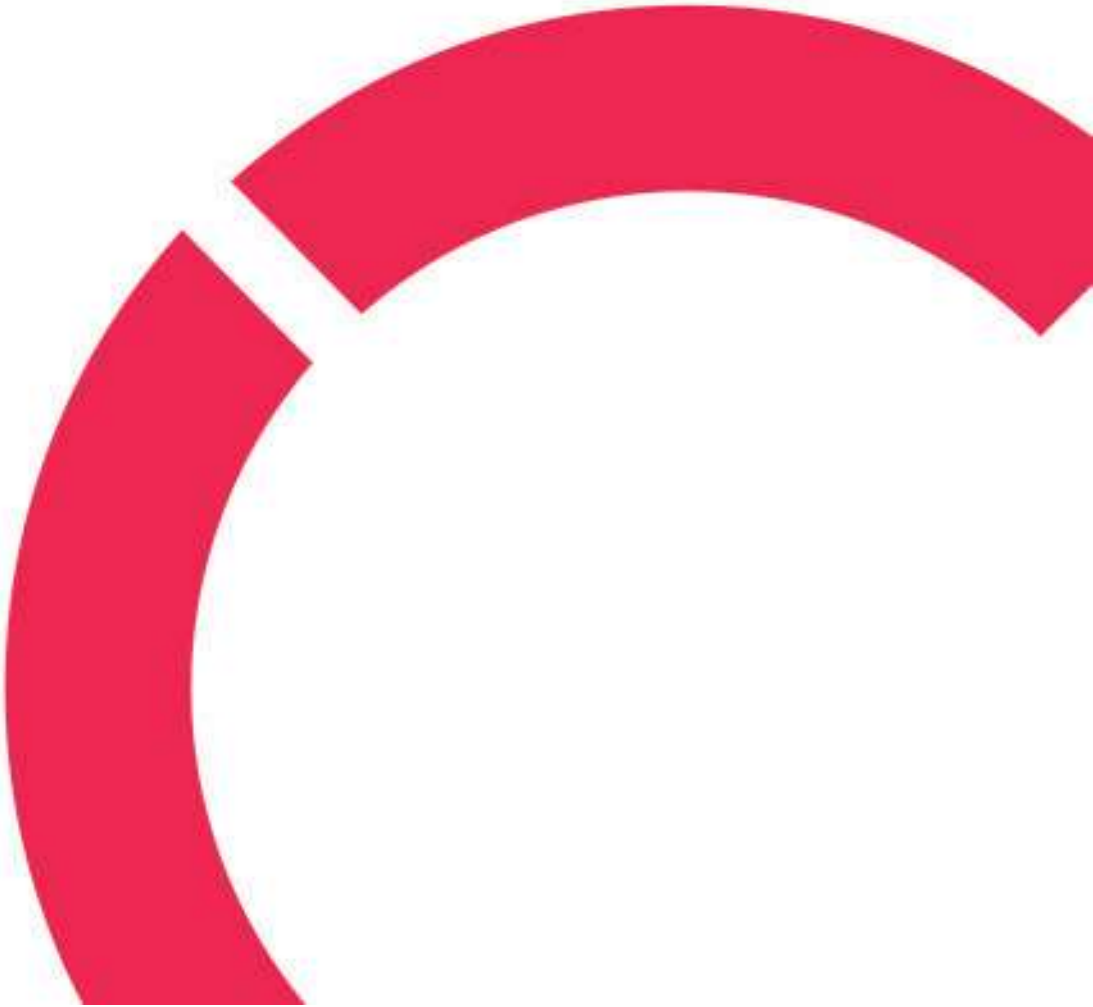
Tran Hoang Minh Long

BUSINESS MANAGEMENT APPLICATION BUILT USING DJANGO

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Bachelor of Engineering, Information Technology
April 2023



ABSTRACT

Centria University of Applied Sciences	**Date** April 2023	**Author** Tran Hoang Minh Long
Degree programme Information Technology		
Name of thesis BUSINESS MANAGEMENT APPLICATION BUILT USING DJANGO		
Centria supervisor Heikki Ahonen		**Pages** 25 + 2
Instructor representing commissioning institution or company		

The purpose of this thesis was to build a web-based application using the Django Framework for Vietnamese fabric businesses. As the current business is still relies heavily on paperwork and technologies are not implemented. Using the application will help in efficiency and reduces the paper uses overall.

The application will provide some services such as authentication system, business management, databases. This application was built with Django framework and other technologies. This thesis will introduce and explain the technologies. This thesis will also explain the code of each feature and its functionality.

Key words
Django, Python, Web Development

CONCEPT DEFINITIONS

UI

(User Interface)

CRUD

(Create Read Update Delete)

DRY

(Don’t repeat yourself)

GUI

(Graphical User Interface)

RSS

(Really Simple Syndication)

SQL

(Structured Query Language)

MVC

(Model View Controller)

MTV

(Model Template View)

CSS

(Cascading Style Sheets)

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

1 INTRODUCTION

Nowadays, there are many frameworks to help developers to build web applications. Therefore, choosing one to start building provides a challenge. With a background in Python development, choosing Python is a suitable choice for me. Moreover, Django is easy to set up, fast, simple, secure, and highly scalable. (Django 2005-2023)

I had been living in Vietnam since I was born. After over 18 years of living there, my parents own a fabric company, and the business has been going well. I want to help the business thrive even more. Building a website that helps manage the business will help tremendously as the business is digitalized. The objectives of the thesis are an explanation of creating an application using the Django framework and related technologies, and how the application works for a small business.

2 TECHNOLOGIES

2.1 Python Virtual Environment

Virtual Environment is a self-contained directory tree that contains a specific version of Python and its additional packages. It is essential for multiple Python applications because most Python applications often use packages and modules from communities, which are not part of the standard libraries. This might cause unwanted compatibilities errors as the application might need specific versions of packages to run properly. Version 1.0 might not be the same as Version 2.0, despite being under the same package's name. (Mohammad 2022, 113). With Virtual Environment, developing multiple Python applications on the same hardware with different versions of packages is possible. Resolving conflicting requirements between Python applications. The Figure illustrated this matter below.

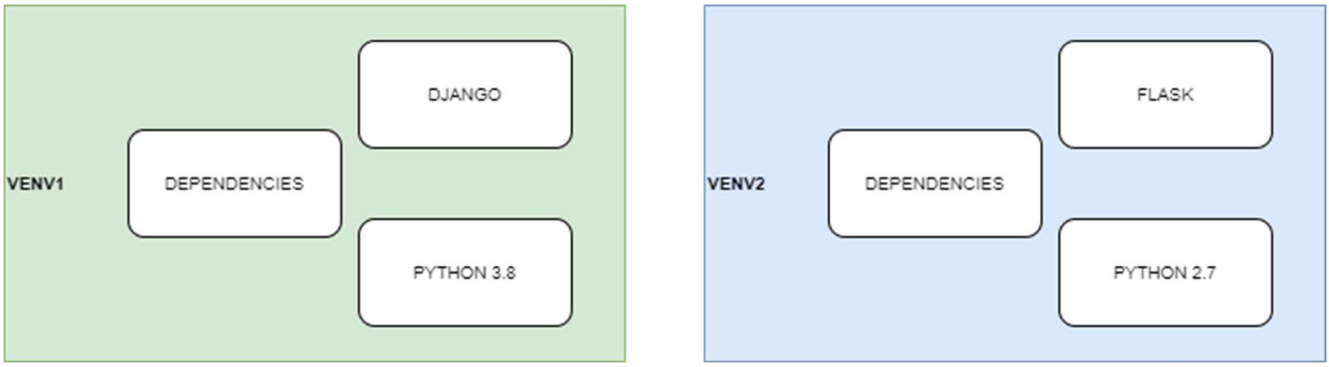


Figure 1: Virtual Environment with separate Application, dependencies, and Python version (Sailer, Z. 2018)2

