**WORK: ARCHITECTURAL DESIGN RECORD[ADR]**

APP NAME: DIME DART GROUP NAME: CODING SQUAD SUBMITTED TO: JASPREET GILL

The t**emplate** used is by **Michael Nygard** and was discussed in class.

1. ADR 1: Type of Mobile App  
     
   Context: Dime Dart App must choose the kind of mobile application to create, considering compatibility with both iOS and Android systems.   
     
   Options Examined:   
     
    1 creating a mobile hybrid application that runs on both iOS and Android.   
    2 not making use of front-end CSS.

Conclusion: We've decided to develop a hybrid mobile application that runs on both iOS and Android.   
  
Rationale: By creating a hybrid application, we want to improve accessibility and user-friendliness while guaranteeing inclusiveness for users on both the iOS and Android platforms.   
  
Consequences: This choice suggests that platform-specific application development work for Android and iOS is required. The Flutter framework will be utilized to ensure a smooth and uniform experience on both platforms.

Follow-up: To ensure seamless performance on both the iOS and Android platforms, make sure optimization and extensive testing are carried out.

1. ADR 2: User Interface Design   
     
   Context: Dime Dart App must choose a UI framework to build quicker and with uniform design on both the iOS and Android platforms.   
     
   Options Examined:   
     
    1 Making use of React Native.   
    2 not using CSS Tailwind.

Conclusion: React Native will be our UI framework of choice.   
  
Rationale: React Native offers a well-rounded method for creating UI elements that work well on both the iOS and Android platforms, enabling quicker development and guaranteeing uniform design.   
  
Consequences: By making this choice, work will go more quickly, and the design will remain similar on all platforms. To strike a balance between personalization and usability, though, may take some thought.

Follow-up Actions: To keep a balanced approach to UI development, make well-informed customisation decisions.

1. Backend Language (ADR 3)   
     
   Context: A backend language must be chosen by Dime Dart App to get great performance and scalability.   
     
   Options Examined:   
     
    1 With Node.js.   
    2 Examining alternative backend languages.

Conclusion: Node.js will be the backend language.   
  
Rationale: Node.js provides great performance and scalability, which meet project objectives and facilitate effective management of backend operations.   
  
Consequences: Reliance on Node.js packages and modules might be dangerous if they are inadequate or of low quality, which could have an impact on the functionality of the application.

Follow-up Measures: To reduce the possibility of performance problems, continuously check and evaluate the quality of the Node.js packages used in the project.

1. ADR 4: Permissions  
     
   Context: To ensure user access and security, Dime Dart App must ascertain which permissions are necessary.   
     
   Options Examined:   
     
    1 need a login to use the application.   
    2 without applying stringent authorization requirements.

Conclusion: To protect user data, the program will only launch if the user is logged in.   
  
Rationale: Enforcing users to log in improves data security and lowers the possibility of unwanted access, protecting user information.   
  
Consequences: Strict permission enforcement improves security, but it may aggravate users when providing rights.

Follow-up Measures: To reduce user annoyance while upholding security standards, provide user-friendly permission request interfaces.

1. ADR 5: Data Storage  
     
   Context: Dime Dart App must choose the database that will be used to store application data.   
     
   Options Examined:   
     
    1 Making use of SQL Database.   
    2 Thinking about MongoDB.

Conclusion: SQL databases will be used to store data.   
  
Rationale: SQL databases are recommended because they are user-friendly and well-known to team members, allowing for effective data archiving and retrieval.   
  
Consequences: Scalability and ease of data administration are provided by using a SQL database. That could, however, be less flexible than MongoDB.

Follow-up Actions: Train team members or provide them tools to improve their understanding of SQL databases and resolve any restrictions on database flexibility.