

**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.Ride Planner Detailed Desing Document 31](#_Toc187702186)

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

[6.1.1 User Modules Responsibilities 32](#_Toc187702188)

[6.1.2 User Module Specifications 33](#_Toc187702189)

[6.2.1 Bus Schedule Module Responsibilities 34](#_Toc187702190)

[5.2.2 Bus Schedule Module Specifications 35](#_Toc187702191)

[5.2.2 Bus Schedule Module Specifications 36](#_Toc187702192)

[5.3.1 TaxiReservation Modules Responsibilities 38](#_Toc187702193)

[5.3.2 TaxiReservation Module Specifications 39](#_Toc187702194)

[5.4.1 TaxiDriverDashboard Modules Responsibilities 39](#_Toc187702195)

[5.4.2 TaxiDriverDashboard Module Specifications 40](#_Toc187702196)

[6.5 Low-Level Desing 42](#_Toc187702197)

[6.5.2 Detailed View of the Modules of Low Levels 43](#_Toc187702198)

[6.6 Mapping Beetwen Models 44](#_Toc187702199)

[6.7 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)

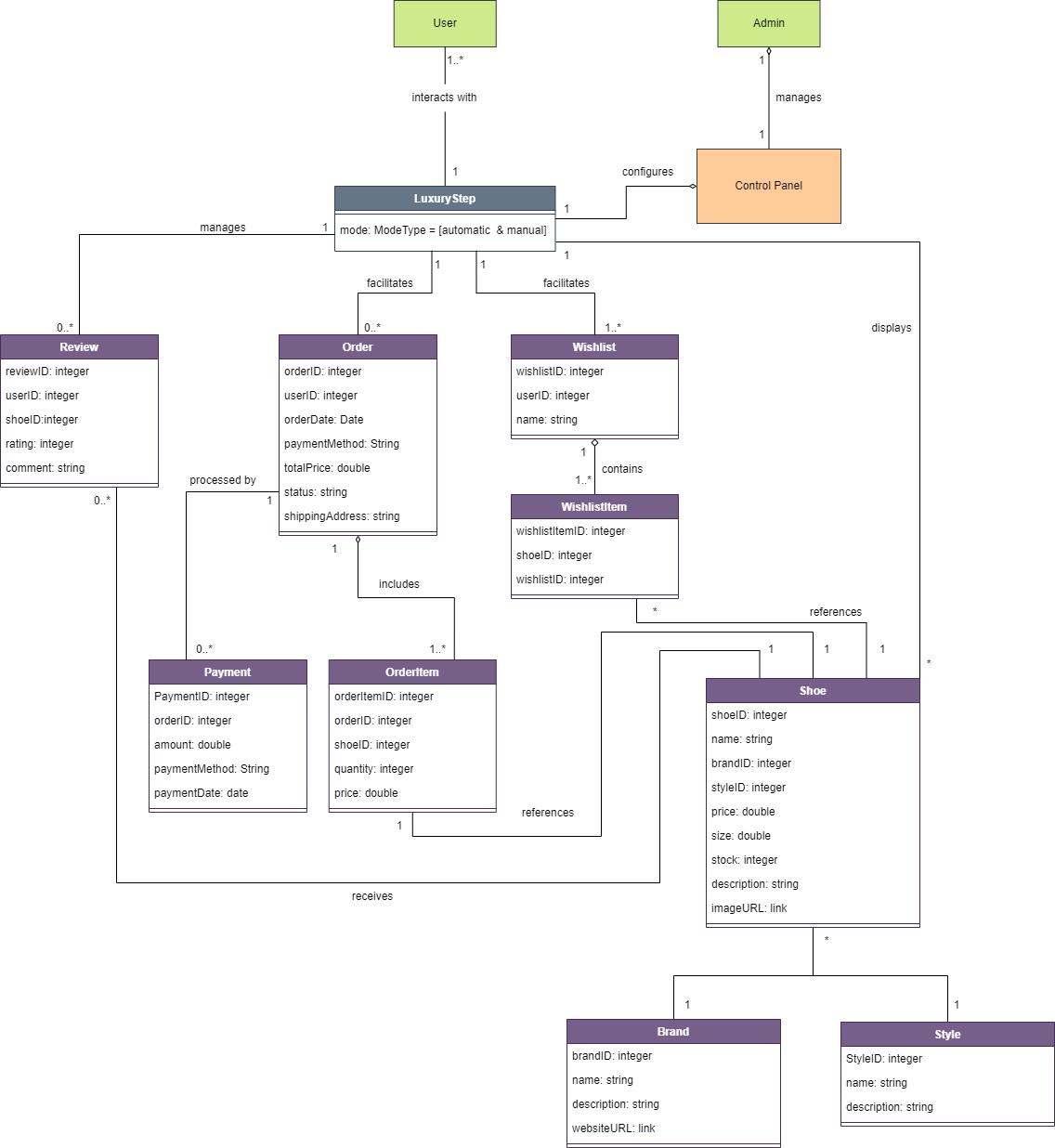
[7.2.2 Taxi Reservation 52](#_Toc187702207)

