



GDSC Inner Core Enrollments '25

Technical Task Round-2

| | |
|--|-----------|
| General Instructions | 1 |
| Tech: Frontend | 3 |
| Instructions | 3 |
| Task One: Responsive Social Media Website | 3 |
| Task Two: AI-Powered Mood-Based Theme Switcher | 4 |
| Tech: Fullstack | 6 |
| Instructions | 6 |
| Task: Real-Time Collaborative Communication System | 6 |
| Tech: Backend | 8 |
| Instructions: | 8 |
| Task one: E-Commerce Marketplace Backend with Order Management | 8 |
| Task two: Smart Library System | 9 |
| Tech: App | 11 |
| Instructions | 11 |
| FLUTTER/ANDROID: Developer Focussed Chat App | 11 |
| IOS APP DEVELOPMENT: Code Challenger Tracker! | 12 |
| Tech: Game | 14 |
| Task: Hypercasual Game Development | 14 |
| Tech: ML | 15 |
| Task Name: CIFAR-10 Image Classification with Deep Learning | 15 |
| Task: AI Agent with RAG and Function | 16 |
| Tech: Python | 17 |
| Task: 6502 or CHIP-8 Emulator Development | 17 |
| Task two: Discord Bot! | 18 |
| Tech: Cybersecurity | 19 |
| General Instructions | 19 |
| Task: Pack and Ship | 19 |
| Task: Hidden Pixels | 20 |

General Instructions

- **These instructions apply to all Tech subdomains.**
- In case a domain has multiple tasks, you may attempt **either/both** task(s). Your attempt will be graded according to the features you have implemented, and their corresponding complexity and/or value.
- Create a public GitHub repo with a branch for each task you are attempting, and try to make **frequent, atomic commits** while working instead of one big commit containing everything at the end.
- Do **as much as possible**. Even if you're not able to complete a task, make sure to have a README detailing your process and the steps you would take next given more time.
- Have fun :D

Tech: Frontend

Instructions

- Try to **deploy** your finished task on Netlify/Vercel/etc.
- Feel free to use vanilla JS, TS or any framework/library of your choice like React, Next, Svelte, etc.

Task One: Responsive Social Media Website

Develop a fully responsive social media website that allows users to view and interact with posts using the [DummyJSON API](#). The implementation can be done using **HTML, CSS, and Vanilla JS** or **any frontend framework of your choice** (React, Vue, Angular, etc.). The evaluation will focus on the completeness and quality of feature implementation rather than the tech stack used.

Core Features We Expect:

- **User Authentication**
 - Implement **JWT-based authentication** with a login screen.
 - Use **protected routes** to restrict access to authenticated users only.
- **Post Feed**
 - Fetch and display all posts from the **DummyJSON API**.
 - Show details like **post content, likes, dislikes, and tags**.
- **Search, Sort, and Pagination**
 - Implement **search functionality** to find posts by keywords.
 - Enable **sorting by likes, dates, or popularity**.
 - Add **pagination** to optimize post-loading.
- **Post Details & Comments**
 - Clicking a post should navigate to a **detailed post view**.
 - Display **comments** for the post and other relevant details from the API.
- **Tag Filtering**
 - Allow users to **filter posts by tags** to find relevant content easily.
- **User Profile Page**
 - Clicking a user's post or comment should navigate to their **profile page**.
 - Display the user's details and a **list of posts made by them**.

Implement the following for brownie points 🧁:

Theme Switching:

- Implement a **light/dark mode toggle** to enhance user experience.
- Refer to <https://monkeytype.com/> themes to add dynamic themes to the website.(Check footer for multiple themes)

Resources:

- **Deployment Platforms:** [Netlify](#), [Vercel](#)
- **Data Sources:** [DummyJSON API](#).

Task Two: AI-Powered Mood-Based Theme Switcher


Develop a **theme switcher** that dynamically adjusts UI themes based on the user's selected mood. The application should allow users to pick a mood and automatically change the theme accordingly. **State management, component reusability, event handling, and conditional rendering** are key focus areas for this task.

Core Features We Expect:

- **State Management**
 - Use **useState** to store the selected **mood** and **theme**.
- **Props & Component Reusability**
 - Implement a **ThemeProvider** component that manages UI theme updates dynamically.
- **Event Handling**
 - Allow users to **select moods** using buttons or a dropdown.
- **Conditional Rendering**
 - Update **UI colours, typography, and background** based on the chosen mood.
- **(Optional) Context API for Global State Management**
 - Use React's **Context API** to make the theme state accessible across the app.

