# **Software Requirements Specification (SRS) for ZenithAchievers**

## 1. Introduction

### 1.1 Purpose

The purpose of this document is to outline the requirements for the development of the ZenithAchievers platform, focusing on the admin panel, user functionalities, and database implementation.

### 1.2 Scope

The system will include an admin panel for managing user logins, deposits, referrals, and product reviews. Users will be able to download products, post them on WhatsApp, and submit views and screenshots for earnings. The system will also track referrals and package purchases.

## 2. Overall Description

### 2.1 System Features

* **Admin Panel**: Manage user logins, view user activities, monitor deposits, approve/reject deposits, and view referral networks.
* **User Features**: Download products, post on WhatsApp, submit views and screenshots, and track referrals.
* **Database**: Implement tables for deposits, package purchases, WhatsApp earnings, product reviews, and referrals.

### 2.2 User Classes

* **Admin**: Manages user accounts, deposits, and product reviews.
* **Users**: Download products, post on WhatsApp, and track earnings and referrals.

## 3. Functional Requirements

### 3.1 Admin Panel

1. **Control All Logins**
   * View and manage all user logins.
   * Disable or reset user passwords.
2. **View All User Logins**
   * Dashboard showing all user logins with timestamps, IP addresses, and metadata.
   * Search/filter by username, date, or status.
3. **View All Downline (Referrals)**
   * Display a network structure of direct and indirect referrals.
   * Implement a tree or hierarchical view.
4. **Monitor Deposits**
   * View all deposits with filters by date, amount, user, or status.
   * Approve, reject, or mark deposits as pending.
5. **Approve Deposits**
   * Approve or reject deposits after reviewing transaction details.
   * Each deposit has a status (Pending, Approved, Rejected).

### 3.2 Database Implementation

1. **Deposits Table**
   * Fields: Deposit ID, User ID, Amount, Status, Date, Payment Method, Screenshot.
2. **Package Purchases Table**
   * Fields: Package ID, User ID, Purchase Date, Package Details, Payment Status.
3. **WhatsApp Earnings**
   * Fields: User ID, Date, Product ID, Earnings, Views, Screenshot URL.
4. **Product Reviews**
   * Fields: Review ID, Product ID, User ID, Rating, Review Text, Download Date.
5. **WhatsApp Section**
   * Fields: User ID, Product ID, Number of Views, Screenshot URL.
6. **Referral Monitoring**
   * Fields: User ID, Referrals, Total Referrals, Earnings from Referrals.

### 3.3 Frontend Features for Users

1. **Download Product and Post on WhatsApp**
   * Download product, post on WhatsApp, and submit views and screenshot after 24 hours.
2. **Referral Program**
   * View referrals and track progress.
3. **WhatsApp Section**
   * Input number of views and upload screenshot for earnings tracking.

## 4. Non-Functional Requirements

### 4.1 Authentication and Security

* Secure login for admin.
* Two-factor authentication.
* Log and monitor user activities.

### 4.2 Notifications

* Admin notifications for new deposits and WhatsApp earnings submissions.

## 5. System Attributes

### 5.1 Usability

* Intuitive admin panel and user interface.

### 5.2 Performance

* Efficient handling of user data and transactions.

### 5.3 Scalability

* Ability to handle increasing numbers of users and transactions.

## 6. Additional Considerations

* Ensure data privacy and security.
* Regular backups and system updates.

**Chat Extracts**

*I would like to have an admin panel implemented where I can:*

*1. Control all the logins.*

*2. See all user logins.*

*3. View all my downline.*

*4. Monitor my deposits.*

*5. Approve deposits.*

*Please let me know if you need any additional information or have any questions. Thank you!"*

*The only important databases that need to be implemented are as follows:*

*1. Deposits.*

*2. Package purchases.*

*3. WhatsApp earnings reflecting on the dashboard.*

*4. Product reviews where users will download the product and post on their status.*

*5. A section labeled 'WhatsApp' where users can add the number of views and drop a screenshot, so that they can reflect on the users' dashboard.*

*6. Monitor deposits and the number of referrals.*

*7. An admin panel with the ability to approve deposits and log in to users' accounts.*

*On the product review page, there should be a section to download the product, which will then appear in the gallery. Users can post the product on WhatsApp and, after 24 hours, they should be able to submit the number of views and a screenshot as proof. The earnings can then reflect on the WhatsApp Earnings dashboard.*

*\*Contact Admin:\* Since this is not my company alone, there shouldn't be an email listed under the 'Contact Admin' section.*

*2. \*Package Purchase:\* When someone presses the purchase button on a package, it should show whether they've paid for the package or not. If they haven't paid, it should indicate that they need to make a purchase.*

*3. \*Withdrawal Confirmation:\* When someone withdraws WhatsApp earnings or makes any withdrawal, it should show a confirmation that they've successfully made the withdrawal.*

*4. \*Share Plan:\* There should be a package just like Brifox, actually two packages—one for Ksh 6000 and the other for Ksh 10,000.*

