**PHASE 2 – MOBILE APPLICATION DEVELOPMENT**

**APPLICATION ARCHITECTURE**

**MEMBERS**

**Dong Chen**

**Manminder Singh**

**Yaling Wei**

**Kris Schneider**

**Ngoc Tam Nguyen**

**Target devices: Android**

**CSS framework: Bootstrap**

**Development framework: TO BE UPDATED BY CHEN**

**ADR**

* **Decision Title**: Selection of Development Framework for Cross-Platform Mobile Application.
* **Context**: As we begin the development of our cross-platform mobile application, we need to select a suitable development framework. This decision will impact our development speed, performance, user experience, and ability to leverage the team's existing skillset.
* **Decision Options**:

|  |  |  |
| --- | --- | --- |
|  | **Pros** | **Cons** |
| **React Native** | * **Performance**: High performance, closer to native due to compiled code. * **UI Components**: Provides native-like UI, suitable for a seamless user experience. * **Community**: Large community and resources available for troubleshooting. | * **Learning Curve**: Requires familiarity with JavaScript and React, may be steep for some. * **App Size**: Larger initial app size compared to pure native apps. |
| **Ionic** | * **Cross-Platform**: Code once, deploy on Android, iOS, and web. * **UI Components**: Large set of UI components, easy to build visually appealing apps. * **Faster Prototyping**: Quick setup and prototyping with familiar web technologies. | * **Performance**: Slower than React Native for complex apps due to reliance on WebView. * **Limited Native Access**: May require third-party plugins for more advanced hardware access. |
| **Framework7** | * **iOS Look**: Designed to give apps an iOS feel, beneficial if iOS aesthetic is desired. * **No Build Tools Required**: Simple for web developers, uses HTML, CSS, and JavaScript. * **Integrated UI Elements**: Many pre-built components for quicker development. | * **Limited to Smaller Apps**: Not ideal for complex or resource-intensive apps. * **Community**: Smaller community compared to React Native and Ionic, limited resources. |

* **Final Decision**: we decided to take react native as our development framework.
* **Rationale**:

**Performance and Native Feel**: React Native provides a high-performance experience that closely resembles a native app, which is crucial for real-time functionalities like GPS tracking and providing directions.

**Strong Community Support**: The large and active community provides extensive resources, libraries, and plugins, enabling faster troubleshooting and access to third-party integrations.

* **Consequences**:

**Increased App Size**: Thev eg resulting app may have a larger file size compared to purely web-based apps, potentially impacting download times and storage on users' devices.

**Dependency on Third-Party Libraries**: While React Native has a rich ecosystem, reliance on third-party libraries for certain functionalities (like specific hardware access) can introduce risks related to updates and compatibility over time.

**Navigation strategy: UPDATED BY MANMINDER**

**ADR**

* **Decision Title**: Selecting the Navigation method for application
* **Context**: As its already decided that we are going to use react native as a development framework for the application, now it is turn of navigation strategy

We are looking for the strategy which is suitable to apply on cross platform and support the modern OS system of the mobile devices

* **Decision Options**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Navigation strategy** | **Explanation** | **Pros** | **Cons** |
| **STACK NAVIGATION** | It is like when screens stacked on other screens and pop up and pop out just as minimize function works on the windows or mobiles | * **Clear Structure** for usage * Easy to make it work on **different platforms** | **Complex navigation method** specially when it comes to configure directions |
| **TAB NAVIGATION (Used for the project)** | The screen elements come into play to do tab navigation, it is like touch and go  For example;  Instagram uses the tab navigation where everything is based on the provided elements on the screen | * Quick access to the functions * Easy for user to identify the functions of the elements by observing the logos * Easy to setup for cross platform devices | It makes the application static (very few dynamics which may affect the flow on the user side |
| **DRAWER NAVIGATION** | This navigation method is a type of hidden menu on the side of application | * It can fit more options than any other method **without covering the space on the screen** * **Do not affect the performance** of the UI | * Not easy for users to locate * operate this navigation strategy   it will put extra load on the performance of the system |
| **BREAD-CRUM NAVIGATION** | It is based on links which sets up trail for the user just like click this link to navigate on the desired page mostly suitable for webpages instead of mobile applications | * Easy for users to locate directions such like go back and forward * There is very less chance of getting lost between the navigation when using bread-Crums | Mostly not used for developing mobile applications and screen size of mobile phones is a big challenge to fit bread-Crum in it with efficient way. |
| **SEARCH BASED NAVIGATION** | This navigation method is very simple for operating as users just need to search in the search bar whatever they are looking for | * It uses large datasets to answer the queries of   the users   * Quick response as compared to any other method as it has API integration with datasets | * As it looks very simple for user but at same point it is very difficult for the developers to manage the API integrations at the backend of the application * quality of the results may vary as it depends on many factors like user input, wrong spelling, |