1. **Hyrje**

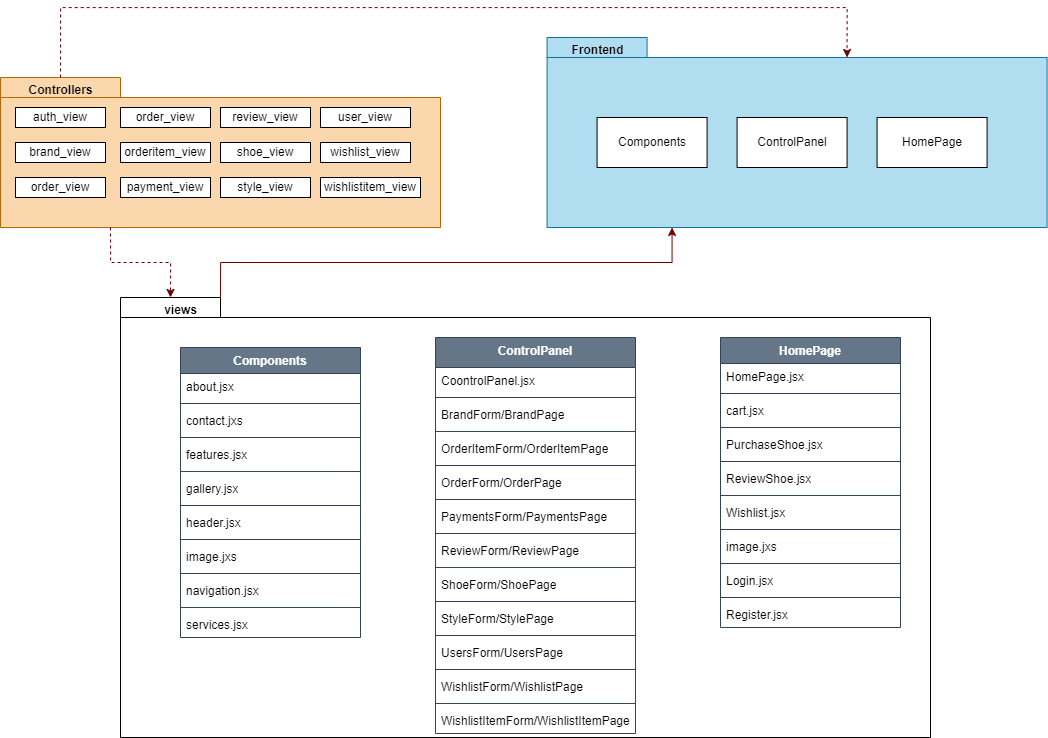
**LuxuryStep** është një platformë inovative që kombinon fuqinë e teknologjisë me pasionin për këpucët luksoze, duke krijuar një hapësirë ku përdoruesit mund të eksplorojnë, rishikojnë dhe blejnë produkte me cilësi të lartë. E ndërtuar me **Django** për backend dhe **React** për frontend, **LuxuryStep** 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:

