**Project Report**

**on**

**“Studify”**

**Submitted to:**

**Faculty of Technology and Computer Application**

**Amrapali Institute of Applied Sciences, Haldwani**

**In the partial fulfillment for the award of degree of**

**Bachelor of Computer Application**

**Project Guide: Dr. Suresh Wariyal**

**Submitted By: Chandra Mohan Pandey**

****

**Amrapali Institute of Applied Sciences, Haldwani**

**Project Report**

**on**

**“Studify”**

**Submitted to:**

**Faculty of Technology and Computer Application**

**Amrapali Institute of Applied Sciences, Haldwani**

**In the partial fulfillment for the award of degree of**

**Bachelor of Computer Application**

**Project Guide: Dr. Suresh Wariyal**

**Submitted By: Chandra Mohan Pandey**

****

**Amrapali Institute of Applied Sciences, Haldwani**

**Declaration**

“I confirm that the work within the completed assignment is all my own work, and does not include any work completed by anyone other than myself unless referenced. I have completed the assignment in accordance with instructions given by the assigned mentor and within the time limits set by my mentor.

By signing my name below, I am declaring the authenticity of the work done.”

Name: Chandra Mohan Pandey

Place: Haldwani

Date: 09-10-2020

**Acknowledgement**

I would like to express my special thanks of gratitude to our honorable Director & Professor (Dr. M. K. Pandey) and my mentor (Dr. Suresh Wariyal) who gave me the golden opportunity to do this wonderful project (Studify), which also helped me in doing a lot of research and I came to know about so many new things I am really thankful to them.

Secondly, I would like to thank the Faculty of Technology and Computer Application, Amrapali Institute of Applied Sciences, Haldwani.

At last, I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

**Index**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Topic** | **Page No.** |
| 1 | Abstract | 7 |
| 2 | System Requirements  2.1 Product Definition  2.2 Problem Statement  2.3 Functions to be provided  2.4 Processing Environment  2.5 Solution strategy  2.6 User Story & Acceptance Criteria  2.7 Feasibility Analysis  2.8 Project Plan | 8-10  8  8  8  8  9  9  9-10  10 |
| 3 | System Requirement Specifications  3.1 Developing/ Operating/ Maintenance Environment  3.2 User Display Format  3.3 Data Dictionary | 11-22  11  12-18  19-22 |
| 4 | Design  Definition  4.1 Use Case Diagram  4.2 Data Flow Diagrams  4.3 Entity Relationship Diagram | 23-36  23-26  27-28  29-33  34-36 |
| 5 | Source Code  Definition  6.1 index.js  6.2 App.js  6.3 MarkAttendanceScreen.js  6.4 SubjectsScreen.js  6.5 TimeTableScreen.js  6.6 TasksScreen.js  6.7 ConcentrateScreen.js  6.8 ProfileScreen.js | 37-77  37-41  42  42-44  45-50  51-54  55-57  58-61  62-71  72-77 |
| 6 | Test Plan | 78-80 |
| 7 | Conclusion  7.1 Current Status of Project  7.2 Future Scope  7.3 Limitations  7.4 Bibliography | 81-84  81  82  83  84 |

**Abstract**

Studify is a student utility app.

It is designed by keeping-in-mind the needs of the students of today’s generation.

It provides its users with many useful functionalities such as the following:

1. Keeping a track of their attendance.
2. Listing their current semester/ class subjects.
3. One-tap accessibility to the time table.
4. Keeping a list of upcoming tasks.
5. Listen to concentration music for focus work.
6. Manage a profile that indicates their current overall attendance.

In one line, “everything that a student need”.

It’s an open-source project and its full source code can be accessed at <https://github.com/cmcodes1/Studify>

This app is hosted on Google Play Store and currently has an overall 5.0 star rating and dozens of downloads and active users. It can be downloaded from the link: <https://play.google.com/store/apps/details?id=com.cmcodes.studify>

**System Requirements**

**2.1 Product Definition**

Studify is a student utility app which provides its users with functionalities that improve their academic performance.

**2.2 Problem Statement**

A lot of students often struggle to keep up with the consistency in today’s world which is full of distractions all around. They have little idea regarding their current attendance or the upcoming tasks and assignments or their subjects or their timetable. Here’s when **Studify** come to the rescue!

**2.3 Functions to be provided**

* Attendance management
* Subjects listing
* Time Table accessibility
* Tasks
* Listen concentration music
* Profile maintenance

**2.4 Processing Environment**

Hardware recommendation:

* Installed Memory (RAM): 4GB DDR3 1333MHz
* Internal Storage: 512GB HDD
* Processor: Intel Core i3-3220 CPU @ 3.30 GHz
* Processor Type: x64-based Processor

Software recommendation:

* Code Editor: Visual Studio Code 1.46.0
* Runtime Environment: Node.js 12.16.1
* Package Manager: NPM 6.13.4
* Framework: React Native 0.62.2
* Operating System: Windows 10 Home
* OS Type: 64-bit Operating System

**2.5 Solution strategy**

The most optimized and accessible solution for the given problem came out to be a mobile application which works without any need of an internet connection or high device specifications.

**2.6 User Story & Acceptance Criteria**

User Story: As a student, the user wants evaluation of his progress and reminder of upcoming tasks to improve his academic performance.

Acceptance Criteria:

* Display student’s current attendance for each subject.
* Display list of the concerned subjects of current semester/ class.
* Display time table in just one tap.
* Display upcoming tasks/ assignments.
* Display profile and overall attendance.

**2.7 Feasibility Analysis**

Feasibility Type:

* Economic Feasibility: No need for any major capital.
* Technical Feasibility: Can be coded on any latest and working system.
* Operational Feasibility: Can be executed and tested on a smartphone emulator or a mobile device.
* Schedule Feasibility: Can be completed within the given timeline.

**2.8 Project Plan**

* Team structure: One developer.
* Development Schedule: Everyday 09:00 to 15:00.
* Programming Language: JavaScript
* Framework: React Native
* Runtime Environment: Node.js
* Development Tool: Visual Studio Code

