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		Deadline Date			
Student Name	Duncan Aquilina	ID Number	0227505L	Class / Group	6.2A

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 modelling 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 MongoDB Atlas database to store event management data. The system will manage events, attendees, venues, ticket bookings and multimedia assets including event posters (images), promotional videos and venue photos. 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.

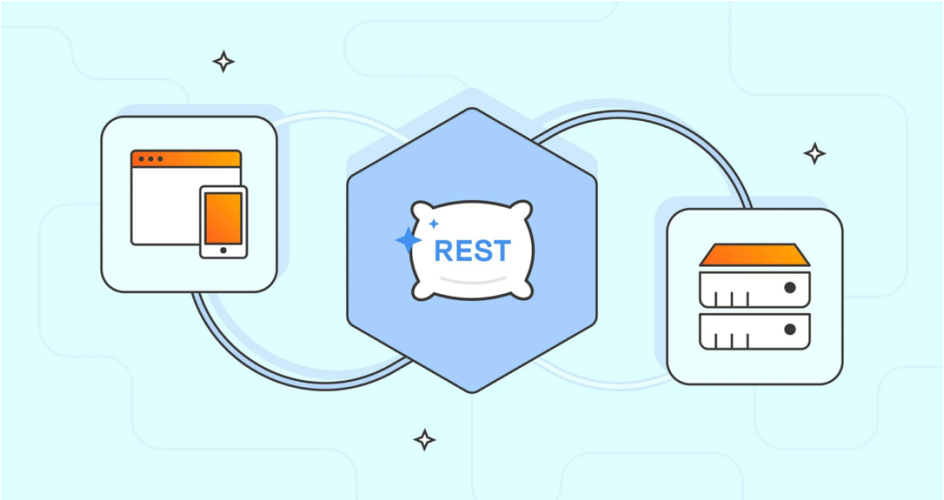

Table of Contents

Assignment Objectives

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

- Design and deploy a MongoDB database on MongoDB Atlas.
- Program a RESTful API to manage event data including events, attendees, venues and ticket bookings.
- Implement file upload and retrieval functionality for multimedia assets such as event posters (images), promotional videos and venue photos.
- 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 MongoDB database using DataGrip or MongoDB Compass. The schema must include collections for events, attendees, venues, bookings and multimedia files (event posters, promotional videos, venue photos).

b) Schema Deployment (5 marks): Connect and execute the schema on MongoDB Atlas (or through Vercel directly) exactly how we did it during the lectures by populating documents with mock data.

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:
   i. Creation, retrieval, update and deletion of events, attendees, venues and ticket bookings
   ii. Uploading and retrieval of event posters (images)
   iii. Uploading and retrieval of promotional videos
   iv. Uploading and retrieval of venue photos

See Appendix B for code examples.

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, with special attention to how files are stored and retrieved.

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).

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. The example utilises asynchronous operations and a connection to a Mongo Atlas database via Motor, an asynchronous driver for MongoDB.

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 .venv
source .venv/bin/activate  (or .venv\Scripts\activate on Windows)
```

2. **Create a .env file:**

```
MONGO_URI=mongodb+srv://user:pass@cluster.mongodb.net/dbname
```

IMPORTANT: ADD .ENV TO YOUR .GITIGNORE FILE!

3. **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.
4. **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.

```
import os
from fastapi import FastAPI, File, UploadFile, HTTPException
from fastapi.responses import StreamingResponse
from pydantic import BaseModel
from typing import Optional
from datetime import datetime
from dotenv import load_dotenv
import motor.motor_asyncio
import io

# Load environment variables from .env file
load_dotenv()

app = FastAPI()

# Connect to MongoDB Atlas
client = motor.motor_asyncio.AsyncIOMotorClient("your_mongo_connection_string")
db = client.event_management_db

# Data Models
class Event(BaseModel):
    name: str
    description: str
    date: str
    venue_id: str
    max_attendees: int

class Attendee(BaseModel):
    name: str
    email: str
    phone: Optional[str] = None

class Venue(BaseModel):
    name: str
    address: str
    capacity: int

class Booking(BaseModel):
    event_id: str
    attendee_id: str
    ticket_type: str
    quantity: int