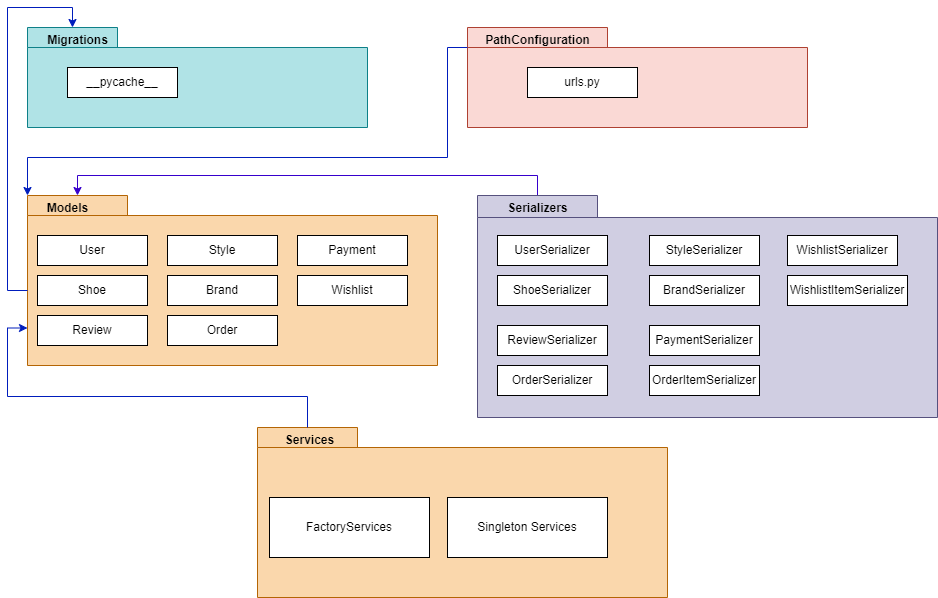
1. **Lehtësimi i Zgjedhjes për Blerësit** – Përdoruesit mund të lexojnë rishikime autentike dhe të bazuara në eksperiencën e konsumatorëve të tjerë para se të marrin një vendim për blerje.
2. **Rritja e Besimit përmes Transparencës** – Çdo produkt ka një vlerësim të detajuar dhe rishikime të verifikuara 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**

****

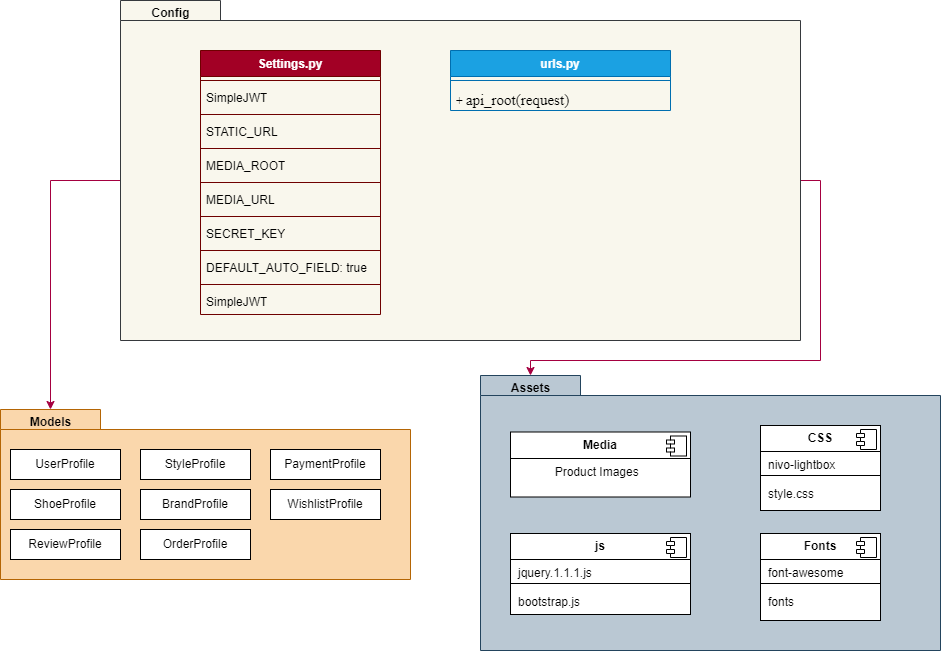
* 1. **Conceptual Model-Presentation Layer**

****

* 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.