2.2 Django Framework

The Django framework is written in Python which is a high-level programming language. Python's biggest advantage is its simplicity in readability and defining code blocks. Python excels in popularity with many application domains such as Web Development, Machine Learning, Data Science, Business applications, and Scripting. (Miller 2021). It was launched in 2005 and it is an open-source Python project supported by a non-profit independent foundation with goals to advance the state of the art in Web development (Django 2005-2023). As Django Framework is fast and easy to set up with built-in tasks such as authentication, content administration, site maps, RSS feeds, and others. Moreover, it is secure as it helps developers to avoid security mistakes such as SQL injection, cross-site scripting, cross-site

request forgery, and clickjacking. Django is also versatile because of its scalability to handle traffic demands. (Django 2005-2023). Although the naming conventions of Django are different, Django is an MVC framework. Django's interpretation of MVC is MTV. (Django 2005-2023)

MVC	Django MTV (file)
Model	Model (models.py)
View	Template (template.html)
Controller	View (views.py)

Table 1. COMPARISON BETWEEN THE MVC AND DJANGO MTV (Django 2005-2023)

The MVC framework helps to create complex applications easier by separating the different aspects of the application; these are input logic, business logic, and UI logic. Model objects handle retrieving and storing model states in a database as the business logic. View components handle displaying the user interface as the UI logic. Controller components handle user interaction and act as a bridge between view and model components as the input logic. When the users interact with the application, they use the controller to cause the view to send the signals to the model. Then the model manipulates the data, and the model will send the signal for the view to update itself to provide new information to the user (W3Schools 2009-2023). This is illustrated in Figure 2.

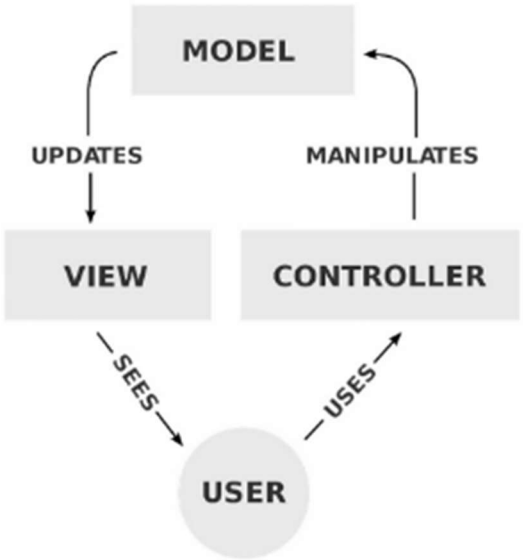


Figure 2. MVC MODEL (W3Schools 2009-2023)

Django supports the common tasks in web development by including user authentication, testing, database models, admin interface, security and performance upgrades, and support for multiple database backends within the framework. Moreover, the Django framework is maintained and updated every 8 months with bug fixes or security fixes. On top of that, Django uses Python which is readable but powerful. That makes Django framework a wonderful choice for web development. (Vincent W)

2.3 Docker

Docker is an open-source platform that simply builds, deploys, runs, update, and manage containers. Containers are made by isolating and virtualizing capabilities of built-in Linux. Docker containers can run anywhere as it is self-sufficient that can run on the cloud and multi-platforms such as Linux, Windows, and MacOS. Therefore, providing developers with consistency and reliability during the development. (Microsoft 2022) Moreover, there are advantages to using Docker Containers compared to Virtual Machines. The comparison is also illustrated in Table 2. An application on VMs depends on infrastructure, OS, Host OS, Hypervisor, and dependencies. Meanwhile, Docker Containers require only the infrastructure, OS, and container engine to maintain isolation but share the base OS services. Therefore, containers require fewer resources, are easier, and deploy faster. Meaning running more services on the same hardware and costing less. (Microsoft 2022)

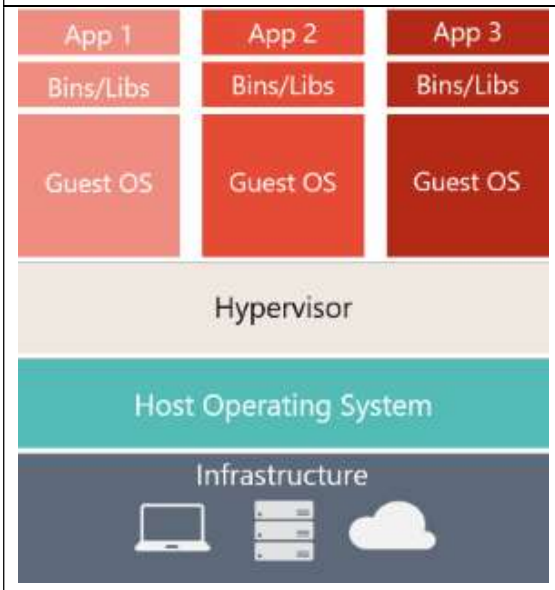
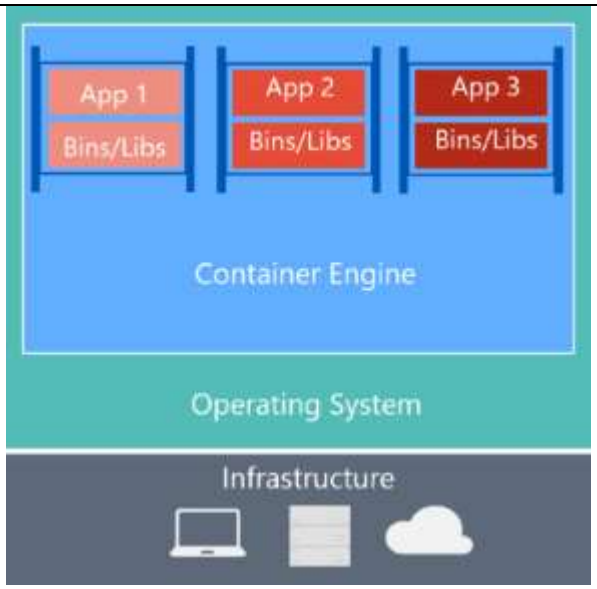
Virtual Machines	Docker Containers
	
Virtual machines require Host OS, Hypervisor, Libs, and Guest OS for each app.	Docker Container require fewer resources as they share the Host OS kernel with other containers.

