**CHAPTER 5: IMPLEMENTATION**

**5.1 Mapping Models to Code**

The **Ethiopian Stock Market Simulation Platform** transforms the designed models into functional Django components. Each application (Users, Stocks, Regulations, Notifications, etc.) is developed with clear responsibilities and well-organized code.

**5.1.1 Users App**

**Model:**

**CustomUser**  
Below is a simplified sample of the custom user model extending Django’s AbstractUser:

# users/models.py

from django.contrib.auth.models import AbstractUser

from django.db import models

from django.utils import timezone

from ethio\_stock\_simulation.utils import generate\_otp

class CustomUser(AbstractUser):

ROLE\_CHOICES = [

('trader', 'Trader'),

('regulator', 'Regulator'),

('company\_admin', 'Company Admin'),

]

role = models.CharField(max\_length=15, choices=ROLE\_CHOICES, default='trader')

is\_approved = models.BooleanField(default=False)

kyc\_document = models.FileField(upload\_to='kyc\_documents/', blank=True, null=True)

kyc\_verified = models.BooleanField(default=False)

company\_id = models.IntegerField(null=True, blank=True)

account\_balance = models.DecimalField(max\_digits=15, decimal\_places=2, default=0.00, null=True, blank=True)

profit\_balance = models.DecimalField(max\_digits=15, decimal\_places=2, default=0.00, null=True, blank=True)

date\_registered = models.DateTimeField(default=timezone.now)

last\_login = models.DateTimeField(null=True, blank=True)

# OTP Fields

otp\_code = models.CharField(max\_length=6, blank=True, null=True)

otp\_sent\_at = models.DateTimeField(null=True, blank=True)

otp\_verified = models.BooleanField(default=False)

otp\_attempts = models.IntegerField(default=0)

**Serializer:**

**UserSerializer**  
Defines the fields and validation logic for user registration and updates:

# users/serializers.py

from rest\_framework import serializers

from .models import CustomUser

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = CustomUser

fields = [

'id', 'username', 'email', 'password', 'role', 'kyc\_document',

'kyc\_verified', 'account\_balance', 'profit\_balance',

'otp\_verified', 'otp\_attempts', 'is\_approved'

]

extra\_kwargs = {

'password': {'write\_only': True},

'kyc\_verified': {'read\_only': True},

'is\_approved': {'read\_only': True},

'account\_balance': {'read\_only': True},

'profit\_balance': {'read\_only': True},

}

def create(self, validated\_data):

user = CustomUser.objects.create\_user(

username=validated\_data['username'],

password=validated\_data['password'],

email=validated\_data['email'],

role=validated\_data['role'],

kyc\_document=validated\_data.get('kyc\_document', None),

)

return user

**View:**

**RegisterUser**  
Manages the user registration process, including OTP generation and notification:

# users/views.py

from rest\_framework import generics, status

from rest\_framework.response import Response

from django.utils import timezone

from .serializers import UserSerializer

from .models import CustomUser

from notifications.utils import notify\_user\_email

class RegisterUser(generics.CreateAPIView):

queryset = CustomUser.objects.all()

serializer\_class = UserSerializer

def create(self, request, \*args, \*\*kwargs):

serializer = self.get\_serializer(data=request.data)

serializer.is\_valid(raise\_exception=True)

user = serializer.save()

# Generate and send OTP (simplified for example)

otp = "123456"

user.otp\_code = otp

user.otp\_sent\_at = timezone.now()

user.save()

subject = "Verify Your ESX Account"

message = f"Dear {user.username},\n\nYour OTP is {otp}.\n\nThank you!"

notify\_user\_email(subject, message, [user.email])

return Response(

{"detail": "Registration successful. OTP sent to your email."},

status=status.HTTP\_201\_CREATED

)

**Key Functionalities:**

* **User Registration:** Allows new users to register by providing essential details. An OTP is generated and sent to the user’s email for verification.
* **KYC Document Handling:** Users can upload KYC documents during registration.
* **Role Assignment:** Assigns roles (Trader, Regulator, Company Admin) based on user input.