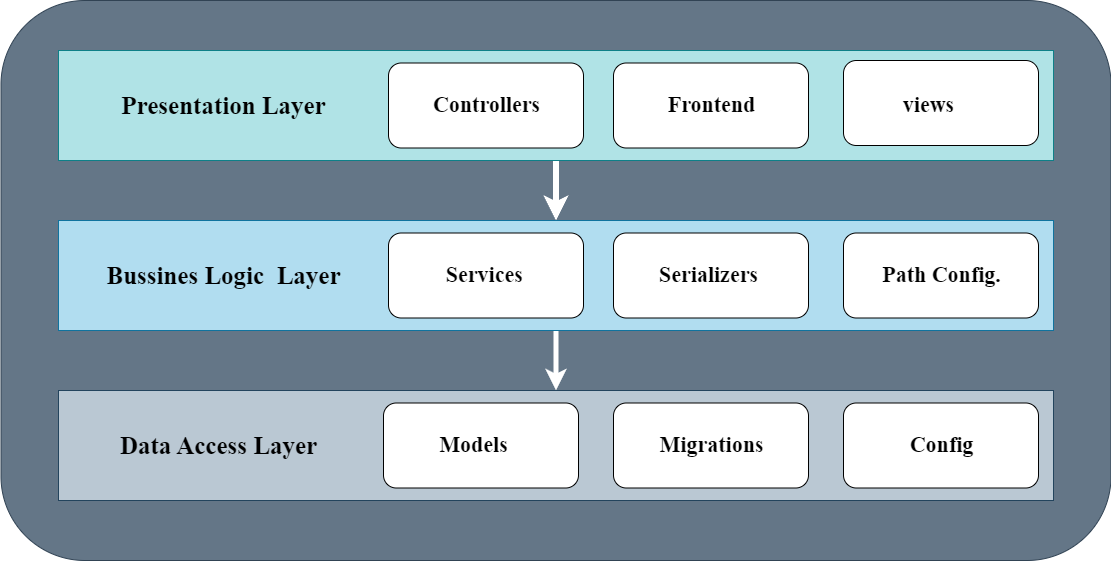
 **Ndërfaqe Interaktive për Përdoruesit:**

* 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**

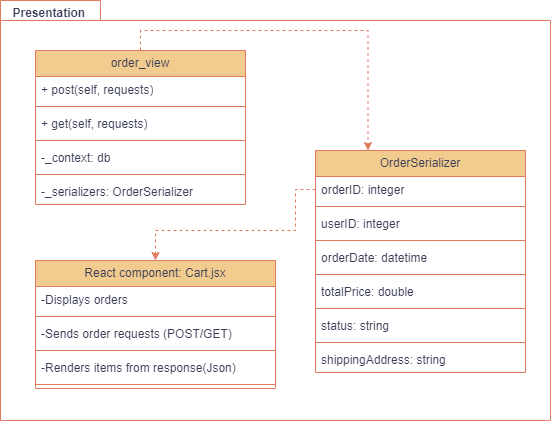
|  |  |
| --- | --- |
| **Layer** | **Përshkrimi** |

|  |  |
| --- | --- |
| **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.

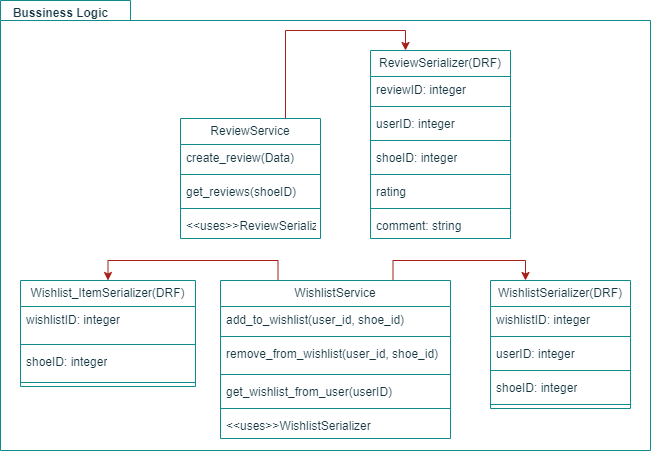
**5.2.2 Presentation Layer Rationale**

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.

