Unit Test Plan

Introduction

The unit test plan for the Shorty URL Shortener project was developed to validate the reliability and correctness of the backend application, built using Django and MongoDB. A focused testing approach was implemented to verify that key functionality—including authentication, URL creation, analytics, and error handling—performed as expected. The tests were designed to be repeatable, isolated, and fast, allowing early detection of issues throughout development.

Purpose

This unit test plan ensures that all essential components of the Shorty backend behave as intended. Testing focused on the creation of short URLs, user authentication flows, secure handling of analytics data, and input validation. The purpose was to verify the application's logic in isolation from the frontend, providing confidence in backend stability and security before deployment.

Overview

Testing was conducted using Django's built-in unittest framework along with APITestCase for API-level validation. The tests simulated common user actions, including:

- Logging in via session-authenticated endpoints
- Creating short URLs with and without custom codes

- Accessing analytics dashboards with filtered, user-specific data
- Handling invalid URLs or malformed inputs

Each test scenario was designed to reflect real-world usage patterns and edge cases. Where failures were encountered, fixes were implemented, and tests were rerun to confirm resolution.

Test Plan

Items

- Django backend application
- MongoDB Atlas test cluster (separate from production data)
- Django Allauth authentication system with GitHub/Google
- API endpoints under /api/shorten/, /api/analytics/, and OAuth login routes

Features

- Session-based authentication using OAuth
- POST endpoint for URL shortening
- GET endpoint for analytics (user-scoped)
- URL validation and conflict resolution
- Shortcode lookup and redirection

Deliverables

- tests.py file in the Django app
- test_results.txt file containing test output
- Screenshots showing test success and edge-case handling

Tasks

- Setup and configure Django test settings and MongoDB connection
- Write test cases using Django's testing tools
- Use pre-defined test credentials and fake user sessions
- Run test suite with python manage.py test
- Capture output and analyze errors or failures
- Apply fixes and revalidate changes with new test runs

Needs

- Python 3.12
- Django 4.x
- Django REST Framework
- MongoDB Atlas test URI
- .env.test file for isolated environment variables
- Access to Allauth-configured OAuth providers

Pass/Fail Criteria

The following criteria were used to determine whether each feature in the application was functioning correctly:

- URL Shortening

  A test passed if the application successfully created a new short URL and

responded with a status code of 201 Created, along with a JSON response containing the shortened URL. The test failed if the system returned a 400 Bad Request due to an invalid URL format, or a 409 Conflict if a custom shortcode was already in use.

- Authentication

  A test passed if a user could log in using valid credentials, and their session remained active for subsequent requests. Any request made without proper authentication was expected to be denied, returning a 403 Forbidden or redirecting the user to the login screen. Failure occurred if protected endpoints could be accessed without authentication.

- Analytics Retrieval

  A test passed if the authenticated user could retrieve a list of only their own shortened URLs, complete with analytics data such as click counts and timestamps. A failure occurred if the endpoint returned an empty list when valid data existed, or if an unauthenticated user was able to access the analytics.

- Redirection

  A test passed if visiting a valid short URL successfully redirected the user to the corresponding original URL. The test failed if the server returned a 404 Not Found when given a valid shortcode, or if the redirect logic did not work correctly.

- Input Validation

  A test passed if the backend correctly identified and rejected invalid input, such as improperly formatted URLs or shortcodes with disallowed characters, and

responded with clear error messages. It failed if invalid input was accepted or if the application crashed instead of handling the error gracefully