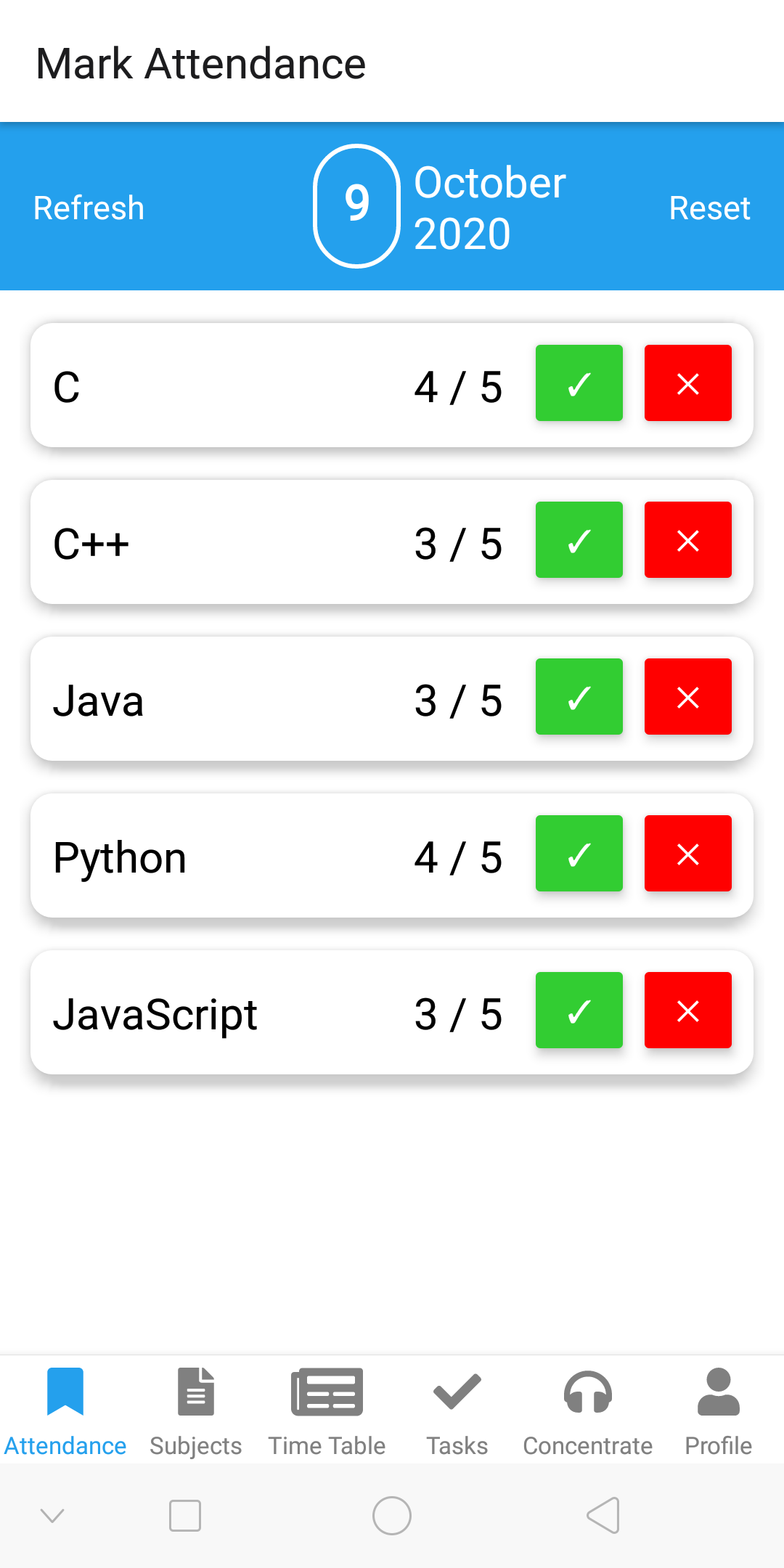
**System Requirement Specifications**

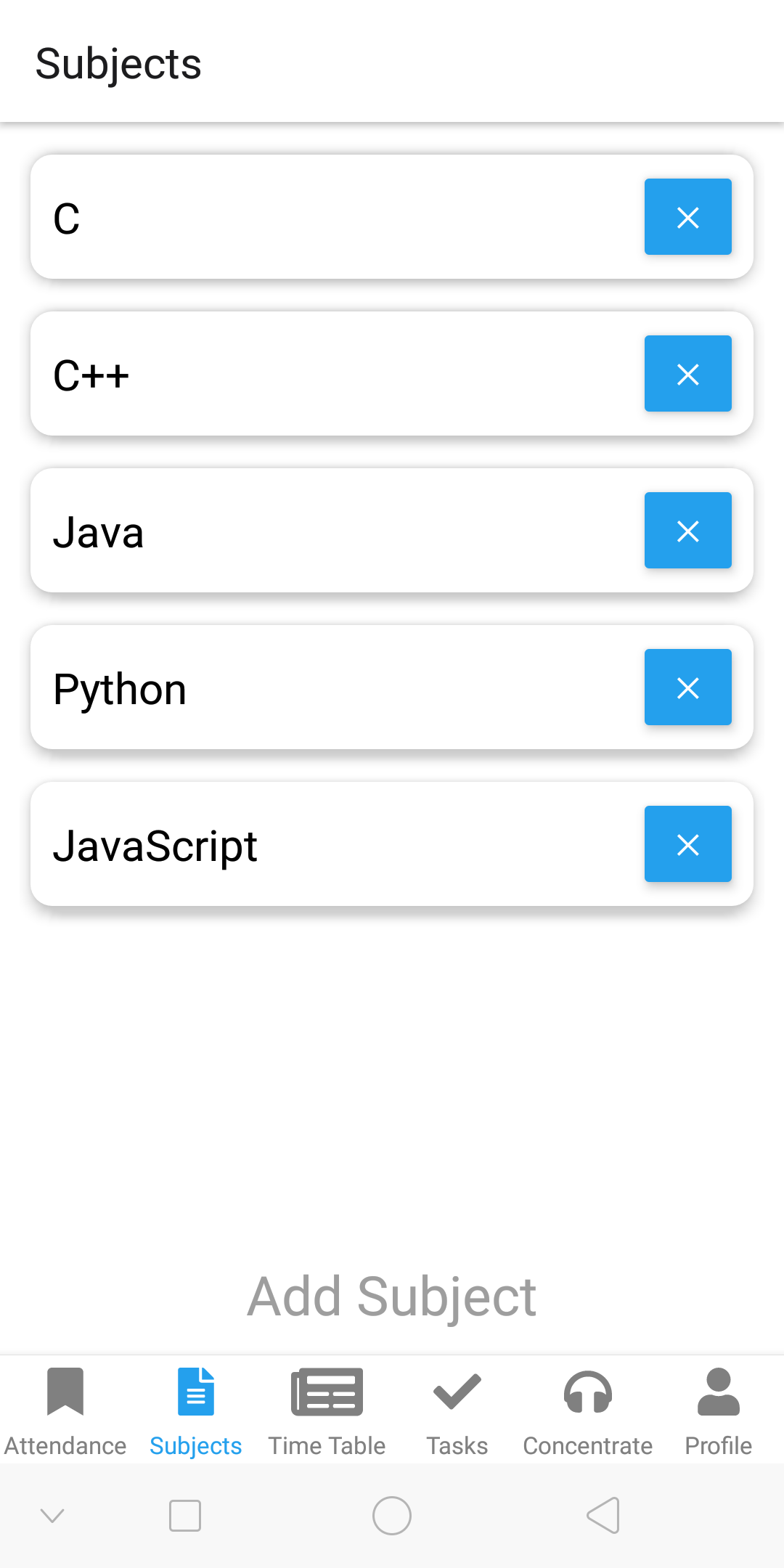
**3.1 Developing/ Operating/ Maintenance Environment**

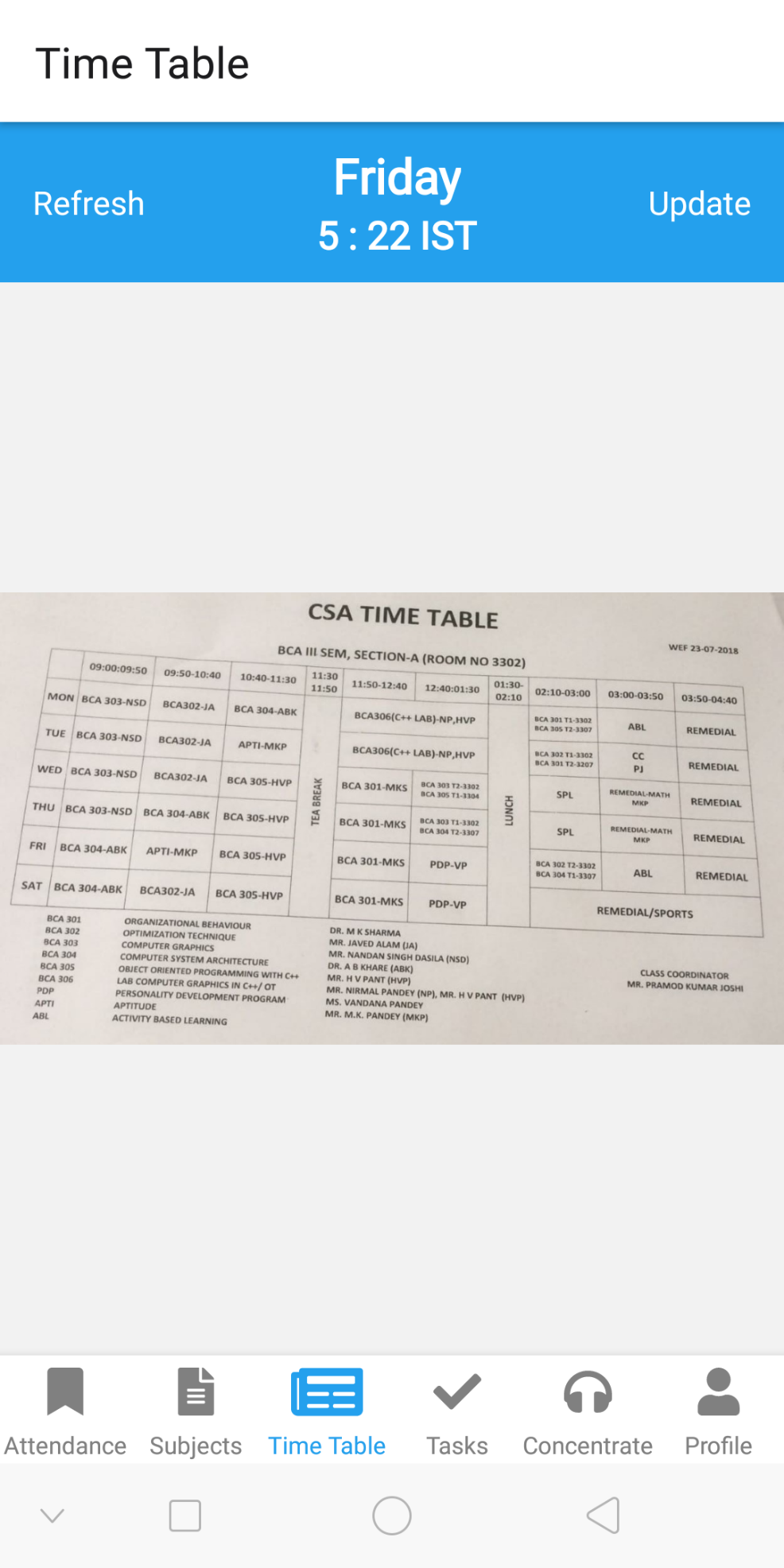
* Installed Memory (RAM): 4GB DDR3 1333MHz
* Internal Storage: 512GB HDD
* Processor: Intel Core i3-3220 CPU @ 3.30 GHz
* Processor Type: x64-based Processor
* Code Editor: Visual Studio Code 1.46.0
* Runtime Environment: Node.js 12.16.1
* Package Manager: NPM 6.13.4
* Framework: React Native 0.62.2
* Operating System: Windows 10 Home
* OS Type: 64-bit Operating System

