АКАДЕМИЈА ТЕХНИЧКО-УМЕТНИЧКИХ СТРУКОВНИХ СТУДИЈА БЕОГРАД

ОДСЕК ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА

Рашовић Коста

Пајтон апликација за продавницу

- завршни рад -

Београд, јун 2023.

Кандидат: **Рашовић Коста**

Број индекса: **РТ-68/19**

Студијски програм: **Рачунарска техника**

Тема: **Пајтон апликација за продавницу**

Основни задаци:

1. **Опис елемената програмског језика пајтон који је коришћен у раду**
2. **Имплементација пајтон апликације за продавницу**
3. **Опис корисничког интерфејса**

Ментор:

Београд, месец 2023 године.

_____

др Перица Штрбац, проф. АТУС

РЕЗИМЕ:

У дипломском раду описани су елементи програмског језика пајтон као и опис корисничког интерфејса кроз имплементацију пајтон апликације за продавницу у Ђанго оквору. Описане су све фазе процеса, од израде Базе података, пропратног „API" све до крајње фронт-енд стране.

Кључне речи: : пајтон, база података, „API", фронт-енд страна, Ђанго.

ABSTRACT:

The thesis describes the elements of the Python programming language, as well as the description of the user interface through the implementation of Python applications for a store.This project was implemented trought Django framework, for database was used SQLite3 while frontends side was implemented trought HTML with CSS styling. All phases of the process are described, from database creation and API development to the final front-end side on the web application.

Key words: Python, database, API, front-end side, Django. HTML,CSS.

САДРЖАЈ:

1. УВОД

У данашњем дигиталном добу, интернет апликације (*web application*) су постале неизоставни алати за различите области пословања. Једна од најзначајнијих области у којој се интернет апликација највише користе јесте продаја или е-трговина. Сврха коришћења интернет продавница је да омогуће ефикасан, брз и практичан начин куповине и продаје производа или услуга на интернету. Развој интернета довео је до значајног повећања коришћења интернет апликација за продавницу. Широка доступност интернета омогућила је продавницама да циљају глобално тржиште и да досегну широку публику која предходно није била доступна.

Један од кључних фактора који је олакшао израду интернет апликација за продавницу је развој „Django" оквира. „Django" је моћани високо продуктиван оквир за израду веб апликација, који омогућава брзо и ефикасно креирање комплексних и скалабилних апликација. Са својом јасном структуром и уграђеним функционалностима, „Django" пружа програмерима алате за олакшану израду апликација за продавницу.

Како би и сам приказ корисничког интерфејса био имплементиран кроз Ђанго оквир, коришћени су ђанго шаблони (*енгл. Templates*). Који служе за рендеровање HTML елемената на страницу директно кроз ђанго оквир.

Да би сам приказ странице био стилизован, коришћен је CSS.

За чување података из апликације направљена је SQLite3 база података.

У Поглављу 2 дат је преглед и опис свих технологија, библиотека, пакета, као и функција коришћених у изради апликације

У Поглављу 3 обрађена је имплементација апликације, свих њених функционалности и могућности, као и начин функционисања базе података у којој су складиштени подаци везани за апликацију.

У Поглављу 4 описан је кориснички интерфејс, његова израда као и сами изглед истог.

У Поглављу 5 представљен је закључак уз осврт на коришћену технологију, као и потенцијална рјешења за даљи ток развоја апликације.

2. ОПИС БИБЛИОТЕКЕ ЂАНГО И ЊЕГОВИХ МОДУЛА

Апликација је развијена коришћењем Ђанго оквира. Разне функционалности имплементиране су уз помоц следећих модула:

- *Django*
- *Django db.models*
- *Django http*
- *Django urls*
- *Django views*
- *Django mail*
- *Django Session*

Због лакше имплементације одређених функционалности и акција, направљена је скрипта са *JavaScript* кодом.Такође, због љепшег стилизовања корисничких страница написани су *CSS* статички фајлови.

2.1. DJANGO

Django је популаран *open-source* веб оквир који се користи за развој брзих и скалабилних веб апликација. Написан је у python програмског језику, док су његови творци Adrian Holovaty и Jacob Kaplan-Mossom, а први пут је представљен јавности 2005. год. Он пружа скуп алата и библиотека који олакшавају израду апликација. Ђанго има много компоненти и функционалности које олакшавају развој апликација, као што су:

1. *ORM (Object-Relational Mapping):*Слој који омогућава програмерима да манипулишу базом података користећи пајтон објекте, што уклања потребу за писањем *SQL* упита.

2. *URL* рутирање: Ђанго омогућава да се дефинишу УРЛ-ви апликације на једноставан начин, мапирајући их на одговарајуће функције и класе које обрађују захтјеве корисника.

3. Системски шаблон: Ђанго користи свој властити системски (*template system*) који омогућава програмерима да стварају динамички генерисане *HTML* шаблоне за приказивање података корисницима.

4. Административна страна: Ђјанго долази с уграђеном административном страном која омогућава брзо стварање функционалности за управљање садржајемвеб апликације.

Ђанго прати MVC (Model-View-Control) архитектуру, гдје су модели (modles) одговорни за представљање податак аи пословне логике, погледи (views) су задужени за интеракцију с`корисничким захтјевима и генерисањем одговарајућих одговора, док је контрола (contorl) одговорна за управљање током апликације.

Сам развој апликације рађен је у програму *Visual Studio Code* који је креирао виртуелно окружење. Након тога инсталиран је ђанго користећи следећу команду u терминалу:

```
pip install django
```

Након инсталације ђанга било је потребно креирати ђанго пројекта, што је учињено следећом командом:

```
django-admin startprject Store
```

Ова команда ће створити основну структуру пројекта са именом „Store". Унутар пројекта биће креирана скрипта *manage.py* која омогућава управљање различитим аспектима пројекта, као сто су покретање развојног сервера, извршавање миграција базе података и управљање командама специфичним за ђанго, као што је и наредна команда у низу за креирања апликације. Потребно је креирати ђанго апликацију која представља модуларну компоненту пројекта а у даљем току имплементације обављати ће специфичне задатке везане за саму апликацију.

Позиционирањем у креирани пројекат у терминалу позивамо команду за креирање апликације:

```
python manage.py startapp bmdSite
```

2.1.1. Render

Рендер функција у ђангу је основна функција која се користи за генерисање *HTTP* одговора који се шаљу кориснику. Она комбинује *HTML* шаблоне са подацима и генерише одговарајући *HTML* који се приказује у прегледачу.

Render функција има следећи облик:

```
render(request, template_name, context, content_type, status, using)
```

- *Request*: Захтјев који се прослиједјује функцији и који садржи информације о корисничком захтјеву.
- *template_name*: Име шаблона који се користи за генерисање *HTML*-а
- *context*: Опциони аргумент који садржи податке који се прослиједјују шаблону како би се попунили динамички дијелови *HTML-a*
- *content_type:* Опциони аргумент који одеђује тип садржаја који се шаље као одговор
- *status:* Опциони аргумент који дефинише *HTTP* статус код одговора (нпр. *status=200* за успјешан, *status=404* за грешку).
- *using*: Опциони аргумент којидефинише име базе података коју желимо да користимо приликом операције читања података.

2.2. DJANGO MODELS

Modles пакет је кључни дио *farmework-a* који омогућава дефинисање

модела података и интеракцију са базом података. Овај пакет пружа разне класе и функционалности које олакшавају рад са подацима, као што су: дефинисање структуре података, интеракција са базом података и манипулација подацима кроз моделовање објеката.

Неке од функција и класа су:

- *Model*
- *Fields*
- *Migrations*
- *QuerySet*
- *Методе*

2.2.1. Model

Model у ђангу представља кључну компоненту која омогућава дефинисање структуре података и интеракцију са базом података. Модели су класе које описују ентитете или концепт у апликацији. Такође омогућује бољу организацију, приступ и манипулацију над подацима на систематичан начин.

Поред дефинисања атрибута, модели такође подржавају дефинисање метода које описују понашање објеката модела. Ове операције могу извршавати различите операције над подацима и пружити додатне функционалности.

2.2.2. Fields

Fields (поље) служе за дефинисање атрибута или карактеристика модела. Они одређују тип податка који ће бити чуван за одређено поље и пружају информацију о томе како ће се ти подаци чувати у бази података.

