BSc (Hons) in Computing in Games Development
BSc (Hons) in Computing in Software Development

CA 5 - Project

Multi-User Client-Server Application

CA Value: **20%**

This Project is a Group Project and must be completed in the designated groups.

You **must use GitHub to show incremental progress** on your assignment. Either add your lecturer as a collaborator or make your project public and share the link with them.

On completion each student should be able to explain the code, design philosophy, and patterns used. Everyone **must** demo their project in person to the lecturer.

Objectives

To complete the following:

- Design, implement and test a database-driven client-server application
- Design and implement a Data Access Layer for database access
- Implement a Multithreaded Server
- Design, develop and document a communications protocol using a sequence diagram showing all messages that pass between client and server
- Implement the exchange of data between client and server using the JSON format, and serialize and deserialize the data as required using You are required to use the **org.json** Parser(NOT GSon)
- Implement the exchange and display/storage of a binary stream of data
- Design and implement relevant unit tests
- Demonstrate the **ongoing** use of version control (this is mandatory)
- Incorporate relevant code design patterns (such as Factory, Singleton, Command and/or Decorator)
- Incorporate Collection classes where appropriate
- Apply DRY program design principles. (DRY=Don't Repeat Yourself)
- Draw an annotated high-level Architecture diagram for your system (one page)

All of the above items should be incorporated in the implementation of the features described below and will form part of the criteria for grading.

Requirements

You are required to select a theme/topic of interest to you and to identify one or more entities/classes that are relevant to your theme. (For example – if you select Premiership Soccer as a theme, then the following entities may be relevant – Player, Match, Ground, Manager etc. Once identified, you can develop classes to represent these entities. Below, a number of required **features** are specified. You are required to implement those features for your specific theme (system) and you are required to apply best practices in software engineering design, implementation and testing, and use of collections and structures that provide the best-practice solution when implementing a feature.

An important learning outcome for you is the completion of features by a specified deadline. Hence, there are strict deadlines for delivery at each milestone. Another important aspect in grading your work is an assessment of your **understanding** of the code that you have presented. You must demonstrate a clear understanding - and be able to explain your design and implementation choices in all aspects of your code. As we will not be specifying detailed implementation approaches, you should discuss your proposed approach with your team/colleagues and/or ask your lecturer for guidance as you proceed.

Schedule of Deliverables

Active Week	**Deadline**	**Deliverable**
Week 7	Sunday 9th March	*Milestone 1* Features:1 - 6 GitHub Repo
Week 8		*(demo Milestone 1 to lecturer in class)*
Week 9		
Week 10		
Week 11	Sunday 6th April	*Milestone 2* Features: 7 - 10
Week 12		*(demo Milestone 2 to lecturer in class)*
Easter		
Easter	Sunday 27th April	*Milestone 3* Features 11 -14 and all other requirements. Final Submission of all requirements
Week 13		Demo/Interviews in lab.

You must use Git Version Control and create a Repository (Repo) on GitHub or GitLab and make the Repo available to your lecturer. You must push your work to your Repo at regular intervals during the development of each feature and at the completion of **each** feature, using meaningful commit descriptions. **A poor Commit History will severely reduce your grade** (see Grading Rubric). Features must also be demonstrated to your lecturer in lab-based progress demo/interviews. No marks will be awarded in the absence of an interview. If a milestone/deadline is missed – the maximum available mark for the component due at that milestone will become 40%. Specific penalties described below are applicable to the Final Deadline.

Feature Specifications

Select one of the main entities (classes/tables) for your proposed system. (In this example we use a Player class, but you will name your own table appropriately). It must be a class that has at least one integer field, one floating point field and one String field. You class must have a field called "id". and it must be an autoincrement field of a suitably sized Integer type (INT ?).

1. Create and populate a MySQL Database Table to store your main entity. i.e. one table is required with at least 10 rows of data. You must create a "MySqlSetup.sql" file with the SQL required to create and populate the database table so that it can be easily recreated. Add this file in your project directory. When you have completed features 1-7, you can add **additional related tables (1:M) or (M:M)** to improve the quality of your system and demonstrate your mastery of the material.

2. Create a corresponding Data Access Layer with a DTO, a DAO and corresponding Interfaces that allow access to your database table using the structure provided in the Data Access Layer code sample. (See Features below)

3. Create a simple menu system that will allow you to select the options stated below: (Commit and push each feature to your shared remote Repo as you develop them (2-3 commits at least per feature would be expected). Marks will be lost if a consistent and spaced history of Repo commits & pushes is not in evidence.

Basic Features (DAO Methods)

Feature 1 – Get all Entities (assuming Player entities in this example)
e.g. List<Player> getAllPlayers() - return a List of all the entities and display the returned list.

Feature 2 – Find and Display (a single) Entity by Key
e.g. Player getPlayerById(id) – return a single entity (DTO) and display its contents.

Feature 3 – Delete an Entity by key
e.g. deletePlayerById(id) – remove specified entity from database

Feature 4 – Insert an Entity
(gather data, instantiate a Player object, pass into DAO method for insertion in DB) e.g. Player insertPlayer(Player p)
return new entity (Player DTO) that includes the assigned auto-id.