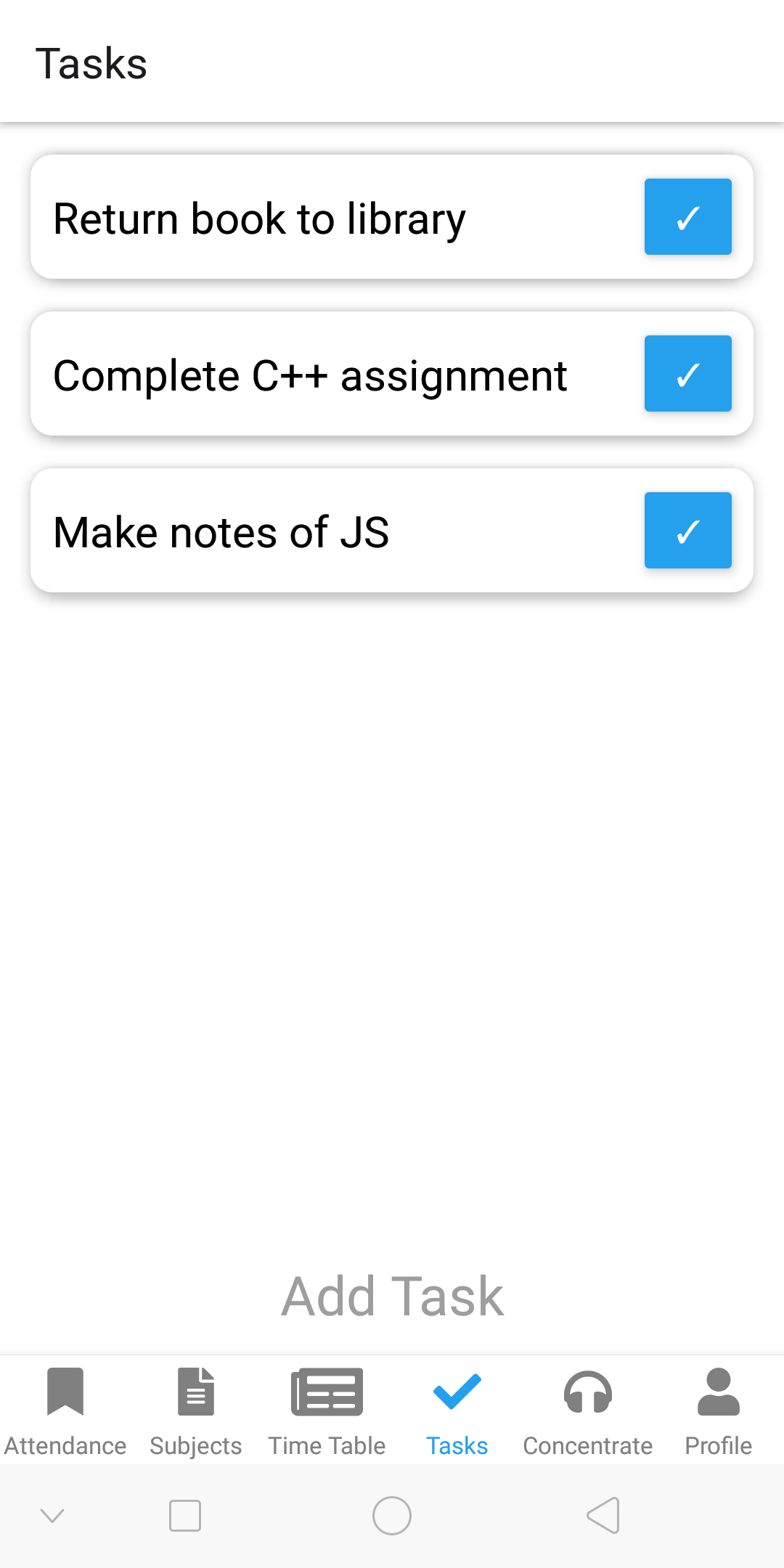
**3.2 User Display Format**

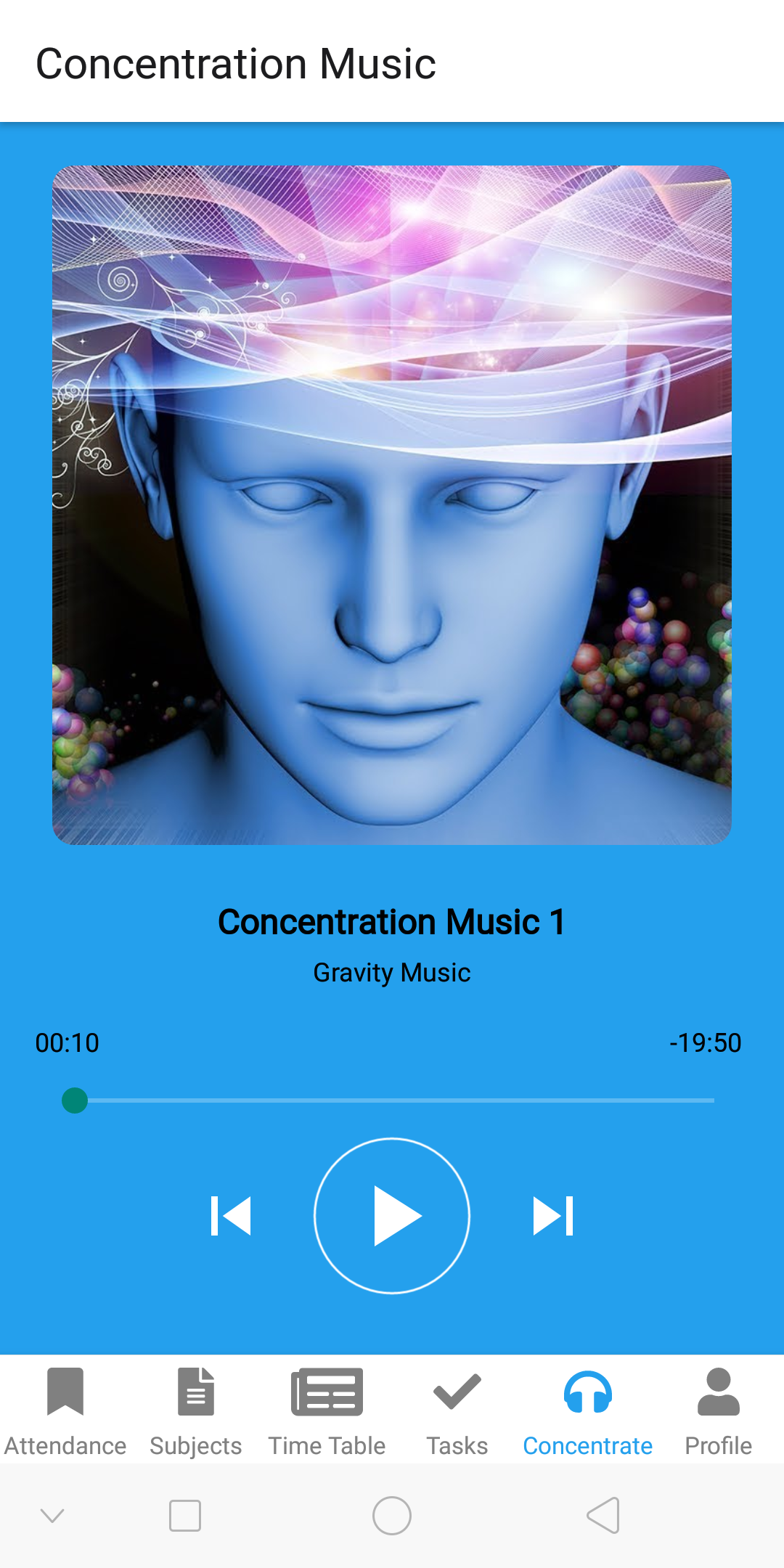
****

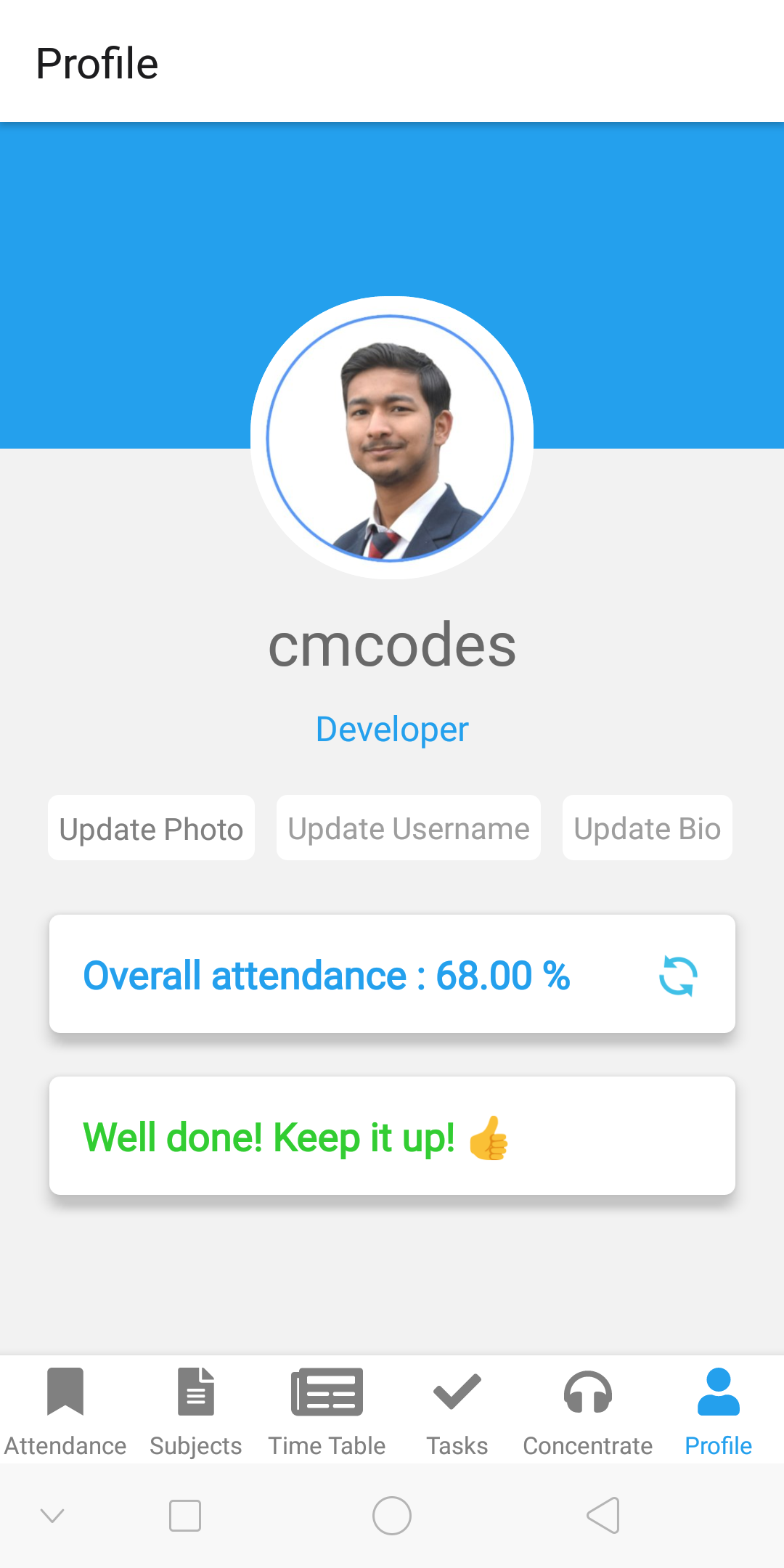


****









**3.3 Data Dictionary**

A data dictionary, or metadata repository, as defined in the IBM Dictionary of Computing, is a "centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format". Oracle defines it as a collection of tables with metadata. The term can have one of several closely related meanings pertaining to databases and database management systems (DBMS):

* A document describing a database or collection of databases.
* An integral component of a DBMS that is required to determine its structure.
* A piece of middleware that extends or supplants the native data dictionary of a DBMS.

**Documentation**

The terms data dictionary and data repository indicate a more general software utility than a catalogue. A catalogue is closely coupled with the DBMS software. It provides the information stored in it to the user and the DBA, but it is mainly accessed by the various software modules of the DBMS itself, such as DDL and DML compilers, the query optimizer, the transaction processor, report generators, and the constraint enforcer. On the other hand, a data dictionary is a data structure that stores metadata, i.e., (structured) data about information. The software package for a stand-alone data dictionary or data repository may interact with the software modules of the DBMS, but it is mainly used by the designers, users and administrators of a computer system for information resource management. These systems maintain information on system hardware and software configuration, documentation, application and users as well as other information relevant to system administration.

If a data dictionary system is used only by the designers, users, and administrators and not by the DBMS Software, it is called a passive data dictionary. Otherwise, it is called an active data dictionary or data dictionary. When a passive data dictionary is updated, it is done so manually and independently from any changes to a DBMS (database) structure. With an active data dictionary, the dictionary is updated first and changes occur in the DBMS automatically as a result.

Database users and application developers can benefit from an authoritative data dictionary document that catalogs the organization, contents, and conventions of one or more databases. This typically includes the names and descriptions of various tables (records or Entities) and their contents (fields) plus additional details, like the type and length of each data element. Another important piece of information that a data dictionary can provide is the relationship between Tables. This is sometimes referred to in Entity-Relationship diagrams, or if using Set descriptors, identifying which Sets database Tables participate in.