Неки од типова поља су:

- *CharField:* поље које се користи за чување податка који је тип *char*
- *IntegerField:* поље које се користи за чување податка који је тип *intiger*
- *ImageField:* поље које се користи за чување податка који је тип *image*
- *DateField* :поље које се користи за чување податка који је тип *date*

Неке од додатних опција које се могу користити приликом дефинисања атрибута или карактеристика модела су:

- *null:*поље које дефинише да ли је дозвољена *null* вриједност у бази
- *default:* поље које дефинише *default* вриједност код сачувавања податка у базу
- *unique:* поље које дефинише да податак има јединствену вриједност

- *max_length:* поље које дефинише максималну величину податка
- *related_name:* поље које додјељује име приликом коришћења податка

2.2.3. Migartions

Migrations (миграције) су механизам који омогућава аутоматско ажурирање базе података како би се одржале примјене у моделима. Одговорне су за синхронизацију између дефиниције модела и стварне структуре табеле у бази података.

Ради тако што приликом дефинисања или измјене модела креира миграционе скрипте које описују кораке које треба предузети да би се модели промјенили у бази података. Тј. свака миграција представља логички корак који може укључивати креирање, измјену или брисање табела или поља.

Приликом креирања базе података потребно је генерисати миграционе фајлове следећом командом:

```
python manage.py makemigrations
```

Да би примјенили миграције на базу података потребно је позвати следећу команду:

```
python manage.py migrate
```

Миграције чине процес ажурирања и одржавања базе података ефикаснијим и поузданијим, штедећи вријеме и смањујући ризик од грешке.

2.2.4. QuerySet

QuerySet представља кључни концепт у Ђанго-овом *ORM*-у(*Object-Relational Mapping*) који представља резултат упита над базом податак. *QuerySet* омогућава да се лакше изврше сложени упити, манипулација над подацима као и лакши приступ истим са великом ефикасношћу.

Неке од метода су:

- *filter():* филтрира податке по предходно постављеним критеријумима
- *update(): ажурирање података у бази*
- *delete(): брисање објекта из базе*
- *create(): креирање нових објеката у бази*
- *order_by(): сортирање објеката у резултату упита на основу одређеног поља*
- *count(): враћа број објеката у резултату упита*
- *sum(): сабирање нумеричких вриједности*
- *avg(): израчунавање просјечне вриједности*

Један од примјера коришћења QuerySet функционалности из практичног

дијела:

```
store= Article.objects.filter(store__slug=slug).exclude(quantity__lte=1)
```

Наиме, у примјеру изнад враћа се објекат који је филтриран тако да има исту вриједност *slug* промјењиве са вриједношћу *slug* у *store* табели у бази података, с`тим што подлијеже још једном критеријуму гдје *quantity* поље у бази није мање од 1.

2.2.5. Методе

Методе у *models* пакету су функције које се дефинишу унутар модела и пружају додатну функционалност за рад са објектима тог модела. Ове методе се могу користити за извршавање различитих операција, манипулацију подацима или пружање додатних информација.

Неке од уобичајних метода које се могу дефинисати унутар модела су:

1. *__str__():* Ова метода дефинише како ће се објекат приказивати приликом конверзије у *string* тип. Често се користи за приказивање читљивих информација о објекту. Један од примјера је Admin страна на којој називе табела и њихове ентитете можемо приказати на читљив начин кроз имплементирање ове методе.

2. *Custom* методе: Ове методе се могу дефинисати по сопственом нахођењу унутар модела. И могу обављати специфичне операције над објектима тог модела. Као што је то:

```
def get_store(self):
    return self.sotore.name
```

Метода враћа *name* атрибут објекта.

2.3. DJANGO HTTP

У пакету *http* налази се скуп класа, функција и константи кое пружају функционалности за манипулацију *HTTP* захтјевима и одговорима у ђанго веб апликацији. Омогућава размјену података између сервера и клијента.

Неке од кључних компоненти пакета *http су:*

- *HttpRequest*
- *HttpResponse*
- *HttpResponseRedirect*

2.3.1. HttpRequest

Класа *HttpRequest* је компонента у ђанго-у која представља *HTTP* захтјев који је примљен од стране сервера. Ова класа пружа интерфејс за приступ различитим дијеловима захтјева и омогућава манипулацију над тим подацима.

Функционалности класе су:

- Метода захтјева

- УРЛ и путања
- Подаци о сесији
- Подаци из форме

2.3.1.1. Методе захтјева

Метода захтјева тј. *method* је атрибут објекта *HttpRequest* који садржи *HTTP* методу захтјева који је примљен од стране сервера, и та информација идентификује врсту операције коју клијент жели да изврши над ресурсом.

Методе могу бити:

- *GET*:

Метода *GET* се користи за добављање података са сервера. Када клијент шаље *GET* захтјев, тражи се ресурс са задатим УРЛ-ом. Користи се за приказивање страница или добављање података из базе.

- *POST:*

Метода *POST* се користи за слање података на сервер како би се извршила нека акција. Обично се користи за слање података из *form*-и или за извршавање одређене операције на серверу, као што је додавање новог ресурса.

- *DELETE*:

Метода *DELETE* се користи за брисање ресурса са сервера. Клијент шаље захтјев да се ресурс са задатим УРЛ-ом обрише. Ова метода се користи за трајно уклањање података са сервера.

Метода *method* омогућава да се тачно утврди коју је методу је клијент користио захтјевајући одређени ресурс и на основу тога прилагођава се логика обраде захтјева.

2.3.1.2. УРЛ и путања

УРЛ и путања (*path , get_full_path()*):

Атирбут *path* представља путању која је затражена у УРЛ-у, без домена или параметара.

Метода *get_full_path()* враћа цјелокупну путању са свим параметрима.

2.3.1.3. Подаци о сесији

Атрибут *session* омогућава приступ сесији за тренутни захтјев. Сесија се користи за чување података између различитих захтјева

2.3.1.4. Подаци из форме

Подаци из форме (*POST* и *GET*) су атрибути који садрже податке који су послати у захтјеву кроз тијело или УРЛ параметре. *POST* се користи за приступ подацима послати методом *POST,* док се *GET* користи за приступ

подаким апослати методом *GET.*

2.3.2. HttpResponse

HttpResponse је класа у ђангу која се користи за генерисање *HTTP* одговора који ће бити послат клијенту. Овај одговор може садржати информације као што су *HTML* садржај, *JSON* подаци, слике или било који други тип података који се може пренијети преко *HTTP* протокола.

Неки од атрибута су:

content који представља садржај *HTTP* одговора и може бити податак различитог типа.

status који представља статусни код који се прослиједује и означава успијех, грешку или друге ситуације током *HTTP* захтјева.

2.3.3. HttpResponseRedirect

HttpResponseRedirect је класа у ђангу која се користи за преусмеравање клијента на другу УРЛ адресу након обраде *HTTP* захтјева. Ова класа генерише *HTTP* одговор са статусним кодом 302 (*Found*) и одговарајућим заглављем *Location,* које садржи циљну УРЛ адресу.

Основна сврха је да преусмјери клијента на другу УРЛ адресу. Као што је то учињено у следећем коду:

```
def post(self, request):
    if method.POST:
            print("usao u post requesT")
            return HttpResponseRedirect("/box")
```

2.4. DJANGO URLS

Пакет *urls* омогућава дефинисање и управљање рутама (УРЛ адресама) у пројекту. Као систем рутирања користи се за повезивање УРЛ адреса са одговарајућим погледима (*views*) који ће обрадити захтјев и генерисати одговор.

Неке од кључних метода које су кориштене у пројекту су:

- *path()*
- *include()*

2.4.1. Path

Ова метода се користи за дефинисање рута користећи синтаксу путање () у ђангу. Она прима три аргумета:

- *route:* представља део УРЛ адресе која се односи на ту руту
- *view:*представља поглед који ће се позвати за обраду захтјева на тој рути
- *kwargs:*представља додатне аргументе који се могу прослиједити

погледу

У следећем дијелу кода урађено је повезивање рута и погледа:

```
from django.urls import path, include
from . import views
urlpatterns = [
        path('', views.MainPageView.as_view(), name="main-page"),
        path('store<slug:slug>', views.StoreView.as_view(), name="store-page" ),
        path('box/', views.BoxView.as_view(), name="box-page"),
        path('boxx/', views.Add_to_session, name="add-session")
    ]
```

2.4.2. Include

Ова метода се користи за укључивање других УРЛ конфигурација из других модула или апликација. То омогућава организацију рута у модуларне компоненте и поделе УРЛ конфигурација на више дијелова.

У следећем дијелу кода урађено је повезивање руте из „Store" и УРЛ конфигурација из апликације „bmdSite".

