**IT-IVA**

**HOMEPAGE**

1. Function of IT-IVA
2. How Vulnerability Assessment Works

**USERS**

1. Register and Login (email, google and facebook)
2. Fill details about your business e.g., You name, location, Business Name, Business Objective/Description, website
3. Integrate Gemini AI for summarizing
4. Answer Vulnerability Assessment Questionnaire
5. View Vulnerability Assessment Result (Contains Recommendations based on Criteria and overall assessment, large image of Radar Chart)
6. Download pdf report of Recommendations. Contains Recommendations based on Criteria and overall assessment, large image of Radar Chart

**ADMIN**

1. View Security Landscape on a map with different colours representing each business based on their vulnerability assessment score.
2. Editing Questionnaire, Marking Grid and Recommendations
3. View reports of user
4. View Vulnerability Landscape from clickable Map

**Django Backend & PostgreSQL Database Architecture**

This document outlines the proposed structure for the ITIVA backend, using Django and PostgreSQL, to support the functionalities of the Vue.js frontend.

**1. Updated Entity Relationship (ER) Model**

Based on your new requirements, the database schema needs to be more comprehensive.

+----------------+ +------------------+ +----------------+

| Users | | Businesses | | VAResults |

|----------------| |------------------| |----------------|

| PK user\_id |----o<| FK user\_id |----o<| FK user\_id |

| email | | PK business\_id | | PK result\_id |

| password\_hash| | business\_name | | website\_score|

| full\_name | | description | | network\_score|

+----------------+ | website | | compliance\_score|

| address | | cybersec\_score|

| latitude | | overall\_score |

| longitude | | created\_at |

+------------------+ +----------------+

+----------------+ +----------------+ +----------------+

| Questions | | Options | | UserAnswers |

|----------------| |----------------| |----------------|

| PK question\_id |----o<| PK option\_id |----o<| PK answer\_id |

| text | | FK question\_id | | FK user\_id |

| category | | option\_letter | | FK question\_id |

+----------------+ | text | | FK option\_id |

+----------------+ +----------------+

|

|1

o|

+----------------+ +-------------------+

| MarkingGrid | | Recommendations |

|----------------| |-------------------|

| PK grid\_id | | PK rec\_id |

| FK option\_id | | FK option\_id |

| score | | recommendation\_text|

| explanation | +-------------------+

+----------------+

**2. PostgreSQL Database Setup**

1. **Create Database:** CREATE DATABASE itiva\_db;
2. **Tables:** The tables will be created automatically by Django based on the models defined below.
3. **Data Ingestion:** Create custom Django management commands to read from your questionnaire.csv, marking\_grid.csv, and recommendations.csv files and populate the Questions, Options, MarkingGrid, and Recommendations tables. This is a one-time setup process.
   * python manage.py import\_questionnaire
   * python manage.py import\_markinggrid
   * python manage.py import\_recommendations

**3. Django Project Structure**

itiva\_project/

├── itiva\_app/

│ ├── migrations/

│ ├── templates/

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── models.py <-- Define DB schema here

│ ├── serializers.py

│ ├── tests.py

│ ├── urls.py

│ └── views.py <-- Define API endpoints here

└── itiva\_project/

├── \_\_init\_\_.py

├── asgi.py

├── settings.py

├── urls.py

└── wsgi.py

**4. Django Models (**models.py**)**

from django.contrib.auth.models import AbstractUser

from django.db import models

class User(AbstractUser):

full\_name = models.CharField(max\_length=255)

# email, username, password are included in AbstractUser

class Business(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

business\_name = models.CharField(max\_length=255)

description = models.TextField(blank=True, null=True)

website = models.URLField(blank=True, null=True)

address = models.CharField(max\_length=512)

latitude = models.DecimalField(max\_digits=9, decimal\_places=6, null=True)

longitude = models.DecimalField(max\_digits=9, decimal\_places=6, null=True)

class Question(models.Model):

CATEGORY\_CHOICES = [

('WS', 'Website Strength'),

('DN', 'Devices & Network'),

('CD', 'Compliance Documentation'),

('CS', 'Cyber Security Implementations'),

]

text = models.TextField()

category = models.CharField(max\_length=2, choices=CATEGORY\_CHOICES)

class Option(models.Model):

question = models.ForeignKey(Question, related\_name='options', on\_delete=models.CASCADE)

text = models.CharField(max\_length=512)

class MarkingGrid(models.Model):

option = models.OneToOneField(Option, on\_delete=models.CASCADE)

score = models.IntegerField()

explanation = models.TextField()

class Recommendation(models.Model):

option = models.OneToOneField(Option, on\_delete=models.CASCADE)

recommendation\_text = models.TextField()

class UserAnswer(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE)

question = models.ForeignKey(Question, on\_delete=models.CASCADE)

selected\_option = models.ForeignKey(Option, on\_delete=models.CASCADE)

session\_id = models.CharField(max\_length=100) # To group answers for a single assessment

timestamp = models.DateTimeField(auto\_now\_add=True)

class VAResult(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE)

session\_id = models.CharField(max\_length=100, unique=True)

website\_score = models.IntegerField()

network\_score = models.IntegerField()

compliance\_score = models.IntegerField()

cybersec\_score = models.IntegerField()

overall\_score = models.IntegerField()

created\_at = models.DateTimeField(auto\_now\_add=True)

**5. API Endpoints (**views.py **&** urls.py**)**

Using Django Rest Framework, you would create the following endpoints:

* /api/auth/register/: (POST) Create a new User and Business.
* /api/auth/login/: (POST) Authenticate user, return JWT token.
* /api/questionnaire/: (GET) Return all Questions and their nested Options.
* /api/assessment/submit/: (POST) Accepts a list of {question\_id, option\_id}. It calculates scores, creates UserAnswer and VAResult objects, and returns the result\_id.
* /api/results/{result\_id}/: (GET) Returns the scores for a specific assessment.
* /api/rankings/: (GET) Returns a list of the top 10 businesses based on their latest VAResult.overall\_score.
* /api/map-data/: (GET) Returns business locations and scores for the Admin map.

This backend architecture provides a clear path to connect with the Vue.js frontend, creating the full-stack application you envisioned.