Implement the following for brownie points 🍰 :

- **Innovative UI Enhancements**
 - **Framer Motion Animations:** Apply mood-based animations (e.g., bouncy effects for "Happy," slow fades for "Calm").
 - **Gradient Backgrounds:** Dynamic backgrounds that change based on mood intensity.
 - **Interactive Elements:** Buttons and UI elements that react differently depending on the selected mood.
- **AI-Powered Features**
 - **AI-Generated Theme Suggestions:** Use **OpenAI API** to recommend a theme colour palette based on mood.
 - **Sentiment Analysis:** If using text input, analyze sentiment and auto-select a mood.
- **Personalization & Persistence**

- **Local Storage Support:** Save the last selected **mood and theme** for user consistency.
- **User-Customized Mood Themes:** Allow users to tweak colors and styles for each mood.
- **Independent Dark/Light Mode Toggle:** Separate from mood-based theme changes, for accessibility.
- **Fun Additions**
 - **Mood-Based Music Recommendation:** Fetch a playlist suggestion from **Spotify API**.
 - **Mood-Based UI Sounds:** Add subtle sound effects based on mood selection (e.g., soft chimes for “Calm”).
-  **Bonus Challenge**
 - **“Surprise Me” Button:** Randomly selects a **mood and its corresponding theme**

Resources:

- **UI Design Tips:** If uncertain about backend integrations, begin by developing the user interface and simulate interactions using mock/test APIs to populate the UI dynamically.
- **OpenAI API for AI-powered suggestions:** <https://platform.openai.com/docs/>
- **Spotify API for music recommendations:** <https://developer.spotify.com/documentation/web-api/>

Tech: Fullstack

Instructions

- Feel free to attempt this task if you've been selected for Frontend and/or Backend, or even otherwise:)
- **Dockerization** is a must. Try to deploy the frontend on a platform like Vercel/Netlify, and if possible the backend on a platform where you have credits (a VM, or platforms like Fly).
- Frontend: Feel free to use vanilla JS, TS or any framework/library of your choice like React, Vue, Svelte, etc. Backend: use the language of your choice:)

Task: Real-Time Collaborative Communication System

Develop a real-time collaborative platform where multiple users can work on a shared project simultaneously. The system should support **live cursor tracking** and **instant updates** for all participants, ensuring a synchronized experience (similar to Figma). Users should be able to draw **basic shapes on a canvas**, and others should see the updates in real time. The interface should be smooth, responsive, and designed for **seamless collaboration**.

Reference Examples: [Figma](#), [Google Docs](#), [Miro](#)

Core Features We Expect:

- **User Authentication**
 - Implement login and signup functionality using at least one of the following methods: email/password or social auth.
- **Real-time Collaboration**
 - Users should be able to join a shared workspace and see real-time updates.
 - Changes made by one user (drawing, modifying shapes) should instantly reflect others.
- **Live Cursor Tracking**
 - Each user's cursor should be visible to others in real time.
 - Cursor movements should be smooth and efficient to avoid lag.
- **Canvas Drawing**
 - Users can draw **basic shapes** (rectangles, circles, lines) on the shared canvas.
 - Every shape and modification should be visible to all participants.
- **WebSocket Integration**
 - Use **WebSockets** to maintain real-time communication between users.

Implement the following for brownie points 🍰 :

- **Anonymous User Participation**
 - Allow users to participate in games anonymously, without creating an account, though these users should not have access to a game history feature.
- **Rooms Feature**
 - Users can **create or join rooms** using a shareable **room code**.
 - Each room should be independent, allowing different teams to collaborate separately.
- ⚡ **No Socket.io Challenge (Instant Recruit Bonus)**
 - Implement WebSocket functionality **without using Socket.io**.
 - Directly handle raw WebSocket connections for optimized performance.

🔗 Resources:

- **Real-Time Interaction:** Explore technologies like [WebSocket](#) or [Firebase](#) for managing real-time data exchanges necessary for interactive gameplay.
- **WebSockets API:** [MDN WebSockets API](#)
- **Figma Collaboration:** [How Figma's Multiplayer Tech Works](#)
- **Figma Live Collaboration:** [Figma Collaboration Docs](#)

Tech: Backend

Instructions:

- Use a suitable backend framework (e.g., Django, Flask, Express.js) and database (e.g., SQLite, PostgreSQL) for the development.
- Ensure secure API endpoints with appropriate authentication and authorization mechanisms.
- Implement error handling and logging for better debugging and user experience.
- Write clear documentation for API endpoints, including request/response formats and usage examples.