Table 2. COMPARISON BETWEEN VM AND DOCKER CONTAINERS (Microsoft 2022)

2.4 Bootstrap CSS

CSS framework is a set of rules for HTML styling. Bootstrap is one of the most popular CSS frameworks as it is open-source and used for creating responsive, mobile-first sites. Originally released in 2011 by Twitter developers Mark Otto and Jacob Thornton (Klein 2021). Usually uses for positioning the page more quickly with responsive gird that can adapt to any device and screen resolution. Bootstrap is also compatible with many browsers like Chrome, Safari, and many devices from computers to smartphones. The bootstrap community is supportive and provides themes, and templates with sleek designs. Providing the front-end designing with ease and a well documentation. Therefore, choosing Bootstrap is straightforward. (Klein 2021)

3 PROJECT

3.1 Project goal

As the fabric market was explained, the project goal is to build a simple Django application for the business owner to manage the business easier by having a database for the customers, fabric rolls, and orders with intuitive UI and UX. By having the application, the shop owner can serve the customers faster with a database search for the desired fabric rolls. Moreover, the amount of paperwork and a number of papers will be decreased as they are now stored in the database. Despite being a basic web application, it is important to take all the problems and needs of the business into account. There are certain goals to be achieved such as listings of customers, products, and orders; security system; CRUD operations.

3.2 Settings up development environment

Before the development of this project, the development environment must be set up. The project of this thesis will be developed on Windows 10. The development environment can be achieved on other OS, but it will not be entirely the same. Therefore, please refer to application's documentations accordingly. The Django will be hosted on lightweight Linux using Docker. However, the following programs must be installed, and installation might be different depending on the OS. Specifically, these programs are Python 3.11.1, pip, Django framework, Docker Desktop, and GitHub Desktop.

For the reason of isolating the different versions of Django, the Python Virtual Environment will be used in the development of the project. This will avoid conflicts if there are multiple Django framework versions and dependencies. The command below "Code 1" will create a virtual environment in the current directory. The command creates the target directory which we can give a name which in this case is "`venvdjango`". Inside the directory are a file names "`pyvenv.cfg`" and 3 subdirectories which are Include, Lib, Scripts. (Python Software Foundation 2001-2023)

```
$ python -m venv venvdjango
```

Code 1. Command to create a virtual environment "venvdjango"

The following command "Code 2" will activate the Virtual Environment that was created using the command above. On Windows, simply accessing the Scripts directory and sourcing an activate.bat script

will activate the Virtual Environment. This might not be the case with POSIX platform as the folder would be “bin” instead of “Scripts”. (Python Software Foundation 2001-2023)

```
$ venvdjango\Scripts\activate.bat
```

Code 2. Command to activate the virtual environment “venvdjango”

Once the Virtual Environment is activated, installing Django version 4.1.5 can be done with the following command “Code 4” using the Python pip. It is an installer for Python packages and other indexes. (Sailer, Z. 2018). After the installation, make a Django project with the following command “Code 5”. The command auto-generates code that establishes the Django project which includes the necessary files such as settings, database configurations, and application settings. (Django. 2005-2023)

```
$ pip install Django==4.1.5
```

Code 3. Command using pip to install Django version 4.1.5

```
$ django-admin startproject MinhPhungCompany
```

Code 4. Command to create a new Django project “MinhPhungCompany”

3.3 Django app development

Django gives the ability to the developer to create an app. This is essential to isolate different functionality or so-called “app” within the same project or website. An example of this could be an e-commerce website with different “apps” such as user authentication, payments, or listing. This makes the complex project easier to develop and maintain. This provides cleaner code overall and isolating features of the app away. The figure demonstrates this. (Vincent, W)

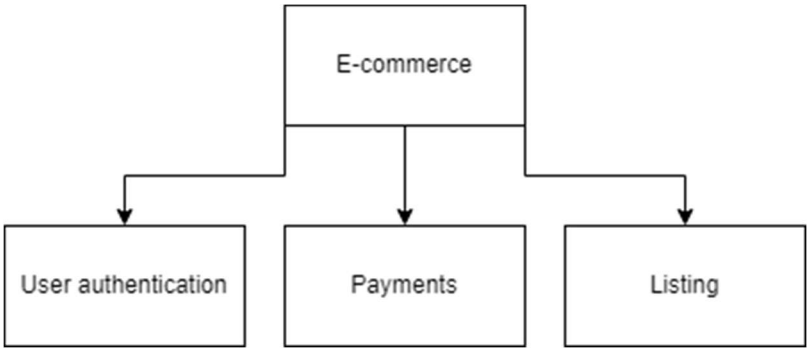


Figure 4. Potential apps for an e-commerce website.

Alongside created apps, every project has a based app name after the project itself. In my case, it locates in MinhPhungCompany/MinhPhungCompany. The following terminal output “Code 6” will show what is inside the base app. These files are asgi.py, settings.py, urls.py, wsgi.py, __init__.py. The main files are settings.py and urls.py. Where the settings.py is essentially a Django configuration file. INSTALLED_APPS and MIDDLEWARE can be registered. Others such as STATIC_FILES, WSGI_APPLICATION or DATABASES configuration are also can be configured here.

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         03-Feb-23     18:32                __pycache__
-a----         03-Feb-23     18:15            425 asgi.py
-a----         03-Feb-23     18:15           3553 settings.py
-a----         03-Feb-23     18:15            868 urls.py
-a----         03-Feb-23     18:15            425 wsgi.py
-a----         03-Feb-23     18:15              0 __init__.py
```

Code 5. Inside the directory MinhPhungCompany (GitHub repo)

Specifically, INSTALLED_APPS is a python list of apps that are currently registered inside Django. Management is a newly created app, and it is needed to be registered inside this list for Django to be recognized. STATIC_FILES in “Code 7” such as CSS, JavaScript, and Images are needed to be configured by defining the STATIC_URL to the directory “static/” inside the project folder. Directory “static/” is where all CSS, JS, and Images are stored.

```
STATIC_URL = 'static/'
```

Code 6. STATIC_URL in settings.py

DATABASES is a python dictionary containing the databases that Django uses. The “Code 7” below is generated by the Django framework and currently using the sqlite3 database. Depending on the developers and the project, this can be changed into other databases such as MariaDB, MySQL, and PostgreSQL and implemented accordingly.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Code 7. DATABASES dictionary in settings.py

3.3.1 Creating and install a Django Application

Management is a main objective; therefore, a management app is needed. Creating the app can be done with the following command “Code 8”.

```
$ python manage.py startapp management
```
Code 8. Command using manage.py file to create a new app “management”.

The command creates a new app in the project folder containing the necessary files for an app functionality (Vincent W 2010). Given the developers the structure of an app and files for creating databases, registering databases, making classes for views, and testing. Conventionally, additional files and folders can be created inside the app folder for further development could be as the templates folder where all necessary HTML is stored and urls.py for routing URLs. (Vincent W 2010) After creating the app, it must be registered inside the settings.py inside the based app’s directory which Django created for the developers. Adding the app’s name to the INSTALLED_APPS list will register it.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'management',
]
```
Code 9. Listing of installed apps inside settings.py in based app directory (GitHub repo)

The newly created app ‘management’ is now added to the list along with other Django apps that come with the framework. Firstly, `django.contrib.admin` is one of the most powerful apps and it is an automatic admin interface that the admin uses as an internal management tool (Django 2005-2023). Secondly, the `django.contrib.auth` is a component of Django’s authentication system (Django. 2005-2023). Thirdly, `django.contrib.contenttypes` provides tracking of the models installed within the Django project (Django. 2005-2023). Fourthly, the `django.contrib.sessions` can be used for storing and retrieving data on a per-site-visitor basis (Django. 2005-2023). Fifthly, `django.contrib.messages` provide support for flash messages or one-time notification messages

after user input such as form (Django. 2005-2023). Lastly, `django.contrib.staticfiles` serves by collecting the static files from each application into a single location (Django. 2005-2023).

