**Project E.U - EmpowerUs**

Overview:

Project E.U aims to create a robust and user-friendly web application to empower both workers and investors. The platform will accommodate a large user base, ranging from 19 to 70 years old. The application will have distinct sections for workers and investors, allowing seamless interaction and transactions.

1. Facts:

i. Web Application

ii. Capacity for 4000+ users

iii. User-friendly navigation for ages 19-70

2. Account Creation:

For Workers:

i. Click "Become a Worker"

ii. Login/Sign-up

iii. Agree to company policy

iv. Provide personal information

v. Choose subscription plan (monthly)

- 5,000frs = 2 hours daily

- 10,000frs = 4 hours daily

- 15,000frs = 6 hours daily

- 20,000frs = 8 hours daily

vi. Payment via Banks, MTN Mobile Money, Orange Money

vii. Agree to terms and conditions

viii. Task assignment and completion within subscribed hours

ix. Marketing task: Create a personalized video recommendation

For Investors:

i. Create an Investor account

ii. Provide ID Card/Passport, business contact, and domain of interest

iii. Select payment method (Bank, MTN Mobile Money, Orange Money)

iv. Manual approval by platform owners

3. Graphics:

i. Investors upload products

ii. Workers market products with clickable "Buy" buttons

iii. Buyers remain on their platform for purchases

iv. Real-time notifications for investors on purchases

v. Efficient traffic routing and high-speed accessibility

4. Time:

i. Investors can upload products 24/7

ii. Buyers have 2 minutes to read/fill information, then "Buy" again

iii. Dormant accounts auto-renew policy

iv. Subscription renewal reminders

5. Investments:

i. Investors create accounts with ID verification

ii. Select domain of interest

iii. Provide contact and business details

iv. Manual approval by platform owners

DevOps Methodology:

Continuous Integration/Continuous Deployment (CI/CD)

Version control (Git)

Automated testing

Infrastructure as Code (IaC)

Agile development

Pseudocode/Algorithm:

(Note: Pseudocode is a simplified representation)

**ALGORITHM-Pseudocode(Python)**

**This Python pseudocode represents the key functionalities and processes of Project E.U, providing a clear and structured outline for implementation.**

**# Project E.U - EmpowerUs**

**# Overview**

**# Project E.U aims to create a robust and user-friendly web application to empower both workers and investors.**

**# The platform will accommodate a large user base, ranging from 19 to 70 years old.**

**# The application will have distinct sections for workers and investors, allowing seamless interaction and transactions.**

**# 1. Facts**

**# i. Web Application**

**# ii. Capacity for 4000+ users**

**# iii. User-friendly navigation for ages 19-70**

**# 2.a) Account Creation for Workers**

**def worker\_account\_creation():**

**click("Become a Worker")**

**login\_or\_sign\_up()**

**agree\_to\_policy()**

**provide\_personal\_information()**

**choose\_subscription\_plan()**

**make\_payment()**

**agree\_to\_terms\_and\_conditions()**

**assign\_task()**

**create\_marketing\_video()**

**# 2.b) Account Creation for Investors**

**def investor\_account\_creation():**

**create\_investor\_account()**

**provide\_id\_verification()**

**select\_domain\_of\_interest()**

**provide\_contact\_details()**

**make\_payment()**

**manual\_approval\_by\_platform\_owners()**

**# 3. Graphics**

**# i. Investors upload products**

**# ii. Workers market products with clickable "Buy" buttons**

**# iii. Buyers remain on their platform for purchases**

**# iv. Real-time notifications for investors on purchases**

**# v. Efficient traffic routing and high-speed accessibility**

**# 4. Time**

**# i. Investors can upload products 24/7**

**# ii. Buyers have 2 minutes to read/fill information, then "Buy" again**

**# iii. Dormant accounts auto-renew policy**

**# iv. Subscription renewal reminders**

**# 5. Investments**

**# i. Investors create accounts with ID verification**

**# ii. Select domain of interest**

**# iii. Provide contact and business details**

**# iv. Manual approval by platform owners**

**# DevOps Methodology**

# - Continuous Integration/Continuous Deployment (CI/CD)

# - Version control (Git)

# - Automated testing

# - Infrastructure as Code (IaC)

# - Agile development

**# Pseudocode/Algorithm in Python**

**# Worker Account Creation**

def worker\_account\_creation():

click("Become a Worker")

login\_or\_sign\_up()

agree\_to\_policy()

provide\_personal\_information()

choose\_subscription\_plan()

make\_payment()

agree\_to\_terms\_and\_conditions()

assign\_task()

create\_marketing\_video()