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
         path('admin/', admin.site.urls),
         path('', include("bmdSite.urls"))
        ]
```

2.5. DJANGO VIEWS

Пакет *views* у ђангу је одговоран за обраду *HTTP* захтјева и генерисања одговора. Пружа могућности генерисања погледа (views) који садрже логику за обраду захтјева, приступање подацима, припреми контекста и генерисање одговора који се шаљу назад клијенту.

Погледи се могу дефинисати као фукције или класе (C*lass-based-views*) и могу садржати логику пословне логике, приступање моделима из базе података, валидацију и друге операције које су потребне за обраду захтјева.

Примјер погледа дефинисаног кроз функцију:

```
def Add_to_session(request):
        if request.method=="POST":
            item_checked=request.session["item_checked"]
            print("usao u dodaj kolicinu")
            kljuc=request.POST["kljuc"]
            quantity=request.POST["vrijednost"]
            item_checked[kljuc]=int(quantity)
            print(kljuc, quantity)
            request.session["item_checked"]=item_checked
         return HttpResponse('OK')
```

У овом погледу обрађује се *POST* захтјев и након извршавања тијела функције враћа се *HttpResponse* са параметром „OK".

2.5.1. Class-based views

Класа базираних погледа у ђангу је алтренативни приступ дефинисању погледа који се заснива на употреби *Python* класа умјесто функција. *Class-based views* пружају моћан и флексибилан начин за организовање и поновно коришћење логике погледа.

Неки од уобичајних класа базираних погледа који су коришћени у изради пројекта су:

- *ListView.*
- *View*

2.5.1.1. ListView

ListView је класа која се користи за приказивање листе објеката из базе података. Она олакшава добијање података из базе, припрему контекста и генерисање одговора у облику листе објеката. Наиме *ListView* аутоматски дохвата листу објеката из базе података на основу модела који је наведен у класи погледа тј. постављања атрибута *model* у класи.

Припрема контекста који ће бити прослијеђен шаблону за приказивање листе објеката, може се прилагодити дефинисањем метода *get_queryset()* и *get_context_data()* у класи *ListView.*

За постављање шаблона потребно је ручно поставити име шаблона у атрибут *template_name* у класи *ListView.* Постоји и аутоматско тражење шаблона у односу на постављени модел.

Примјер коришћења *ListView* у погледима:

```
class MainPageView(ListView):
        template_name="bmdSite/main.html"
        model=Store
         def get_context_data(self, **kwargs):
              context= super().get_context_data(**kwargs)
                context["stores"]=Store.objects.all()
                 return context
```

2.5.1.2. View

Класа *View* је основна класа у ђанго-овом пакету *views* која се користи за дефинисање класа базираних погледа (C*lass-based-views*). Класа дефинише различите методе које се користе за различите *HTTP* захтјеве, као што су *GET, POST, PUT, DELETE,* тиме омогућава да се дефинише логика за обраду различитих захтјева.

- *get()*:

Ова метода се позива када се прими *HTTP GET* захтјев. Обично се користи за приказивање података кориснику. Извршавају се операције дохватања података из базе, филтрирање, сортирање или било коју другу логику која је птребна. Након тога може се генерисати одговор који ће бити послат назад кориснику.

- ***post()***:

Ова метода се позива када се прими *HTTP POST* захтјев. Користи се за обраду података које корисник шаље на сервер. У овој методи може се извршити валидација унесених података, снимање података у базу, слање мејлова... Након обраде могуће је генерусати одговор који ће бити послат кориснику.

Методе *get()* и *post()* имају приступ објекту *self* који представља инстанцу класе *View,* класа садржи и атрибут *request* који садрже информације о тренутно захтјеву.

2.6. DJANGO MAIL

Mail пакет користи се за слање е-поште из веб апликације. Да би се користио потребно је прво кофигурисати поставке е-поште у ђјанго пројекту, конкретну у setting.py фајлу. То подразумјева постављање *SMTP* послужитеља, адресе пошиљаоца и други опционалних поставки као што су *SSL* енкрипција или провјера аутентичности.

Приказ постављања параметара у пројекту веб апликације за продавницу, у стању развоја користи се конзлни приказ е-поште:

```
EMAIL_BACKEND='django.core.mail.backends.console.EmailBackend'
```

Приказ постављања свих параметара у пројекту веб апликације за продавницу:

```
EMAIL_BACKEND='django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST='smtp.example.com' #adresa SMTP servera
EMAIL_PORT=587 #Port SMTP servera
EMAIL_HOST_USER='kostarasovic25gmail.com'
EMAIL_HOST_PASSWORD=''
EMAIL_USE_TLS=True #TLS enkripcija
```

Када је конфигурација постављена може се користити ђанго *API* за слање е-поште. Постоји неколико различитих метода за слање е-поште, за израду пројекта коришћена је метода *send_mail().*

send_mail() функција прима аргументе попут наслова поруке, тијела поруке, адресе примаоца и адресе пошиљаоца.

Приказ дијела погледа из пројекта веб апликације за продавницу у којој је реализовано слање е-поште са подацима добијеним из *request*-а:

```
elif "buy_articles" in request.POST:
        context={}
        articles=Article.objects.filter(id__in=item_checked)
         email=request.POST["user_email"]
         phone_num=request.POST["phone_num"]
         total_price=str(request.POST["total_price"])
         print("kupi artickle")
         context["total_price"]=total_price
        context["email"]=email
```

```
        context["phone"]=phone_num
         messages=""
 for article in articles:
        messages+=f'{article.type_arttcle.name} (Cijena:{article.type_arttcle.price})'
         subject='Potvrda o uspijesnoj kupovini'
         message=f'{messages} Ukupna cijena racuna:{total_price}'
         from_email= "kostarasovic25@gmail.com"
        to_email=[from_email,email]
        send_mail(subject,message,from_email,to_email,fail_silently=False)
```

У овом дијелу кода из *request* се добијају информације које је корисник прослиједио, на основу тих података врши се слање е-поште користећи методу *send_mail().*

2.7. DJANGO SESSION

Пакет *session* омогућава чување података и информација о кориснику, као и друге релевантне податке, док корисник остаје активан корисник на веб апликацији. Пакет пружа неколико кључних функционалности, могуће је чувати произвољне информације о кориснику током сесије. Ови подаци се чувају на серверској страни и могу се читати и мијењати током трајања сесије.

Такође *session* пакет омогућава да се дефинише временско ограничење за сесију.Када сесија истекне подаци се аутоматски бришу. Ђанго сесије подржавају сигурност. Подаци се аутоматски енкриптују како би се заштитили од неовлашћеног приступа.

Свака сесија има јединствени идентификатор који се користи за идентификацију током сесије. Овај идентификатор се чува као колачић у прегледачу корисника и користи се за повезивање корисника са одговарајућим садржајем са северске стране.

Конкретно у пројекту пајтон апликације за продавницу коришћене су за складиштење информација о корпи одређеног корисника.

3. ИМПЛЕМЕНТАЦИЈА ПАЈТОН АПЛИКАЦИЈЕ ЗА ПРОДАВНИЦУ

Како би имплементирао пајтон апликацију за продавницу у ђангу било је потребно прво направити базу података, у којој ће се чувати подаци директно везани за рад апликације. Такође потребно је дефинисати више рута, и направити конекције истих са погледима (views) који ће бити окинути приликом рутирања и извршити одређене акције или ће имитирати *REST API* тј. вршиће манипулације над подацима из базе података кроз саме шаблоне (*template*).

3.1. ИЗРАДА БАЗЕ ПОДАТАКА

Приликом израде саме базе података одабрао сам *SQLite3* базу. Коју сам искористио за израду самих модела, креирање релација између модела, креирање миграција и чување података у саму базу података. С`тим у вези направљена је релациона база.

Као први корак у креирању базе података у фајлу *models.py* било је потребно направити класе које представљају моделе. Да би се имплементирала таква логика потребно је изврсити наслиједјивање класе *Model* која је импортована из пакет *models.*

```
from django.db import models
```

Креирање модела за продавницу.

```
class Store(models.Model):
    name=models.CharField(null=False, max_length=100)
    image=models.ImageField(upload_to="images", null=True)
    address=models.CharField(null=False, max_length=200)
    slug= models.SlugField(unique=True, db_index=True)
    def __str__(self):
        return f"{self.name}"
```

У коду изнад креирана је класа *Store* која представља модел и садржи атрибуте који као цјелина дефинишу продавницу. Атрибути имају поља која представљају тип вриједности који се уноси и имају својства везана за базу података о начину складиштења података.

Такође, за потребе валидације вриједности у самим атрибутима импортовано је:

```
from django.core.validators import MinLengthValidator, MinValueValidator
```

MinLengthValidator и *MinValueValidator* представљају својства која имају за сврху валидацијз података по минималној дужин*и stringa* и минималној вриједности података

У наредном коду креиран је модел *Type_Article* који са својим атрибутима представља тип артикла и садржи информације о истом.