Feature 5 – Update an existing Entity by ID using supplied Player object
e.g. updatePlayer(int id, Player p) – executes specified updates

Feature 6 – Get list of entities matching a filter (based on DTO object)
e.g. List<Player> findPlayersApplyFilter(playerAgeComparator)

4. Create two methods (in a JsonConverter Class) that will convert data into JSON format:

   Feature 7 - Convert List of Entities to a JSON String
   e.g. String playersListToJsonString(List<Player> list)

   Feature 8 – Convert a single Entity by Key into a JSON String
   e.g. String playerToJsonString(Player p)

5. Test your features by creating a number of meaningful JUnit tests. (These tests form part of your submission)

Client-Server Features

The following features will allow you to integrate the components that you have completed previously with the client-server material that has been provided in class. Your Data Access Layer (DAOs) will reside on the server, and your Menu/GUI will reside on the client.

It is recommended that you implement the client with a GUI, using JavaFX. A well written console application is also acceptable, but you will not be eligible for full marks (refer to the grading rubric for 'Excellent' and 'Exceptional' grading criteria.) The best approach may be to implement a console based interface initially and afterwards add the GUI.

Feature 9 - "Display Entity by Id"
Implement a client-side menu item that allows a user to select the option "Display Entity by ID" where the Entity is your selected entity class (e.g. Display Player by ID). The client will send a request (command) to the server, along with the user inputted identifier (ID), in accordance with your specified protocol. The server will process the request, extract the ID, and call an appropriate DAO method (on server side) to retrieve the entity by ID from the database and instantiate a Player object. The Player object will then be serialized into JSON representation, and sent from server to client via a socket stream. The client will receive the JSON data, parse the data and instantiate and populate an entity object with the data. The data will then be retrieved from the entity object and displayed on the client screen in a neatly formatted manner.

Feature 10 - "Display all Entities"
Implement a client-side menu option "Display all Entities" that will send a request to the server to obtain a list of all entities. The server will process the request (command) and will use a DAO method to retrieve a list of entities. This list will be passed to a method that will convert it to JSON format (array of json objects), and return it as a JSON String. The JSON String will be sent from server to client via a socket. The client will parse the received JSON data and use it to populate a List of entities. All entities will be displayed and neatly formatted from the list of entity objects. Parsing of JSON must be done using the ***org.json*** library.

Feature 11 – "Add an Entity"
Implement a client-side menu item that will allow the user to input data for an entity, **serialize** the data into a JSON formatted request and send the JSON request to the server. The server will extract the received JSON data and add the entity details to the database using a relevant DAO method (INSERT), and will send a success/failure response to the client. On successful insertion, the response will return the new Entity object (as JSON data) incorporating the newly allocated ID (if the ID was auto generated). This will be sent from server to client, and the client will display the newly added entity, along with its auto generated ID. If the insert fails, and appropriate error (as JSON) should be returned to the client and an appropriate client-side message displayed for the user.

Feature 12 – "Delete Entity by ID"
In a manner similar to above, provide a menu item that will delete an entity by ID, send a command (as JSON) to the server to undertake the delete, and return and display an appropriate message on the client.

Feature 13 – "Get Images List"
The client menu will provide a menu item that will allow the user to request a list of image file names available for download from the server. This list of image file names (as String type) is transferred in JSON format. The client user can then select one image and a request for this image will be sent to the server. The server will read the request and then opens the file and sends the file as a binary stream to the client. The client
reads the binary stream, using buffering, and writes the data to a local file (on client side), or displays the image in a GUI. As a more advanced challenge you can consider how to download all listed images at the same time and either store them or display them.

Feature 14 – Exit
Notify the server that this client is quitting and terminate this client.

Submission Checklist

*Submit all the following items to Moodle or your submission **will not be marked**:*

GitHub Regular commits from all students, relevant commit messages, minimum 2 branches. Must be either public, or have invited your teammates and lecturer as a collaborators.
Screencast 5 minutes (max) – all students talk through a running demo of your project with two clients including extra functionality, point out relevant code
Coversheet Separate sheets signed by each student
Zipped project Include everything needed to run the project, (Repo URL alone is not acceptable) including DB configuration and seed data (SQL). One submission per group is sufficient but include all coversheets.

Students will demonstrate their work, and each student will be interviewed on **ALL** aspects of the project.

Academic Integrity

The assignment must be entirely the work of each student – in your own words. Students are not permitted to share any pseudocode or source code from their solution with any other student in the class. Students may not distribute the source code of their solution to any other student in any format (i.e., electronic, verbal, or hardcopy transmission). Any suspected plagiarism will be investigated, pursued, and reported to the Plagiarism Committee.

Generative artificial intelligence (AI) tools cannot be used in this assessment task. In this assessment, **you must not use** generative artificial intelligence (AI) (ChatGPT, ChatSonic, Bing Chat, Lex, DALL-E 2, or other tools) to generate any materials or content in relation to the assessment task.

The DkIT Academic Integrity Policy and Procedures, https://www.dkit.ie/about-dkit/policies-and-guidelines/academic-policies.html) states the following:

"Using generative artificial intelligence tools (e.g. ChatGPT) in an assessment unless explicitly permitted to do so and with proper acknowledgement, is a form of plagiarism".

Late Submissions

The following late submission penalties will apply:

- The marks awarded to the assessment element will be reduced by 20% for material submitted after the deadline and **within 24hrs** of the deadline.
- The marks awarded to the assessment element will be capped at 40% for material submitted after (the deadline + 24hrs), and before 72hrs after the deadline.
- Assessment material submitted more than 72 hours after the deadline will be given a mark of zero.

Standard DkIT policies will apply in relation to legitimate verifiable absence from assessment or late submission of assessment.