**# Investor Account Creation**

def investor\_account\_creation():

create\_investor\_account()

provide\_id\_verification()

select\_domain\_of\_interest()

provide\_contact\_details()

make\_payment()

manual\_approval\_by\_platform\_owners()

**# Product Marketing Task**

def product\_marketing\_task():

upload\_products()

workers\_market\_products()

create\_clickable\_buy\_buttons()

seamless\_transfer\_of\_images\_with\_buy\_buttons()

buyers\_purchase\_on\_their\_platform()

notify\_investors\_on\_purchases()

efficient\_traffic\_routing()

**# Subscription Renewal Reminder**

def subscription\_renewal\_reminder():

if account\_dormant\_for\_month():

send\_renewal\_reminder()

else:

continue()

**# Finance**

**This pseudocode ensures that the profit distribution is calculated based on predetermined percentage values, and the record\_profit function handles the recording of profits for each group. The profit distribution starts with the owners receiving the highest percentage, followed by investors and then marketers. Adjust the percentage values according to your desired profit-sharing structure.**

**# Constants**

OWNER\_PROFIT\_PERCENTAGE = 0.40 # 40% of total profit

INVESTOR\_PROFIT\_PERCENTAGE = 0.30 # 30% of total profit

MARKETER\_PROFIT\_PERCENTAGE = 0.10 # 10% of total profit

**# Other constants and functions remain the same**

**# Function to calculate profit distribution**

def calculate\_profit\_distribution(total\_profit):

owner\_profit = total\_profit \* OWNER\_PROFIT\_PERCENTAGE

investor\_profit = total\_profit \* INVESTOR\_PROFIT\_PERCENTAGE

marketer\_profit = total\_profit \* MARKETER\_PROFIT\_PERCENTAGE

return owner\_profit, investor\_profit, marketer\_profit

**# Worker Account Creation**

def worker\_account\_creation():

**# Existing worker account creation code**

**# ...**

**# Calculate and distribute profit**

total\_profit = calculate\_total\_profit(subscription\_fee)

owner\_profit, investor\_profit, marketer\_profit = calculate\_profit\_distribution(total\_profit)

record\_profit(owner\_profit, investor\_profit, marketer\_profit)

**# Investor Account Creation**

def investor\_account\_creation():

**# Existing investor account creation code**

**# ...**

**# Calculate and distribute profit**

total\_profit = calculate\_total\_profit(investment\_amount)

owner\_profit, investor\_profit, marketer\_profit = calculate\_profit\_distribution(total\_profit)

record\_profit(owner\_profit, investor\_profit, marketer\_profit)

**# Product Marketing Task**

def product\_marketing\_task():

**# Existing product marketing task code**

**# ...**

**# Calculate and distribute profit**

total\_profit = calculate\_total\_profit(product\_sales)

owner\_profit, investor\_profit, marketer\_profit = calculate\_profit\_distribution(total\_profit)

record\_profit(owner\_profit, investor\_profit, marketer\_profit)

**# Function to calculate total profit based on different activities**

def calculate\_total\_profit(amount):

**# Add any other factors influencing profit**

return amount

**# Function to record profit distribution**

def record\_profit(owner\_profit, investor\_profit, marketer\_profit):

owners\_record\_profit(owner\_profit)

investors\_record\_profit(investor\_profit)

marketers\_record\_profit(marketer\_profit)

**# Additional functions to handle profit records for owners,** investors, and marketers

def owners\_record\_profit(profit):

**# Record profit for owners (hidden from others)**

**# ...**

**def investors\_record\_profit(profit):**

**# Record profit for investors**

**# ...**

**def marketers\_record\_profit(profit):**

**# Record profit for marketers**

**# ...**

**TASK IN FORM OF TICKETS**

**Ticket 1: Web Application Setup**

**Role**: DevOps Engineer

**Description**:

Set up the infrastructure and deploy the web application.

**Pseudocode:**

# Infrastructure as Code (IaC)

def setup\_infrastructure():

# Use tools like Terraform, AWS CloudFormation, or Azure Resource Manager

# to define and provision the required infrastructure.

# Continuous Integration/Continuous Deployment (CI/CD)

def configure\_ci\_cd():

# Set up CI/CD pipelines using Jenkins, GitLab CI, or GitHub Actions.

# Ensure automated deployment to the web server after successful builds.

**Ticket 2: Version Control**

**Role:** DevOps Engineer

**Description:**

Implement version control for the project using Git.