In an active data dictionary constraint may be placed upon the underlying data. For instance, a Range may be imposed on the value of numeric data in a data element (field), or a Record in a Table may be FORCED to participate in a set relationship with another Record-Type. Additionally, a distributed DBMS may have certain location specifics described within its active data dictionary (e.g. where Tables are physically located).

The data dictionary consists of record types (tables) created in the database by systems generated command files, tailored for each supported back-end DBMS. Oracle has a list of specific views for the "sys" user. This allows users to look up the exact information that is needed. Command files contain SQL Statements for CREATE TABLE, CREATE UNIQUE INDEX, ALTER TABLE (for referential integrity), etc., using the specific statement required by that type of database.

There is no universal standard as to the level of detail in such a document.

**Platform-specific examples**

Developers use a data description specification (DDS) to describe data attributes in file descriptions that are external to the application program that processes the data, in the context of an IBM System i. The sys.ts$ table in Oracle stores information about every table in the database. It is part of the data dictionary that is created when the Oracle Database is created.

**Typical attributes**

Here is a non-exhaustive list of typical items found in a data dictionary for columns or fields:

* Entity or form name or their ID (EntityID or FormID). The group this field belongs to.
* Field name, such as RDBMS field name
* Displayed field title. May default to field name if blank.
* Field type (string, integer, date, etc.)
* Dimension(s) such as min and max values, display width, or number of decimal places.
* Field display order or tab order
* Coordinates on screen (if a positional or grid-based UI)
* Default value
* Prompt type, such as drop-down list, combo-box, check-boxes, range, etc.
* Is-required (Boolean) - If 'true', the value can't be blank, null, or only white-spaces
* Is-read-only (Boolean)
* Reference table name, if a foreign key. Can be used for validation or selection lists.
* Various event handlers or references to. Example: "on-click", "on-validate", etc. See event-driven programming.
* Format code, such as a regular expression or COBOL-style "PIC" statements
* Description or synopsis
* Database index characteristics or specification