**5.1.2 Stocks App**

**Model:**

**Stocks**  
Represents company stocks and associated metadata:

# stocks/models.py

from django.db import models

from django.utils import timezone

class Stocks(models.Model):

ticker\_symbol = models.CharField(max\_length=10, unique=True)

company = models.ForeignKey('ListedCompany', on\_delete=models.CASCADE, related\_name='stocks')

total\_shares = models.IntegerField()

current\_price = models.DecimalField(max\_digits=15, decimal\_places=2)

available\_shares = models.IntegerField()

max\_trader\_buy\_limit = models.IntegerField(default=1000)

created\_at = models.DateTimeField(default=timezone.now)

def \_\_str\_\_(self):

return f"{self.ticker\_symbol} ({self.company.company\_name})"

**Serializer:**

**StocksSerializer**

# stocks/serializers.py

from rest\_framework import serializers

from .models import Stocks

class StocksSerializer(serializers.ModelSerializer):

class Meta:

model = Stocks

fields = '\_\_all\_\_'

**View:**

**StocksViewSet**  
Implements RESTful endpoints for managing stocks:

# stocks/views.py

from rest\_framework import viewsets

from .models import Stocks

from .serializers import StocksSerializer

from rest\_framework.permissions import IsAuthenticated

class StocksViewSet(viewsets.ModelViewSet):

queryset = Stocks.objects.all()

serializer\_class = StocksSerializer

permission\_classes = [IsAuthenticated]

**Key Functionalities:**

* **Stock Management:** Handles essential operations on stocks, including viewing, creating, and updating.
* **Direct Purchase Logic:** Can be extended to bypass order matching for direct stock purchases.

**Key Functionalities:**

* **Email Dispatching:** Sends notifications (trade confirmations, KYC updates, regulatory alerts) without storing them in the database, optimizing performance.

**5.1.3 Suspicious Activity App**

**Model: SuspiciousActivity**

Tracks and manages suspicious trading activities.

# suspicious\_activity/models.py

from django.db import models

from django.utils import timezone

from trades.models import Trade

class SuspiciousActivity(models.Model):

reason = models.TextField()

flagged\_at = models.DateTimeField(default=timezone.now)

reviewed = models.BooleanField(default=False)

trade = models.ForeignKey(Trade, on\_delete=models.CASCADE, related\_name='suspicious\_activities')

def \_\_str\_\_(self):

return f"Suspicious Activity for Trade ID: {self.trade.id}"

**Serializer: SuspiciousActivitySerializer**

Serializes suspicious activity data.

# suspicious\_activity/serializers.py

from rest\_framework import serializers

from .models import SuspiciousActivity

class SuspiciousActivitySerializer(serializers.ModelSerializer):

class Meta:

model = SuspiciousActivity

fields = '\_\_all\_\_'

read\_only\_fields = ['flagged\_at', 'reviewed']

**View: SuspiciousActivityViewSet**

Manages creation and review of suspicious activities.

# suspicious\_activity/views.py

from rest\_framework import viewsets, status

from rest\_framework.response import Response

from rest\_framework.permissions import IsAuthenticated

from .models import SuspiciousActivity

from .serializers import SuspiciousActivitySerializer

class SuspiciousActivityViewSet(viewsets.ModelViewSet):

queryset = SuspiciousActivity.objects.all()

serializer\_class = SuspiciousActivitySerializer

permission\_classes = [IsAuthenticated]

def create(self, request, \*args, \*\*kwargs):

serializer = self.get\_serializer(data=request.data)

serializer.is\_valid(raise\_exception=True)

suspicious\_activity = serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

def update(self, request, \*args, \*\*kwargs):

instance = self.get\_object()

instance.reviewed = True

instance.save()

serializer = self.get\_serializer(instance)

return Response(serializer.data)

**Key Functionalities:**

* **Activity Monitoring:** Detects and flags unusual trading patterns.
* **Review Process:** Allows regulators to review and mark activities as reviewed, ensuring accountability.

### ****5.2 Testing and Evaluation****

Upon completing the development of the **Ethiopian Stock Market Simulation Platform**, a comprehensive testing and evaluation process is essential to ensure its functionality, reliability, and security.

**Unit testing** is conducted to verify that each individual component, such as models, serializers, and views, operates correctly. The goal is to achieve at least 80% test coverage, ensuring that most parts of the application are thoroughly tested.

**Integration testing** follows, where different modules like Users, Stocks, and Trading are tested together to confirm they interact seamlessly. This includes scenarios such as user registration, trade execution, and notification dispatching. Additionally, **performance and stress testing** assesses the platform's ability to handle a numbers of simultaneous users and transactions, ensuring smooth operation.