Specifications

Example test case for URL shortening:

```python
from django.test import TestCase, Client
from unittest.mock import patch
from bson import ObjectId
from datetime import datetime
from django.contrib.auth.models import User
from rest_framework.test import APIClient

class URLShortenerTests(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='testuser', password='testpass')
        self.client = APIClient()
        self.client.force_authenticate(user=self.user)

        # You must also set the session variable manually
        session = self.client.session
        session['username'] = self.user.username
        session.save()

    @patch('url_shortener.views.get_urls_collection')
    def test_delete_url_success(self, mock_collection):
        mock_result = type('MockResult', (), {'deleted_count': 1})()
        mock_collection.return_value.delete_one.return_value = mock_result

        response = self.client.delete('/api/analytics/test123/')
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.json()['message'], 'URL deleted successfully')

    @patch('url_shortener.views.get_urls_collection')
    def test_delete_url_not_found(self, mock_collection):
        mock_result = type('MockResult', (), {'deleted_count': 0})()
        mock_collection.return_value.delete_one.return_value = mock_result

        response = self.client.delete('/api/analytics/fakecode/')
        self.assertEqual(response.status_code, 404)
        self.assertEqual(response.json()['error'], 'URL not found or not owned by you')

    @patch('url_shortener.views.get_urls_collection')
    def test_user_analytics(self, mock_collection):
        mock_collection.return_value.find.return_value = [
            {
                'original_url': 'https://example.com',
                'short_code': 'abc123',
                'clicks': 5,
                'created_at': datetime.now()
            }
        ]

        response = self.client.get('/api/analytics/')
        self.assertEqual(response.status_code, 200)
        self.assertIsInstance(response.json(), list)
        self.assertEqual(response.json()[0]['short_code'], 'abc123')

    @patch('url_shortener.views.get_urls_collection')
    def test_redirect_url_found(self, mock_collection):
        mock_collection.return_value.find_one.return_value = {
            '_id': ObjectId(),
            'original_url': 'https://redirect.com',
            'short_code': 'go123'
        }
        mock_collection.return_value.update_one.return_value = None

        response = self.client.get('/s/go123/')
        self.assertEqual(response.status_code, 302)
        self.assertEqual(response['Location'], 'https://redirect.com')

    @patch('url_shortener.views.get_urls_collection')
    def test_stats_view(self, mock_collection):
        mock_collection.return_value.find_one.return_value = {
            'original_url': 'https://example.com',
            'short_code': 'abc123',
            'created_at': datetime.now(),
            'clicks': 42
        }

        response = self.client.get('/api/stats/abc123/')
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.json()['clicks'], 42)
```

Procedures

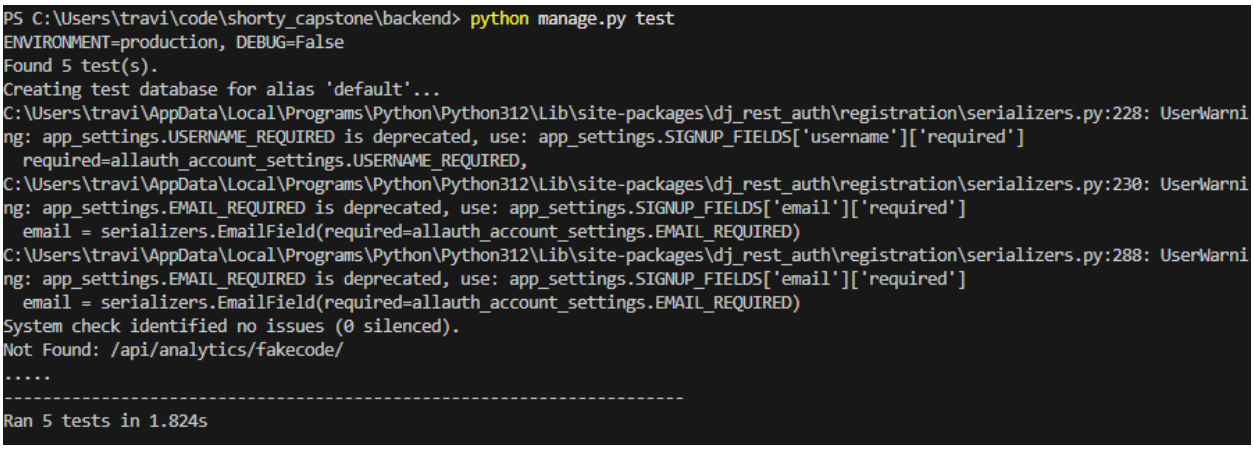
- Prepare test user accounts and test database
- Write and validate test cases for each critical backend function
- Run python manage.py test and redirect output to a file
- Capture screenshots of terminal results and failed responses (if applicable)
- Update or refactor backend logic based on test feedback
- Repeat the test cycle until all results pass

Results

The full suite of unit tests executed successfully. Key results included:

- Short URLs were generated and returned correctly with valid input
- Requests with invalid URLs or taken custom codes failed with expected errors
- Analytics endpoint returned correct, user-filtered data
- Authentication was enforced across protected endpoints

Screenshots and output logs were captured and stored in the /docs directory.



Remediation

Where failures occurred (e.g., duplicate short codes or unauthenticated access), I traced errors using logs and test output, adjusted backend logic, and updated the corresponding test case. Once issues were resolved, the full test suite was re-executed to confirm success.