Task one: E-Commerce Marketplace Backend with Order Management

Develop the backend system for an **e-commerce marketplace** where users can **browse products, manage shopping carts, and place orders**. Sellers should be able to **list and update products**, while authentication ensures **secure access** to different features.

Core Features We Expect:

- **Product Management**
 - Implement **CRUD** (Create, Read, Update, Delete) operations for sellers to **manage products**.
 - Users can **fetch product listings** and search by **categories, price range, and keywords**.
- **Shopping Cart & Checkout**
 - Users can **add, update, and remove** items from their shopping cart.
 - The cart should display **total price** and **remaining account balance**.
 - Implement **order processing** after checkout.
- **User Authentication & Profiles**
 - **JWT Authentication**: Secure login and user sessions.
 - **Role-Based Access Control (RBAC)**:
 - **Buyers** can manage their cart, browse products, and place orders.
 - **Sellers** can add, update, and remove products.
 - Implement **middleware** to enforce access control.
- **Database & Backend Considerations**
 - Use **Django, Flask, or Express.js** as the backend framework.
 - Use **SQLite, PostgreSQL, or MongoDB** for database management.
 - Secure API endpoints with **authentication and authorization mechanisms**.
 - Implement **error handling and logging** for debugging and user experience.
 - Write **clear API documentation** with request/response formats and usage examples.

Implement the following for brownie points 🍰 :

- 📄 **Discount/Coupon Feature**
 - Sellers can **create custom coupon codes** for discounts.
 - Buyers can **apply coupons** during checkout for price reductions.
- 📧 **Webhook Functionality**
 - Implement a webhook to **send notifications** when the cart status changes.

🔗 **Resources:**

- **Task Details:** [Google Docs Link](#)

Task two: Smart Library System

Develop a **modern library system** where students can **find, borrow, and return books** easily. The system should also offer **book recommendations** and **analytics** to enhance the user experience.

Core Features We Expect:

- **Book Management** 📖
 - **Add new books to the library.**
 - **Update book details** (title, author, genre, copies, etc.).
 - **Remove books when necessary.**
 - **Search for books** (Google-like search but simpler).
 - **Allow users to rate and review books.**
- **Smart Borrowing System**
 - **Handle book checkouts and returns.**
 - **Calculate due dates for borrowed books.**
 - **Fine Calculation:** Track overdue books and calculate fines.
 - **Reservation System:** Users can reserve books when they are unavailable.
- **Book Recommendations**
 - **Suggest books based on user reading history.**
 - **Display popular books across different categories.**
 - **Consider user ratings when recommending books.**

Implement the following for brownie points 🍰:

- **Advanced Search Functionality**
 - Implement **filters** for author, genre, and availability.
- **Basic Analytics Dashboard**
 - Track **most borrowed books** and **popular genres**.
- **User Reading History**
 - Maintain a history of books borrowed by a user.

🔗 **Resources:**

- **Sample Book Structure**

```
{  
  "isbn": "978-3-16-148410-0",  
  "title": "Sample Book",  
  "author": "Author Name",  
  "genre": ["Fiction", "Mystery"],  
  "copies": 3,  
  "available_copies": 2,  
  "location": "Section A-12",  
  "rating": 4.5,  
  "borrowed_count": 15  
}
```

Submission:

- Provide a GitHub repository with the backend codebase, including clear instructions on setting up and running the application.
- Include API documentation detailing endpoints, request/response formats, and any additional features implemented.
- Optionally, provide a demo or walkthrough video showcasing the implemented features and functionalities.

Tech: App

Instructions

- Unless specified otherwise, feel free to use whichever language or framework you feel most comfortable with to build the mobile application.

FLUTTER/ANDROID: Developer Focussed Chat App

Create a private messaging app tailored for developers, featuring coding-friendly discussions, syntax-highlighted messages, and real-time collaboration. The interface should be clean and intuitive, inspired by Discord's private messaging system.

Tech Stack: Flutter + Firebase

Feel free to use React Native, Native(Kotlin/Java) with Firebase as well

Core Features We Expect:

- Implement secure user authentication using Firebase Auth for sign-up and login.
- Users can send and receive messages in one-on-one conversations.
- Messages containing code snippets should automatically display with syntax highlighting.
- Enable users to reply to specific messages with a quoted reference, similar to Discord's reply system.

Implement the following for brownie points 🍰 :

- Allow users to engage in live voice and video chats within private conversations.
- Users can link their GitHub accounts to receive real-time updates on issues and pull requests in their chat. Utilize GitHub's REST API to fetch relevant data and display notifications.