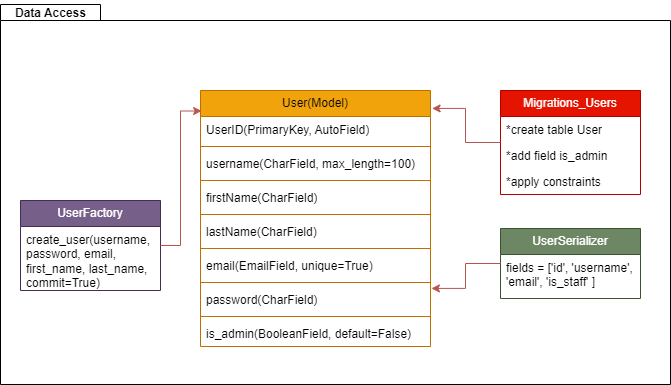
### **Përfitimet e këtij dizajni**

* **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:

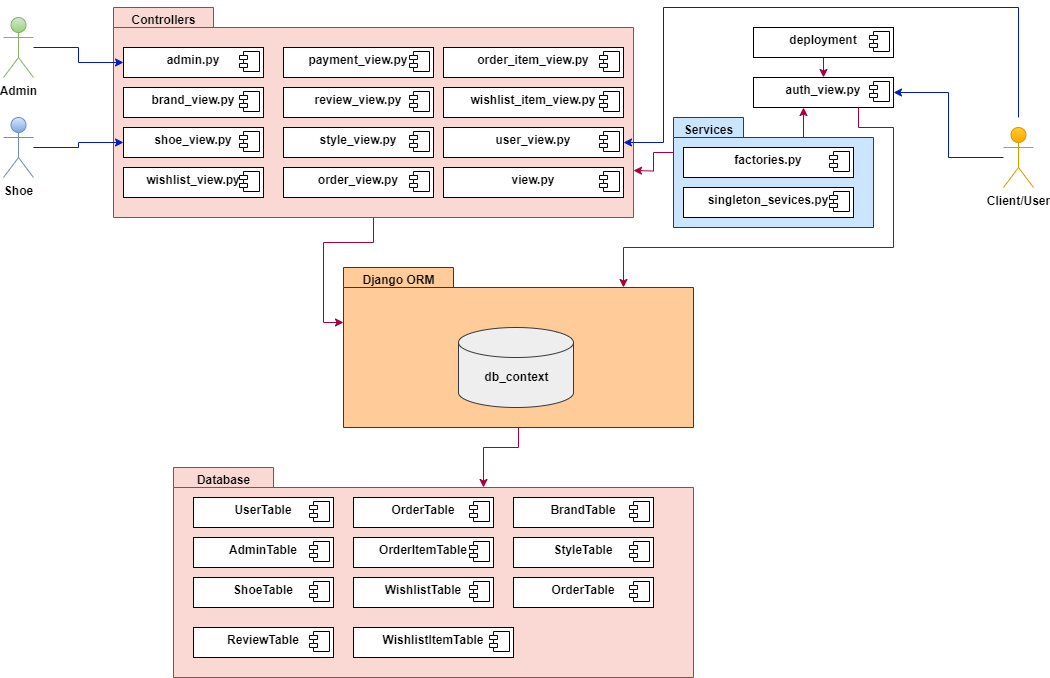
* Menaxhimin e modeleve te te dhenave(Data Models)
* Migrimet(Migrations)
* Filtrat golbale(Global Query Filters)

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**

**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 PostgreSQL/MySQL)

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 (Django ORM & PostgreSQL/MySQL)**

* 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**
* PostgreSQL ose MySQL përdoren 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.