```
class Type_Article(models.Model):
    name=models.CharField(null=False, max_length=100)
```

```
price=models.FloatField(null=True, validators=[MinValueValidator(0.1)])
image=models.ImageField(upload_to="images", null=True)
text=models.TextField(validators=[MinLengthValidator(10)] )
def __str__(self):
    return f"{self.name}"
```

За потребе складиштења слика или мултимедијалног садржаја било је потребно предходно инсталирати библиотеку *Pillow,* што би омогућило коришћење *ImageField* поља*.* Кроз који се прослиједјују информације о локацији складиштења слика.

Следећа креирана табела је *Article* преко које је извршено повезивање тј. креирање релационих веза са остале двије табеле. Што је извршено преко поља *ForeinKey* које представља страни кључ табела *Type_Article* и *Store.* Преко података о страном кључу може се директно приступити подацима из других табела. Тако да табела *Article* са својим атрибутима као цјелина представља један артикал у пордавници и у даљем току имплементације биће кориштена за дохватање постојећих података у циљу приказивања или манипулације над истим.

```
class Article(models.Model):
        store=models.ForeignKey(Store,null=True,on_delete=models.SET_NULL,
related_name="store")
        type_arttcle=models.ForeignKey(Type_Article,on_delete=models.CASCADE,
null=False, related_name="type_article")
        quantity=models.IntegerField(validators=[MinValueValidator(1)])
        def __str__(self):
            return f"{self.type_arttcle}"
         def get_store(self):
             return self.sotore.name
        def get_type_article(self):
             return self.type_arttcle.name
```

У *ForeinKey* као први параметар наводи се име табеле чији је страни кључ односно са којом се повезује. Такође, наводи се *on_delete* параметар који описује шта ће се десити са тренутним пољем уколико дође до брисања табеле из које је изведен страни кључ. У предходном коду наведена су два случаја *CASCADE обрисати ентитет* и *SET_NULL* сетуј вриједност страног кључа на *null* у табели.

У свим моделима направљене су додатне методе које стилизују испис података у *string* формату или враћају одређене податке приликом позива.

3.2. РУТИРАЊЕ

Да би све функционалности и садрђај креиране апликације функционисали потребно је прво извршити рутирање. Као прва етапа имплементације извршено је рутирање у пројекту *Store* који садржи аутоматски ђанго генерисани фајл *urls.py*. У њему је извршено рутирање на ђјанго генерисани *admin side* и укључивање УРЛ конфигурације из апликације *bmdSite*.

```
from django.contrib import admin
```

```
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
        path('admin/', admin.site.urls),
        path('', include("bmdSite.urls"))
        ]+ static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Промјењива *urlpatterns* служи за дефинисње УРЛ-ова. Док метода *static* се у овом случају користи како би се омогућило убацивање слика из статичних фајлова. Тј. у бази података за сваку слику сачуван је података о локацији на којој се налази а у параметрима методе *static* дефинисана је адреса и начин приказивања података кориснику у *УРЛ-у* због сигурносних разлога.

Такође извршеноје рутирање *urls.py* фајла и у *bmdSite,* гдје за функционалност има да повеже УРЛ адресу са поглдом који треба да изврши дефинисану акцију.

```
from django.urls import path, include
from . import views
        urlpatterns = [
        path('', views.MainPageView.as_view(), name="main-page"),
        path('store<slug:slug>', views.StoreView.as_view(), name="store-page" ),
        path('box/', views.BoxView.as_view(), name="box-page"),
        path('boxx/', views.Add_to_session, name="add-session")
        ]
```

3.3. ИМПЛЕМЕНТАЦИЈА ПОГЛЕДА(VIEWS)

Да би се успјешно извршила имплементација свих погледа за све постојеће функционалности прво је потребно импортовати пакете чији алати су потребни, као и моделе базе података.

```
        from typing import Any, Dict
        from django.http import HttpResponseRedirect, HttpResponse
        from django.urls import reverse
        from django.shortcuts import render
        from .models import Store, Article, Type_Article
        from django.views.generic import ListView, DetailView
        from django.views import View
        from django.core.mail import send_mail
        from django.contrib.sessions.models import Session
```

У процесу креирања погледа који се позове на основу дефинасаног рутирања у *urls.py* фајлу а има за сврху рендеровања односно прослиједивања свих потребних података како би се приказала почетну страница.

3.3.1. MainPageView

Дефинисана класа *MainPageView* као поглед насилиједује класу *ListView*(*Class-based view*) која нуди олакшице за враћање података у односу на захтјев корисника. Сврха погледа је да обезбједи успјешно приказивање почетне странице.

```
#Main page for client side. Show all store and direction for store page
class MainPageView(ListView):
        template_name="bmdSite/main.html"
```

```
        model=Store
        def get_context_data(self, **kwargs):
            context= super().get_context_data(**kwargs)
            context["stores"]=Store.objects.all()
            return context
```

У погледу дефинише се име шаблона на који ће бити рендеровани подаци извучени из дефинисаног модела. Преклопљена је метода *get_context_data* у којој се смјешта као листа параметар „*stores*“ све пордавнице које се налазе у бази. За циљ метода има да дефинисаном шаблону пружи прослијеђене податке на располагање под именом „*stores*“.

3.3.2. StoreView

Дефинисана класа *StoreView* као поглед насилијеђује класу *View*(*Class-based view*) која нуди олакшице за враћање података у односу на захтјев корисника. Сврха погледа је да обезбједи потребне акције и податке за приказивање појединачне странице по продавници.

```
class StoreView(View):
    def get(self, request, slug):
        store= Article.objects.filter(store__slug=slug).exclude(quantity__lte=1)
            context={
            "store_slug":slug,
            "store_name":store.first().store.name,
            "store":store
            }
            return render(request, "bmdSite/store.html", context)
    def post(self, request, slug):
            quantity=1
            if request.method=="POST":
                item_checked=request.session.get("item_checked")
                #check is the item already initialized
                if item_checked is None:
                    item_checked={}
                    print("usao u prvi if \n")
                article_id=int(request.POST["article_id"])
                print("this is ID %d" %article_id)
                temp_article= Article.objects.get(id=article_id)
                slug=temp_article.store.slug
                if str(article_id) not in list(item_checked.keys()):
                    item_checked[str(article_id)]=quantity
                    request.session["item_checked"]=item_checked
                    print(request.session["item_checked"])
                    request.session.modified=True
                return HttpResponseRedirect("/store"+slug)
```

Уколико је захтјев био *GET* аутоматски ће се позвати метода *get()*. Која обрађује исти захтјев. Кроз УРЛ рутирање прослијеђује се и *slug* на основу кога се у *get()* методи филтрирају подаци из базе података. Кроз *context* прослијеђују се подаци као што су *slug* продавнице *„име“* продавнице и листа продавница која је повучена из базе кроз филтрирање на основу задатих параметара. Одговор ће бити прослијеђен на основу параметара који су постављени у функцији *render.*

Уколико је захтјев био *POST* аутоматски ће се позвати метода *post()*. Која обрађује исти захтјев. Кроз УРЛ рутирање прослијеђује се и *slug* који се у

post() методи користи за потребе редирекције одговора у *HttpResponseRedirect.* Унутар методе врши се провјера да ли је примљени захтјев *POST,* уколико јесте врши се дохватање сесије и иницијализација промјењиве *item_checked* подацима из сесије. У случају да сесија не постоји, промјењива ће бити иницијализована на *None* тип, што је у следећим линијама кода и провјерено за потребе реиницијализације на *dict* тип. У промјењиву *article_id* смјешта се кастован података прослијеђен кроз захтјев из корисничке форме. Служиће за потребе филтрирања артикала у бази и провјере да ли у сесији постоји исти артикал уколико је сесија предходно постојала. Такође уписиваће се као кључ у библиотеки(*dict*) *item_checked* чија ће вриједност бити *quantity* промјењива. Која приликом првог уписа у сесију као сопствену вриједност смијешта „1“. Промјењива *item_checked* биће уписана у сесију као крајња акција и циљ описане методе *post(),* тј. имаће за циљ чувања података (ид артикла и количина истог) у сесији по активном кориснику. Након успјешно извршених акција враћа се редирект на отворену продавницу.