* **Final Decision**: We decided to go with the Tab navigation for our application

**Rationale**

* **SETUP WILL BE EASY**

One of the biggest reasons of using tab navigation is it is easy to setup for cross platform devices so we can get the environment without facing any big issue with it as a developer it is very important step to configure the tools which fits in the requirement

* **USER FRIENDLY**

Other one more important reason to use this is after application development completes the application will end up as a user-friendly tool which make things easy for the user who wants to locate parking areas or using other features of the application with just one touch

**Consequences**:

* **Access to Nested features get effected**

When it comes to access the functions which are embedded in primary functions it may mislead the user to access them some problems can be arise like

* double tap on the feature
* Page element is not responding to the request after clicking
* Hard to locate and explore every function of the application as it will be hidden behind other functions

**Hardware: TO BE UPDATED BY TAM**

**ADR**

* **Decision Title**: Selecting the hardware for the application.
* **Context**: As we start to develop mobile applications. We need to choose the hardware possible for the application. It can help to develop our features by using it with the mobile application.
* **Decision Options**: (GPS, Speaker, Fingerprint scanner, etc. The hardware our App will invoke or utilize to accomplish our designed features, it could be a combination such as GPS + Speaker + Gyroscope) A list of possible options (three options would be sufficient), along with the pros and cons of each.

|  |  |  |
| --- | --- | --- |
|  | **Pros** | **Cons** |
| GPS | **Location tracking**: easy tracking location of devices which help to improve. It helps to create exactly the function’s location.  Help to identify **slot parking available** and suggest to users.  Recommend and Recommend and Navigate users moving to the nearest slot parking. | The device will spend a lot of power using GPS.  It should connect to the Internet to show the location. |
| Speaker | Users can easily navigate by instruction of audio directions.  Sound notification to alert slot parking available or news of the application. | The sound can distract users when driving.  The old device can limit the support sound. Ex: the sound is not enough to listen to when driving. |
| Fingerprint scanner | It’s easy to log in to the application.  It’s high security for users. | It depends on the support of devices.  Rejection identifies the finger when it is affected by the outside. |

* **Final Decision**: We decided to get GPS as our hardware application.
* **Rationale**:

**Popular almost device:** GPS supports almost all versions of the mobile.

**A lot of location applications using it**: There are a lot of applications both Google Play Store and Apple Store to use it.

* **Consequences**:

**Depending on the acceptance permissions of users**: The application just can use GPS when the user allows it. So, It cannot work exactly when it doesn’t allow on their devices.

**Depending on the user’s network**: GPS cannot be used without the Internet. So, if the user doesn’t connect to the Internet, they cannot use this feature.

**Database storage: TO BE UPDATED BY KRIS**

**ADR**

* **Decision Title**: Selecting the appropriate database that will suit the needs of our application.
* **Context**: During the app development, we eventually realized that it would be easier to manage our data if we made use of a database.
* **Decision Options**: (Local (encrypted or unencrypted), remote, or none) A list of possible options (three options would be sufficient), along with the pros and cons of each.