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Description** | **Example** |
| present\_count | Array | Stores the count of individual classes attended of the respective subjects sequentially. | [2, 3, 5, 7, 5] |
| total\_count | Array | Stores the count of individual classes attended of the respective subjects sequentially. | [4, 7, 6, 7, 8] |
| present | Integer | Stores the total number of classes attended of all the subjects. | 22 |
| absent | Integer | Stores the total number of classes held of all the subjects. | 33 |
| subjects | Array | Stores the name of all the subjects. | [“SE”, “AI”, C#”, “TOC”, “Web Tech”] |
| text | String | Stores the current subject. | “Web Tech” |
| refreshing | Boolean | Stores the current refresh state. | false |
| filePath | String | Stores the path on the device of the saved time table photo. | “val” |
| tasks | Array | Stores the name of all the tasks. | [“Submit assignment”, “Complete the app dev”, “Read book “] |
| title | String | Stores the title of the music. | 'Concentration Music 1' |
| Artist | String | Stores the name of the artist of the music. | 'Gravity Music' |
| albumArtUrl | String | Stores the URL of the album art. | "https://i.imgur.com/bDutmFm.jpg" |
| audioUrl | String | Stores the audio URL of the music. | https://www.mboxdrive.com/concentration\_music\_01.mp3 |
| dpPath | Array | Stores the path on the device of the saved profile photo. | “path” |
| username | String | Stores the name of the user. | “cmcodes” |
| bio | String | Stores the bio of the user. | “Developer” |

**Design**

A design is a plan or specification for the construction of an object or system or for the implementation of an activity or process, or the result of that plan or specification in the form of a prototype, product or process. The verb to design expresses the process of developing a design. In some cases, the direct construction of an object without an explicit prior plan (such as in craftwork, some engineering, coding, and graphic design) may also be considered to be a design activity. The design usually has to satisfy certain goals and constraints, may take into account aesthetic, functional, economic, or socio-political considerations, and is expected to interact with a certain environment. Major examples of designs include architectural blueprints, engineering drawings, business processes, circuit diagrams, and sewing patterns.

The person who produces a design is called a designer, which is a term generally used for people who work professionally in one of the various design areas—usually specifying which area is being dealt with (such as a textile designer, fashion designer, product designer, concept designer, web designer or interior designer), but also others such as architects and engineers. A designer's sequence of activities is called a design process, possibly using design methods. The process of creating a design can be brief (a quick sketch) or lengthy and complicated, involving considerable research, negotiation, reflection, modeling, interactive adjustment and re-design.

**Design as a process**

Substantial disagreement exists concerning how designers in many fields, whether amateur or professional, alone or in teams, produce designs. Kees Dorst and Judith Dijkhuis, both designers themselves, argued that "there are many ways of describing design processes" and discussed "two basic and fundamentally different ways", both of which have several names. The prevailing view has been called "the rational model", "technical problem solving" and "the reason-centric perspective". The alternative view has been called "reflection-in-action", "co-evolution", and "the action-centric perspective".

**The rational model**

The rational model was independently developed by Herbert A. Simon, an American scientist, and two German engineering design theorists, Gerhard Pahl and Wolfgang Beitz. It posits that:

Designers attempt to optimize a design candidate for known constraints and objectives.

The design process is plan-driven.

The design process is understood in terms of a discrete sequence of stages.

The rational model is based on a rationalist philosophy and underlies the waterfall model, systems development life cycle, and much of the engineering design literature. According to the rationalist philosophy, design is informed by research and knowledge in a predictable and controlled manner.

**Example sequence of stages**

Typical stages consistent with the rational model include the following:

* Pre-production design
  + Design brief or Parti pris – an early (often the beginning) statement of design goals
  + Analysis – analysis of current design goals
  + Research – investigating similar design solutions in the field or related topics
  + Specification – specifying requirements of a design solution for a product (product design specification)[14] or service.
  + Problem solving – conceptualizing and documenting design solutions
  + Presentation – presenting design solutions
* Design during production
  + Development – continuation and improvement of a designed solution
  + Testing – in situ testing of a designed solution
* Post-production design feedback for future designs
  + Implementation – introducing the designed solution into the environment
  + Evaluation and conclusion – summary of process and results, including constructive criticism and suggestions for future improvements
* Redesign – any or all stages in the design process repeated (with corrections made) at any time before, during, or after production.

Each stage has many associated best practices.

**Criticism of the rational model**

The rational model has been widely criticized on two primary grounds:

Designers do not work this way – extensive empirical evidence has demonstrated that designers do not act as the rational model suggests.

Unrealistic assumptions – goals are often unknown when a design project begins, and the requirements and constraints continue to change.

**The action-centric model**

The action-centric perspective is a label given to a collection of interrelated concepts, which are antithetical to the rational model. It posits that:

Designers use creativity and emotion to generate design candidates.

The design process is improvised.

No universal sequence of stages is apparent – analysis, design and implementation are contemporary and inextricably linked.

The action-centric perspective is based on an empiricist philosophy and broadly consistent with the agile approach and a methodical development. Substantial empirical evidence supports the veracity of this perspective in describing the actions of real designers. Like the rational model, the action-centric model sees design as informed by research and knowledge. However, research and knowledge are brought into the design process through the judgment and common sense of designers – by designers "thinking on their feet" – more than through the predictable and controlled process stipulated by the rational model.

**Descriptions of design activities**

At least two views of design activity are consistent with the action-centric perspective. Both involve three basic activities.

In the reflection-in-action paradigm, designers alternate between "framing", "making moves", and "evaluating moves". "Framing" refers to conceptualizing the problem, i.e., defining goals and objectives. A "move" is a tentative design decision. The evaluation process may lead to further moves in the design.

In the sensemaking–coevolution–implementation framework, designers alternate between its three titular activities. Sensemaking includes both framing and evaluating moves. Implementation is the process of constructing the design object. Coevolution is "the process where the design agent simultaneously refines its mental picture of the design object based on its mental picture of the context, and vice versa".

The concept of the design cycle is understood as a circular time structure, which may start with the thinking of an idea, then expressing it by the use of visual or verbal means of communication (design tools), the sharing and perceiving of the expressed idea, and finally starting a new cycle with the critical rethinking of the perceived idea. Anderson points out that this concept emphasizes the importance of the means of expression, which at the same time are means of perception of any design ideas.

**4.1 Use Case Diagram**

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

**Application**

While a use case itself might drill into a lot of detail about every possibility, a use-case diagram can help provide a higher-level view of the system. It has been said before that "Use case diagrams are the blueprints for your system". They provide the simplified and graphical representation of what the system must actually do.

Due to their simplistic nature, use case diagrams can be a good communication tool for stakeholders. The drawings attempt to mimic the real world and provide a view for the stakeholder to understand how the system is going to be designed. Siau and Lee conducted research to determine if there was a valid situation for use case diagrams at all or if they were unnecessary. What was found was that the use case diagrams conveyed the intent of the system in a more simplified manner to stakeholders and that they were "interpreted more completely than class diagrams".

The purpose of the use case diagrams is simply to provide the high-level view of the system and convey the requirements in laypeople's terms for the stakeholders. Additional diagrams and documentation can be used to provide a complete functional and technical view of the system.

Student

**4.2 Data Flow Diagrams**

A data-flow diagram is a way of representing a flow of data through a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart.

There are several notations for displaying data-flow diagrams. The notation presented above was described in 1979 by Tom DeMarco as part of Structured Analysis.

For each data flow, at least one of the endpoints (source and / or destination) must exist in a process. The refined representation of a process can be done in another data-flow diagram, which subdivides this process into sub-processes.

The data-flow diagram is part of the structured-analysis modeling tools. When using UML, the activity diagram typically takes over the role of the data-flow diagram. A special form of data-flow plan is a site-oriented data-flow plan.

Data-flow diagrams can be regarded as inverted Petri nets, because places in such networks correspond to the semantics of data memories. Analogously, the semantics of transitions from Petri nets and data flows and functions from data-flow diagrams should be considered equivalent.

**0-level DFD:**

Attendance

Student

Subjects

Time Table

Tasks

Concentrate

Profile

**1-level DFD:**

Subjects List

Overall Attendance

Enter Subjects

Enter Profile Details

Select Time Table Photo

Enter Tasks

Student

Stream music

**2-level DFD:**

Enter Subjects

Student

Save Subjects

subjects

present

Retrieve Subjects

subjects

total\_count

total

present\_count

Student

Mark Attendance

Select Time Table Photo

Student

filePath

Enter Tasks

Student

Save Tasks

tasks

Retrieve

total classes

total

present

Retrieve

total presence

bio

username

total

Student

present

Enter Profile Details

filePath

Retrieve title

title

Retrieve artist

artist

Retrieve album art

Retrieve audio URL

albumArtUrl

audioUrl

**4.3 Entity Relationship Diagram**

An entity–relationship model (or ER model) describes interrelated things of interest in a specific domain of knowledge. A basic ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between entities (instances of those entity types).

In software engineering, an ER model is commonly formed to represent things a business needs to remember in order to perform business processes. Consequently, the ER model becomes an abstract data model, that defines a data or information structure which can be implemented in a database, typically a relational database.

