Installations:

IDE's and Simulators:

- We will be using Visual studio as an IDE for entire project development.
- For IOS Applications we need XCode can be downloaded from AppStore to use simulator
- For Android Applications Download Android studio

Dependencies:

- Install node and watchman by using Homebrew
  >>>brew install node
  >>>brew install watchman
- cocoapods manages library dependencies for XCode
  >>>sudo gem install cocoapods

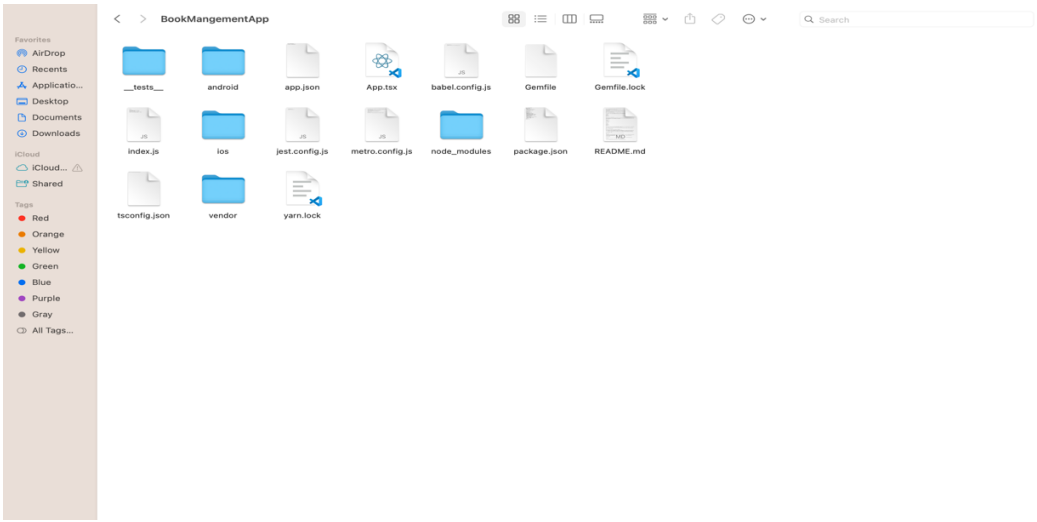
Building Project from scratch:

Backend setup:

- MySQL for database you can use MySQL from command line, or you can also use MySQL workbench
- Create folder Book-Management-Backend and initialize new node.js project
  >>>npm init -y
- This command will setup the project for you which consists of package.json file Which is a manifest of your project that includes the packages and applications it depends on, information about its unique source control, and specific metadata like the project's name, description, and author.
- In this project we are using node.js server using express, Prisma as a ORM and Apollo client for graphQl to install all these dependencies run the command below in the root director of your backend
  >>> npm install express apollo-server-express prisma @prisma/client graphql

Frontend setup:

- open visual studio and run the below command to initialize your project
  >>> npx react-native init BookManagementApp
- This is going to initialize your project by create the files show in below image



- We need Apollo client for communication to backend, for navigation from one screen to another screen we need react-native-navigation library which provides navigation for both platforms.

  >>>npm install @apollo/client graphql
  >>>npm install react-navigation @react-navigation/native @react-navigation/stack

- We also need for formik, yup and react-native gesture handler for creating forms and gestures

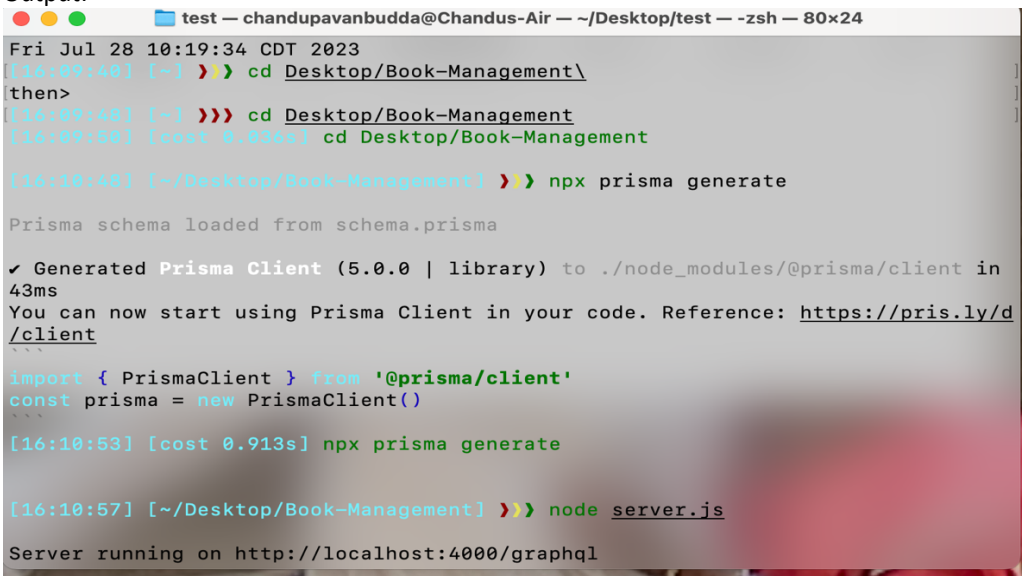
  >>>npm install formik yup react-native-gesture-handler