3.3.3. BoxView

Дефинисана класа Box*View* као поглед насилијеђује класу *View*(*Class-based view*) која нуди олакшице за враћање података у односу на захтјев корисника. Сврха погледа је да обезбједи потребне акције и податке за приказивање корпе по кориснику.

```
class BoxView(View):
        #list all article from the session
  def get(self, request):
        price_total=0
        if "item_checked" in request.session:
          item_checked=request.session.get("item_checked")
        else:
          item_checked=None
          context={}
          #if there is no data
          if item_checked is None or len(item_checked)==0:
            context["articles"]=[]
            context["has_something"]=False
          else:
             temp_dict1={} #dict for showing quantity per article id
             temp_dict2={} #dict for showing total_price per item
             for article_id, quantity in item_checked.items():
                article_Id=int(article_id)
                print("this is quantity %d" %quantity)
                temp_dict1[article_Id]=quantity
                print("article %d quantity %d" %(int(article_id), quantity))
                context["has_something"]=True
               price_per_item=Article.objects.get(id=article_Id).type_arttcle.price*quantity
               price_total+=price_per_item
               temp_dict2[article_Id]=price_per_item
               context["price_total_item"]=temp_dict2
               context["price_total"]=price_total
          context["quantity"]=temp_dict1
          article=Article.objects.filter(id__in=item_checked)
          context["articles"]=article
        return render(request, "bmdSite/box.html", context)
#create session if it is not created yet, and save the article into it.
```

```
def post(self, request):
        print("usao u post requesT")
        item_checked=request.session.get("item_checked")
        print(request.POST)
        #The button delete is clicked
        if "delete_article" in request.POST:
           article_id=int(request.POST["article_id"])
           del item_checked[str(article_id)]
           request.session["item_checked"]=item_checked
           return HttpResponseRedirect("/box")
        # the button submit (buy) whole page is clicked
        elif "buy_articles" in request.POST:
            context={}
            articles=Article.objects.filter(id__in=item_checked)
            email=request.POST["user_email"]
            phone_num=request.POST["phone_num"]
            total_price=str(request.POST["total_price"])
            print("kupi artickle")
            context["total_price"]=total_price
            context["email"]=email
            context["phone"]=phone_num
            messages=""
           for article in articles:
 messages+=f'{article.type_arttcle.name}(Cijena:article.type_arttcle.price})'
           subject='Potvrda o uspijesnoj kupovini'
           message=f'{messages} Ukupna cijena racuna:{total_price}'
           from_email= "kostarasovic25@gmail.com"
           to_email=[from_email,email]
           send_mail(subject,message,from_email,to_email,fail_silently=False)
           delete_session(request)
        return render(request, "bmdSite/success.html",context)
```

Уколико је захтјев био *GET* аутоматски ће се позвати метода *get()*. Која обрађује исти захтјев. Сврха методе је да омогући приказивање странице за корпу у којој се приказују подаци о убаченим артиклима. Подаци о изабраним артиклима који треба да се прикажу у корпи, чувани су уз помоћ сесије и у предходно описаном погледу *StoreView* описано је како се убацују подаци у сесију. Метода *get()* у класи *BoxView* у одговору шаље податке повучене из сесије. Прво се провјерава да ли постоји сесија, у случају да постоји подаци из сесије смјештају се у промјењиву *item_checked* типа *dict.* Даље у коду уколико се испуни услов да је промјењива *item_checked* празна, у *context* се убацују параметри *articles* који је иницијализован на празну листу и параметар *has_something* који добија *Boolean* вриједност чија сврха ће бити дефинисана у рендерованом шаблону. Уколико претходни услов није задовољен, слиједе акције манипулације подацима из сесије и базе података. Наиме, пролази се итеративно кроз податке из сесије који су предходно смјештени у библиотеку *item_checked* и смјешта их у привремено направљену промјењиву типа *dict.* Подаци из те промјењиве ће касније бити послати у одговору кроз *context[“quantity”]* и користиће се за потребе приказивања количине артикала у шаблону за корпу. Такође, у погледу вршиће се манипулације над подацима из базе за потребе рачунања цијена артикала као и укупне вриједности рачуна. Након обрађеног захтјева корисника тј. завршетка свих дефинисаних акција и манипулација над подацима, као резултат погледа су: филтрирани подаци из базе, израчунате вриједности артикала и рачуна. Такви подаци биће прослијеђени у одговору кроз *render* функцију за потребе приказивања и

омогућивања функционалности на шаблону корпе.

Уколико је захтјев био *POST* аутоматски ће се позвати метода *post()*. Која обрађује исти захтјев. Метода *post()* у класи *BoxView* у односу на прослијеђени захтјев корисника може да врати један од два одговора. Захтјев корисника се дефинише у шаблону корпе и може бити: захтјев за брисање одабраног артикла из корпе и захтјев за куповину наведених артикала у корпи. Уколико се ради о првом захтјеву (Захтјев за брисање) у промјењивој *request.POST* налази се информација о врсти захтјева који је дефинисан у форми унутар шаблона корпа. Конкретно за први наведени захтјев потребна је информација „*delete_article*“. Уколико се испуни услов слиједи иницијализација локалне промјењиве податком о артиклу послатом кроз захтјев корисника *request.POST[“article_id”].* Локална промјењива биће иницијализована податком који представља кључ у сесији. Самим тиме имамо приступ објекту који треба да се избрише. Након завршетка акције врши се редирекција на страницу корпе уз помоћ *HttpResponseRedirect.* Уколико се ради о другој врсти захтјева (Захтјев за куповину) у промјењивој *request.POST* биће прослијеђена информација „*buy_articles*“. Након уласка у блок врши се манипулација над подацима из базе уз помоћ филтрирања са подацима повучених из сесије. Такође, у локалне промјењиве уписују се подаци прослијеђени кроз захтјев корисника (информације о кориснику: е-пошта, број телефона). Итеративним проласком кроз листу приступамо објектима из листе добијене филтрирањем података из базе, ти подаци о артиклима уписују се у литерал који ће бити прослијеђен као *messages* у генерисаној е-пошти. Подаци о кориснику заједно са генерисаном текстуалном поруком (*messages*) прослијеђују се као параметри функције *send_mail()* која шаље е-пошту. Након тога позива се функција која има за циљ брисање креиране сесије са свим подацима у њој. Као одговор рендерује се страница success уз помоћ *render* функције.

3.3.4. Add_to_session

Дефинисана функција *Add_to_session* као поглед има за циљ да изврши манипулацију над подацима из сесије. Позваће се тако што се из *JS* скрипте прослиједи дефинисани *ajax* захтјев који има информације о УРЛ рути везаном за овај поглед.

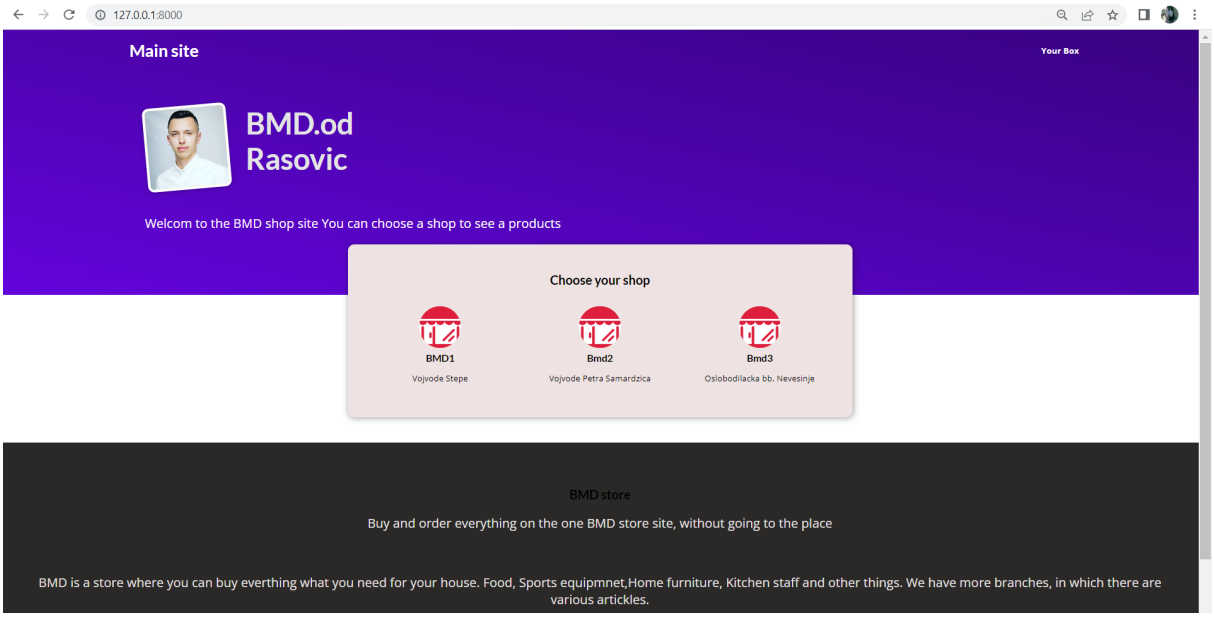
```
def Add_to_session(request):
        if request.method=="POST":
            item_checked=request.session["item_checked"]
            print("usao u dodaj kolicinu")
            kljuc=request.POST["kljuc"]
            quantity=request.POST["vrijednost"]
            item_checked[kljuc]=int(quantity)
            print(kljuc, quantity)
            request.session["item_checked"]=item_checked
        return HttpResponse('OK')
```

