PROWLER

Technical Exercices
Backend Python Django engineer

2025

Introduction

Welcome to the technical exercises for the Python (Django) Backend Engineer position. We are excited to see your skills and creativity in action. These exercises are designed to evaluate your proficiency with these technologies and your ability to implement practical solutions to common development challenges.

In the mail where you received this document you can find the specific date and time.

We understand that these exercises are extensive, and while we encourage thoroughness, we also recognize that you may need to make pragmatic decisions about which aspects to focus on. Please rest assured that we will take this into consideration when reviewing your submission. Please describe your reasoning in the exercises so we understand your approach and decisions.

In addition to the technical implementation, we highly value clarity, good style, and proper grammar in your written explanations. Clear communication is essential in our team, so please ensure that your code comments, documentation, and any written descriptions are well-articulated and easy to follow.

Thank you for your time and effort. We look forward to seeing your work and appreciate your interest in joining our team.

1. Usage and Customization of Prowler

Objective

Demonstrate the candidate's ability to use Prowler to perform a security assessment on an AWS account.

Instructions

1. **Setup**
    - Ensure you have the AWS CLI installed and configured on your machine with appropriate credentials.
    - Install Prowler CLI by following the instructions on the Prowler GitHub page.
2. **Run a Basic Scan**
    - Execute Prowler to perform a basic security scan on your AWS account.
3. **Generate a Report**
4. **Analyze the Report**
    - Review the generated report and identify any high-severity findings.
    - Provide a brief summary of the high-severity issues and potential remediation steps

Submission

Submit the command you executed in each step, including the report generated (json/csv) file along with a summary document outlining the high-severity findings and suggested remediation steps.

2. Django REST Framework CRUD Application

Objective

Create a Django application using Django REST framework to manage Prowler scan checks findings. The application should support basic CRUD operations (Create, Read, Update, Delete), and allow real-time monitoring of scan status via an API endpoint.

You are tasked with creating a feature in a Django web application that allows users to run a Prowler scan, list the Prowler checks, and list the scan findings (scan results). Additionally, the system should support real-time status tracking of scans through the SCAN endpoint. Describe the steps you would take to implement this feature, including model definition, form handling, view logic, API endpoints, serializers, and routing. How would you handle the concurrent execution of multiple scans? What approaches would you implement to ensure scalability and efficiency?

Requirements

Setup Django Project

- Create a new Django project.
- Create a new Django app within the project.

Model Design

- Create models to represent the entities:
    - **Scan**: Represents a scan, and should include:
        - A field to track the **real-time status**
        - A timestamp of when the scan was initiated and when it ended.
    - **Check**: Represents a check performed during a scan.
    - **Finding**: Represents a finding from a check.
- Define relationships between models (e.g., a scan can have many checks, a check can have many findings).

Real-Time Status Tracking

- Add a **scan status tracking mechanism**:
    - Introduce a status field in the Scan model to track different stages (pending, in_progress, completed, failed).
    - Ensure that the **SCAN endpoint** can query the real-time status of a scan for continuous updates or using periodic polling.

Concurrent and Parallel Scans

Ensure the system can handle multiple scans running in parallel:

- Use asynchronous task queues such as **Celery** for running scans in the background.

    - Ensure the system can manage parallel task execution efficiently, using Redis or another message broker for task management.
    - Implement locking mechanisms or unique task identifiers to prevent race conditions when running concurrent scans.

API Endpoints

- Implement the following endpoints using Django REST framework:
    - Scans: List, Create, Retrieve, Update, Delete. Ensure that scan creation triggers an asynchronous scan task and that retrieval includes the current status.
    - Checks: List, Create, Retrieve, Update, Delete.
    - Findings: List, Create, Retrieve, Update, Delete.
    - Scan Status: Add an endpoint (/scans/<scan_id>/status/) to retrieve the real-time status of a scan, leveraging the status field in the Scan model.

Serializers

- Create serializers for each model to handle JSON serialization and deserialization.
- Update the ScanSerializer to include real-time status tracking information.

Views

- Create views using DRF viewsets to handle the API logic for scans, checks, and findings.
- Implement a dedicated view for real-time scan status updates using Django Channels if real-time WebSocket updates are desired.

Routing

- Configure URL routing to expose the API endpoints, including the real-time status endpoint.

Submission

Provide the following:

- The complete source code of the Django project, including the real-time tracking functionality.
- A **README** file or similar with instructions on how to set up, run the project, and manage dependencies (e.g., Celery, Redis).
- Include instructions on running scans asynchronously and how to observe the real-time scan status updates.
- Any details about the implementation that you think are worth to elaborate on.

3. Django Multitenant application

Objective

You need to develop a multi-tenant Django application where each tenant has its own set of data. Describe how you would set up the application to support multi-tenancy. Include details about any specific packages you would use, database schema strategies, and how you would handle tenant-specific data separation and security.
The app must use PostgreSQL as DB engine.

Describe the complex challenges you can anticipate about creating this multi tenant app

Instructions

- Code implementation or prototyping is not mandatory
- Describing how you would solve the problem. Include any resources (explanations, graphs, diagrams) that you think can help us to understand the solution.

Submission

Provide the following:

- PDF with the elaborated answer