🔗 Resources:

- **GitHub API:** <https://docs.github.com/en/rest>
- **Flutter Development:** Explore [Flutter](#) documentation for best practices on mobile development.
- **GitHub Auth API:** <https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/authorizing-oauth-apps>
- **GitHub Webhooks for Real-Time Updates:** <https://docs.github.com/en/webhooks>

IOS APP DEVELOPMENT: Code Challenger Tracker!

Develop an iOS app that allows users to track and view LeetCode and CodeChef leaderboard rankings. The app enables users to create and join groups via a group code, fostering collaborative tracking.

- App should be built using Swift and SwiftUI.
- It should use the Observation Framework (<https://developer.apple.com/documentation/observation>).
- It should use Swift Concurrency a.k.a. async/await. (<https://developer.apple.com/videos/play/wwdc2021/10132/>).
- Should store local data using SwiftData (<https://developer.apple.com/xcode/swiftdata/>).

Core Features We Expect:

- **Group Management**
 - Users can create and join groups for LeetCode and CodeChef challenges.
 - Groups can be joined via a group code.
 - Local storage to persist group information using SwiftData.
- **User Profile Tracking**
 - Add multiple LeetCode and CodeChef profiles.
 - Fetch and display user data from both APIs.
 - Show leaderboard rankings with:
 - Username
 - Score
 - Global Rank
- **Leaderboard Display**
 - Sortable leaderboard rankings for each group.
 - Supports both LeetCode and CodeChef users.
- **Refresh Feature**
 - Fetch the latest ranking updates.

Implement the following for **brownie points** 🍰:

- Use SwiftUI Charts to provide a graphical representation of ranking progression.
- Gamification features like:
 - Alert when a friend surpasses another in rankings.
 - Award badges & achievements for milestones.
 - Encourage healthy competition with progress streaks.

Resources:

- **APIs:**
 - LeetCode API: <https://leetcode-api-faisalshohag.vercel.app/{username}>
 - CodeChef API: <https://codechef-api.vercel.app/handle/{handle}>
- **Docs for Reference:**
 - Swift Data: <https://developer.apple.com/documentation/swiftdata>
 - Observation Framework:
<https://developer.apple.com/documentation/observation>
 - Swift Concurrency:
<https://developer.apple.com/videos/play/wwdc2021/10134/>
 - SwiftUI Charts: <https://developer.apple.com/documentation/charts>

Tech: Game

Task: Hypercasual Game Development

Create a simple hypercasual game in the engine/language of your choice. Hypercasual games are known for their minimal mechanics, easy-to-learn gameplay, and addictive nature (e.g., *Helix Jump*, *Flappy Bird*). We require you to submit a built, playable and available to download game or a video of the game play. The game must be playable on **Windows, Web, or Android**.

Requirements:

- The game should include a **Main Menu**.
- A **Pause Menu** should be implemented, and the game must completely stop when paused.
- The **UI should scale properly** for common aspect ratios:
 - **Landscape:** 16:9, 16:10 | **Portrait:** 9:16, 10:16
- The game should **persistently store the high score locally**.
- A brief **Game Design Document (GDD)** should be included with the submission.

Implement the following for **brownie points** 🍰:

- **VFX elements** (particles, line renderers, etc.)
- **Custom assets** created specifically for the game.
- **Custom Shaders** (advanced feature).
- **XR (VR/AR) support**

Tech: ML

Task Name: CIFAR-10 Image Classification with Deep Learning

Dataset Link: <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>

Train a deep learning neural network to accurately classify an image using the CIFAR-10 dataset of 60,000 images (Check dataset in resources)

Core Features We Expect:

- **Build Deep Learning Model:**
 - Construct a CNN architecture for this (Conv2D, MaxPooling2D, Flatten, Dense, Dropout, and batch normalisation layers)
- **Train and Validate Model:**
 - Use a proper train-test split.
 - Implement early stopping to prevent overfitting.
 - Tune hyperparameters (e.g., learning rate, batch size) for better performance.
- **Evaluate Performance:**
 - Calculate accuracy, precision, recall, and F1-score.
 - Compare training and validation accuracy/loss curves.

Implement the following for brownie points 🍰 :

- Implement Transfer Learning using a pre-trained model like VGG16 or ResNet.
- Optimize the model using techniques like Batch Normalization, Learning Rate Schedulers, or Data Augmentation.

🔗 Resources:

- **Dataset:** <https://www.kaggle.com/c/cifar-10>
- **TensorFlow/Keras:** [Documentation](#)
- **CNN Guide:** <https://cs231n.github.io/convolutional-networks/>

Task: AI Agent with RAG and Function

Develop an AI agent capable of processing files as input using **LlamaIndex** or **LangChain**. The agent should exhibit **reasoning capabilities**, support **Retrieval-Augmented Generation (RAG)**, and implement **function calling** for basic arithmetic operations.