3.3.2 Templates and Inheritance

Inheritance provides reusability by reducing redundant codes. This is very useful because usually, most websites have the same background, navbar, and sidebar that reuses many times for many templates. Django provides us tags such as `{%include%}` to include a whole file into another. Using the main.html as a base for our website eliminates the redundant code such as scripts, navbars, sidebar, heads, etc. In the example below, the navbar was included in using the tag. Doing this provides cleaner code and is easier to maintain in the future.

```
<body class="sb-nav-fixed">
  <!-- NAVBAR -->
  {% include 'management/navbar.html' %}
  <div id="layoutSidenav">
    <!-- SIDENAVBAR -->
    {% include 'management/sidenavbar.html' %}
    <div id="layoutSidenav_content">
      {%block content%} {%endblock%}
      <!-- FOOTER -->
      {% include 'management/footer.html' %}
    </div>
  </div>
```

Code 10. A part of main.html using the tag {%include%} to include navbar.html and sidenavbar.html (GitHub repo)

The other tags are the `{%block content%}` and `{%endblock%}`. These will be used to make a block inside the main.html for our main content. Since every main content is different for every HTML file but the head content that includes the background or navbar is the same, it is only necessary to include the main content as a block in the other HTML file. By using the tag {%extends%} and `{%block%}` on the specified HTML, the inheritance is established, and redundant codes are reduced.

```
{%extends 'management/main.html'%}
{%block content%}
<main>
</main>
{%endblock%}
```

Code 11. A part of index.html using the tag {%block content%} and {%endblock%} (GitHub repo)

3.3.3 URL Routing

The Django framework comes with the urls.py in the main app with the purpose of routing the URLs. Conventionally, a urls.py can be manually created on the management app and include back to the main urls.py with a purpose of easier maintenance. The following “Code 12” demonstrates the main app’s urls.py to include an app’s urls.py. The `urlpatterns` from “Code 13” is a list of paths and whenever the URL is matched with a path, it will trigger a function. For example “Code 14”: the `orderTable` will trigger a function in views.py calls order. This function will return a HTML request and other responses such as API data or SQL data, in this case a SQL query from Order table.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('management.urls'))
]
```

Code 12. The MinhPhungCompany/urls.py in app settings that comes with Django app creation. (GitHub repo)

```
urlpatterns = [
    path("home", views.home, name="home"),
    path("charts", views.charts, name="charts"),
    path("customerTable", views.customerTable, name="customerTable"),
    path("productTable", views.productTable, name="productTable"),
    path("orderTable", views.order, name="orderTable"),
]
```

Code 13. The management/urls.py that the developer created. (GitHub repo)

```
def order(request):
    order = Order.objects.all()
    context = {
        'order' : order,
    }
    return render(request, 'management/tables/orderTable.html', context)
```

Code 14. The function order in management/views.py (GitHub repo)

3.3.4 Database Models

Models.py is a file that allow users to create database tables by writing Python classes inheriting from `django.db.models.Model` (Django 2005-2023). A model contains fields and behaviours of the data. There are different field types such as BinaryField, Boolean Field, CharField, DateField, Decimal-Field, EmailField and many more. There are optional field options such as max_length, primary_key, foreinkey, choices, on_delete; different fields require arguments for them to function. Usually, single class represents a single database table. Moreover, overriding the `__str__` function in a class will make the objects easier to identify in the Django admin panel. If leave untouched, the objects will be returned as `'<class name>'` object. Thus override the function is recommended as it improves the readability.

```
class Customer(models.Model):
    customer_id = models.BigAutoField(primary_key=True)
    lastname = models.CharField(max_length=64)
    middlename = models.CharField(max_length=64, null=True)
    firstname = models.CharField(max_length=64)
    phone = models.CharField(max_length=64, null=True)
    socialnumber = models.IntegerField(null=True)
   customer_date = models.DateTimeField(auto_now_add=True, null=True)

    def __str__(self):
        return f'{self.lastname} {self.middlename} {self.firstname}'
```

Code 15. Class ‘Customer’ in management/models.py (GitHub repo)

Migrations are changes to the database models such as deleting rows or changing a field. This is compulsory whenever there are changes occurred. The `makemigrations` is responsible for making new migrations that are based on the changes to the models and `migrate` is for applying the migrations that `makemigrations` has made. This is done automatically when user runs the following commands. (Django 2005-2023)

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Code 16. Commands “makemigrations” and “migrate” that Django framework provides using for migrating databases.

Optionally, models can be registered into the admin interface. Each created app has the admin.py file. The admin.py is a file that allows developers to do just that. Because it gives an overall view of the data, therefore it is recommended. This can be achieved by passing these classes as arguments for the

`admin.site.register()`. This is demonstrated in “Code 16” below. Now the Django Admin page is set up to edit the Customer objects within the detail pages. (Django 2005-2023)

```
from .models import *
admin.site.register(Customer)
```

Code 17. Registering the Customer table by importing and use admin.site.register() function (GitHub repo)

The Django framework provides the Django Admin page as an internal management tool. That allows users with admin privileges to edit users, groups, and models. This is very useful to see the models that we have created. To access the Django administration, a superuser account is required. This can be achieved by running the following “Code 18” command.

```
$ python manage.py createsuperuser
```

Code 18. Command using manage.py to create a new superuser

After the account is created, superuser can login at http://127.0.0.1:8000/admin. Once the superuser is logged in, the Django administration site can be seen like “Figure 5”. There is “Authentication and Authorization” which contain “Groups” and “Users”. These will be used for managing users and giving certain permissions for certain group of users. Moreover, tables of apps that has been created and registered are represented in Django administration.

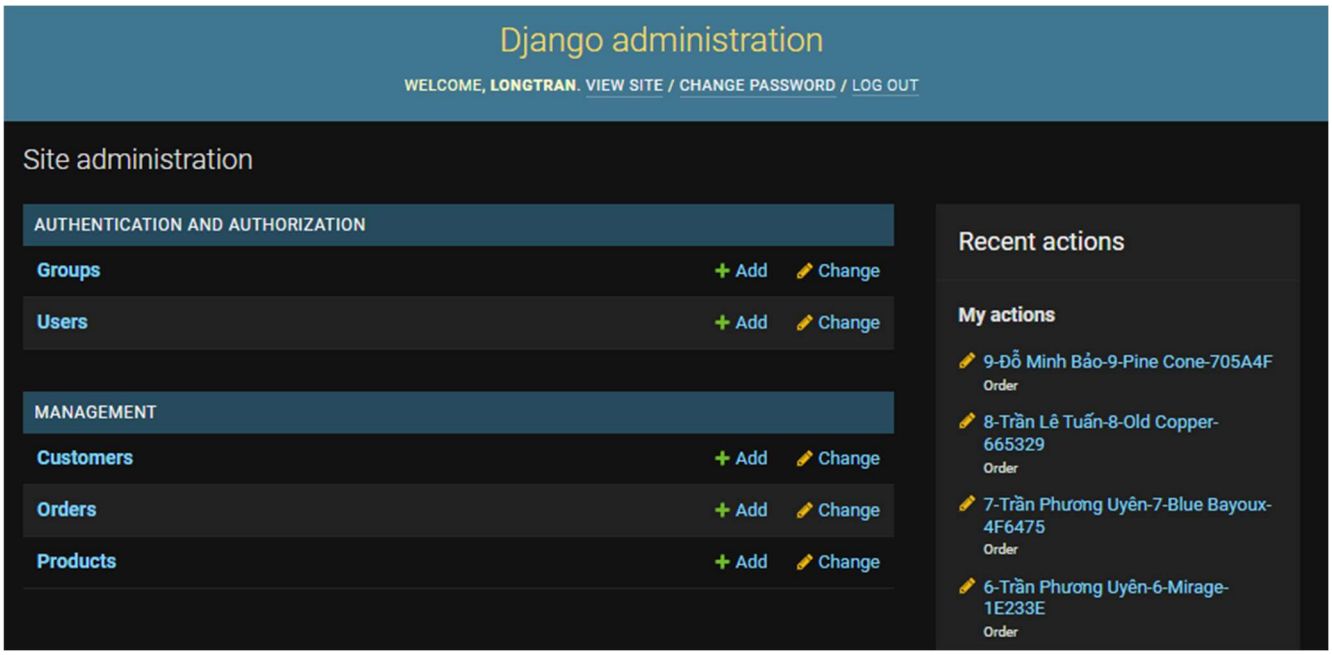


Figure 5. The Django administration the comes with the Django Framework

