**ЗМІСТ**

[**Серверна частина (бекенд)** 2](#_Toc196929304)

[**Основні задачі та їх виконання** 3](#_Toc196929305)

[**Етапи** 4](#_Toc196929306)

[**Створення бекенду поетапно** 5](#_Toc196929307)

[**Тестування API токенів в Postman** 11](#_Toc196929308)

[**Тестування створення полісів в Postman** 13](#_Toc196929309)

[**API Ендпоїтни** 15](#_Toc196929310)

# **Серверна частина (бекенд)**

**Тема:** інформаційна система автострахування, яка дозволяє клієнтам оформляти поліси онлайн, а агентам — керувати страхуванням.

**Використані технології**

* **Python 3.12**
* **Django 5.x**
* **Django REST Framework**
* **PostgreSQL**
* **JWT авторизація** (djangorestframework-simplejwt)
* **Тестування API** Postman

# **Основні задачі та їх виконання**

**Етап 1: Базова настройка проекту**

* Створили Django-проєкт insurance\_project.
* Створили додаток insurance.
* Налаштували базу даних PostgreSQL.
* Налаштували Django REST Framework.

**Етап 2: Створення моделей**

* **Користувач (User)**:
  + Розширили стандартну модель Django (AbstractUser).
  + Додали ролі: is\_customer, is\_agent.
  + Додали поля: phone\_number, address.
* **Страховий поліс (InsurancePolicy)**:
  + Поля: клієнт, номер авто, тип страхування, дата початку і кінця, ціна, статус.

**Етап 3: Реєстрація і аутентифікація користувачів**

* Створили API для **реєстрації користувачів** (/api/register/).
* Підключили **JWT аутентифікацію** (/api/token/, /api/token/refresh/).

**Етап 4: Управління страховими полісами**

* Створили API для перегляду та створення полісів (/api/policies/).
* Автоматично прив'язали залогіненого користувача до страхового полісу при створенні.
* Налаштували захист через токени (IsAuthenticated).

**Етап 5: Розмежування прав доступу**

* **Клієнт** бачить тільки свої поліси.
* **Агент** бачить всі поліси в системі.

# **Етапи**

* 1. Бекенд
* Створити Django-проект
* Створити django-додаток “insurance”
* Підключити базу даних PostgreSQL
* Налаштувати Django REST Framework
* Створити базові моделі: Customer, Agent, InsurancePolicy
* Зробити API для реєстрації, логіну, та створення полісів
* Захистити API токенами (JWT)
  1. Фронтенд
* Створити React-проект
* Налаштувати базову структуру (сторінки, компоненти)
* Підключитися до бекенду через API
* Реалізувати форму реєстрації, логіну, створення полісу

# **Створення бекенду поетапно**

* 1. **Встановлення інструментів**

pip install django djangorestframework djangorestframework-simplejwt psycopg2

* 1. **Створити проект**

django-admin startproject insurance\_project

cd insurance\_project

* 1. **Створення додатку**

python manage.py startapp insurance

* 1. **Підключити додаток та бібліотеки у файлі insurance\_project/settings.py**

INSTALLED\_APPS = [

...

'rest\_framework',

'rest\_framework\_simplejwt',

'insurance', # наш додаток

]

* 1. **Створення БД в Postgre**

**Host:** localhost

**Password:** який вказували при реєстрації при встановленні Postgre

A close-up of a box

AI-generated content may be incorrect.

* 1. **У settings.py**

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'insurance\_db', # ім'я бази даних

'USER': 'postgres', # твій користувач БД

'PASSWORD': 'yourpassword', # твій пароль БД

'HOST': 'localhost',

'PORT': '5432',

}

}

* 1. **У файл models.py додаємо базові класи**

from django.db import models

from django.contrib.auth.models import AbstractUser

# Користувач (розширений User)

class User(AbstractUser):

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

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

phone\_number = models.CharField(max\_length=20, blank=True)

address = models.CharField(max\_length=255, blank=True)

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

return self.username

# Страховий поліс

class InsurancePolicy(models.Model):

POLICY\_TYPES = (

('basic', 'Basic Coverage'),

('full', 'Full Coverage'),

)

client = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='policies')

vehicle\_number = models.CharField(max\_length=10)

insurance\_type = models.CharField(max\_length=10, choices=POLICY\_TYPES)

start\_date = models.DateField()

end\_date = models.DateField()

price = models.DecimalField(max\_digits=10, decimal\_places=2)

status = models.CharField(max\_length=20, default='active')

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

return f"Policy {self.id} for {self.client.username}"

* 1. **У файлі insurance\_project/settings.py додати під INSTALLED\_APPS**

AUTH\_USER\_MODEL = 'insurance.User'