**Security testing** is performed to identify and address potential vulnerabilities, safeguarding user data and maintaining system integrity. Finally, **User Acceptance Testing (UAT)** involves real users Traders, Regulators, and Company Admins to validate the platform's usability and effectiveness in a real-world context. This structured testing approach guarantees that the platform meets high standards of quality and is ready for deployment.

### ****5.3 System Maintenance****

Ensuring the long-term success of the **Ethiopian Stock Market Simulation Platform** requires diligent system maintenance to guarantee its availability, performance, and security. **Preventive maintenance** involves regular monitoring of system performance and conducting routine security assessments to proactively identify and address potential issues. This includes applying necessary software updates and patches to keep the system secure and efficient.

**Corrective maintenance** addresses any unexpected problems or bugs that arise, ensuring they are resolved promptly to minimize downtime and user disruption. Additionally, **system updates** are regularly performed to upgrade the operating system, frameworks, and third-party components, maintaining compatibility and enhancing security. To support scalability, server resources are adjusted.

**Security maintenance** is a continuous effort, involving regular vulnerability scans and access reviews to enforce strict security protocols and protect sensitive data. Comprehensive **documentation and training** are maintained to ensure that the development and maintenance teams are well-equipped to manage the system effectively. By adhering to these maintenance practices, the platform remains reliable, secure, and capable of providing a seamless user experience.

**CHAPTER 6: CONCLUSION AND RECOMMENDATION**

**6.1 Conclusion**

The **Ethiopian Stock Market Simulation Platform** is a groundbreaking initiative aimed at preparing Ethiopian stakeholders for the launch of the Ethiopian Stock Market. By integrating **user management**, **KYC verification**, **stock trading mechanisms**, and **regulatory oversight** within a robust Django framework, the platform effectively simulates real-world trading in a **risk-free environment**. Key features include seamless user registration with OTP verification, comprehensive KYC processes, advanced trading functionalities, real-time notifications, and proactive monitoring of suspicious activities. Leveraging **PostgreSQL’s reliability** and **Django’s extensibility**, the platform is designed to scale and adapt to evolving regulatory requirements. Drawing inspiration from established simulators like **Investopedia's Simulator**, the **Ethiopian Stock Market Simulation Platform** upholds industry-standard security and reliability, positioning itself as an essential tool for **education**, **strategy testing**, and **regulatory training** within Ethiopia's emerging stock market ecosystem.

**6.2 Recommendation**

To further enhance the **Ethiopian Stock Market Simulation Platform** and ensure its effectiveness as a training tool, the following recommendations are proposed:

1. **Advanced Analytical Tools and Dashboards**
   * **Implementation:** Integrate real-time dashboards using libraries like **Chart.js** or **D3.js** to visualize market trends and trading performance.
   * **Benefits:** Provides users with actionable insights and enables regulators to monitor market activities more effectively.
2. **Extended Order Types and Financial Instruments**
   * **Implementation:** Introduce complex order types such as **stop-loss** and **iceberg orders**, and expand support to additional financial instruments like **bonds** and **commodities**.
   * **Benefits:** Offers a more realistic trading environment and accommodates a wider range of trading strategies.
3. **AI-Driven Surveillance and Anomaly Detection**
   * **Implementation:** Develop machine learning models to analyze trading patterns and detect fraudulent activities or market manipulations.
   * **Benefits:** Enhances the platform’s ability to maintain market integrity and reduces reliance on manual monitoring.
4. **Automated Regulatory Compliance and Reporting**
   * **Implementation:** Automate the generation of compliance reports and streamline KYC verification workflows.
   * **Benefits:** Streamlines regulatory processes, ensuring timely and accurate compliance with minimal administrative overhead.
5. **Enhanced User Experience and Interface Improvements**
   * **Implementation:** Continuously refine the user interface based on feedback, incorporating features like **customizable dashboards** and **interactive tutorials**.
   * **Benefits:** Improves user satisfaction and facilitates easier adoption across diverse user groups.
6. **Integration with External Financial Data Providers**
   * **Implementation:** Connect the platform with external APIs to fetch real-time financial data and market indicators.
   * **Benefits:** Enriches the simulation experience by reflecting actual market dynamics, providing users with comprehensive information for informed trading strategies.

By implementing these enhancements, the **Ethiopian Stock Market Simulation Platform** will significantly improve its functionality, user engagement, and compliance capabilities. This will ensure the platform remains a leading tool for stock trading simulations and regulatory training, effectively preparing Ethiopian stakeholders for the successful launch and operation of the Ethiopian Stock Exchange.