Овај поглед је позван кроз рутирање на УРЛ адресу из *ajax-a* који прослијеђује додатне податке о врсти захтјева који је у овом случају *POST,* такође прослијеђује податке о артиклу и постављеној количини артикла у корпи. Позвани поглед има за циљ ажурирање података у сесији за прослијеђени артикал. Након извршене измјене података у сесији као одговор враћа се HttpResponse који као параметар прослијеђује „OK“ као резултат *ajax*

захтјева.

4. ОПИС КОРИСНИЧКОГ ИНТЕРФЕЈСА

За потребе креирања корисничког интерфејса коришћени су ђанго шаблони (*templates*). Који су кроз рутирање у фјаловима *urls.py* повезани са погледима који рендерују и прослиједјују податке потребне за приказ страница односно корисничког интерфејса. *HTML* елементи су у шаблонима имплементирани већином уз помоћ олакшица које пружа ђанго оквир. Док су странице стилизоване *CSS* фајловима који су смјештени у статичким фолдерима.



Слика 4.1 – Почетна страница

У почетној страници корисник види информације о продавници и има опцију да изабер једну од понуђених продавница како би извршио куповину артикала (слика 4.1).

Почетна страница *main.html* креирана је коришћењем ђангових шаблона

```
{% extends "base.html" %}
{% load static %}
{% block title %}
  Glavna stranica
{% endblock title %}
{% block css_files %}
    <link rel="stylesheet" href="{% static 'bmdSite/post.css' %}">
    <link rel="stylesheet" href="{% static 'bmdSite/index.css' %}">
{% endblock css_files %}
{% block content %}
<section id="welcome">
 <header>
<img src="{% static 'bmdSite/images/Me.jpg' %}" alt="Kosta- The Author Of This Blog" />
<h2>BMD.od Rasovic</h2>
 </header>
```

```
<p>Welcom to the BMD shop site You can choose a shop to see a products</p>
</section>
 <section id="latest-posts">
 <h2>Choose your shop</h2>
 <ul>
{% for shop in stores %}
{% include "bmdSite/includes/shop.html" %}
 {% endfor %}
 </ul>
</section>
<div id="about">
 <nav>
 <h1>BMD store</h1>
<p>Buy and order everything on the one BMD store site, without going to the place<p>
 <br>
<p>BMD is a store where you can buy everthing what you need for your house.
Food, Sports equipmnet,Home furniture, Kitchen staff and other things.
We have more branches, in which there are various artickles.
 </p>
 <p><span>App's author: Kosta Rasovic</span></p>
 </nav>
</div>
{% endblock content %}
```

Приказ исјечка који *shop.html* који се користио кроз фор петљу:

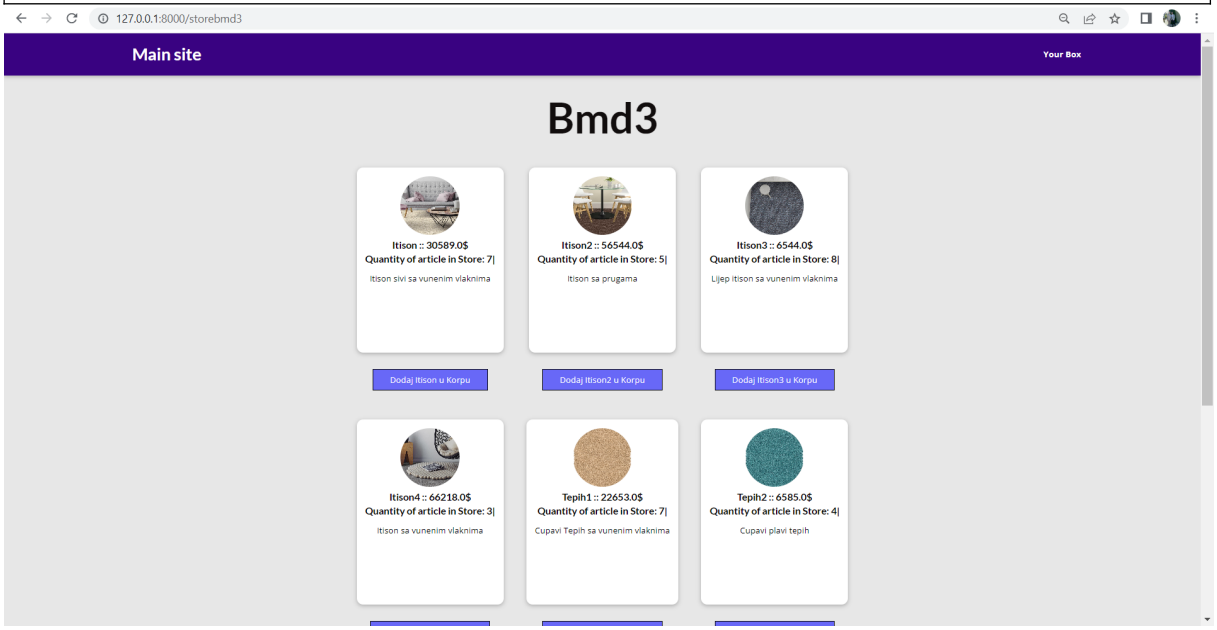
```
<li>
<article class="post">
<a href="{% url 'store-page' shop.slug %}">
<img src="{{shop.image.url}}" alt={{shop.slug}} />
<div class="post__content">
<h3>{{ shop.name }}</h3>
<p>
{{ shop.address }}
</p>
</div>
</a>
</article>
</li>
```

У коду странице *main.html* може да се примјети да су кориштене неке од ђангових олакшица за креирање *HTML* страница. *Header* као и цијели *HTML* шаблон написан је у фајлу *base.html.* Што је омогућило бржу израду страница гдје се не одузима много времена на писање стандардних *HTML* елемената. У заглављу корисник може да се пребацује на тражене странице тј. дијелове веб апликације. Било да је почетна страница или корпа. Слика 4.1.

Приказ имплементације шаблона *base.html* који наслијеђују све странице:

```
<!DOCTYPE html>
{% load static %}
<html>
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<script src="https://code.jquery.com/jquery-latest.min.js"></script>
<title>{% block title %}
```

```
 {% endblock title %}</title>
<link rel="stylesheet" href="{% static 'app.css' %}">
{% block css_files %}
{% endblock css_files %}
</head>
 <body>
<header id="main-navigation">
<h1><a href="{% url "main-page" %}">Main site</a></h1>
<nav>
<a href="{% url 'box-page' %}">Your Box</a>
</nav>
</header>
{% block content %}
 {% endblock content %}
 </body>
</html>
```



Слика 4.2, Отворена продавница

Ово је изглед отворене продавнице која има садржај прослијеђен из погледа, на страници постоје дугмат које упућују акције погледима. Тј. На страници су смјештени артикли који се налазе у одабраној продавници. Испод сваког артикла налази се дугме које има функцију убацивања артикла у корпу. Страница је такође израђена преко ђанго шаблона (слика 4.2).