|  |  |  |
| --- | --- | --- |
|  | **Pros** | **Cons** |
| Local (encrypted or unencrypted) | Encrypted:   * **Enhanced Security:** Encryption significantly improves data protection, making it much harder for unauthorized individuals to access sensitive information even if the device is compromised or lost. This is crucial for applications handling personal data, financial information, or other sensitive content. * **Offline Functionality:** Data remains accessible even without an internet connection, enhancing the user experience, especially in areas with poor network coverage. * **Compliance:** Meeting regulatory requirements (like GDPR, HIPAA) related to data protection is often easier with strong local encryption. * **Greater Control:** You have more control over your data and its security than with cloud-based solutions.   Unencrypted:   * **Simplicity:** Easier to implement and manage compared to an encrypted database. * **Performance:** Generally faster performance than an encrypted database due to the absence of encryption/decryption overhead. * **Accessibility:** Data is readily accessible without the need for decryption. | Encrypted:   * **Complexity:** Implementing and managing encryption correctly requires expertise and can be complex. Errors in implementation can weaken security. * **Performance Overhead:** Encryption and decryption processes can impact application performance, potentially leading to slower response times. The impact depends on the encryption method and device capabilities. * **Key Management:** Securely storing and managing encryption keys is critical. Loss or compromise of the keys renders the data inaccessible. * **Device-Specific:** The encryption method might be tied to the specific mobile operating system or device, limiting portability and potentially creating compatibility issues. * **Backup and Restore:** Backing up and restoring encrypted data requires careful planning and execution to ensure data integrity and accessibility.   Unencrypted   * **Security Risks:** Data is vulnerable to unauthorized access if the device is compromised, lost, or stolen. This is a significant drawback, especially for applications handling sensitive information. * **Compliance Issues:** Failure to protect data adequately can lead to non-compliance with data protection regulations and potential penalties. * **Limited Control:** You have less control over data security compared to an encrypted database. |
| Remote | * **Cost-effectiveness:** Eliminates the need for in-house IT professionals and infrastructure, leading to significant cost savings, particularly for smaller businesses. You pay only for the services used. * **24/7 Support:** Constant monitoring and maintenance ensure quick responses to potential issues, minimizing downtime. * **Centralized Storage and Backups:** Data is stored in a central location, simplifying backups and recovery processes. * **Shared Resources and Collaboration:** Facilitates efficient collaboration and resource sharing across different locations or teams. * **Increased Security (with proper implementation):** Adding another layer of protection, especially when combined with SSL encryption. This can be particularly useful for archiving older, less frequently accessed data. * **Scalability:** Remote databases can often be scaled more easily to handle increasing data volumes and user traffic. * **Flexibility:** Allows access from various locations and devices. | * **Limited Control:** You have less control over the data due to its nature of being remote. * **Security Issues:** Remote databases are vulnerable to security breaches if not properly secured. * **User Privileges/Limited Access:** Depending on the administrator, you may only have access to certain tables within the database or have no access to any part of the database at all. * **Dependence on Network Connectivity:** Reliable internet connectivity is crucial for accessing and using the remote database. Outages can disrupt operations. * **Complexity:** Setting up and managing a remote database can be more complex than a local database. * **Latency:** Network latency can impact performance, especially if the database is geographically distant from the users or applications accessing it. |
| No database | * **Simplicity:** The simplest approach is to have no database at all, especially for very small applications or tasks that don't require persistent data storage. This eliminates the complexity of database setup, management, and maintenance. * **Reduced Costs:** No database means no licensing fees, no server costs (if applicable), and no specialized personnel needed for database administration. * **Faster Development (for small projects):** For very small projects with limited data needs, the development process might be faster without the overhead of database integration. | * **Data Loss:** Data is typically not persistent without a database. If the application closes or crashes, all data is lost. * **Scalability Issues:** Handling even moderately large amounts of data becomes extremely difficult or impossible without a database. * **Data Integrity Problems:** Ensuring data consistency and accuracy is extremely challenging without a database's built-in mechanisms for data validation and constraint enforcement. * **Difficult Data Management:** Searching, sorting, filtering, and updating data become extremely cumbersome and inefficient without a database management system. * **Security Risks:** Data security is virtually non-existent without a database's security features. * **Limited Functionality:** Many application features that rely on data persistence (e.g., user accounts, history tracking, reporting) are impossible to implement without a database. |

* **Final Decision**: For this mobile project, we have decided to use a local database. The main reason for this is because it allows for the fetching of data, even with the API.
* **Rationale**: We have chosen to use a local database for the following reasons:
  + **Offline functionality:** A local database will still be able to operate without depending on the internet.
  + **Enhanced Performance:** Storing and retrieving data locally is significantly faster than accessing a remote database.
  + **Improved Privacy:** Keeping sensitive data locally on the user's device enhances data privacy and security. This is because the data does not need to be transmitted over a network, reducing the risk of interception or unauthorized access.
  + **Reduced Server Load:** By offloading some data storage and data processing to the user’s device, the application reduces the load on the remote server.
* **Consequences**: Here are the consequences of using a local database for our application:  
  **Data Management and Availability:**
  + **Data Isolation:** A local database, by its nature, stores data only on the device where it's installed. This means data is not readily accessible from other devices or locations without specific mechanisms for synchronization or data transfer. This can be a limitation for applications requiring data sharing or collaboration.
  + **Offline Functionality (but also limitation):** While a significant advantage, the reliance on local data also means that the application's functionality is limited if the device is offline, or the local database is corrupted. This is a trade-off; improved offline capabilities come at the cost of potential data inaccessibility in certain situations.
  + **Data Backup and Recovery:** The responsibility for backing up and recovering the local database falls solely on the application developer and the user. Failure to implement proper backup strategies can lead to significant data loss if the device is lost, stolen, or damaged.

**Security:**

* + **Device Security:** The security of a local database is directly tied to the security of the device itself. If the device is compromised (e.g., through malware or physical theft), the local database and its contents are at risk. Encryption can mitigate this risk, but it's not a guarantee.
  + **Data Breach Risk:** While a local database reduces the risk of data breaches during transmission, it doesn't eliminate the risk entirely. A compromised device can still lead to a data breach. Furthermore, if the local database isn't properly secured (e.g., lacking encryption or strong access controls), it's vulnerable to attacks.

**Performance and Scalability:**

* + **Performance Bottlenecks:** A local database's performance is limited by the device's processing power, memory, and storage capacity. This can become a bottleneck for applications with large datasets or high data access rates.
  + **Scalability Challenges:** Scaling a local database to handle a growing amount of data or an increasing number of users can be challenging and may require significant changes to the application's architecture. This is in contrast to cloud databases which can be more easily scaled.