Server Build:

- Create an empty prisma.schema file inside the server root folder where you can write your schema for the database

Change URL to your MySQL URL –>mysql://username:Password@localhost:portnumber/db_name

- To generate the prisma client run the following command
  >>>npx prima generate
  Output:



- Create a file schema.js in the root directory
- The schema.js file plays a crucial role as it defines the GraphQL schema. The schema is a contract between the client and the server that defines the available types, queries, mutations, and subscriptions that the client can interact with. It provides a clear understanding of what data can be requested from the server and what data can be modified.

- Create a resolvers.js file which is responsible for implementing the resolver functions for each field in the GraphQL schema. Resolvers are functions that provide the actual data for each field in the schema during

query execution. They are used to fetch data from the database, manipulate data, or perform any other necessary actions to fulfill the client's requests.

- In our project we have books, book, createBook, updateBook and deleteBook
  - prisma.book.findMany() -> to fetch all the books data in the database
  - prisma.book.findUnique() ->to fetch specific book by its id
  - prisma.book.create()->to create new book
  - prisma.book.update()->to update the data that edited
  - prisma.book.delete()->to delete the book in the database

- Create a server.js file which is typically the entry point of your server-side code. It is where you set up and configure the GraphQL server using a specific library or framework, such as Apollo Server, Express, or any other GraphQL server implementation.
- After creating all these files run the following command
  >>> npx prisma migrate dev

```
[22:47:21] [~/desktop/Book-Management] ››› npx prisma migrate dev
Prisma schema loaded from schema.prisma
Datasource "db": MySQL database "Book" at "localhost:3306"

✔ Enter a name for the new migration: … book_migration
Applying migration `20230730035103_book_migration`

The following migration(s) have been created and applied from new schema changes:

migrations/
  └─ 20230730035103_book_migration/
    └─ migration.sql

Your database is now in sync with your schema.

✔ Generated Prisma Client (5.0.0 | library) to ./node_modules/@prisma/client in
38ms


[22:51:03] [cost 192.975s] npx prisma migrate dev
```

  This command helps to sync the schema to the database

  >>>node server.js

```
[16:10:57] [~/Desktop/Book-Management] ››› node server.js

Server running on http://localhost:4000/graphql
```

  You can see that your server is running on a particular URL if you go to that URL you will redirect to Apollo studio which is a built in API tester where we can test all our queries and mutation.

Frontend Build:

- Just go to root folder of frontend and enter the following commands.

  For IOS:
  >>>npx react-native run-ios
  For Android:
  >>>npx react-native run-android

  It will automatically connect to the metro bundler which manages assets, caches build and performs hot module reloading and opens simulator for respective environment.

  File structure of Front-end:

```
[19:41:49] [~/Desktop/BookMangementApp] ❱❱❱ tree -L 1~/
.
├── App.tsx
├── Gemfile
├── Gemfile.lock
├── README.md
├── __tests__
├── android
├── app.json
├── babel.config.js
├── index.js
├── ios
├── jest.config.js
├── metro.config.js
├── node_modules
├── package-lock.json
├── package.json
├── src
├── tsconfig.json
├── vendor
└── yarn.lock

7 directories, 13 files
```

Except src everything is automatically created while initialization of the project

Inside src:

```
[19:42:14] [~/Desktop/BookMangementApp/src] ❱❱❱ tree -L 1~/
.
├──  SearchAndFilter
├── client
├── constants
├── forms
├── hooks
├── navigation
└── screens

8 directories, 0 files
[19:42:18] [cost 0.036s] tree -L 1~/
```

In this project we have four screens they are:

- Booklist -> where we have the list of books
- Book Details ->where we have details of the book
- Edit Book ->where we can update Book details
- Add Book ->where we can add our Book

Search and Filter Folder-> where we have components for implementations of searching and filtering

Client->where we have connection to the server

Constants->if you have trouble connecting to the server you can also use mock Data (sample data)

Forms -> I used formik where we can manipulate the form state, how form validations and error messages are handled, how form submission handled both edit book and add book uses this form component.
Hooks-> Inside of hooks there is only one file queries.js where we would have graphql quaries but for the sake of simplicity I also implemented queries on the files where they are needed. Using of these hooks is

helpful when you want to migrate from one server to another server (we must go every file to change the queries instead we can directly change in hooks) we can also implement each query as a single hook which decreases the code due to reusability.

Navigation -> which is used to move between different screens

App.tsx -> In a typical React Native project, the entry point of the application is often named App.js or index.js. This file serves as the starting point of your application and contains the setup for rendering the root component and other configurations. Where we will pass the navigation component and navigation component uses rest of the components in src