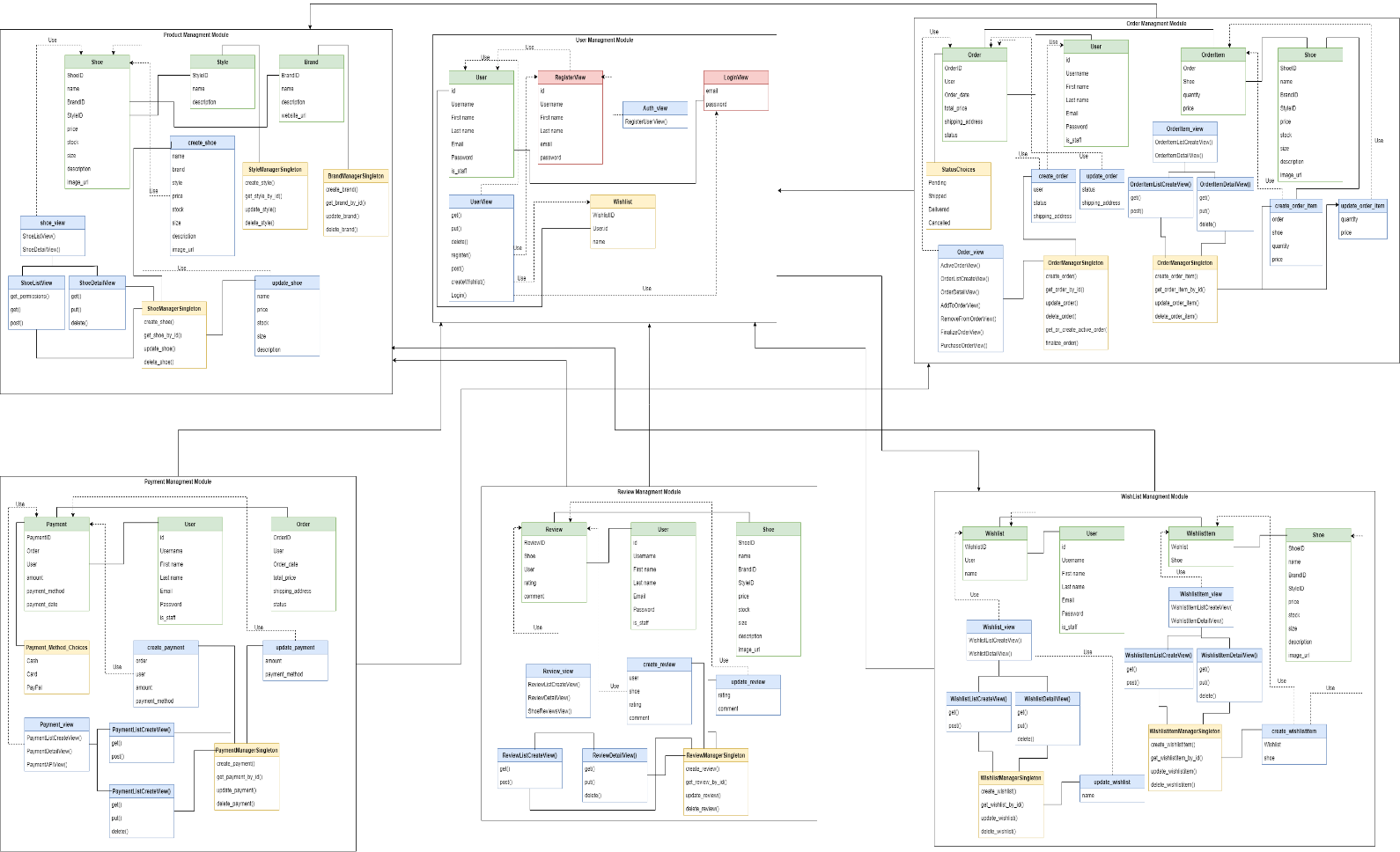
• **Wishlist Item Module** – Përfshin logjikën për menaxhimin e artikujve brenda listave të dëshirave. Përdoren **WishlistItemManagerSingleton** dhe **WishlistItemFactory** për të shtuar, përditësuar dhe fshirë artikujt, ndërveprimi i të cilëve shfaqet në view-t përkatës, duke garantuar se lista e dëshirave përmban artikujt e saktë.

• **User Management Module** – Administron regjistrimin, autentikimin dhe menaxhimin e përdoruesve. Ky modul përfshin view-t si **RegisterView**, **UserListView** dhe **UserDetailView**, të cilat ndërveprojnë drejtpërdrejt me entitetin **User** për operacione të tilla si krijimi, listimi, përditësimi dhe fshirja e përdoruesve, duke siguruar gjithashtu funksionalitetet e autentikimit dhe autorizimit sipas roleve.

## **6.2 Low Level Design Models**

Low-Level Design përfshin module të ndryshme për menaxhimin e përdoruesve, produkteve (këpucëve), porosive, pagesave dhe vlerësimeve.  
Sistemi është i organizuar në modulet e mëposhtme, ku secili modul përfaqëson një pjesë të rëndësishme të funksionalitetit të platformës:

* **User Management Module** – Përdor modelin e integruar të përdoruesit nga Django për menaxhimin e identiteteve të përdoruesve. Ky modul trajton autentifikimin dhe autorizimin e përdoruesve, si dhe administron rolet e tyre (p.sh., staf, klient, etj.), duke përcaktuar nivelet e aksesit brenda platformës.
* **Product Management Module** – Përgjegjës për administrimin e produkteve të disponueshme në dyqan. Përfshin menaxhimin e këpucëve, markave dhe stileve, duke lehtësuar kategorizimin dhe filtrimin e produkteve sipas nevojave të përdoruesve.
* **Order Management Module** – Përmban logjikën për krijimin dhe menaxhimin e porosive të përdoruesve. Çdo porosi përbëhet nga një ose më shumë artikuj (OrderItem), duke mbajtur informacion mbi produktet e blera, sasinë dhe statusin e porosisë.
* **Payment Module** – Menaxhon pagesat e bëra nga përdoruesit për porositë e tyre. Ky modul mbështet disa metoda pagese, duke përfshirë para në dorë (Cash), kartat bankare (Card) dhe PayPal, duke siguruar fleksibilitet për klientët në mënyrën e pagesës.
* **Review & Rating Module** – Përdoruesit mund të lënë vlerësime dhe komente mbi këpucët që kanë blerë. Ky modul përmban një sistem vlerësimi që ndihmon përdoruesit e tjerë të marrin vendime më të informuara mbi produktet e ofruara në dyqan.
* **Wishlist Module** – Lejon përdoruesit të ruajnë një listë dëshirash me produktet që planifikojnë të blejnë më vonë. Çdo përdorues mund të ketë një wishlist të personalizuar, duke përfshirë këpucët e preferuara për një akses më të lehtë në të ardhmen.