Entity–relationship modeling was developed for database and design by Peter Chen and published in a 1976 paper, with variants of the idea existing previously. Some ER models show super and subtype entities connected by generalization-specialization relationships, and an ER model can be used also in the specification of domain-specific ontologies.

An E-R model is usually the result of systematic analysis to define and describe what is important to processes in an area of a business. It does not define the business processes; it only presents a business data schema in graphical form. It is usually drawn in a graphical form as boxes (entities) that are connected by lines (relationships) which express the associations and dependencies between entities. An ER model can also be expressed in a verbal form, for example: one building may be divided into zero or more apartments, but one apartment can only be located in one building.

Entities may be characterized not only by relationships, but also by additional properties (attributes), which include identifiers called "primary keys". Diagrams created to represent attributes as well as entities and relationships may be called entity-attribute-relationship diagrams, rather than entity–relationship models.

An ER model is typically implemented as a database. In a simple relational database implementation, each row of a table represents one instance of an entity type, and each field in a table represents an attribute type. In a relational database a relationship between entities is implemented by storing the primary key of one entity as a pointer or "foreign key" in the table of another entity.

There is a tradition for ER/data models to be built at two or three levels of abstraction. Note that the conceptual-logical-physical hierarchy below is used in other kinds of specification, and is different from the three-schema approach to software engineering.

Subjects List

Attendance

Subjects

attendance

Profile

Tasks

Time Table

Concentrate

**Source Code**

In computing, source code is any collection of code, with or without comments, written using a human-readable programming language, usually as plain text. The source code of a program is specially designed to facilitate the work of computer programmers, who specify the actions to be performed by a computer mostly by writing source code. The source code is often transformed by an assembler or compiler into binary machine code that can be executed by the computer. The machine code might then be stored for execution at a later time. Alternatively, source code may be interpreted and thus immediately executed.

Most application software is distributed in a form that includes only executable files. If the source code were included it would be useful to a user, programmer or a system administrator, any of whom might wish to study or modify the program.

**Definitions**

The Linux Information Project defines source code as:

Source code (also referred to as source or code) is the version of software as it is originally written (i.e., typed into a computer) by a human in plain text (i.e., human readable alphanumeric characters).

The notion of source code may also be taken more broadly, to include machine code and notations in graphical languages, neither of which are textual in nature. An example from an article presented on the annual IEEE conference and on Source Code Analysis and Manipulation.

For the purpose of clarity "source code" is taken to mean any fully executable description of a software system. It is therefore so construed as to include machine code, very high-level languages and executable graphical representations of systems.

Often there are several steps of program translation or minification between the original source code typed by a human and an executable program. While some, like the FSF, argue that an intermediate file "is not real source code and does not count as source code", others find it convenient to refer to each intermediate file as the source code for the next steps.

**History**

The earliest programs for stored-program computers were entered in binary through the front panel switches of the computer. This first-generation programming language had no distinction between source code and machine code.

When IBM first offered software to work with its machine, the source code was provided at no additional charge. At that time, the cost of developing and supporting software was included in the price of the hardware. For decades, IBM distributed source code with its software product licenses, until 1983.

Most early computer magazines published source code as type-in programs.

Occasionally the entire source code to a large program is published as a hardback book, such as Computers and Typesetting, vol. B: TeX, The Program by Donald Knuth, PGP Source Code and Internals by Philip Zimmermann, PC SpeedScript by Randy Thompson, and µC/OS, The Real-Time Kernel by Jean Labrosse.

**Organization**

The source code which constitutes a program is usually held in one or more text files stored on a computer's hard disk; usually these files are carefully arranged into a directory tree, known as a source tree. Source code can also be stored in a database (as is common for stored procedures) or elsewhere.

The source code for a particular piece of software may be contained in a single file or many files. Though the practice is uncommon, a program's source code can be written in different programming languages. For example, a program written primarily in the C programming language, might have portions written in assembly language for optimization purposes. It is also possible for some components of a piece of software to be written and compiled separately, in an arbitrary programming language, and later integrated into the software using a technique called library linking. In some languages, such as Java, this can be done at run time (each class is compiled into a separate file that is linked by the interpreter at runtime).

Yet another method is to make the main program an interpreter for a programming language, [citation needed] either designed specifically for the application in question or general-purpose, and then write the bulk of the actual user functionality as macros or other forms of add-ins in this language, an approach taken for example by the GNU Emacs text editor.

The code base of a computer programming project is the larger collection of all the source code of all the computer programs which make up the project. It has become common practice to maintain code bases in version control systems. Moderately complex software customarily requires the compilation or assembly of several, sometimes dozens or maybe even hundreds, of different source code files. In these cases, instructions for compilations, such as a Makefile, are included with the source code. These describe the programming relationships among the source code files and contain information about how they are to be compiled.

**Purposes**

Source code is primarily used as input to the process that produces an executable program (i.e., it is compiled or interpreted). It is also used as a method of communicating algorithms between people (e.g., code snippets in books).

Computer programmers often find it helpful to review existing source code to learn about programming techniques. The sharing of source code between developers is frequently cited as a contributing factor to the maturation of their programming skills. Some people consider source code an expressive artistic medium.

Porting software to other computer platforms is usually prohibitively difficult without source code. Without the source code for a particular piece of software, portability is generally computationally expensive. [citation needed] Possible porting options include binary translation and emulation of the original platform.

Decompilation of an executable program can be used to generate source code, either in assembly code or in a high-level language.

Programmers frequently adapt source code from one piece of software to use in other projects, a concept known as software reusability.

**Legal Aspects**

The situation varies worldwide, but in the United States before 1974, software and its source code were not copyrightable and therefore always public domain software.

In 1974, the US Commission on New Technological Uses of Copyrighted Works (CONTU) decided that "computer programs, to the extent that they embody an author's original creation, are proper subject matter of copyright".

In 1983 in the United States court case Apple v. Franklin it was ruled that the same applied to object code; and that the Copyright Act gave computer programs the copyright status of literary works.

In 1999, in the United States court case Bernstein v. United States it was further ruled that source code could be considered a constitutionally protected form of free speech. Proponents of free speech argued that because source code conveys information to programmers, is written in a language, and can be used to share humour and other artistic pursuits, it is a protected form of communication.

An author of a non-trivial work like software, has several exclusive rights, among them the copyright for the source code and object code. The author has the right and possibility to grant customers and users of his software some of his exclusive rights in form of software licensing. Software, and its accompanying source code, can be associated with several licensing paradigms; the most important distinction is open source vs proprietary software. This is done by including a copyright notice that declares licensing terms. If no notice is found, then the default of All rights reserved is implied.

Generally speaking, software is open source if the source code is free to use, distribute, modify and study, and proprietary if the source code is kept secret, or is privately owned and restricted. One of the first software licenses to be published and to explicitly grant these freedoms was the GNU General Public License in 1989; the BSD license is another early example from 1990.

For proprietary software, the provisions of the various copyright laws, trade secrecy and patents are used to keep the source code closed. Additionally, many pieces of retail software come with an end-user license agreement (EULA) which typically prohibits decompilation, reverse engineering, analysis, modification, or circumventing of copy protection. Types of source code protection—beyond traditional compilation to object code—include code encryption, code obfuscation or code morphing.

**Quality**

The way a program is written can have important consequences for its maintainers. Coding conventions, which stress readability and some language-specific conventions, are aimed at the maintenance of the software source code, which involves debugging and updating. Other priorities, such as the speed of the program's execution, or the ability to compile the program for multiple architectures, often make code readability a less important consideration, since code quality generally depends on its purpose.

