MCAST

ASSESSMENT AND INTERNAL VERIFICATION FRONT SHEET (Individual Criteria)

(Note : This version is to be used for an assignment brief issued to students via Classter)

Course Title	**IT6-A4-23 | Bachelor of Science (Honours) in Digital Games Development, IT6-A03-23 | Bachelor of Science (Honours) in Creative Computing**		**Lecturer Name & Surname**	**David Deguara**	
Unit Number & Title	ITMSD-506-2301 | Database Essentials				
Assignment Number, Title / Type	Home Assignment – Developing Secure and Effective Database Solutions				
Date Set	17th March 2025	Deadline Date	7th April 2025		
Student Name		ID Number		Class / Group	

Assessment Criteria	Maximum Mark
R&U5 - Identify common tools and technologies used in database development.	5
E&C1 - Design a comprehensive database schema for a multimedia application.	10
R&U2 - Explain the different types of databases and their use cases.	5
R&U4 - Define key concepts in ER modeling and normalisation.	5
R&U1 - Describe the core components of a database system.	5
E&C2- Develop a functional multimedia database application.	10
R&U7 - Describe various database security threats.	5
E&C3 - Propose security measures to protect a multimedia database.	10
Total Mark	55

Notes to Students:

- This assignment brief has been approved and released by the Internal Verifier through Classter.
- Assessment marks and feedback by the lecturer will be available online via Classter (Http://mcast.classter.com) following release by the Internal Verifier
- Students submitting their assignment on Moodle/Unicheck will be requested to confirm online the following statements:

Student's declaration prior to handing-in of assignment

- ❖ I certify that the work submitted for this assignment is my own and that I have read and understood the respective Plagiarism Policy

Student's declaration on assessment special arrangements

- ❖ I certify that adequate support was given to me during the assignment through the Institute and/or the Inclusive Education Unit.
- ❖ I declare that I refused the special support offered by the Institute.

MCAST

A02: Developing Secure and Effective Database Solutions

Overview

This assignment requires the development and hosting of a restful web API that connects to a Mongo Atlas database to store multimedia game assets. These assets include sprites (images), audio files and player scores. The project is entirely practical and code oriented. Although any programming language may be used, Python is recommended.

A REST (Representational State Transfer) API is an architectural style that employs standard HTTP methods (GET, POST, PUT, DELETE) to manage resources identified by unique URIs. Its stateless nature and use of standard protocols enable scalability and simplicity.

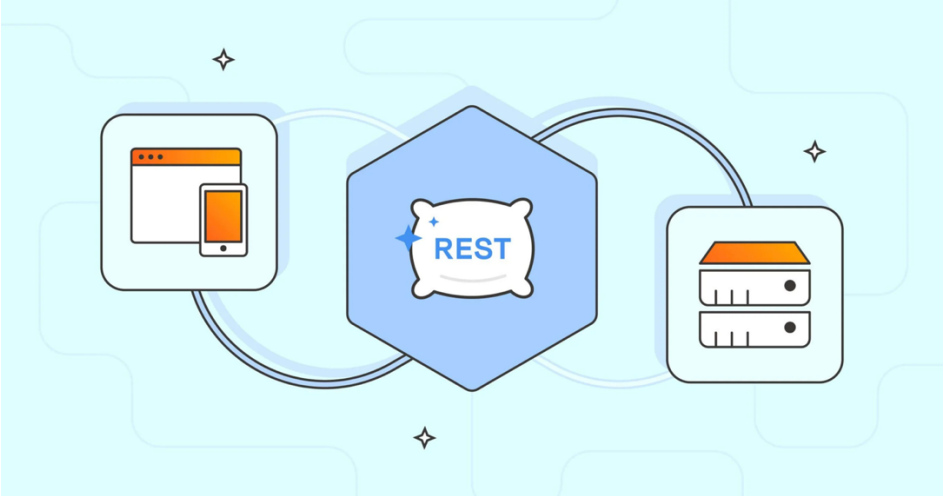

Table of Contents

A02: Database Essentials – Home Assignment 1
Overview 1
Assignment Objectives 2
Submission Guidelines 2
Assignment Tasks 2
Task 1: Environment Setup and Technology Selection (5 marks) 2
Task 2: Database Schema (15 marks) 2
Task 3: Development of the Web API (20 marks) 3
Task 4: Configuring Database Security (15 marks) 3
Appendices 3
Appendix A: Submission Checklist 3
Appendix B: Python FastAPI Code Example 4
Appendix C: Setting Up a FastAPI Project 4
Appendix D: Installing Project Dependencies 5
Appendix E: Testing with Postman 6
Appendix F: Deploying FastAPI on Vercel 6

MCAST

Assignment Objectives

In this home assignment, you are required to do the following:

- Design and deploy a Mongo database on Mongo Atlas.
- Program a restful API to manage multimedia game assets including images, audio files and player scores.
- Host the API for external access.
- Demonstrate robust database design, security measures and comprehensive code documentation.