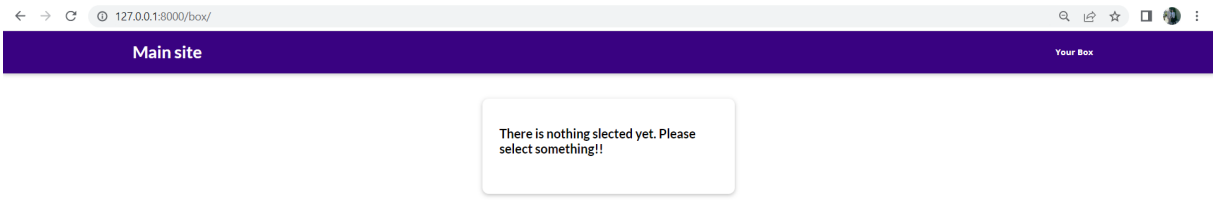
Приказ кода за страницу *store.html:*

```
{% extends "base.html" %}
{% load static %}
{% block title %}
{{store_slug}}
{% endblock title %}
{% block css_files %}
<link rel="stylesheet" href="{% static 'bmdSite/post.css' %}">
<link rel="stylesheet" href={% static 'bmdSite/all-posts.css' %}>
{% endblock css_files %}
```

```
{% block content %}
<section id="all-posts">
<h2>{{store_name}}</h2>
<ul>
{% for article in store %}
{% include "bmdSite/includes/article.html" %}
{% endfor %}
 </ul>
 </article>
 </section>
{% endblock content %}
```

Исјечак који се убацује итеративно кроз фор петљу:

```
<li>
<article class="post">
<a href="">
<img src="{{article.type_arttcle.image.url}}"alt={{article.type_arttcle.name}} />
<div class="post__content">
<h3>{{article.type_arttcle.name}}  :: {{article.type_arttcle.price}}$</h3>
<h3>Quantity of article in Store: {{article.quantity}}| </h3>
<p>
{ article.type_arttcle.text }}
</p>
</div>
</a>
<div id="read-later">
<form action="{% url 'store-page' article.store.slug %}" method='POST'>
{% csrf_token %}
<input type="hidden" value="{{article.id}}" name="article_id">
<button type="submit"> Dodaj   {{article.type_arttcle.name}} u Korpu</button>
 </form>
</div>
</article>
</li>
```



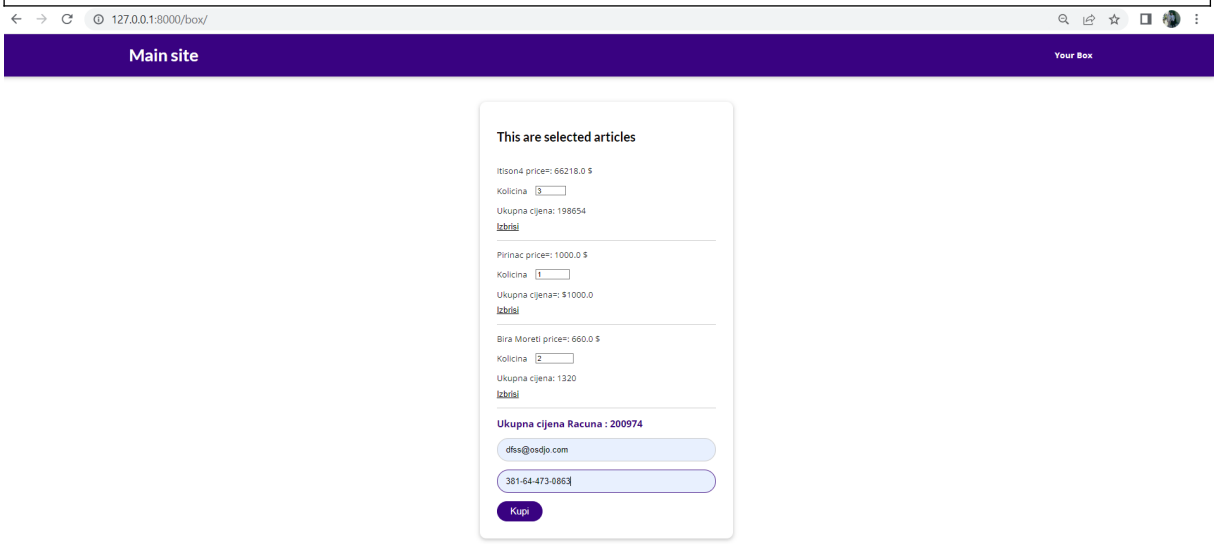
Слика 4.3, Празна корпа

Уласком у корпу корисник има увид о селектованим артиклима, слика изнад приказује ситуацију гдје корисник није ништа селектовао од артикала тј. корпа је празна (слика 4.3).

Приказ имплементираног кода у фајлу *box.html:*

```
{% extends "base.html" %}
{% load static %}
{% block title %}
Your box
{% endblock title %}
{% block css_files %}
<link rel="stylesheet" href="{% static 'bmdSite/box.css' %}">
{% endblock css_files %}
{% block content %}
<section id="stored-posts">
<ul>
{% if has_something %}
<h2>This are selected articles</h2>
{% comment "list" %}
List all selected article with their data.
{% endcomment %}
{% for article in articles %}
<li>
<p>{{article.type_arttcle.name}}price=:<spanclass="single_item_price">{{article.type_arttcle.p
rice}}</span> $</p>
{% for article_id, quantity_self in quantity.items  %}
{% if article_id == article.id %}
<input type="hidden" value="{{quantity_self}}" name="{{article.id}}" id="article_total_price"
class="article_total_price">
<div class="number-wrapper">
<p class="text-number">Kolicina</p>
<input type="number" value="{{quantity_self}}" class='numberOf_article'
id="numberOf_article" name="numberOf_article" min="1" max="{{article.quantity}}">
</div>
{% endif %}
{% endfor %}
{% for article_id, price in price_total_item.items %}
{% if article_id == article.id %}
<p id="total-price" class="total-price" >Ukupna cijena=: ${{price}}</p>
{% endif %}
{% endfor %}
form action="{% url 'box-page' %}" method="POST">
{% csrf_token %}
<input type="hidden" value="{{article.id}}" name="article_id">
<button type="submit" class="delete-button" name="delete_article">Izbrisi</button>
<div class="border-basket"></div>
</form>
</li>
{% endfor %}
<form action="{% url 'box-page' %}" method="POST">
{% csrf_token %}
<input type="hidden" value="${{price_total}}" class="form_total" name="total_price">
<p id="total_price_r" class="total_price_r" name="total_price_r">Ukupna cijena Racuna=:
${{price_total}}</p>
<input type="email" name="user_email"placeholder="Your email" class="input-basket"
required ><br>
```

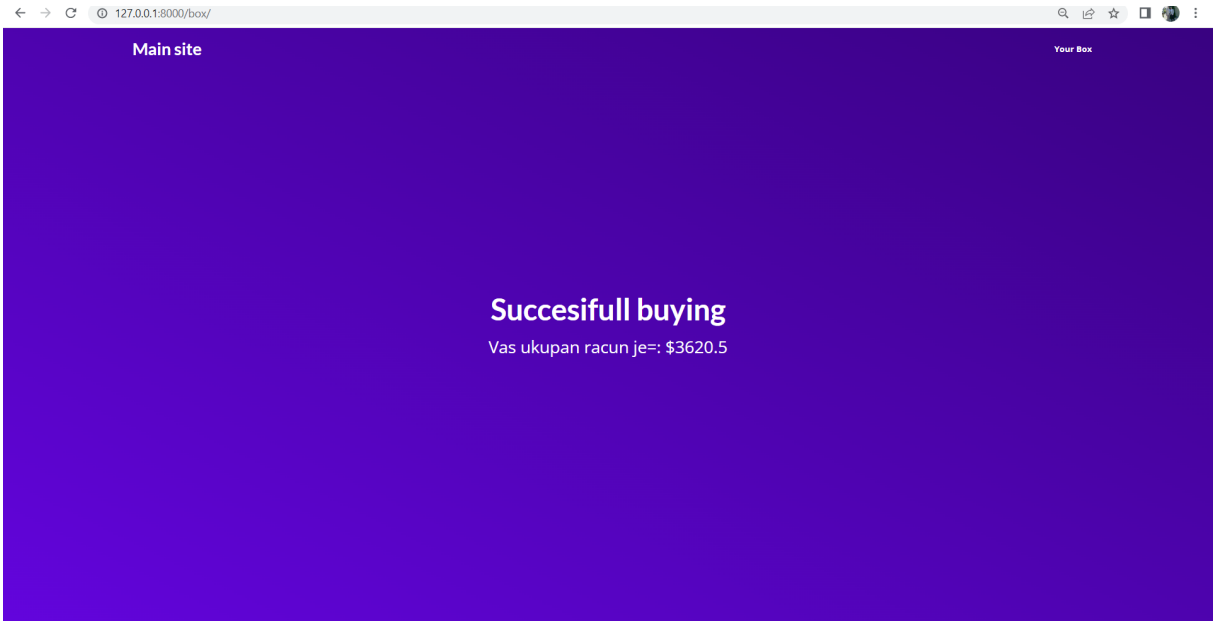
```
        <input type="tel" name="phone_num" pattern="[0-9]{3}-[0-9]{2}-[0-9]{3}-[0-9]{4}" placeholder="Your phone number" class="input-basket" required >
        <br>
        <button type="submit" name="buy_articles" class="by-button">Kupi</button>
        </form>
        {% else %}
        <h2>There is nothing slected yet. Please select something!!</h2>
        {% endif %}
        </ul>
        </section>
```



Слика 4.4, Пуна корпа

Ово је приказ корпе када у њој постоје селектовани артикли. Корисник има могућност за избор више функционалности. Као што су повећавање и смањивање количине артикала, брисање артикала као и куповином свих селектованих артикала. Овај дио странице имплеметиран је приказаним кодом изнад. Корисник уноси жељену количину артикала, уноси своју е-мејл адресу и број телефона. Након тога притисне дугме „Купи“ за куповину селектованих артикала (слика 4.4).