****

**6.2.1 User Module Responsibilities**

| **Model** | **Responsibilities** |
| --- | --- |
| **UserEntity** | - Mbajtja e informacionit bazë të një përdoruesi, si UserId,Username, FirstName, LastName, Email, dhe PasswordHash. - Menaxhimi i rolit të përdoruesit (p.sh., is\_staff) Krijimi i një Wishliste për User kur të regjistrohet . |
| **Register View** | - Përmban fushat e nevojshme për regjistrimin e një përdoruesi të ri, përshirë fjalëkalimin dhe konfirmimin e tij. |
| **Auth\_View** | - Menaxhon sesionet e autentikimit të përdoruesit. - Përfshin logjikën për regjistrimin dhe hyrjen e përdoruesve. - Administron token-at e autentikimit për përdoruesit e regjistruar. |
| **Login View** | - Përmban fushat e nevojshme për hyrjen e një përdoruesi (email, password). - Verifikon kredencialet e përdoruesit dhe gjeneron token autentikimi në rast suksesi. |
| **Wishlist Entity** | - Përfaqëson listën e dëshirave të përdoruesit (WishlistId, UserId). - Përdoruesi mund të ruajë produkte në wishlist për një blerje të mundshme në të ardhmen (Krijohet në fazën e regjistrimit). |
| **UserView** | - **get()** – Kthen informacionin e përdoruesit ose një listë përdoruesish.  - **put()** – Përditëson të dhënat e një përdoruesi ekzistues.  - **post()** – Krijon një përdorues të ri.  - **delete()** – Fshin një përdorues nga sistemi.  - **register()** – Regjistron një përdorues të ri.  - **login()** – Verifikon kredencialet dhe menaxhon sesionin e përdoruesit.  - **createwishlist()** – Shton një wishlist të re për përdoruesin (në regjistrim). |

**User Module Specifications**

| **Functionality** | **Syntax** | **Pre** | **Post** |
| --- | --- | --- | --- |
| Regjistron një përdorues të ri | Task<User> RegisterAsync(RegisterView view/controller); | model nuk duhet të jetë null. model.Email duhet të jetë një email valid. model.Password duhet të plotësojë kriteret e sigurisë. | Një përdorues i ri shtohet në bazën e të dhënave. Kthen objektin User me ID të gjeneruar. |
| Autentifikon përdoruesin | Task<User> LoginAsync(string email, string password); | email dhe password nuk duhet të jenë null. email duhet të jetë në formatin e duhur. | Kthen objektin User nëse kredencialet janë të sakta. Kthen null nëse kredencialet janë të pasakta. |
| Merr të dhënat e përdoruesit | Task<User> GetUserByIdAsync(int id); | id duhet të jetë një vlerë pozitive. | Kthen objektin User nëse përdoruesi ekziston. Kthen null nëse përdoruesi nuk ekziston. |
| Fshin një përdorues nga sistemi | Task<bool> DeleteUserAsync(int id); | id duhet të jetë një vlerë e vlefshme (>0). Përdoruesi duhet të ekzistojë në databazë. | Kthen true nëse fshirja kryhet me sukses, false përndryshe. |
| Përditëson informacionin e një përdoruesi | Task<bool> EditUserAsync(int userId, EditUserView view/controller); | userId duhet të jetë > 0. model nuk duhet të jetë null. | Të dhënat e përdoruesit përditësohen në bazën e të dhënave. Kthen true nëse përditësimi bëhet me sukses, false përndryshe. |
| Krijon një wishlist për përdoruesin | Task<Wishlist> CreateWishlistAsync(int userId); | userId duhet të jetë një vlerë valide dhe përdoruesi duhet të ekzistojë. | Krijohet një wishlist e re për përdoruesin dhe ruhet në databazë. |
| Kthen një wishlist të përdoruesit | Task<Wishlist> GetWishlistByUserIdAsync(int userId); | userId duhet të jetë një vlerë valide. Wishlist duhet të ekzistojë për këtë përdorues. | Kthen objektin Wishlist nëse ekziston, përndryshe kthen null. |

**6.2.2 Product/Shoe Module Responsibilities**

|  |  |
| --- | --- |
| **Model** | **Responsibilities** |
| **Shoe Entity** | **-** Përfaqëson një produkt (këpucë) në sistem. - Përmban atributet si ShoeId, Name, Description, Price, BrandId, StyleId, Stock, Size, image\_url**.** |
| **Style Entity** | **-** Përfaqëson kategorinë ose stilin e këpucëve. - Përmban atributet si StyleId dhe Name. |
| **Brand Entity** | - Përfaqëson markën e këpucëve. - Përmban atributet si BrandId dhe Name. |
| **Shoe\_View** | **-** Përmban logjikën e menaxhimit të këpucëve. - Përfshin ShoeListView dhe ShoeDetailView për listimin dhe menaxhimin e detajeve të këpucëve. |
| **ShoeListView** | **- get\_permissions():** Menaxhon lejet për shikimin dhe menaxhimin e këpucëve. **- get():** Kthen një listë të këpucëve. **- post():** Krijon një produkt të ri (këpucë). |
| **ShoeDetailView** | **- get():** Merr detajet e një këpuce të caktuar**. - put():** Përditëson informacionin e një këpuce. **- delete():** Përdoret për fshirjen e një këpuce. |
| **ShoeManagerSingleton** | **-** Menaxhon operacionet CRUD për këpucët. **- create\_shoe():** Krijon një këpucë të re. **- update\_shoe():** Përditëson një këpucëekzistuese. **- get\_shoe\_by\_id**(): Merr një këpucë sipas ID-së. **- delete\_shoe():** Fshin një këpucë nga sistemi. |
| **Create\_shoe** | - Përmban logjikën për krijimin e një këpuce të re. |
| **Update\_shoe** | - Përmban logjikën për përditësimin e një këpucë ekzistuese. |
| **StyleManagerSingleton** | - Menaxhon stilet e këpucëve (CRUD për Style Entity). |
| **BrandManagerSingleton** | - Menaxhon markat e këpucëve (CRUD për Brand Entity). |