Requirements:

- Implement file ingestion using **LlamaIndex** or **LangChain**.
- Integrate **RAG** to enhance responses with the retrieved information.
- Support **basic arithmetic function calling** (e.g., addition, subtraction, multiplication, division).
- The agent should demonstrate **reasoning capabilities** to process and infer information from the provided files.

Implement the following for **brownie points** 🍰:

- Implement **function calling** to **generate notes and summaries** based on the provided content.
- Enable the agent to **search the web** for relevant information to enhance its responses.

Tech: Python

Task: 6502 or CHIP-8 Emulator Development

Develop an **emulator** for either the **6502** or **CHIP-8** processor. You will be provided with a **binary compiled for the chosen platform**, containing a subset of the available instructions. The emulator should be capable of running this binary and **producing the correct output**.

Core Features we expect:



- **Core FCPU Emulation**
 - Implement an instruction decoder that reads opcodes from memory.
 - Execute the correct operation based on the opcode.
- **Memory Management**
 - Implement a simple memory model to store:
 - Program instructions
 - Registers
 - Stack (if applicable)
 - Correctly load the provided binary into memory.
- **Registers & Stack Handling**
 - Implement all necessary registers (e.g., general-purpose registers, program counter, stack pointer).
 - Handle stack operations like pushing and popping values.
- **Input/Output Handling**
 - Implement a way to display output (for CHIP-8, this means a simple graphics display; for 6502, it could be a command-line output).
 - Handle keyboard input where necessary.
- **Execution Loop**
 - Implement a fetch-decode-execute cycle for running instructions continuously.
 - Ensure proper timing and clock emulation if necessary.

Implement the following for brownie points 🍰 :

- **Extended Instruction Support**
 - Implement **more opcodes** beyond the minimal required subset.
- 🎮 **Graphical Output (For CHIP-8)**
 - Implement a **graphical display** using SDL, OpenGL, or another library.
- 🔄 **Save/Load State Functionality**
 - Allow users to **save** and **restore** the emulator's state.

🔗 Resources:

- **Languages Suggestion:** C, C++, Rust, Python
- **Libraries (For CHIP-8 graphics):** SDL2, Raylib, OpenGL

-  [CHIP-8 Wikipedia](#)
-  [6502 Emulator Guide](#)

Task two: Discord Bot!

Build a Discord Bot with Chat, Polls, Reminders, and Music Functionality

Core Features We Expect:

The bot should be able to:

- Respond to user messages in the Discord server using the Gemini API.
- Create, delete, and modify reminders for users in the Discord server. Users should be able to set a reminder by sending a message with the time and date of the reminder in a specific format.
- Users should be able to create Polls.

Implement the following for brownie points 🍰 :

- **AI-powered Summaries** – Users can ask the bot to summarize long messages.
- **Custom Welcome Messages** – The bot welcomes new members.
- **Music Queue System** – Users can view and manage a queue of songs.
- **Auto-Delete Expired Reminders** – To keep the system clean.

Tech: Cybersecurity

General Instructions

Each task will have a flag that has to be discovered and your approach needs to be documented and presented at the review. The flag follows the format: gdsc{...}

Task: Pack and Ship

Our GDSC Tech Team accidentally left some sensitive data in their latest binary release! They attempted to protect it with basic obfuscation, but we need you to evaluate how secure it really is. Your challenge is to recover the hidden flag.

Files Provided:

- [Binary Release](#)
- SHA256:
56548282b3d968e12d14c0760fc2ac778d5a9bc9f10c78d1c8233aa97f5b7329

Flag Format:

The flag follows the format: gdsc{...}

Instructions:

- Download the binary file.
- Grant execution permissions and run the binary locally:

```
chmod +x unpacking  
./unpacking
```

Task: Hidden Pixels

During GDSC's Flutter App Development workshop, someone took a screenshot of our debug console. We noticed some unusual artifacts in the image, and our Flutter mentor hinted that it might contain clues about an upcoming workshop topic. Can you analyze this debug screenshot and uncover the hidden information?

Files Provided:

- [Screenshot](#)
- SHA256:
53c7ddbf269ea3dcb2c22e73c09a0f597fb302677a59c846102a206a9db72b9e

Flag Format:

The flag follows the format: `gdsc{...}`

Instructions:

1. Download the provided image file.
2. Analyse the image for any hidden messages.