Након обављене акције прослиједјује му се порука о успјешности куповине(слика 4.5).



Слика 4.5, Успјешна куповина артикала

Приказ написане *JS* скрипте са имплементрианим функционалности које утичу на приказ корисничког интерфејса:

```
<script>
 const quantityInput= document.getElementsByClassName('numberOf_article');
const totalPricePerItem= document.getElementsByClassName('total-price');
const single_item_price= document.getElementsByClassName('single_item_price');
 const total_price_r= document.getElementsByClassName('total_price_r');
const form_total=document.getElementsByClassName('form_total');
const article_total_price=document.getElementsByClassName('article_total_price');
 var csrftoken='{{csrf_token}}'
let total_price = 0;
var name
console.log(total_price_r.value)
for(let i = 0; i < quantityInput.length; i++){
quantityInput[i].addEventListener('input', function(){
total_price=0
let quantity= quantityInput[i].value;
 let pricePerItem = single_item_price[i].innerHTML;
let total=quantity*pricePerItem;
totalPricePerItem[i].textContent=`Ukupna cijena: ${total}`;
article_total_price[i].setAttribute('value',`${total}`);
for(let j = 0; j < quantityInput.length; j++){
quantity= quantityInput[j].value;
pricePerItem = single_item_price[j].innerHTML;
```

```
let total=quantity*pricePerItem;
 total_price += total
 console.log(total_price)
total_price_r[0].textContent=`Ukupna cijena Racuna : ${total_price}`;
form_total[0].setAttribute('value',`${total_price}`);
 console.log(article_total_price[j].name,quantity)
dodajUSesiju(article_total_price[j].name,quantity) }
//dodajUSesiju(article_total_price[i].name,quantity)})}
function dodajUSesiju(kljuc, vrijednost){
console.log(kljuc,vrijednost)
$.ajax({
type:"POST",
url:"{% url 'add-session' %}",
headers:{
"X-CSRFToken":csrftoken, },
data: {
'kljuc':kljuc,
 'vrijednost': vrijednost},
success: function(response){
console.log("Podaci su dodati u sesiju")}})}
</script>
```

Написани код у скрипти има задатак да ослушкује елемент *input* како би сваки пут кад се деси промјена на елементу послао *ajax* захтјев серверу и послао нове податке који треба да се сачувају у сесију. Такође врши се аутоматско рачунање укупне цијене по артиклу и цијене рачуна.

Приказ стилизовања *article.html* странице:

```
body {
background-color: #e7e7e7;  }
#main-navigation {
background-color: #390281;
box-shadow: 0 2px 8px rgba(0, 0, 0, 0.25);}
#all-posts {
margin: 7rem auto;
width: 90%;
max-width: 60rem;}
#all-posts h2 {
text-align: center;
font-size: 5rem;
color: #110d0d;
margin: 3rem 0;}
ul {
list-style: none;
margin: 0;
padding: 0;
display: grid;
grid-template-columns: repeat(auto-fill, minmax(17rem, 1fr));
gap: 1.5rem;}
.post img {
width: 7rem;
height: 7rem;}
.post a {
```

```
height: 22rem;
transform-origin: center;
box-shadow: 0 2px 8px rgba(0, 0, 0, 0.25);
border-radius: 12px;
background-color: white;}
#read-later {
margin: 2rem 0;}
#read-later button {
background-color: rgb(104, 104, 247);
border: 1px solid rgb(8, 8, 8);
padding: 0.5rem 2rem;
font: inherit;
color: white;
cursor: pointer;}
#read-later button:hover,
#read-later button:active {
background-color: white;
color: #390281;}
```

CSS фајл *post.css* као и остали *css* фјалови чувају се у статичним фолдерима. И одговорни су за стилизовани изглед корисничке стране интернет апликацијије за продавницу.

5. ЗАКЉУЧАК

Као што смо могли да видимо у току рада на интернет апликацији за продавницу, ђанго оквир за развој веб апликација у пајтон језику отвара могућности брзог и ефикасног развоја апликација.У овоме пројекту користио сам многе пакете из ђанго оквира који су олакшали развој као што је *models* за развој модела базе података, *views* пакет који је омогућио креирање погледа који су одговорни за све акције и функционалности што на серверској тако и на клијентској страни. И чине спону између базе података и клијента тј. корисничког интерфејса.

За даљи развој апликације могло би да се имплементура много функционалности које би унапредило моћ саме апликације. Свакако потребно је олакшати унос података у базу, направити банковну форму за унос података и комуникацију са банком за потребе плаћања.

Такође сама база података није савршена за скалирање, нарочито уколико би се радило о великој количини података који би морали да се складиште у њу. За те потребе било би боље замјенити *SQLite3* са *MySQL* ili *PostgreSQL*.

6. ИНДЕКС ПОЈМОВА

7. ИНДЕКС СЛИКА

8. ЛИТЕРАТУРА

[1] P. Barry: "*Head First Python*",O'Reilly Media "incorporated", 2010.

[2]Steven Lott, "Functional Python Programing", Packet Publishing, 2015.

[3] B. Shaw, S. Badhwar, A. Bird, "Web Development with Django", *Packet Publishing*, 2021.

[4]Nigel George, "Mastering Django: Core" *Packet Publishing*,Dec 2016.

[5]John Ducket,"HTML and CSS", Wiley, Nov 2011.

[6]Zachary Shute, "Advanced JavaScript", Packet Publishing, 2019.

[7]Django documetation, https://docs.djangoproject.com/en/4.2/, преузето:мај 2023.

[8]JavaScript documentation,

https://developer.mozilla.org/en-US/docs/Web/JavaScript, преузето:мај 2023.

[9]Django template documentation,

https://docs.djangoproject.com/en/4.2/topics/templates/, преузето:мај 2023.

[10]Django views documentation,

https://docs.djangoproject.com/en/4.2/topics/http/views/, преузето:мај 2023.

[11]Django models documetation,

https://docs.djangoproject.com/en/4.2/topics/db/models/, преузето:мај 2023.

9. ИЗЈАВА О АКАДЕМСКОЈ ЧЕСТИТОСТИ

ИЗЈАВА О АКАДЕМСКОЈ ЧЕСТИТОСТИ

Студент (име, име једног родитеља и презиме): **Коста Драган Рашовић**

Број индекса: **РТ-68/19**

Под пуном моралном, материјалном, дисциплинском и кривичном одговорношћу изјављујем да је завршни рад, под насловом:

Пајтон апликација за продавницу

1. резултат сопственог истраживачког рада;
2. да овај рад, ни у целини, нити у деловима, нисам пријављиво/ла на другим високошколским установама;
3. да нисам повредио/ла ауторска права, нити злоупотребио/ла интелектуалну својину других лица;
4. да сам рад и мишљења других аутора које сам користио/ла у овом раду назначио/ла или цитирао/ла у складу са Упутством;
5. да су сви радови и мишљења других аутора наведени у списку литературе/референци који је саставни део овог рада, пописани у складу са Упутством;
6. да сам свестан/свесна да је плагијат коришћење туђих радова у било ком облику (као цитата, прафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, вебсајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторском и сродним правима), као и других закона и одговарајућих аката Високе школе електротехнике и рачунарства струковних студија у Београду;
7. да је електронска верзија овог рада идентична штампаном примерку овог рада и да пристајем на његово објављивање под условима прописаним актима Високе школе електротехнике и рачунарства струковних студија у Београду;
8. да сам свестан/свесна последица уколико се докаже да је овај рад плагијат.

У Београду, __. __. 2023. године

Својеручни потпис студента

_____