**Product/Shoe Module Specifications**

| **Functionality** | **Syntax** | **Pre** | **Post** |
| --- | --- | --- | --- |
| **Merr listën e këpucëve** | Task<List<Shoe>> GetShoesAsync() | - Duhet Të ketë këpucë në databazë | - Kthen një listë të këpucëve ekzistuese. |
| **Merr detajet e një këpuce** | Task<Shoe> GetShoeByIdAsync(int shoeId) | - shoeId duhet të jetë një vlerë pozitive (>0). | - Kthen një objekt Shoe nëse ekziston.  - Kthen null nëse këpuca nuk ekziston. |
| **Krijon një këpucë të re** | Task<Shoe> CreateShoeAsync(Shoe shoe) | - shoe nuk duhet të jetë null.  - Atributet Name, Price, BrandId, StyleId, Stock, dhe Size duhet të jenë të vlefshme. | - Shton këpucën në sistem dhe kthen objektin Shoe. |
| **Përditëson një këpucë ekzistuese** | Task<bool> UpdateShoeAsync(int shoeId, Shoe shoe) | - shoeId duhet të jetë një vlerë pozitive (>0).  - shoe nuk duhet të jetë null.  - Këpuca me shoeId duhet të ekzistojë në sistem. | - Përditëson të dhënat e këpucës ekzistuese.  - Kthen true nëse përditësimi është i suksesshëm, përndryshe false. |
| **Fshin një këpucë nga sistemi** | Task<bool> DeleteShoeAsync(int shoeId) | - shoeId duhet të jetë një vlerë pozitive (>0).  - Këpuca duhet të ekzistojë në sistem. | - Fshin këpucën nëse ekziston.  - Kthen true nëse fshirja është e suksesshme, përndryshe false. |
| **Menaxhon stilet e këpucëve** | Task<List<Style>> GetStylesAsync() | - Duhet Të ketë Stil në databazë | - Kthen një listë me stilet e këpucëve. |
| **Merr një stil të caktuar** | Task<Style> GetStyleByIdAsync(int styleId) | - styleId duhet të jetë një vlerë pozitive (>0). | - Kthen një objekt Style nëse ekziston.  - Kthen null nëse stili nuk ekziston. |
| **Shton një stil të ri** | Task<Style> CreateStyleAsync(Style style) | - style nuk duhet të jetë null.  - Name duhet të jetë unik. | - Krijon një stil të ri dhe e ruan në sistem. |
| **Fshin një stil ekzistues** | Task<bool> DeleteStyleAsync(int styleId) | - styleId duhet të jetë një vlerë pozitive (>0).  - Stili duhet të ekzistojë në sistem. | - Fshin stilin nëse ekziston.  - Kthen true nëse fshirja është e suksesshme, përndryshe false. |
| **Menaxhon markat e këpucëve** | Task<List<Brand>> GetBrandsAsync() | - Duhet Të ketë Brand në databazë | - Kthen një listë me markat e këpucëve. |
| **Merr një markë të caktuar** | Task<Brand> GetBrandByIdAsync(int brandId) | - brandId duhet të jetë një vlerë pozitive (>0). | - Kthen një objekt Brand nëse ekziston.  - Kthen null nëse marka nuk ekziston. |
| **Shton një markë të re** | Task<Brand> CreateBrandAsync(Brand brand) | - brand nuk duhet të jetë null.  - Name duhet të jetë unik. | - Krijon një markë të re dhe e ruan në sistem. |
| **Fshin një markë ekzistuese** | Task<bool> DeleteBrandAsync(int brandId) | - brandId duhet të jetë një vlerë pozitive (>0).  - Marka duhet të ekzistojë në sistem. | - Fshin markën nëse ekziston.  - Kthen true nëse fshirja është e suksesshme, përndryshe false. |

**6.2.3 Payment Module Responsibilities**

|  |  |
| --- | --- |
| **Model** | **Responsibilities** |
| **Payment Entity** | - Përfaqëson një pagesë në sistem. - Përmban atributet si PaymentId, UserId, OrderId, Amount, Payment\_method, payment\_date. |
| **User Entity** | - Përfaqëson përdoruesin që bën pagesën. - Përmban atributet e User Entity. |
| **Order Entity** | - Përfaqëson një porosi të bërë nga një përdorues. - Përmban atributet e Order Entity |
| **Payment\_View** | - Përmban logjikën e menaxhimit të pagesave. - Përfshin PaymentListCreateView dhe PaymentDetailView për listimin dhe menaxhimin e pagesave. |
| **PaymentListCreateView** | **- get():** Kthen një listë të pagesave. **- post():** Krijon një pagesë të re. |
| **PaymentDetailView** | **- get():** Merr detajet e një pagese të caktuar. **- put():** Përditëson një pagesë ekzistuese. **- delete():** Fshin një pagesë nga sistemi. |
| **PaymentManagerSingleton** | - Menaxhon operacionet CRUD për pagesat. **- create\_payment():** Krijon një pagesë të re. **- update\_payment():** Përditëson një pagesë ekzistuese. **- get\_payment\_by\_id():** Merr një pagesë sipas ID-së. **- delete\_payment():** Fshin një pagesë nga sistemi. |
| **Create\_payment** | - Përmban logjikën për krijimin e një pagese të re. |
| **Update\_payment** | - Përmban logjikën për përditësimin e një pagese ekzistuese. |
| **Payment\_Method\_Choices** | - Përmban opsionet për mënyrat e pagesës: **Cash, Card, PayPal.** |