```
class Order(models.Model):
    order_id = models.BigAutoField(primary_key=True)
    customer = models.ForeignKey(Customer, null=True, on_delete=mod-
els.SET_NULL)
    product = models.ForeignKey(Product, null=True, on_delete=mod-
els.SET_NULL)
    order_date = models.DateTimeField(auto_now_add=True, null=True)

    def __str__(self):
        return f'{self.customer}-{self.product}'
```

Code 19. Class 'Order' in management/models.py uses to create table Order (GitHub repo)

With Django framework, there are ways to obtain the most common types of database relationships such as many-to-one, many-to-many, and one-to-one. There are tables for customers, products, and orders. For the management app case, the relationships between these are a customer can have many orders and, in order, have one product. `ForeignKey` allows us to create many-to-one relationships between the tables. `On_delete` allows us to prevent Django to delete the order whenever a customer is deleted.

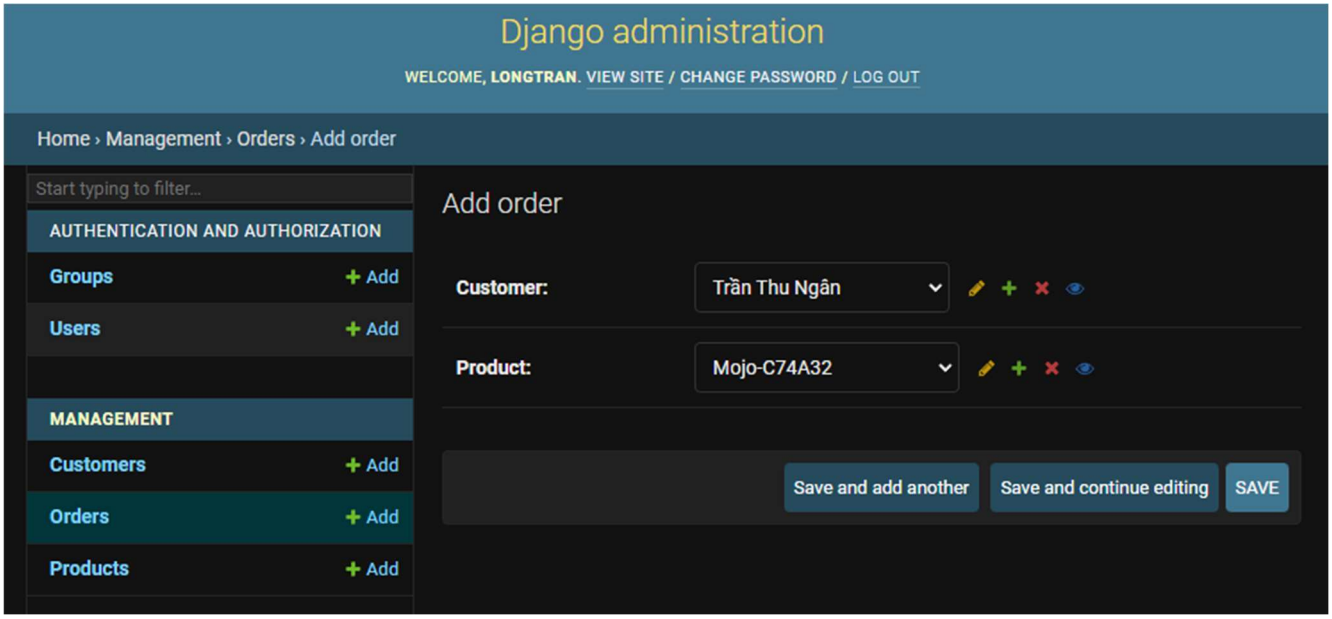


Figure 6. Add a new order to the Orders table on the Django admin site that comes with the Django Framework.

3.3.5 Databases interacting and Dynamic URL routing

When a business owner wants to see a client's order, the web application must have a page rendering all customer's orders. The Django framework provides URLconf for Dynamic URL routing. This is for creating a clean, elegant URL scheme and it is important for any web application. By using the PrimaryKey of the models `<str:pk>` inside the patterns, Django can render the order of customer bases on their primary key. This is demonstrated in "Code 20".

```
urlpatterns = [
    path("customer/<str:pk>", views.customer, name="customer"),
]
```

Code 20. Path of customer's order using the primarykey inside urlpattern list in management/urls.py (GitHub repo)

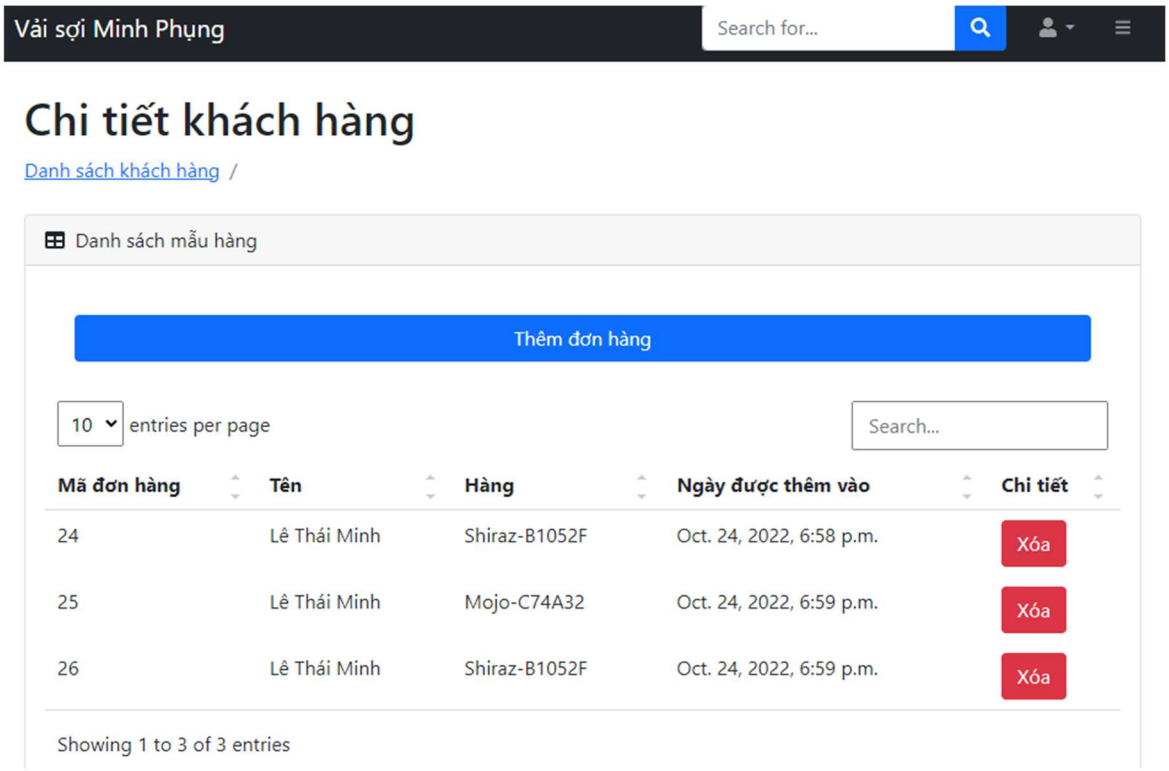


Figure 7. All of Minh's order on the website hosted on localhost

Being able to create, update, and delete is very important as it is one of the project goals. Django provides `django.forms` as tools and libraries to help developers build forms and take users' input from the website. (Django 2005-2023). Conventionally, having a separate file for our forms would achieve cleaner code overall and easier maintenance. This user-created file, named form.py, contains written classes that inherit the subclass `ModelForm` that comes with the Django framework. The `ModelForm` will build a form along with the appropriate fields and attribute bases of a `Model` class. Defining the

appropriate model and choosing the fields, the class is now ready and can be imported to views.py. (Django 2005-2023)