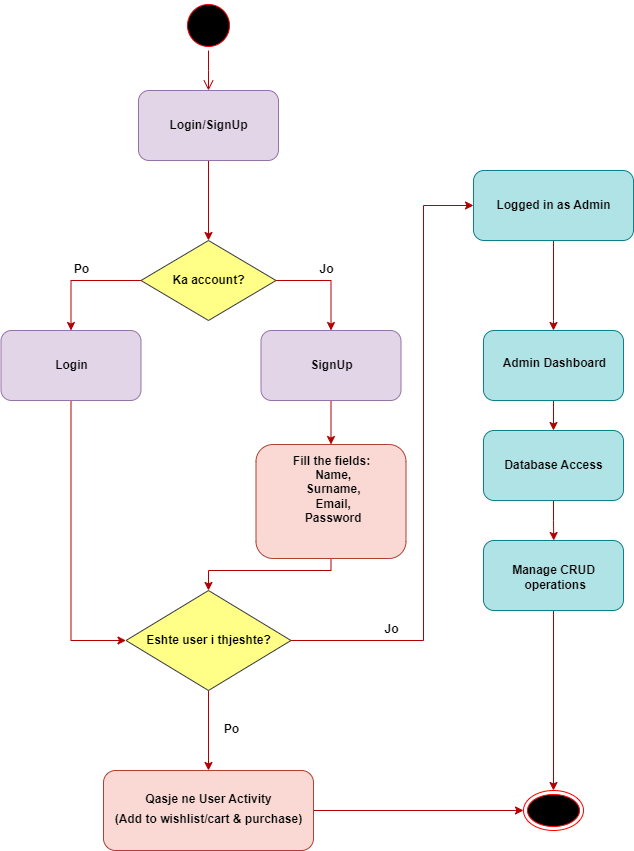
## **6.LuxuryStep Detailed Design Document**

## **6.1 Mid Level Design Models**

****

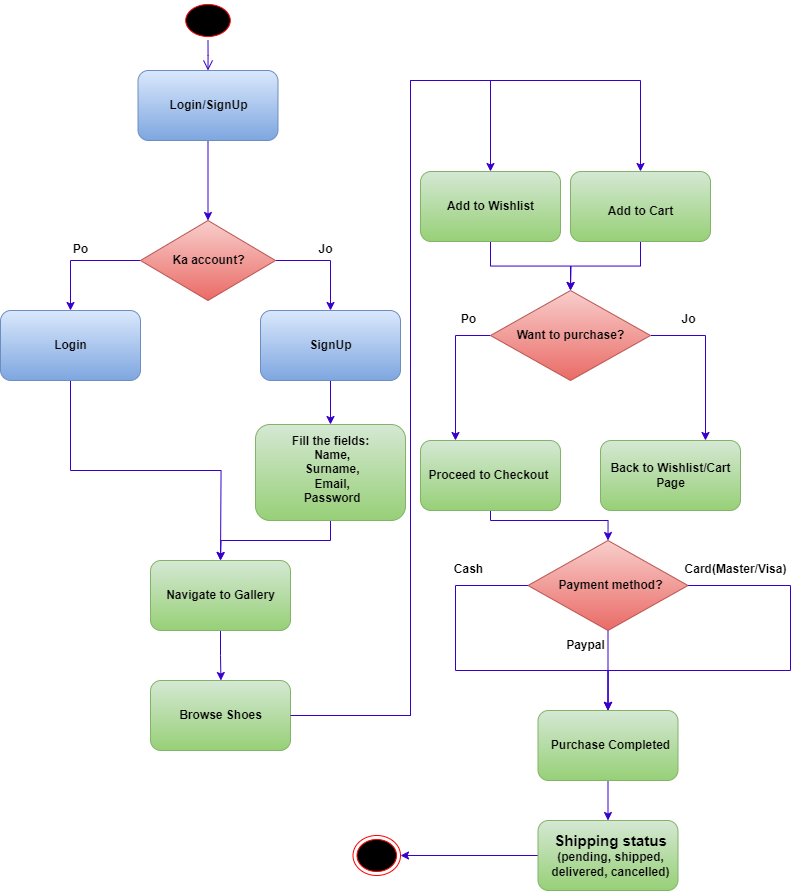
**7. Activity Diagrams**

**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**.