**Payment Module Specifications**

| **Functionality** | **Syntax** | **Pre** | **Post** |
| --- | --- | --- | --- |
| **Merr listën e pagesave** | Task<List<Payment>> GetPaymentsAsync() | - Duhet Të ketë Payment në databazë | - Kthen një listë të pagesave ekzistuese. |
| **Merr detajet e një pagese** | Task<Payment> GetPaymentByIdAsync(int paymentId) | - paymentId duhet të jetë një vlerë pozitive (>0). | - Kthen një objekt Payment nëse ekziston.  - Kthen null nëse pagesa nuk ekziston. |
| **Krijon një pagesë të re** | Task<Payment> CreatePaymentAsync(Payment payment) | - payment nuk duhet të jetë null.  - UserId dhe OrderId duhet të jenë të vlefshëm dhe ekzistues në sistem.  - Amount duhet të jetë një vlerë pozitive (>0).  - Payment\_method duhet të jetë një vlerë e vlefshme nga Payment\_Method\_Choices. | - Shton pagesën në sistem dhe kthen objektin Payment. |
| **Përditëson një pagesë ekzistuese** | Task<bool> UpdatePaymentAsync(int paymentId, Payment payment) | - paymentId duhet të jetë një vlerë pozitive (>0).  - payment nuk duhet të jetë null.  - Pagesa me paymentId duhet të ekzistojë në sistem. | - Përditëson të dhënat e pagesës ekzistuese.  - Kthen true nëse përditësimi është i suksesshëm, përndryshe false. |
| **Fshin një pagesë nga sistemi** | Task<bool> DeletePaymentAsync(int paymentId) | - paymentId duhet të jetë një vlerë pozitive (>0).  - Pagesa duhet të ekzistojë në sistem. | - Fshin pagesën nëse ekziston.  - Kthen true nëse fshirja është e suksesshme, përndryshe false. |
| **Menaxhon mënyrat e pagesës** | Task<List<string>> GetPaymentMethodsAsync() | - Nuk ka parakushte. | - Kthen një listë të mënyrave të vlefshme të pagesës (Cash, Card, PayPal). |

**6.2.4 Review Module Responsibilities**

|  |  |
| --- | --- |
| **Model** | **Responsibilities** |
| **Review Entity** | - Përfaqëson një vlerësim të bërë nga një përdorues për një këpucë. - Përmban atributet si ReviewId, UserId, ShoeId, Rating, Comment. |
| **User Entity** | - Përfaqëson përdoruesin që krijon vlerësimin. - Përmban atributet e User Entity. |
| **Shoe Entity** | - Përfaqëson këpucën për të cilën është bërë vlerësimi. - Përmban atributet e Shoe Entity. |
| **Review\_View** | - Përmban logjikën për menaxhimin e vlerësimeve. - Përfshin ReviewListCreateView dhe ReviewDetailView për listimin dhe menaxhimin e detajeve të vlerësimeve. |
| **ReviewListCreateView** | **- get():** Kthen një listë vlerësimesh. **- post():** Krijon një vlerësim të ri. |
| **ReviewDetailView** | **- get():** Merr detajet e një vlerësimi të caktuar. **- put():** Përditëson një vlerësimekzistues. **- delete():** Fshin një vlerësimnga sistemi. |
| **ReviewManagerSingleton** | - Menaxhon operacionet CRUD për vlerësimet. **- create\_review():** Krijon një vlerësim të ri. **- update\_review():** Përditëson një vlerësimekzistues. **- get\_review\_by\_id():** Merr një vlerësim sipas ID-së. **- delete\_review():** Fshin një vlerësim nga sistemi. |
| **Create\_Review** | - Përmban logjikën për krijimin e një vlerësim të ri. |
| **Update\_Review** | - Përmban logjikën për përditësimin e një vlerësim ekzistues. |

**Review Module Specifications**

| **Functionality** | **Syntax** | **Pre** | **Post** |
| --- | --- | --- | --- |
| **Merr listën e vlerësimeve** | Task<List<Review>> GetReviewsAsync() | - Duhet Të ketë Review në databazë | - Kthen një listë me të gjitha vlerësimet ekzistuese. |
| **Merr një vlerësim të caktuar** | Task<Review> GetReviewByIdAsync(int reviewId) | - reviewId duhet të jetë një vlerë pozitive (>0). | - Kthen një objekt Review nëse ekziston.  - Kthen null nëse vlerësimi nuk ekziston. |
| **Krijon një vlerësim të ri** | Task<Review> CreateReviewAsync(Review review) | - review nuk duhet të jetë null.  - UserId dhe ShoeId duhet të jenë të vlefshëm dhe ekzistues në sistem.  - Rating duhet të jetë një vlerë ndërmjet 1 dhe 5. | - Shton vlerësimin në sistem dhe kthen objektin Review. |
| **Përditëson një vlerësim ekzistues** | Task<bool> UpdateReviewAsync(int reviewId, Review review) | - reviewId duhet të jetë një vlerë pozitive (>0).  - review nuk duhet të jetë null.  - Vlerësimi me reviewId duhet të ekzistojë në sistem. | - Përditëson të dhënat e vlerësimit ekzistues.  - Kthen true nëse përditësimi është i suksesshëm, përndryshe false. |
| **Fshin një vlerësim nga sistemi** | Task<bool> DeleteReviewAsync(int reviewId) | - reviewId duhet të jetë një vlerë pozitive (>0).  - Vlerësimi duhet të ekzistojë në sistem. | - Fshin vlerësimin nëse ekziston.  - Kthen true nëse fshirja është e suksesshme, përndryshe false. |