```
class CustomerForm(ModelForm):
    class Meta:
        model = Customer
        fields = "__all__"
```

Code 21. Class CustomerForm() that has subclass ModelForm as an argument in management/forms.py. (GitHub repo)

Creating a new customer can be done by using the forms that were created inside the forms.py and a few conditions. Firstly, we create a form by calling the function and assign to a variable. Secondly, if there is a POST request from the user, take those data and check if the form is valid by using the `is_valid()` to validate all the fields that are included on the form. If it is, then save it using `save()` to save objects back to the database (Django 2005-2023) and redirect the user back to the customer table.

```
def formCustomer(request):

    form = CustomerForm()
    if request.method == 'POST':
        form = CustomerForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('customerTable')
    context = {
        'form' : form,
    }
    return render(request, 'management/forms/formCustomer.html', con-
text)
```

Code 22. Function formCustomer in management/views.py (GitHub repo)

Updating is mostly the same as creating a new customer but with minor changes where querying all the relational objects by the primary keys of the objects and passing these onto the instance. This instance can display the existing data and can be updated according. Following validating by using `is_valid()` and saving using `save()`, after that redirect the user to Customer table and now the update function is finished.

```
def updateCustomer(request, pk):
    customer = Customer.objects.get(customer_id=pk)
    form = CustomerForm(instance=customer)
```

```
    if request.method == "POST":
        form = CustomerForm(request.POST, instance=customer)
        if form.is_valid():
            form.save()
            return redirect("customerTable")

    context = {
        "form": form,
    }
    return render(request, "management/forms/formCustomer.html", con-
text)
```

Code 23. Function updateCustomer in management/views.py (GitHub repo)

Deleting is mostly the same as creating a new customer but with minor changes where querying all the relational objects by the primary keys of the objects. If there is a POST request method, using the delete() on the customer objects will delete the Customer object, after that redirect the user to Customer table and now the delete function is finished.

```
def deleteCustomer(request, pk):
    customer = Customer.objects.get(customer_id=pk)

    if request.method == 'POST':
        customer.delete()
        return redirect('customerTable')

    context = {
        'customer' : customer,
    }
     return  render(request,  'management/forms/deleteCustomer.html',
context)
```

Code 24. Function deleteCustomer in management/views.py (GitHub repo)

3.3.6 Identification, authentication, and authorization

A security system is a process of letting authorized users access the system and keeping unauthorized users out. Users need to be identified and then they must prove their identity which means authentication. Moreover, they also must be authorized to perform certain actions (Renaud, K 96-97). Django framework provides API for building authentication backend. Specifically, `django.contrib.auth` has authenticate(), login(), and logout(). (Django 2005-2023) - These can be imported into our views.py and write the logic for identification, authentication, and authorization. For identification and authentication,

the following login function and decorator will be sufficient to keep unauthorized users out of the website and let authenticated users be able to log in.

The `login_required()` decorator allows only logged-in users to be able to enter the site by checking if the user is logged in or not. If the user is not logged in, they will be redirected to the login page by passing the absolute path in the query string. Otherwise, render out the view normally. (Django 2005-2023)

```
@login_required(login_url="login")
```
Code 25. Decorator login_required

For the login decorator to works accordingly, the login logic needs to be written. Later, the decorator can be added to certain views that requires the functionality of decorator. Thus, the following code snippet “Code 26”. Whenever there is a POST method from the user, the function will get the username and password and store them into variables. These variables are used as arguments of API `authenticate()` that comes with the Django framework. Once called, it returns either an authenticated user or None. An if statement will check if the user is not None which means the user is authenticated, then redirect them to the homepage or which means login. Otherwise, if it is None then either the username or the password is incorrect, and the message will appear on the login page.

```
@unauthenticated_user
def login(request):
    if request.method == "POST":
        username = request.POST.get("username")
        password = request.POST.get("password")
       user = authenticate(request, username=username, password=pass-
word)

        if user is not None:
            auth_login(request, user)
            return redirect("home")
        else:
           messages.info(request, "Tên người dùng hoặc mật khẩu sai.")
    context = {}
    return render(request, "management/login.html", context)
```

Code 26. Function login inside the views.py (GitHub repo)

However, there is a slight problem with the function without the `unauthenticated_user` decorator. The authenticated user can still access the login page and go through the authentication again. The

decorator is a function that calls another function and applies certain logic before executing the former. This is where `is_authenticated` argument that comes with the Django framework helps to check if the user is authenticated or not. If the user is authenticated then redirect them to the homepage, otherwise run `view_func()` which means the login() function where the login process is executed.

```
def unauthenticated_user(view_func):
    def wrarpper_func(request, *args, **kwargs):
        if request.user.is_authenticated:
            return redirect("home")
        else:
            return view_func(request, *args, **kwargs)

    return wrarpper_func
```

Code 27. Unauthenticated_user decorator inside decorator.py (GitHub repo)

For authorization, Django provides Groups that we can assign to users. With certain users, some pages will be restricted to them as they do not belong to authorized Groups. By writing function `allowed_user` taking a list of `allowed_roles` and some if statements. The first if statement is checking if the user belongs to a group, if so get the name of the first list and stored it in the variable names group. The second if statement is checking if the group variable is the same in the `allowed_roles` list and then render the view normally. Otherwise, return a HttpResponse as "You don't have permission to enter".