**6.1 index.js**

import { AppRegistry } from 'react-native';

import App from './App';

import { name as appName } from './app.json';

AppRegistry.registerComponent(appName, () => App);

**6.2 App.js**

import 'react-native-gesture-handler';

import React, { useEffect } from 'react';

import { NavigationContainer } from '@react-navigation/native';

import { createStackNavigator } from '@react-navigation/stack';

import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';

import { faBookmark, faNewspaper, faFileAlt, faCheck, faUser, faHeadphones } from '@fortawesome/free-solid-svg-icons';

import SplashScreen from 'react-native-splash-screen';

import MarkAttendanceScreen from './screens/MarkAttendanceScreen';

import TimeTableScreen from './screens/TimeTableScreen';

import SubjectsScreen from './screens/SubjectsScreen';

import TasksScreen from './screens/TasksScreen';

import ConcentrateScreen from './screens/ConcentrateScreen';

import ProfileScreen from './screens/ProfileScreen';

const Tab = createBottomTabNavigator();

const MarkAttendanceStack = createStackNavigator();

const TimeTableStack = createStackNavigator();

const SubjectsStack = createStackNavigator();

const TasksStack = createStackNavigator();

const ConcentrateStack = createStackNavigator();

const ProfileStack = createStackNavigator();

function MarkAttendanceStackScreen() {

  return (

    <MarkAttendanceStack.Navigator>

      <MarkAttendanceStack.Screen name="Mark Attendance"

component={MarkAttendanceScreen} />

    </MarkAttendanceStack.Navigator>

  );

}

function TimeTableStackScreen() {

  return (

    <TimeTableStack.Navigator>

      <TimeTableStack.Screen name="Time Table" component={TimeTableScreen} />

    </TimeTableStack.Navigator>

  );

}

function SubjectsStackScreen() {

  return (

    <SubjectsStack.Navigator>

      <SubjectsStack.Screen name="Subjects" component={SubjectsScreen} />

    </SubjectsStack.Navigator>

  );

}

function TasksStackScreen() {

  return (

    <TasksStack.Navigator>

      <TasksStack.Screen name="Tasks" component={TasksScreen} />

    </TasksStack.Navigator>

  );

}

function ConcentrateStackScreen() {

  return (

    <ConcentrateStack.Navigator>

      <ConcentrateStack.Screen name="Concentration Music"

component={ConcentrateScreen} />

    </ConcentrateStack.Navigator>

  );

}

function ProfileStackScreen() {

  return (

    <ProfileStack.Navigator>

      <ProfileStack.Screen name="Profile" component={ProfileScreen} />

    </ProfileStack.Navigator>

  );

}

export default function App() {

  useEffect(() => {

    SplashScreen.hide();

  }, []);

  return (

    <NavigationContainer>

      <Tab.Navigator

        initialRouteName="Profile"

        screenOptions={({ route }) => ({

          tabBarIcon: ({ color }) => {

            if (route.name === 'Attendance') {

              return (<FontAwesomeIcon icon={faBookmark} size={22}

color={color} />);

            } else if (route.name === 'Subjects') {

              return <FontAwesomeIcon icon={faFileAlt} size={22}

color={color} />;

            } else if (route.name === 'Time Table') {

              return (<FontAwesomeIcon icon={faNewspaper} size={33}

color={color} />);

            } else if (route.name === 'Tasks') {

              return <FontAwesomeIcon icon={faCheck} size={22}

color={color} />;

            } else if (route.name === 'Concentrate') {

              return <FontAwesomeIcon icon={faHeadphones} size={22}

color={color} />;

            } else if (route.name === 'Profile') {

              return <FontAwesomeIcon icon={faUser} size={22}

color={color} />;

            }

          },

        })}

        tabBarOptions={{ activeTintColor: '#24a0ed', inactiveTintColor: 'gray' }}

      >

        <Tab.Screen name="Attendance" component={MarkAttendanceStackScreen} />

        <Tab.Screen name="Subjects" component={SubjectsStackScreen} />

        <Tab.Screen name="Time Table" component={TimeTableStackScreen} />

        <Tab.Screen name="Tasks" component={TasksStackScreen} />

        <Tab.Screen name="Concentrate" component={ConcentrateStackScreen} />

        <Tab.Screen name="Profile" component={ProfileStackScreen} />

      </Tab.Navigator>

    </NavigationContainer>

  );

}

**6.3 MarkAttendanceScreen.js**

import React, { Component } from 'react';

import { View, Text, Button, StyleSheet, FlatList, ScrollView, RefreshControl, TouchableOpacity, Alert } from 'react-native';

import AsyncStorage from '@react-native-community/async-storage';

function fetchData(timeout) {

  return new Promise((resolve) => {

    setTimeout(resolve, timeout);

  });

}

export default class MarkAttendanceScreen extends Component {

  \_isMounted = false;

  constructor(props) {

    super(props);

    this.state = {

      subjects: [],

      text: '',

      present\_count: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

      total\_count: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

      present: 0,

      total: 0,

      refreshing: false,

    };

  }

  present = (i) => {

    let present\_count = [...this.state.present\_count];

    present\_count[i]++;

    let total\_count = [...this.state.total\_count];

    total\_count[i]++;

    let present = this.state.present;

    present++;

    let total = this.state.total;

    total++;

    this.setState({ present\_count, total\_count, present, total });

    AsyncStorage.setItem('PRESENT\_COUNT', JSON.stringify(present\_count));

    AsyncStorage.setItem('TOTAL\_COUNT', JSON.stringify(total\_count));

    AsyncStorage.setItem('PRESENT', JSON.stringify(present));

    AsyncStorage.setItem('TOTAL', JSON.stringify(total));

  };

  total = (i) => {

    let total\_count = [...this.state.total\_count];

    total\_count[i]++;

    let total = this.state.total;

    total++;

    this.setState({ total\_count, total });

    AsyncStorage.setItem('TOTAL\_COUNT', JSON.stringify(total\_count));

    AsyncStorage.setItem('TOTAL', JSON.stringify(total));

  };

  resetvalues = () => {

    Alert.alert(

      'Reset attendance',

      'Are you sure you want to reset your attendance?',

      [

        { text: 'NO', style: 'cancel' },

        {

          text: 'YES', onPress: () => {

            {

              AsyncStorage.setItem('PRESENT\_COUNT',

JSON.stringify([0, 0, 0, 0, 0, 0, 0, 0, 0, 0]));

              AsyncStorage.setItem('TOTAL\_COUNT',

JSON.stringify([0, 0, 0, 0, 0, 0, 0, 0, 0, 0]));

              AsyncStorage.setItem('PRESENT', JSON.stringify(0));

              AsyncStorage.setItem('TOTAL', JSON.stringify(0));

            }

            this.\_onRefresh();

          }

        },

      ]

    );

  };

  \_onRefresh = () => {

    this.setState({ refreshing: true }, () => {

      this.componentDidMount();

    });

    fetchData().then(() => {

      this.setState({ refreshing: false });

    });

  };

  componentDidMount() {

    this.\_isMounted = true;

    this.setState({ refreshing: false });

    Subjects.all((subjects) => this.setState({ subjects: subjects || [] }));

    AsyncStorage.getItem('PRESENT\_COUNT').then((value) => {

      if (value) { this.setState({ present\_count:

JSON.parse(value || this.state.present\_count), }); }

    });

    AsyncStorage.getItem('TOTAL\_COUNT').then((value) => {

      if (value) { this.setState({ total\_count:

JSON.parse(value || this.state.total\_count), }); }

    });

    AsyncStorage.getItem('PRESENT').then((value) => {

      if (value) { this.setState({ present:

JSON.parse(value || this.state.present) }); }

    });

    AsyncStorage.getItem('TOTAL').then((value) => {

      if (value) { this.setState({ total:

JSON.parse(value || this.state.total) }); }

    });

  }

  componentWillUnmount() {

    this.\_isMounted = false;

  }

  render() {

    let tick = '\u2713', cross = '\u2573';

    var date = new Date().getDate();

    var month = new Date().getMonth();

    const monthNames = ["January", "February", "March", "April", "May",

"June", "July", "August", "September", "October", "November", "December"];

    var year = new Date().getFullYear();

    return (

      <View style={styles.container}>

        <>

          {

            this.state.subjects != ''

              ?

              (

                <>

                  <View style={styles.dateContainer}>

                    <TouchableOpacity onPress={this.\_onRefresh.bind(this)}>

                      <Text style={{ fontSize: 15, color: "#fff",

paddingLeft: 15 }}>Refresh</Text>

                    </TouchableOpacity>

                    <View style={{ flex: 1, alignItems: "flex-end" }}>

                      <Text style={styles.date}>{date}</Text>

                    </View>

                    <View style={{ flex: 0.05 }}>

                    </View>

                    <View style={{ flex: 1, alignItems: "flex-start" }}>

                      <Text style={{ fontSize: 20, color: "#fff", }}>

{monthNames[month]}{"\n"}{year}</Text>

                    </View>

                    <TouchableOpacity onPress={this.resetvalues.bind(this)}>

                      <Text

style={{ fontSize: 15, color: "#fff", paddingRight: 15 }}>Reset</Text>

                    </TouchableOpacity>

                  </View>

                  <FlatList

                    style={styles.list}

                    contentContainerStyle={{ paddingBottom: 20 }}

                    data={this.state.subjects}

                    renderItem={({ item, index }) => {

                      return (

                        <View>

                          <View style={styles.listItemCont}>

                            <Text style={styles.listItem}> {item.text} </Text>

                            <View style={styles.buttonContainer}>

                              <Text style={styles.listItem}>

                                {this.state.present\_count[index]} {"/"}

{this.state.total\_count[index]}{' '}

                              </Text>

                              <View style={styles.button}>

                                <Button

                                  title={tick}

                                  onPress={() => this.present(index)}

                                  color="limegreen"

                                />

                              </View>

                              <View style={styles.button}>

                                <Button

                                  title={cross}

                                  onPress={() => this.total(index)}

                                  color="red"

                                />

                              </View>

                            </View>

                          </View>

                        </View>

                      );

                    }}

                    keyExtractor={(item, index) => index.toString()}

                    refreshing={this.state.refreshing}

                    onRefresh={this.\_onRefresh}

                  />

                </>

              ) : (

                <ScrollView

                  showsVerticalScrollIndicator={false}

                  refreshControl={

                    <RefreshControl

                      refreshing={this.state.refreshing}

                      onRefresh={this.\_onRefresh}

                    />

                  }>

                  <Text style={{ textAlign: 'center', marginTop: 250 }}>

                    Go to the Subjects tab & add your subjects first.{"\n"}

Then come to this tab and pull to refresh.