**6.2.5 Wishlist Module Responsibilities**

|  |  |
| --- | --- |
| **Model** | **Responsibilities** |
| **Wishlist Entity** | Përfaqëson një listë dëshirash të përdoruesit. Përdor atributet si WishlistId, UserId, emri. |
| **User Entity** | Përfaqëson përdoruesin që ka listën e dëshirave. Përdor atributet e entitetit të përdoruesit. |
| **Wishlist\_View** | Menaxhon logjikën për shfaqjen e pamjeve të listës së dëshirave. Përfshin WishlistListView dhe WishlistDetailView për listimin dhe menaxhimin e listave të dëshirave. |
| **WishlistListCreateView** | **get():**Merr një listë të të gjitha listave të dëshirave.  **post():**Krijon një listë të re dëshirash për një përdorues. |
| **WishlistDetailView** | **get():**Merr informacion të detajuar për një listë të caktuar dëshirash. **put():**Përditëson detajet e një liste të ekzistuese dëshirash.. **delete():**Fshin një listë dëshirash.. |
| **WishlistManagerSingleton** | Menaxhon operacionet CRUD për listat e dëshirave.   * create\_Wishlist(): Krijon një listë të re dëshirash. * update\_Wishlist(): Përditëson një listë ekzistuese dëshirash. * get\_Wishlist\_by\_id(): Merr një listë dëshirash sipas ID-së. * delete\_wishlistitem(): Fshin një artikull të veçantë nga lista e dëshirave. |
| **Create\_Wishlist** | Përmban logjikën për krijimin e një liste të re dëshirash. (Krijon listën e dëshirave kur një përdorues regjistrohet). |
| **Update\_Wishlist** | Përmban logjikën për përditësimin e një liste ekzistuese dëshirash. |
| **WishlistItem Entity** | Përfaqëson një artikull në listën e dëshirave. Përdor atributet si WishlistItemId, WishlistId, ShoeId. |
| **Shoe Entity** | Përfaqëson një artikull këpucë në sistem. Përdor atributet e entitetit të këpucëve. |
| **WishlistItem\_View** | Menaxhon logjikën për shfaqjen e artikujve të listës së dëshirave. Përfshin WishlistItemListView dhe WishlistItemDetailView. |
| **WishlistItemListCreateView** | **get():**Merr një listë të të gjitha artikujve në listën e dëshirave. **post():**Shton një artikull të ri në listën e dëshirave. |
| **WishlistItemDetailView** | **get():**Merr detaje për një artikull të veçantë në listën e dëshirave.. **put():**Përditëson një artikull të listës së dëshirave. **delete():**Fshin një artikull të listës së dëshirave. |
| **WishlistItemManagerSingleton** | Menaxhon operacionet CRUD për artikujt e listës së dëshirave.   * create\_Wishlistitem(): Krijon një artikull të ri në listën e dëshirave. * update\_Wishlistitem(): Përditëson një artikull ekzistues në listën e dëshirave. * get\_Wishlistitem\_by\_id(): Merr një artikull të listës së dëshirave sipas ID-së. * delete\_wishlistitem(): Fshin një artikull nga lista e dëshirave. |
| **Create\_Wishlistitem** | Përmban logjikën për shtimin e një artikulli të ri në listën e dëshirave. |
| **Update\_Wishlistitem** | Përmban logjikën për përditësimin e një artikulli në listën e dëshirave. |

**Wishlist Module Specifications**

| **Functionality** | **Syntax** | **Pre** | **Post** |
| --- | --- | --- | --- |
| Merr të gjitha listat e dëshirave | Task<List<Wishlist>> GetAllWishlistsAsync() | - Duhet Të ketë Wishlist në databazë | - Kthen të gjitha listat ekzistuese të dëshirave |
| Krijon një listë të re dëshirash | Task<Wishlist> CreateWishlistAsync(Wishlist wishlist) | - wishlist nuk është null  - UserId është valid dhe ekzistues në sistem. | - Lista ruhet në databazë |
| Merr një listë dëshirash me ID | Task<Wishlist> GetWishlistByIdAsync(int wishlistId) | - wishlistId duhet të jetë një vlerë pozitive (>0). | - Kthen listën nëse ekziston, ndryshe null |
| Përditëson një listë ekzistuese dëshirash | Task<bool> UpdateWishlistAsync(int wishlistId, Wishlist updated) | - wishlistId duhet të jetë një vlerë pozitive (>0). - Wishlist nuk duhet të jetë null  - Wishlist me WishlistId duhet të ekzistojë në system. | - Lista përditësohet, true nëse suksesshëm |
| Fshin një listë dëshirash | Task<bool> DeleteWishlistAsync(int wishlistId) | - wishlistId duhet të jetë një vlerë pozitive (>0).  - Wishlist me WishlistId duhet të ekzistojë në system. | - Lista fshihet nëse ekziston |
| Merr të gjithë artikujt në listën e dëshirave | Task<List<WishlistItem>> GetAllWishlistItemsAsync() | - Duhet Të ketë Wishlistitem në databazë | - Kthen listën e artikujve |
| Shton një artikull të ri | Task<WishlistItem> CreateWishlistItemAsync(WishlistItem item) | - item nuk është null  - WishlistId dhe ShoeId janë valide dhe ekzistojn në sistem. | - Artikulli ruhet në databazë |
| Merr një artikull sipas ID-së | Task<WishlistItem> GetWishlistItemByIdAsync(int itemId) | - wishlistitemId duhet të jetë një vlerë pozitive (>0). | - Kthen artikullin ose null |
| Përditëson një artikull ekzistues | Task<bool> UpdateWishlistItemAsync(int itemId, WishlistItem updated) | - wishlistitemId duhet të jetë një vlerë pozitive (>0). - Wishlistitem nuk duhet të jetë null  - Wishlistitem me WishlistitemId duhet të ekzistojë në system. | - Artikulli përditësohet nëse ekziston |
| Fshin një artikull nga lista | Task<bool> DeleteWishlistItemAsync(int itemId) | - wishlistitemId duhet të jetë një vlerë pozitive (>0).  - Wishlistitem me WishlistitemId duhet të ekzistojë në system. | - Artikulli fshihet nëse ekziston |