# Event Endpoints
@app.post("/events")
async def create_event(event: Event):
    event_doc = event.dict()
    result = await db.events.insert_one(event_doc)
    return {"message": "Event created", "id": str(result.inserted_id)}

@app.get("/events")
async def get_events():
    events = await db.events.find().to_list(100)
    for event in events:
        event["_id"] = str(event["_id"])
    return events

# Upload Event Poster (Image)
@app.post("/upload_event_poster/{event_id}")
async def upload_event_poster(event_id: str, file: UploadFile = File(...)):
    content = await file.read()
    poster_doc = {
        "event_id": event_id,
        "filename": file.filename,
        "content_type": file.content_type,
        "content": content,
        "uploaded_at": datetime.utcnow()
    }
    result = await db.event_posters.insert_one(poster_doc)
    return {"message": "Event poster uploaded", "id": str(result.inserted_id)}
```

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.

```
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: *fastapi uvicorn motor pydantic python-dotenv requests*

Appendix E: Testing with Postman

1. **Create a New Request:**

- Open Postman and create a new request.
- Set the request method to POST or GET as appropriate.

2. **Endpoint Configuration:**

- For creating events, set the URL to *http://127.0.0.1:8000/events*
- For registering attendees, set the URL to *http://127.0.0.1:8000/attendees*
- For adding venues, set the URL to *http://127.0.0.1:8000/venues*
- For creating bookings, set the URL to *http://127.0.0.1:8000/bookings*
- For uploading event posters, set the URL to *http://127.0.0.1:8000/upload_event_poster/{event_id}*
- For uploading promotional videos, set the URL to *http://127.0.0.1:8000/upload_promo_video/{event_id}*
- For uploading venue photos, set the URL to *http://127.0.0.1:8000/upload_venue_photo/{venue_id}*

3. **Request Body Setup for JSON Data:**
- Select "raw", set the type to JSON and provide a JSON object, for example:

```
{
    "name": "Tech Conference 2025",
    "description": "Annual technology summit",
    "date": "2025-06-15",
    "venue_id": "venue123",
    "max_attendees": 500
}
```

4. **Request Body Setup for File Uploads:**
- Select "form-data" in the Body tab.
- Add a key named "file" and change the type from "Text" to "File" using the dropdown.
- Click "Select Files" to choose an image or video file from your computer.

5. **Sending and Verifying Requests:**
- Click "Send" to execute the request.
- Verify the API response for confirmation of the operation.
- For file retrieval endpoints (GET), the response will display or download the file directly.

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.

MCAST

Appendix G: Assessment Rubric

The following rubric will be used to assess your submission. Total marks: 55

Criteria	Excellent	Good	Satisfactory	Needs Work
Task 1: Environment Setup *(5 marks)*	**5** Complete setup, detailed README, regular commits	**4** Good setup, README present	**3** Basic setup, minimal docs	**1-2** Incomplete setup
Task 2a: Schema Design *(10 marks)*	**9-10** Well-designed with all collections	**7-8** Good schema, minor issues	**5-6** Basic schema	**1-4** Poor design
Task 2b: Schema Deployment *(5 marks)*	**5** Fully deployed with mock data	**4** Deployed, some data	**3** Deployed, minimal data	**1-2** Partially deployed
Task 3a: API Endpoints *(7 marks)*	**7** All CRUD + file endpoints working	**5-6** Most endpoints work	**3-4** Basic endpoints only	**1-2** Few endpoints
Task 3b: API Hosting *(5 marks)*	**5** Fully hosted online	**4** Hosted, minor issues	**3** Hosted with issues	**1-2** Localhost only
Task 3c: Documentation *(8 marks)*	**7-8** Comprehensive comments	**5-6** Good documentation	**3-4** Basic comments	**1-2** Minimal docs
Task 4a: Credentials *(5 marks)*	**5** Secure, env vars used	**4** Good credentials	**3** Basic setup	**1-2** Hardcoded creds
Task 4b: IP Whitelisting *(5 marks)*	**5** Proper whitelisting	**4** Whitelisting present	**3** Basic (0.0.0.0)	**1-2** No whitelisting
Task 4c: Injection Prevention *(5 marks)*	**5** Input validation implemented	**4** Good validation	**3** Pydantic only	**1-2** No validation