```
def allowed_users(allowed_roles=[]):
    def decorator(view_func):
        def wrapper_func(request, *args, **kwargs):
            group = None
            if request.user.groups.exists():
                group = request.user.groups.all()[0].name

            if group in allowed_roles:
                return view_func(request, *args, **kwargs)
            else:
                return HttpResponse("Bạn không có quyền truy cập")

        return wrapper_func

    return decorator
```

Code 28. Decorator allowed_users within management/decorators.py (GitHub repo)

Creating and assigning Groups straight-forward can be done on the Admin Panel that comes with the Django Framework. It can be accessed as a superuser account under “host/admin” which in my case is “http://127.0.0.1:8000/admin”.

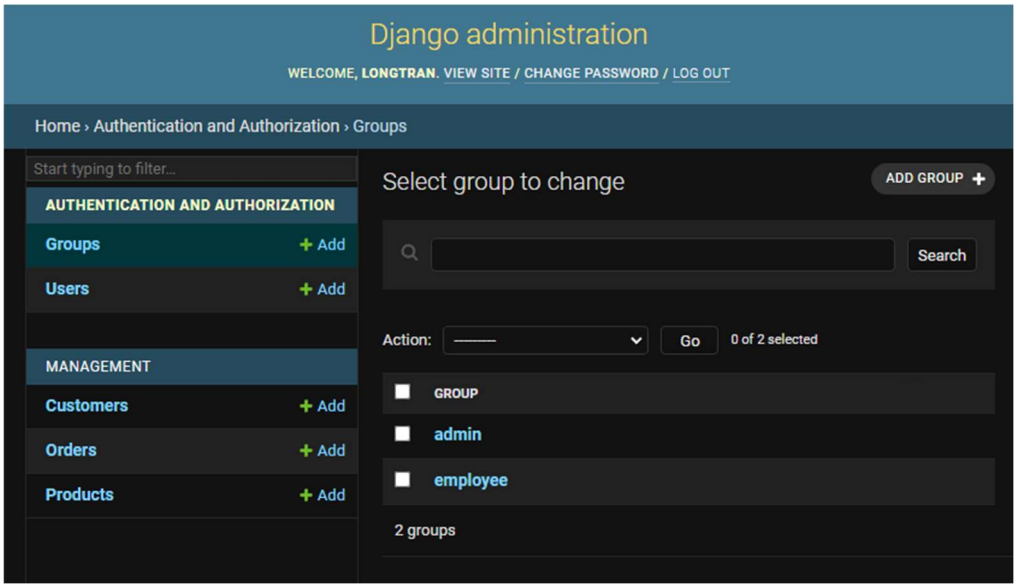


Figure 8. Groups on Django Administration Panel that comes with the Django Framework

The decorator can now be used after importing into views.py and by attaching names of the authorized Groups to @allowed_users(allowed_roles=[]) accordingly. An example, the function home is now allowed only users within the admin group to view the site.

```
@login_required(login_url="login")
@allowed_users(allowed_roles=["admin"])
def home(request):
    customer = Customer.objects.all()
    order = Order.objects.all()
    product = Product.objects.all()

    total_customer = customer.count()
    total_order = order.count()
    total_product = product.count()

    context = {
        "customer": customer,
        "order": order,
        "product": product,
        "total_customer": total_customer,
        "total_order": total_order,
        "total_product": total_product,
    }
```

```
    return render(request, "management/index.html", context)
```

Code 29. Function home in management/views.py after applying decorators (GitHub repo)

3.3.7 Dockerize the Django app

To build a Docker Container, it is required to build a Docker Image using Dockerfile. Docker Image is a package with dependencies and configuration needed to create a container. These dependencies and configurations are instructed in Dockerfile. The Dockerfile is demonstrated in "Code 30" below.

```
FROM python:3.8-slim-buster
ENV PYTHONUNBUFFERED = 1
WORKDIR /django
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
```

Code 30. The Dockerfile (GitHub repo)

The Dockerfile will instruct what to do after we run Docker. Installing a lightweight linux OS and output Python logs or errors to Docker Terminal without buffering them. Therefore, running the slim Linux-based OS, create a new directory names /django and copy the existing requirements.txt to a new created requirements.txt inside the /django directory. After that, install the dependencies in the requirements.txt that was being generated by a pip command: `$ pip freeze > requirements.txt`

```
asgiref==3.5.2
black==22.12.0
click==8.1.3
colorama==0.4.6
Django==4.1.4
mypy-extensions==0.4.3
pathspec==0.10.3
platformdirs==2.6.0
sqlparse==0.4.3
tzdata==2022.7
```

Code 31. The requirements.txt generated by using command $ pip freeze (GitHub repo)

After the Dockerfile is specified, we can create a new file name "docker-compose.yml" for Docker Compose. It is a CLI tool for defining and running multi-container Docker applications which use a YAML file to define the application's services. Docker Compose is helpful for an application that needs

multiple microservices and with a single command line such as $ docker-compose up, developers can create and start all the services. (Docker Docs 2013-2023)