**6.2.6 Order Module Responsibilities**

|  |  |
| --- | --- |
| **Model** | **Responsibilities** |
| **Order Entity** | - Përfaqëson një porosi të bërë nga një përdorues. - Përmban atribute si OrderId, UserId, OrderDate, Status, Total\_price, shipping\_address. |
| **User Entity** | - Përfaqëson përdoruesin që ka bërë porosinë. - Përmban atributet e entitetit të përdoruesit. |
| **Order\_View** | - Menaxhon logjikën për shfaqjen e pamjeve të porosive. - Përfshin `OrderListView`, `OrderDetailView`, `ActiveOrderView`, `AddToOrderView`, `RemoveFromOrderView`, `FinalizeOrderView`, dhe `PurchaseOrderView` për menaxhimin e porosive. |
| **OrderListView** | **- get():** Merr një listë të të gjitha porosive. **- post():** Krijon një porosi të re për një përdorues. |
| **OrderDetailView** | **- get():** Merr informacion të detajuar për një porosi të caktuar**. - put():** Përditëson informacionin për një porosi ekzistuese. **- delete():** Fshin një porosi. |
| **ActiveOrderView** | - Shfaq porositë aktive që janë në pritje ose në proces. |
| **AddToOrderView** | - Shton një produkt (këpucë) në një porosi aktive. |
| **RemoveFromOrderView** | - Heq një produkt (këpucë) nga një porosi aktive. |
| **FinalizeOrderView** | - Përfundon një porosi dhe e përgatit për dërgesë. |
| **PurchaseOrderView** | - Shfaq dhe mundëson blerjen e porosisë. |
| **OrderManagerSingleton** | **-** Menaxhon operacionet CRUD për porositë. **- create\_Order():** Krijon një porosi të re. **- update\_Order**(): Përditëson një porosi ekzistuese. **- get\_Order\_by\_id():** Merr një porosi sipas ID-së. **- delete\_Order():** Fshin një porosi. **- get\_or\_create\_active\_order():** Merr një porosi aktive ose krijon një të re**. - finalize\_order():** Përfundon një porosi. |
| **Create\_Order** | - Përmban logjikën për krijimin e një porosi të re. |
| **Update\_Order** | - Përmban logjikën për përditësimin e një porosi ekzistuese. |
| **StatusChoices** | - Përmban opsione statusi: `pending`, `shipped`, `delivered`, `cancelled`. |
| **Orderitem Entity** | - Përfaqëson një produkt (këpucë) në një porosi. - Përmban atributet si OrderId, ShoeId, Quantity, Price. |
| **Shoe Entity** | - Përfaqëson një produkt këpucë në sistem. - Përmban atributet e entitetit të këpucëve. |
| **Orderitem\_View** | - Menaxhon logjikën për shfaqjen e produkteve të porosisë. - Përfshin `OrderitemListView` dhe `OrderitemDetailView`. |
| **OrderitemListCreateView** | **- get():** Merr një listë të të gjitha produkteve të porosisë. **- post():** Shton një produkt të ri në një porosi. |
| **OrderitemDetailView** | **- get():** Merr informacion të detajuar për një produkt të caktuar në një porosi. **- put():** Përditëson informacionin e një produkti të porosisë. **- delete():** Fshin një produkt nga porosia. |
| **OrderitemManagerSingleton** | - Menaxhon operacionet CRUD për produktet e porosisë. **- create\_Orderitem():** Krijon një produkt të ri në porosi. **- update\_Orderitem():** Përditëson një produkt të ekzistueshëm në porosi. **- get\_Orderitem\_by\_id():** Merr një produkt të porosisë sipas ID-së. **- delete\_Orderitem**(): Fshin një produkt nga porosia. |
| **Create\_Orderitem** | - Përmban logjikën për shtimin e një produkti të ri në porosi. |
| **Update\_Orderitem** | - Përmban logjikën për përditësimin e një produkti ekzistues në porosi. |

**Order Module Specifications**

| **Functionality** | **Syntax** | **Pre** | **Post** |
| --- | --- | --- | --- |
| Merr të gjitha porositë | Task<List<Order>> GetAllOrdersAsync() | - Asnjë | - Kthen listën e të gjitha porosive |
| Krijon një porosi të re | Task<Order> CreateOrderAsync(Order order) | - order nuk është null  - UserId valid dhe ekzistues në sistem. | - Porosia ruhet në databazë |
| Merr një porosi me ID | Task<Order> GetOrderByIdAsync(int orderId) | - OrderId duhet të jetë një vlerë pozitive (>0). | - Kthen porosinë ose null |
| Përditëson një porosi ekzistuese | Task<bool> UpdateOrderAsync(int orderId, Order updated) | - OrderId duhet të jetë një vlerë pozitive (>0). - Order nuk duhet të jetë null  - Order me OrderId duhet të ekzistojë në system. | - Porosia përditësohet nëse ekziston |
| Fshin një porosi | Task<bool> DeleteOrderAsync(int orderId) | - OrderId duhet të jetë një vlerë pozitive (>0). | - Porosia fshihet nëse ekziston |
| Merr ose krijon porosi aktive për përdoruesin | Task<Order> GetOrCreateActiveOrderAsync(int userId) | - userId valid dhe ekzistues ne sistem. | - Kthen porosinë aktive ekzistuese ose krijon një të re |
| Përfundon një porosi | Task<bool> FinalizeOrderAsync(int orderId) | - Porosia duhet të jetë në statusin pending | - Statusi ndryshohet në shipped, delivered ose cancelled |
| Merr porositë aktive | Task<List<Order>> GetActiveOrdersAsync(int userId) | - userId valid dhe ekzistues ne sistem. | - Kthen të gjitha porositë aktive për përdoruesin |
| Blerja e porosisë (Checkout) | Task<bool> PurchaseOrderAsync(int orderId) | - Porosia duhet të jetë finalized | - Kryhet blerja dhe ndryshon statusi në delivered |
| Shton një produkt në porosi | Task<OrderItem> AddToOrderAsync(int orderId, OrderItem item) | - orderId valid  - item valid  - Porosia është aktive | - Artikulli shtohet në porosi |
| Heq një produkt nga porosi | Task<bool> RemoveFromOrderAsync(int orderItemId) | - orderitemId duhet të jetë një vlerë pozitive (>0). | - Artikulli fshihet nëse ekziston |
| Merr të gjithë produktet në porosi | Task<List<OrderItem>> GetAllOrderItemsAsync(int orderId) | - orderitemId valid dhe ekzistues ne sistem | - Kthen të gjitha produktet në porosi |
| Merr një produkt të caktuar në porosi | Task<OrderItem> GetOrderItemByIdAsync(int itemId) | - orderitemId duhet të jetë një vlerë pozitive (>0). | - Kthen produktin ose null |
| Përditëson një produkt ekzistues në porosi | Task<bool> UpdateOrderItemAsync(int itemId, OrderItem updated) | - OrderitemId duhet të jetë një vlerë pozitive (>0). - Orderitem nuk duhet të jetë null  - Orderitem me OrderitemId duhet të ekzistojë në system. | - Produkti përditësohet në porosi |
| Fshin një produkt nga porosia | Task<bool> DeleteOrderItemAsync(int itemId) | - OrderitemId duhet të jetë një vlerë pozitive (>0). | - Produkti fshihet nga porosia nëse ekziston |