*5. \*Finances:\* Referral withdrawals have not been indicated.*

*6. \*History Section:\* It should show the number of referrals and the packages they've purchased.*

*7. \*Dollar Zone:\* There should be packages available for purchase as well.*

*8.*

To implement a referral system in Django, follow these steps:

### 1. ****Modify the Profile Model****

Uncomment and modify the invited\_by field to establish a self-referential relationship.

python

CopyEdit

class Profile(models.Model):

user = models.OneToOneField(

settings.AUTH\_USER\_MODEL,

on\_delete=models.CASCADE

)

invited\_by = models.ForeignKey(

settings.AUTH\_USER\_MODEL,

related\_name='invited\_users',

on\_delete=models.SET\_NULL,

null=True,

blank=True

)

unique\_referral = models.CharField(max\_length=20, unique=True, blank=True, null=True)

def save(self, \*args, \*\*kwargs):

if not self.unique\_referral:

self.unique\_referral = self.user.username # Generate referral code from username

super().save(\*args, \*\*kwargs)

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

return f'Profile of {self.user.username}'

### 2. ****Generate the Referral Link****

Each user gets a unique referral link based on their username.

python

CopyEdit

def get\_referral\_link(user):

return f"https://website.com/register/?invitedby={user.username}"

### 3. ****Modify the Registration Form to Accept Referral Code****

Update the UserRegistrationForm to accept a referral code.

python

CopyEdit

class UserRegistrationForm(forms.ModelForm):

password = forms.CharField(

label='Password',

widget=forms.PasswordInput

)

password2 = forms.CharField(

label='Repeat password',

widget=forms.PasswordInput

)

referral\_code = forms.CharField(

label='Referral Code (Optional)',

required=False

)

class Meta:

model = get\_user\_model()

fields = ['username', 'email']

def clean\_password2(self):

cd = self.cleaned\_data

if cd['password'] != cd['password2']:

raise forms.ValidationError("Passwords don't match.")

return cd['password2']

### 4. ****Modify the View to Handle Referral Code****

Modify your Django registration view to capture the referral code and link the new user to the referrer.

python

CopyEdit

def register(request):

referral\_code = request.GET.get('invitedby', None) # Get referral code from URL

if request.method == 'POST':

form = UserRegistrationForm(request.POST)

if form.is\_valid():

user = form.save(commit=False)

user.set\_password(form.cleaned\_data['password'])

user.save()

# Create profile

profile = Profile.objects.create(user=user)

# Assign the referrer if a valid referral code is provided

if referral\_code:

try:

referrer = get\_user\_model().objects.get(username=referral\_code)

profile.invited\_by = referrer

profile.save()

except get\_user\_model().DoesNotExist:

pass # Ignore if referrer does not exist

return redirect('login') # Redirect to login or another page

else:

form = UserRegistrationForm()

return render(request, 'register.html', {'form': form, 'referral\_code': referral\_code})

### 5. ****Update the Registration Template****

Modify register.html to prefill the referral input field.

html

CopyEdit

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<input type="hidden" name="referral\_code" value="{{ referral\_code }}">

<button type="submit">Register</button>

</form>

Here's how you can create a Django form, view, and model to allow users to submit a phone number, number of views, and a screenshot image.

### 1. Create the Django Model (models.py)

python

CopyEdit

from django.db import models

class Submission(models.Model):

phone\_number = models.CharField(max\_length=15)

number\_of\_views = models.PositiveIntegerField()

screenshot = models.ImageField(upload\_to='screenshots/')

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

return self.phone\_number

### 2. Create the Django Form (forms.py)

python

CopyEdit

from django import forms

from .models import Submission

class SubmissionForm(forms.ModelForm):

class Meta:

model = Submission

fields = ['phone\_number', 'number\_of\_views', 'screenshot']

def clean\_screenshot(self):

screenshot = self.cleaned\_data.get('screenshot')

if screenshot:

ext = screenshot.name.split('.')[-1].lower()

if ext not in ['png', 'jpg', 'jpeg']:

raise forms.ValidationError("Only .png, .jpg, and .jpeg files are allowed.")

return screenshot

### 3. Create the Django View (views.py)

python

CopyEdit

from django.shortcuts import render, redirect

from .forms import SubmissionForm

def submit\_data(request):

if request.method == "POST":

form = SubmissionForm(request.POST, request.FILES)

if form.is\_valid():

form.save()

return redirect('success\_page') # Change 'success\_page' to your success URL name

else:

form = SubmissionForm()

return render(request, 'submit\_form.html', {'form': form})

### 4. Create the Template (submit\_form.html)

html

<form method="post" enctype="multipart/form-data">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Submit</button>

</form>

### 5. Configure urls.py

python

CopyEdit

from django.urls import path

from .views import submit\_data

urlpatterns = [

path('submit/', submit\_data, name='submit\_data'),

]

### 6. Configure settings.py for Media Files

Add the following settings to handle file uploads:

python

CopyEdit

import os

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

Also, update urls.py to serve media files during development:

python

CopyEdit

from django.conf import settings

from django.conf.urls.static import static

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)