Submission Guidelines

The assignment is to be completed on your own and submitted online on vle.mcast.edu.mt. A git repository with regular commits is required as proof that you did the work yourself. Failure to include a repository link may lead to an in-depth interview where you will be asked several questions about your code to prove that you did the work yourself. You are required to submit **two separate files**:

1. A **.zip file** containing the readme file and complete source code for the web API including in-code comments explaining the design decisions and implementation details.
2. A **Microsoft Word document (.docx)** containing the contents listed in Appendix A, namely the git repository url, the hosted API url and relevant screenshots.

~~Assignment Tasks~~

~~To successfully accomplish this assignment, it is advised to reference the Mongo examples we did in class and see the detailed step by step guides in the appendices section of this assignment brief.~~

~~Task 1: Environment Setup and Technology Selection (5 marks)~~

~~*Criteria: R&U5*~~

~~a) Establish the development environment (e.g. creating a virtual environment, installing necessary packages such as FastAPI, Uvicorn and Motor). See appendices C & D.~~

~~b) Utilise a git repository and ensure that the project structure is clearly organised and all dependencies are managed (mandatory step).~~

~~c) **Setup Documentation (5 marks)**: Document the chosen setup in a Readme.md file.~~

Task 2: Database Schema (15 marks)

Criteria: E&C1 and R&U2

a) **Schema Design (10 marks)**: Design the schema of the Mongo database using Datagrip or Mongo Compass.

b) **Schema Deployment (5 marks)**: Connect and execute the schema on Mongo Atlas exactly how we did it during the lectures by populating documents with mock (fake) data.

MCAST

Task 3: Development of the Web API (20 marks)

Criteria: R&U1, R&U4 and E&C2

a) **Develop API endpoints (7 marks)**: Develop the endpoints that facilitate the uploading and retrieval of sprites, audio files and player scores. See appendix B.

b) **Host the API (5 marks)**: Ensure that the API is fully functional by testing it using Postman and publicly accessible online via the browser. See appendices E and F.

c) **Documentation (8 marks)**: Provide detailed code documentation, including explanations of how each endpoint operates and interacts with the database.

Task 4: Configuring Database Security (15 marks)

Criteria: R&U7 and E&C3

a) **Setting appropriate credentials (5 marks):** Create secure credentials for the Mongo Atlas database.

b) **Whitelisting IP address/s (5 marks):** Implement IP whitelisting to restrict database access to a trusted IP address/es (Vercel API server).

c) **Preventing SQL injection attacks (5 marks):** Implement measures to prevent SQL injection attacks (despite Mongo being a NoSQL database), ensuring that user inputs are properly validated and sanitised.

Appendices

Appendix A: Submission Checklist

Public GIT (Github/Bitbucket/Gitlab) url:

Public API (Vercel or otherwise) url:

Task 1 Screenshot of your development environment setup:

Task 2: Screenshot/s of your Mongo Database on MongoAtlas:

Task 3A: Screenshot of the API running on localhost in your browser:

Task 3B: Screenshot of the hosted API running on a public url such as Vercel:

Task 4A: Screenshot showing credential setup.

Task 4B: Screenshot showing IP whitelisting.

Task 4C: Screenshot showing how you are preventing SQL injection.

Appendix B: Python FastAPI Code Example

Below is an example of a Python FastAPI application that provides endpoints for uploading sprites and audio files, as well as for submitting player scores. The example utilises asynchronous operations and a connection to a Mongo Atlas database via Motor, an asynchronous driver for MongoDB.

```
from fastapi import FastAPI, File, UploadFile, HTTPException
from pydantic import BaseModel
import motor.motor_asyncio

app = FastAPI()

# Connect to Mongo Atlas
client = motor.motor_asyncio.AsyncIOMotorClient("your_mongo_connection_string")
db = client.multimedia_db

class PlayerScore(BaseModel):
    player_name: str
    score: int

@app.post("/upload_sprite")
async def upload_sprite(file: UploadFile = File(...)):
    # In a real application, the file should be saved to a storage service
    content = await file.read()
    sprite_doc = {"filename": file.filename, "content": content}
    result = await db.sprites.insert_one(sprite_doc)
    return {"message": "Sprite uploaded", "id": str(result.inserted_id)}

@app.post("/upload_audio")
async def upload_audio(file: UploadFile = File(...)):
    content = await file.read()
    audio_doc = {"filename": file.filename, "content": content}
    result = await db.audio.insert_one(audio_doc)
    return {"message": "Audio file uploaded", "id": str(result.inserted_id)}

@app.post("/player_score")
async def add_score(score: PlayerScore):
    score_doc = score.dict()
    result = await db.scores.insert_one(score_doc)
    return {"message": "Score recorded", "id": str(result.inserted_id)}
```