**6.3 Mapping Between Models**

| **Presentation Layer (Controllers, Frontend, Views)** | **Business Logic Layer (Services, Serializers, Path Config.)** | **Data Access Layer (Models, Migrations, Config)** |
| --- | --- | --- |
| **Wishlist\_view** (Menaxhon krijimin dhe shikimin e wishlisteve) | **WishlistSerializer** (Përpunon logjikën e wishlistave) | **WishlistModel** (Përfaqëson një wishlist) |
| **WishlistItem\_view** (Menaxhon artikujt e wishlistave) | **WishlistItemSerializer** (Përpunon logjikën e wishlist items) | **WishlistItemModel** (Përfaqëson një artikull wishlisti) |
| **Order\_view** (Menaxhon porositë e përdoruesve) | **OrderSerializer** (Përpunon logjikën e porosive) | **OrderModel** (Përfaqëson një porosi) |
| **OrderItem\_view** (Menaxhon artikujt e porosive) | **OrderItemSerializer** (Përpunon logjikën e artikujve të porosive) | **OrderItemModel** (Përfaqëson një artikull porosie) |
| **Payment\_view** (Menaxhon pagesat) | **PaymentSerializer** (Përpunon logjikën e pagesave) | **PaymentModel** (Përfaqëson një pagesë) |
| **Review\_view** (Menaxhon komentet dhe vlerësimet) | **ReviewSerializer** (Logjika për menaxhimin e komenteve) | **ReviewModel** (Përfaqëson një koment ose vlerësim) |
| **Shoe\_view** (Menaxhon këpucët në sistem) | **ShoeSerializer** (Logjika për menaxhimin e këpucëve) | **ShoeModel** (Përfaqëson një produkt këpucësh) |
| **User\_view** (Menaxhon përdoruesit dhe profilet) | **UserSerializer** (Logjika për menaxhimin e përdoruesve) | **UserModel** (Përfaqëson një përdorues) |

#### Përshkrimi i shtresave:

* **Presentation Layer**: Përmban Controllers, Frontend dhe Views që trajtojnë ndërveprimin me përdoruesit.
* **Business Logic Layer**: Përmban Services, Serializers dhe Path Config që përpunojnë logjikën e biznesit.
* **Data Access Layer**: Përmban Models, Migrations dhe Config për ruajtjen e të dhënave dhe konfigurimet e databazës.

**6.4 Detail Design Rationale**Funksionalitetet e ndryshme të sistemit janë ndarë në module të pavarura si **Wishlist Module, Order Module, Payment Module, Review Module**, etj., duke siguruar një dizajn modular. Kjo ndarje u bë për të përmirësuar shkallëzimin dhe për ta bërë më të thjeshtë mirëmbajtjen e sistemit.

Për shembull, **Order Module** përqendrohet në menaxhimin e porosive dhe ndërvepron me module të tjera si **User Module** për të marrë informacionin e përdoruesve dhe **Payment Module** për të përpunuar pagesat. Ndarja e qartë e përgjegjësive midis moduleve siguron që çdo modul të mund të zhvillohet dhe testohet në mënyrë të pavarur.

Arkitektura e ndjekur për këtë sistem bazohet në parimet e **Model-View-Template (MVT)**, duke ndarë **logjikën e biznesit (Model), ndërfaqen e përdoruesit (View), dhe menaxhimin e të dhënave përmes Templates**. Ky model ndarje rrit modularitetin dhe siguron ripërdorshmëri më të lartë të kodit.

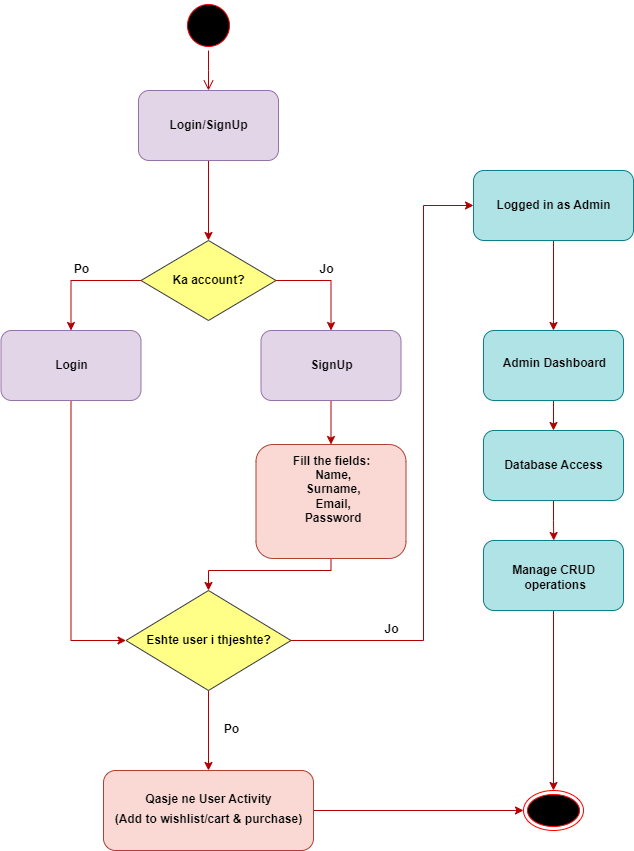
Modulet si **Review Module** dhe **Wishlist Module** janë zhvilluar në mënyrë që të jenë të pavarura nga pjesët e tjera të sistemit. Kjo i lejon zhvilluesit të shtojnë ose të modifikojnë funksionalitete specifike (p.sh., përmirësimin e mekanizmit të vlerësimeve) pa ndikuar në pjesët e tjera të sistemit.