**Pseudocode:**

# Version Control (Git)

def setup\_git\_repository():

# Create a Git repository for the project.

# Initialize the repository and commit the initial codebase.

**Ticket 3: User Account Creation - Workers**

**Role:** Backend Developer

**Description:**

Implement the backend logic for worker account creation.

**Pseudocode:**

# Worker Account Creation Backend Logic

def worker\_account\_creation\_backend():

# Define API endpoints for worker account creation.

# Implement backend logic to handle the account creation process.

# Integrate with databases for storing user information.

**Ticket 4: User Account Creation - Investors**

**Role**: Backend Developer

**Description:**

Implement the backend logic for investor account creation.

**Pseudocode:**

# Investor Account Creation Backend Logic

def investor\_account\_creation\_backend():

# Define API endpoints for investor account creation.

# Implement backend logic to handle the account creation process.

# Integrate with databases for storing investor information.

**Ticket 5: Frontend - Account Creation Forms**

**Role:** Frontend Developer

**Description**:

Design and implement the user interface for worker and investor account creation forms.

**Pseudocode:**

# Frontend - Worker Account Creation Form

def worker\_account\_creation\_form():

# Design and implement the UI for worker account creation.

# Include form fields for personal information, subscription plan, etc.

# Frontend - Investor Account Creation Form

def investor\_account\_creation\_form():

# Design and implement the UI for investor account creation.

# Include form fields for ID verification, domain of interest, etc.

**Ticket 6: Payment Integration**

**Role**: Backend Developer

**Description:**

Integrate payment methods for both workers and investors.

**Pseudocode:**

# Payment Integration Backend Logic

def payment\_integration\_backend():

# Integrate payment gateways such as Banks, MTN Mobile Money, Orange Money.

# Implement logic for processing payments and subscription renewals.

**Ticket 7: Graphics and Real-time Notifications**

**Role**: Frontend Developer

**Description:**

Implement graphics for product uploads and real-time notifications for investors.

**Pseudocode:**

# Frontend - Product Upload Graphics

def product\_upload\_graphics():

# Design and implement graphics for investors to upload products.

# Frontend - Real-time Notifications

def real\_time\_notifications():

# Implement real-time notifications for investors on purchases.

**Ticket 8: Time Management**

**Role**: Backend Developer

**Description:**

Implement time-related functionalities for product uploads, buyer actions, and subscription renewals.

**Pseudocode:**

# Time Management Backend Logic

def time\_management\_backend():

# Implement logic for product uploads 24/7, buyer actions, and subscription renewals.

# Set up reminders for subscription renewals.

**Ticket 9: Investments and Manual Approval**

**Role:** Backend Developer

**Description:**

Implement the investor registration process and manual approval by platform owners.

**Pseudocode:**

# Investor Registration and Approval Backend Logic

def investor\_registration\_and\_approval():

# Implement logic for investors to create accounts with ID verification.

# Include manual approval process by platform owners.

**Ticket 10: Profit Distribution**

**Role**: Backend Developer

**Description:**

Implement the logic for calculating and distributing profits to owners, investors, and marketers.

**Pseudocode:**

# Profit Distribution Backend Logic

def profit\_distribution\_backend():

# Implement logic to calculate total profit based on different activities.

# Distribute profits to owners, investors, and marketers.

# Record profit distribution for each group.

**Ticket 11: Task Assignment and Marketing Video**

**Role**: Backend Developer

**Description:**

Implement the backend logic for task assignment and creating marketing videos.

**Pseudocode:**

# Task Assignment and Marketing Video Backend Logic

def task\_assignment\_and\_marketing\_video():

# Implement logic for assigning tasks to workers.

# Include logic for workers to create personalized marketing videos.

**Ticket 12: Subscription Renewal Reminder**

**Role:** Backend Developer

**Description**:

Implement the logic for sending subscription renewal reminders.

**Pseudocode**:

# Subscription Renewal Reminder Backend Logic

def subscription\_renewal\_reminder\_backend():

# Implement logic to check for dormant accounts and send renewal reminders.

These tickets cover various aspects of the Project E.U - EmpowerUs and can be assigned to different team members based on their expertise. Each ticket includes specific tasks and corresponding pseudocode to guide the implementation.