* 1. **В консолі – директорія проекту, створити міграції і застосувати їх**

python manage.py makemigrations

python manage.py migrate

* 1. **У папці insurance створити serializers.py і зробити схеми реалізації**

from rest\_framework import serializers

from .models import User, InsurancePolicy

# Схема серіалізації для користувача

class UserSerializer(serializers.ModelSerializer):

password = serializers.CharField(write\_only=True)

class Meta:

model = User

fields = ['id', 'username', 'email', 'password', 'is\_customer', 'is\_agent', 'phone\_number', 'address']

def create(self, validated\_data):

password = validated\_data.pop('password')

user = User(\*\*validated\_data)

user.set\_password(password)

user.save()

return user

# Схема серіалізації для страхового полісу

class InsurancePolicySerializer(serializers.ModelSerializer):

class Meta:

model = InsurancePolicy

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

* 1. **В тій ж директорії створити views.py і написати код для реєстрації користувачів**

from rest\_framework import generics

from .models import User, InsurancePolicy

from .serializers import UserSerializer, InsurancePolicySerializer

# Реєстрація користувача

class UserRegistrationView(generics.CreateAPIView):

queryset = User.objects.all()

serializer\_class = UserSerializer

# Список та створення страхових полісів

class InsurancePolicyView(generics.ListCreateAPIView):

queryset = InsurancePolicy.objects.all()

serializer\_class = InsurancePolicySerializer

* 1. **У папці insurance/ створи або доповни файл urls.py**

from django.urls import path

from .views import UserRegistrationView, InsurancePolicyView

urlpatterns = [

path('register/', UserRegistrationView.as\_view(), name='user-register'),

path('policies/', InsurancePolicyView.as\_view(), name='policies'),

]

* 1. **Підключити insurance.urls у головному файлі insurance\_project/urls.py**

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('api/', include('insurance.urls')), # додали сюди наші API

]

* 1. **Запустити сервер через консоль в директорії проекту**

python manage.py runserver

* 1. **Налаштовуємо DFR SimpleJWT у settings.py (Це вказує Django використовувати JWT для авторизації.)**

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': (

'rest\_framework\_simplejwt.authentication.JWTAuthentication',

)

}

* 1. **Додати маршрути для видачі токенів у insurance/urls.py і перейти до тестування**

from rest\_framework\_simplejwt.views import (

TokenObtainPairView,

TokenRefreshView,

)

urlpatterns = [

path('register/', UserRegistrationView.as\_view(), name='user-register'),

path('policies/', InsurancePolicyView.as\_view(), name='policies'),

path('token/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'), # Логін

path('token/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'), # Оновлення токена

]

# **Тестування API токенів в Postman**

Спочатку зареєструємо тестового користувача на <http://127.0.0.1:8000/api/register/>

A screenshot of a black screen

AI-generated content may be incorrect. Створюємо новий запит POST

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

Після виконання запиту повинен бути код 200 про успішне виконання запиту

* 1. **Робимо захист для API в insurance/views.py вписати permission\_classes = [IsAuthenticated] в клас InsuranecPolicyView що б виглядало так**

from rest\_framework.permissions import IsAuthenticated

class InsurancePolicyView(generics.ListCreateAPIView):

queryset = InsurancePolicy.objects.all()

serializer\_class = InsurancePolicySerializer

permission\_classes = [IsAuthenticated]

* 1. **Відкрити insurance/serializers.py і онови InsurancePolicySerializer ось так:**

class InsurancePolicySerializer(serializers.ModelSerializer):

class Meta:

model = InsurancePolicy

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

extra\_kwargs = {

'client': {'read\_only': True} #зробити поле клієнта лише для читання

}

# **Тестування створення полісів в Postman**

Тут вказуємо токен access отриманий після тестування токенів, дійсний не більше 5 хв після перевіркиA black screen with white text

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.

Після перевірки повинен бути код 201 про успішний запит

Ми вже в моделі користувача (User) маємо поля is\_customer і is\_agent.

Тепер у InsurancePolicyView будемо відрізняти:

* Якщо користувач — агент, він бачить усі поліси.
* Якщо користувач — клієнт, він бачить тільки свої поліси.
  1. **Відкрити insurance/views.py і заміни queryset на метод get\_queryset():**

from rest\_framework.permissions import IsAuthenticated

from rest\_framework import generics

from .models import InsurancePolicy

from .serializers import InsurancePolicySerializer

class InsurancePolicyView(generics.ListCreateAPIView):

serializer\_class = InsurancePolicySerializer

permission\_classes = [IsAuthenticated]

def get\_queryset(self):

user = self.request.user

if user.is\_agent:

return InsurancePolicy.objects.all()

return InsurancePolicy.objects.filter(client=user)

def perform\_create(self, serializer):

serializer.save(client=self.request.user)

**Тепер:** Якщо клієнт зробить GET /api/policies/ — він побачить тільки свої поліси.

Якщо агент зробить GET /api/policies/ — він побачить усі поліси всіх клієнтів.

# **API Ендпоїтни**

**Задачі при створенні API**

|  |  |
| --- | --- |
| **Етап** | **Виконано** |
| Створити API для реєстрації – логіну | **+** |
| Створити API для полюсів | **+** |
| Захистити API через JWT токени | **+** |
| Автоматична прив’язка користувача до полісу | **+** |
| Розмежування права доступу (клієнт – свій, агент - всі) | **+** |

**API Ендпоїнти**

| **Шлях** | **Метод** | **Опис** |
| --- | --- | --- |
| /api/register/ | POST | Реєстрація нового користувача |
| /api/token/ | POST | Отримання JWT токену |
| /api/token/refresh/ | POST | Оновлення access токену |
| /api/policies/ | GET | Перегляд полісів |
| /api/policies/ | POST | Створення нового поліса |

**Аутентифікація**

Всі запити до захищених API вимагають заголовок:

Authorization: Bearer <access\_token>