```
version: "3.8"
services:
  management-app:
    build: .
    volumes:
      - .:/django
    ports:
     - 8000:8000
    image: management-app:django
    container_name: management-app
    command: python manage.py runserver 0.0.0.0:8000
```

Code 32. The docker-compose.yml file (GitHub repo)

Choosing the version of the Docker Compose by specifying “3.8” which is currently the latest version by 14 March 2023 (Docker Docs 2013-2023). After that, services can now be specified. The service names management-app is now a service for the Django project. Whenever the docker-compose.yml file runs, it uses a lightweight Linux OS and makes directory /django. After that, maps port 8000 of the container to the PC and runs the runserver command. The Docker Image will be created name management-app with a tag names Django by running the command $ docker-compose build. Moreover, after the Image runs by executing a command $ docker-compose run; a container will be created named management-app.

4 CONCLUSION

The purpose of the thesis is to build a web application for a Vietnamese fabric company using the Django framework. Moreover, doing the research for the thesis gives me insight into web development. It is also rewarding to be able to find a project that benefits and brings value to a real-world company.

At the beginning of the thesis and project, I wanted to cover the full stack of Django and React. Specifically, Django as backends and React as frontends. But the project itself was too simple and adding React as a frontend produces complications with unnecessary technologies, causes it too complicated for what it is. Moreover, I wanted to include DevOps CI/CD using the GitLab pipelines. However, including DevOps in detail would be excursive for the thesis.

As for the goals of the project, these have been achieved. The Django app, specifically the management app has databases for customers, fabric rolls, and orders with easy-to-use UI. The owner can interact with the databases and search for the desired fabric rolls. Moreover, the app can display specific orders from one customer. The app also has a security system that prevents unauthorized users from accessing the website. Furthermore, the Django app has been Dockerized and is ready for CI/CD pipelines in the future.

In the future, the web application can be improved by having graphs to help the business owner visualize and monitor the improvements of the business weekly or monthly. Moreover, having built-in reminders or checklists would be beneficial. Also, having a photo feature and scanning HEX code of the fabric roll and saving it to the database would be beneficial. On the technical side, having Redis for caching and Celery for task scheduling can improve the performance of web applications further.

During the thesis, I learned many new information about the Django framework and web development. Not just the backends and technical problems but also the front end. I have also learned more about Docker and DevOps CI/CD. Moreover, my Python programming language was also sharpened during the project. The Django framework has most of the features that allow developers to build a website that is convenient and saves a lot of time. It is also flexible and very customizable, allowing developers to build web applications that the project inquired about.

5 REFERENCES

Chacon, S. & Straub, B. 2014. Pro Git. Available at: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control. Accessed 14 March 2023.

Django. 2005-2023. *About the Django Software Foundation*. Available at: https://www.djangoproject.com/foundation/ Accessed 14 March 2023.

Django. 2005-2023. *Django FAQ: General*. Available at: https://docs.djangoproject.com/en/4.1/faq/general/#why-does-this-project-exist Accessed 24 February 2023.

Django. 2005-2023. *Documentation - Models*. Available at: https://docs.djangoproject.com/en/4.1/topics/db/models/. Accessed 14 March 2023.

Django. 2005-2023. *Documentation - django.contrib.auth.* Available at: https://docs.djangoproject.com/en/4.1/ref/contrib/auth/. Accessed 14 March 2023.

Django. 2005-2023. *Documentation - How to use sessions*. Available at: https://docs.djangoproject.com/en/4.1/topics/http/sessions/. Accessed 14 March 2023.

Django. 2005-2023. *Documentation – Migrations*. Available at: https://docs.djangoproject.com/en/4.1/topics/migrations/ Accessed 14 March 2023.

Django. 2005-2023. *Documentation - Model instance reference*. Available at: https://docs.djangoproject.com/en/4.1/ref/models/instances/ Accessed 16 March 2023.

Django. 2005-2023. *Documentation – The contenttypes framework.* Available at: https://docs.djangoproject.com/en/4.1/ref/contrib/contenttypes/. Accessed 14 March 2023.

Django. 2005-2023. *Documentation - The Django admin site* . Available at: https://docs.djangoproject.com/en/4.1/ref/contrib/admin/ . Accessed 14 March 2023.

Django. 2005-2023. *Documentation – The messages framework.* Available at: https://docs.djangoproject.com/en/4.1/ref/contrib/messages/. Accessed 14 March 2023.

Django. 2005-2023. *Documentation – The staticfiles app*. Available at: https://docs.djangoproject.com/en/4.1/ref/contrib/staticfiles/. Accessed 14 March 2023.

Django. 2005-2023. *Documentation – URL dispatcher*. Available at: https://docs.djangoproject.com/en/4.1/topics/http/urls/ Accessed 10 March 2023

Django. 2005-2023. *Documentation – Using the Django authentication system*. Available at: https://docs.djangoproject.com/en/4.1/topics/auth/default/ Accessed 10 March 2023

Django. 2005-2023. *Documentation – Working with forms*. Available at: https://docs.djangoproject.com/en/4.1/topics/forms/ Accessed 15 March 2023.

Django. 2005-2023. *Why Django?* Available at: https://www.djangoproject.com/start/overview/ Accessed 24 February 2023.

Django. 2005-2023. *Writing your first Django app, part 1*. Available at: https://docs.djangoproject.com/en/4.1/intro/tutorial01/ Accessed 15 March 2023

Docker Docs. 2013-2023. *Compose file versions and upgrading*. Available at: https://docs.docker.com/compose/compose-file/compose-versioning/. Accessed 14 March 2023.

Docker Docs. 2013-2023. *Docker Compose overview*. Available at: https://docs.docker.com/compose/ Accessed 14 March 2023

Docker Docs. 2013-2023. *Docker Compose*. Available at: https://docs.docker.com/compose/ Accessed 24 February 2023.

Docker Docs. 2013-2023. *Docker Docs*. Available at: https://docs.docker.com/get-started/ Accessed 15 March 2023

Douglas, K & Douglas, S. 2003. *PostgreSQL – A comprehensive guide to building, programming, and administering PostgreSQL databases*. Available at: https://books.google.fi/books?hl=en&lr=&id=gkQVL9pyFVYC&oi=fnd&pg=PA1&dq=postgresql&ots=E9ygWQFBkT&sig=ErA72K5gYMRBMRJOC0Opbpisa4I&redir_esc=y#v=onepage&q=postgresql&f=false Accessed 10 March 2023.

Klein, M. 2021. *What is bootstrap?* Available at: https://www.codecademy.com/resources/blog/what-is-bootstrap/. Accessed 24 Feburary 2023.

Microsoft, 2022. *What is Docker?*. Available at: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined. Accessed 24 Feburary 2023.

Microsoft. 2023. *Docker terminology*. Available at: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-terminology Accessed 24 February 2023.

Miller, S. 2021. *What Is Python Used For?* Available at: https://www.codecademy.com/resources/blog/what-is-python-used-for/. Accessed 24 Fedburary 2023.

Mohammad, O. 2022. *New Virtual Environment Based on Python Programming*. Available at: https://www.iasj.net/iasj/download/0e6332a7c15d8f4e. Accessed 15 March 2023.

Python Software Foundation, 2001-2023. *Python Virtual Environment*. Available at: https://docs.python.org/3/library/venv.html. Accessed 24 Feburary 2023.

Python Software Foundation, 2001-2023. *Virtual Environments and Packages*. Available at: https://docs.python.org/3/tutorial/venv.html Accessed 24 February 2023.

Python Software Foundation, 2001-2023. *What is Python? Executive Summary*. Available at: https://www.python.org/doc/essays/blurb/ Accessed 15 March 2023.

Renaud, K. *QUANTIFYING THE QUALITY OF WEB AUTHENTICATION MECHANISMSA USABILITY PERSPECTIVE*. Available at: https://journals.riverpublishers.com/index.php/JWE/article/view/4341/3109 Accessed 09 March 2023.

Sailer, Z. 2018. *python-for-scientists - Documentation – Virtual Environments.* Available at: https://python-for-scientists.readthedocs.io/en/latest/_pages/environments.html Accessed 15 March 2023

The pip developers. 2023. *Pypi - Pip 23.0.1 – Project description*. Available at: https://pypi.org/project/pip/ Accessed 15 March 2023.

Tran, L. 2023. *GitHub repo*. Available at: https://github.com/milotr/Management_Website Accessed 16 March 2023.

Vincent, W. *Django for Beginners: Build websites with Python and Django.* Available at: https://books.google.fi/books?hl=en&lr=&id=GVxwDwAAQBAJ&oi=fnd&pg=PT7&dq=django+installed+apps&ots=fvrvviGkSV&sig=81ocMQ4hDrewWoUPD8eWiSqIqzE&redir_esc=y#v=onepage&q=django%20installed%20apps&f=false. Accessed 14 March 2023.

W3Schools. 2009-2023. *MVC Overview*. Available at: https://www.w3schools.in/mvc-architecture Accessed 24 February 2023.

APPENDIX 1/1

This section explains how these fabric shop works by explaining the current paperwork system and its problems. Therefore, suggest a web-based application to support the business.

Figure 3. BOARD OF FABRIC SAMPLES

One of the problems is it's difficult to keep track of the quantities and lengths of these fabric rolls.

As we can see that the basic web-based application with databases and CRUD will help the business tremendously. The databases can store the information about the customers, fabric rolls and even orders that customers have made. Due to the nature of being a web-based application, the retailers can use phones, tablets, or computers to access his or her information about fabric rolls, customers, orders easily.

The current problems:

- It's hard to keep tracks when new fabric rolls are imported
- It's hard to get the information when customers ask
- It's hard to locate the location of the fabric rolls
- It's hard to know what length the fabric roll is
- It's hard to keep track of the customers
- It's hard to keep track of the orders

APPENDIX 1/2

The objectives:

- A platform to manage the fabric business
- Databases for customers, fabric rolls, orders
- Simple, easy to use UI
- Authentication
- CRUD the data on the databases

Focus on:

- Authentication
- CRUD the data on the databases
- Database
- Dockerize