Në **Order Module**, ndarja midis **Model, View, dhe Template** siguron që ndryshimet në logjikën e porosive (Model) ose në mënyrën e paraqitjes së informacionit të porosive (View) nuk ndikojnë në logjikën e lidhjes së tyre (Template). **Template Engine** siguron gjithashtu që ndërfaqet e ndryshme (p.sh., aplikacioni web dhe dashboard-i i administratorit) mund të përdorin të njëjtën logjikë pa konflikte.

Ndarja në module dhe përdorimi i **MVT** e bën më të lehtë përfshirjen e funksionaliteteve të reja. Modulariteti dhe dizajni i pastër e bëjnë më të lehtë mirëmbajtjen e kodit dhe kryerjen e testeve unitare. Çdo modul mund të testohet veçmas për të siguruar që funksionaliteti i tij është i saktë dhe nuk ndikon në pjesët e tjera të sistemit. Dizajni modular i bazuar në parimet **MVT** rrit gjithashtu fleksibilitetin dhe aftësinë për të përballuar kërkesat në rritje të sistemit në të ardhmen.

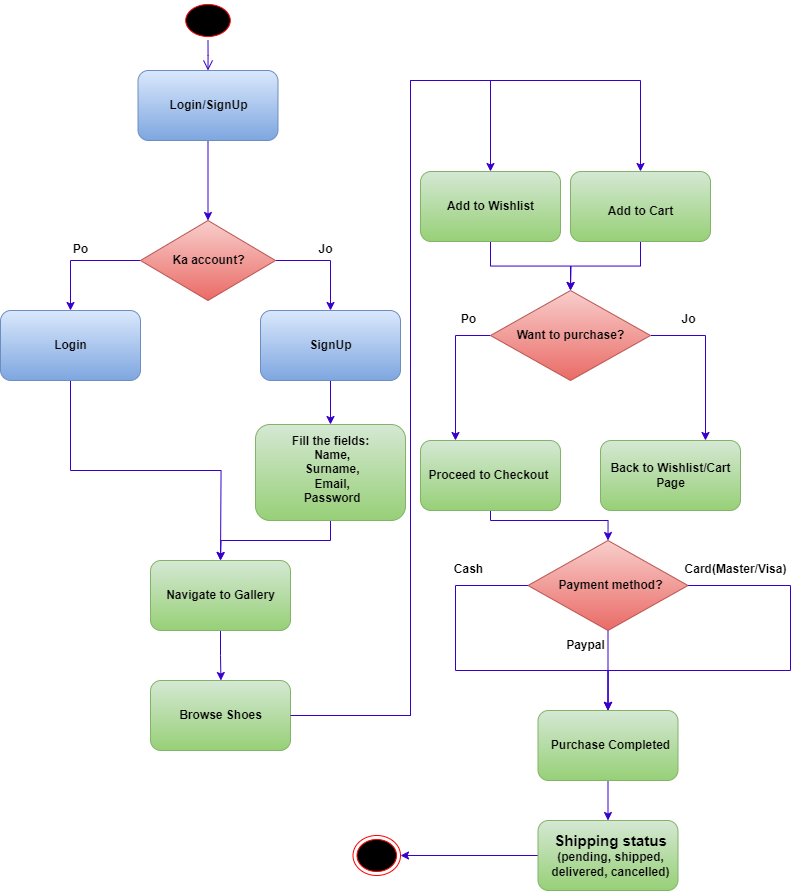
**7. Activity Diagrams**

**7.1 User Authentication**



Ky diagram përshkruan procesin e autentifikimit dhe aktivitetit të përdoruesit në platformën **LuxuryStep**. Fillon me një përdorues që zgjedh mes **Login/SignUp**.Nëse tashmë ka një llogari, ai hyn drejtpërdrejt, përndryshe duhet të plotësojë fushat për regjistrim (**Name, Surname, Email, Password**). Pas kësaj, sistemi kontrollon nëse përdoruesi është i thjeshtë apo administrator. Nëse është përdorues i thjeshtë, ai merr qasje në veprimtarinë e tij, si **shtimi në wishlist, shporta dhe blerja**. Nëse është administrator, ai hyn në **Admin Dashboard**, ku ka akses në bazën e të dhënave dhe mund të kryejë operacione **CRUD** për menaxhimin e sistemit. Diagrami përfundon me një pikë terminale që tregon fundin e rrjedhës së procesit.

**7.2 Client shoe-purchase process**

****

Procesi i blerjes së këpucëve nga klienti fillon me **Login/SignUp**, ku përdoruesi duhet të hyjë ose të regjistrohet. Pasi të identifikohet, ai navigon në galerinë e produktit dhe shfleton këpucët.

Përdoruesi ka mundësinë të shtojë produktet në **Wishlist** ose **Cart**. Nëse zgjedh të bëjë blerjen, ai procedon me checkout, përndryshe mund të kthehet në wishlist ose cart. Gjatë procesit të checkout-it, përdoruesi zgjedh metodën e pagesës: **Cash**, **Card (Master/Visa)** ose **Paypal**.

Pasi të kryhet pagesa, blerja përfundon dhe porosia kalon në statusin e dërgesës, e cila mund të jetë: **pending, shipped, delivered, ose cancelled**.

**7.4 Sequence Diagram**

Ky sequence diagram paraqet procesin e autentifikimit dhe përdorimit të funksioneve kryesore në platformën *LuxuryStep*. Pasi përdoruesi hyn në sistem (ose i kërkohet të regjistrohet nëse nuk ka një llogari valide), ai mund të shfletojë këpucët e disponueshme. Në varësi të zgjedhjes, përdoruesi mund t’i shtojë këpucët ose në Wishlist, ose direkt në Cart. Nëse një artikull ndodhet në Wishlist, përdoruesi ka mundësi ta transferojë në Cart kur dëshiron ta blejë. Më pas, nga Cart-i, kryhet procesi i pagesës për finalizimin e porosisë. Diagrami ilustron bashkëpunimin midis komponentëve si “Authentication Service”, “Shoes” (për marrjen e informacionit rreth këpucëve), “Wishlist” dhe “Cart/Order” (për menaxhimin e artikujve), si dhe “Payment” (për transaksionin përfundimtar). Ky rrjedhë e integruar siguron një përvojë të plotë, nga hyrja në sistem deri te përfundimi i blerjes.