Appendix C: Setting Up a FastAPI Project

1. **Environment Setup:**
   - Ensure that you have python (version 3.11+ recommended) installed on your machine.
   - Open VS Code or Jetbrains Pycharm. If using VS Code, make sure to install the python plugins (you should have installed them during the lecture).
   - Create a virtual environment by executing this code in VS Code terminal:

     ```
     python -m venv env
     source env/bin/activate  (or env\Scripts\activate on Windows)
     ```

- Before installing the dependencies in the VS Code terminal, ensure that you are in the environment by checking for (env) before the command.

2. **Project Structure:**
   - Create a file named `main.py` and paste the FastAPI code provided in the appendices to start.
   - Optionally, organise the code into modules.
3. **Running the API:**
   - Launch the API using by typing this command in VS Code terminal or hitting the play button:

     ```
     uvicorn main:app --reload
     ```

   - The API will be accessible in the browser at http://127.0.0.1:8000/**docs**.

   Note: To install the required dependencies and generate a requirements.txt file required for later, please see Appendix C.

Appendix D: Installing Project Dependencies

Below is a list of Python packages required for the project along with their installation commands:

1. **FastAPI** - A modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints.

   ```
   pip install fastapi
   ```
2. **Uvicorn** - A lightning-fast ASGI server implementation, using `uvloop` and `httptools`.

   ```
   pip install uvicorn
   ```
3. **Motor** - The async driver for MongoDB, designed to work with Tornado or asyncio.

   ```
   pip install motor
   ```
4. **Pydantic** - Data validation and settings management using Python type annotations.

   ```
   pip install pydantic
   ```
5. **Python-dotenv** - Reads key-value pairs from a `.env` file and can set them as environment variables.

   ```
   pip install python-dotenv
   ```
6. **Requests** - A simple, yet elegant HTTP library for Python, built for human beings.

   ```
   pip install requests
   ```

After installing all the necessary dependencies for your project, you can generate a `requirements.txt` file using the `pip freeze` command. This file will list all the installed packages and their versions, ensuring that anyone who wants to run your project can install the exact same dependencies. Here's how to do it:

Run the following command in your terminal:

```
pip freeze > requirements.txt
```

This command will create a `requirements.txt` file in your current directory with a list of all installed packages and their versions.

Ensure all these packages are listed in your requirements.txt file for easy installation: text fastapi uvicorn motor pydantic python-dotenv requests

MCAST

Appendix E: Testing with Postman

1. **Create a New Request:**
   - Open Postman and create a new request.
   - Set the request method to POST.
2. **Endpoint Configuration:**
   - For uploading sprites, set the URL to `http://127.0.0.1:8000/upload_sprite`.
   - For uploading audio files, set the URL to `http://127.0.0.1:8000/upload_audio`.
   - For adding player scores, set the URL to `http://127.0.0.1:8000/player_score`.
3. **Request Body Setup:**
   - For file uploads, select "form-data" and add a key with type "File" to attach the file.
   - For player scores, select "raw", set the type to JSON and provide a JSON object, for example:

```
{
    "player_name": "John Doe",
    "score": 1500
}
```

4. **Sending and Verifying Requests:**
   - Click "Send" to execute the request.
   - Verify the API response for confirmation of the operation.

Appendix F: Deploying FastAPI on Vercel

Deploying a FastAPI application on Vercel using the Vercel website is straightforward. Follow these steps:

1. **Create a Vercel Account:** - Visit the Vercel website. - Sign up for an account using your GitHub, GitLab, or Bitbucket account.
2. **Prepare Your FastAPI Project:** - Ensure your FastAPI project is set up as described in Appendix B. - Your project structure should look something like this:

```
my_fastapi_app/
├── main.py
├── requirements.txt
└── vercel.json
```

3. **Create a `vercel.json` Configuration File:** - In the root of your project, create a `vercel.json` file with the following content:

```
{
      "version": 2,
      "builds": [
       {
         "src": "main.py",
         "use": "@vercel/python"
       }
      ],
      "routes": [
       {
         "src": "/(.*)",
         "dest": "main.py"
       }
      ]
}
```

4. **Deploy Your Application:** - Push your project to a Git repository (GitHub, GitLab, or Bitbucket). - Go to the Vercel dashboard and click on “New Project”. - Import your project from the connected Git repository. - Follow the prompts to configure and deploy your project.

5. **Access Your Deployed Application:** - Once the deployment is complete, Vercel will provide you with a URL where your FastAPI application is hosted. - You can access your application using this URL. If using FastAPI, add /docs to the url.