                  </Text>

                </ScrollView>

              )

          }

        </>

      </View >

    );

  }

}

let Subjects = {

  convertToArrayOfObject(subjects, callback) {

    return callback(

      subjects

        ? subjects.split('\n').map((subject, i) => ({ key: i, text: subject }))

        : [],

    );

  },

  convertToStringWithSeparators(subjects) {

    return subjects.map((subject) => subject.text).join('\n');

  },

  all(callback) {

    return AsyncStorage.getItem('SUBJECTS', (err, subjects) =>

      this.convertToArrayOfObject(subjects, callback),

    );

  },

};

const styles = StyleSheet.create({

  container: {

    flex: 1,

    justifyContent: 'center',

    alignItems: 'center',

    backgroundColor: 'white',

  },

  dateContainer: {

    backgroundColor: "#24a0ed",

    width: "100%",

    paddingTop: 10,

    paddingBottom: 10,

    flexDirection: "row",

    alignItems: "center",

  },

  date: {

    textAlign: 'center',

    fontSize: 22,

    fontWeight: "bold",

    padding: 12,

    color: "#fff",

    borderWidth: 2,

    borderColor: "#fff",

    borderRadius: 50,

  },

  list: {

    flex: 1,

    width: '95%',

  },

  listItemCont: {

    marginRight: 5,

    marginLeft: 5,

    flexDirection: 'row',

    alignItems: 'center',

    backgroundColor: 'white',

    borderRadius: 10,

    marginTop: 15,

    shadowColor: '#000',

    shadowOffset: { width: 0, height: 2 },

    shadowOpacity: 0.25,

    shadowRadius: 4,

    elevation: 5,

  },

  listItem: {

    paddingTop: 15,

    paddingBottom: 15,

    paddingLeft: 5,

    fontSize: 20,

  },

  buttonContainer: {

    flex: 1,

    flexDirection: 'row',

    justifyContent: 'flex-end',

    marginRight: 10,

  },

  button: {

    paddingTop: 10,

    paddingLeft: 10,

    width: 50,

    height: 20,

  },

});

**6.4 SubjectsScreen.js**

import React, { Component } from 'react';

import { View, Text, Button, TextInput, StyleSheet, FlatList, Keyboard,

Platform } from 'react-native';

import AsyncStorage from '@react-native-community/async-storage';

export default class SubjectsScreen extends Component {

  state = {

    subjects: [],

    text: '',

  };

  changeTextHandler = (text) => {

    this.setState({ text: text });

  };

  addSubject = () => {

    let notEmpty = this.state.text.trim().length > 0;

    if (notEmpty) {

      this.setState(

        (prevState) => {

          let { subjects, text } = prevState;

          return {

            subjects: subjects.concat({ key: subjects.length, text: text }),

            text: '',

          };

        },

        () => Subjects.save(this.state.subjects),

      );

    }

  };

  deleteSubject = (i) => {

    this.setState(

      (prevState) => {

        let subjects = prevState.subjects.slice();

        subjects.splice(i, 1);

        return { subjects: subjects };

      },

      () => Subjects.save(this.state.subjects),

    );

  };

  componentDidMount() {

    Keyboard.addListener(

      isAndroid ? 'keyboardDidShow' : 'keyboardWillShow',

      (e) => this.setState({

viewPadding: e.endCoordinates.height + viewPadding }),

    );

    Keyboard.addListener(

      isAndroid ? 'keyboardDidHide' : 'keyboardWillHide',

      () => this.setState({ viewPadding: viewPadding }),

    );

    Subjects.all((subjects) => this.setState({ subjects: subjects || [] }));

  }

  render() {

    let cross = '\u2573';

    return (

      <View style={styles.container}>

        <FlatList

          contentContainerStyle={{ paddingBottom: 20 }}

          style={styles.list}

          data={this.state.subjects}

          renderItem={({ item, index }) => (

            <View>

              <View style={styles.listItemCont}>

                <Text style={[styles.listItem, { paddingLeft: 5 }]}>

                  {' '}{item.text}{' '}

                </Text>

                <View style={styles.buttonContainer}>

                  <View style={styles.button}>

                    <Button

                      title={cross}

                      onPress={() => this.deleteSubject(index)}

                      color="#24a0ed"

                    />

                  </View>

                </View>

              </View>

            </View>

          )}

          keyExtractor={(item, index) => index.toString()}

        />

        <TextInput

          style={styles.textInput}

          textAlign={'center'}

          onChangeText={this.changeTextHandler}

          onSubmitEditing={this.addSubject}

          value={this.state.text}

          placeholder="Add Subject"

          returnKeyType="done"

          returnKeyLabel="done"></TextInput>

      </View>

    );

  }

}

let Subjects = {

  convertToArrayOfObject(subjects, callback) {

    return callback(

      subjects

        ? subjects.split('\n').map((subject, i) => ({ key: i, text: subject }))

        : [],

    );

  },

  convertToStringWithSeparators(subjects) {

    return subjects.map((subject) => subject.text).join('\n');

  },

  all(callback) {

    return AsyncStorage.getItem('SUBJECTS', (err, subjects) =>

      this.convertToArrayOfObject(subjects, callback),

    );

  },

  save(subjects) {

    AsyncStorage.setItem(

      'SUBJECTS',

      this.convertToStringWithSeparators(subjects),

    );

  },

};

const isAndroid = Platform.OS == 'android';

const viewPadding = 10;

const styles = StyleSheet.create({

  container: {

    flex: 1,

    justifyContent: 'center',

    alignItems: 'center',

    backgroundColor: 'white',

  },

  list: {

    flex: 1,

    width: '95%',

  },

  listItemCont: {

    marginRight: 5,

    marginLeft: 5,

    flexDirection: 'row',

    alignItems: 'center',

    backgroundColor: 'white',

    borderRadius: 10,

    marginTop: 15,

    shadowColor: '#000',

    shadowOffset: { width: 0, height: 2 },

    shadowOpacity: 0.25,

    shadowRadius: 4,

    elevation: 5,

  },

  listItem: {

    paddingTop: 15,

    paddingBottom: 15,

    paddingLeft: 5,

    fontSize: 20,

  },

  buttonContainer: {

    flex: 1,

    flexDirection: 'row',

    justifyContent: 'flex-end',

    marginRight: 10,

  },

  button: {

    width: 40,

  },

  textInput: {

    fontSize: 25,

    height: 50,

    width: '90%',

    paddingRight: 10,

    paddingLeft: 10,

    borderColor: 'gray',

    borderWidth: isAndroid ? 0 : 1,

    justifyContent: 'flex-end',

  },

});

**6.5 TimeTableScreen.js**

import React, { Component } from 'react';

import { View, Button, StyleSheet, ScrollView, RefreshControl,

TouchableOpacity, Text } from 'react-native';

import AsyncStorage from '@react-native-community/async-storage';

import ImagePicker from 'react-native-image-picker';

import PhotoView from 'react-native-photo-view-ex';

function fetchData(timeout) {

  return new Promise((resolve) => {

    setTimeout(resolve, timeout);

  });

}

export default class TimeTableScreen extends Component {

  constructor() {

    super();

    this.state = { filePath: 'val' };

  }

  \_onRefresh = () => {

    this.setState({ refreshing: true }, () => { this.componentDidMount(); });

    fetchData().then(() => {

      this.setState({ refreshing: false });

    });

  };

  chooseFile = () => {

    var options = {

      title: 'Select Image',

      quality: 1.0,

      storageOptions: { skipBackup: true, path: 'images' },

    };

    ImagePicker.showImagePicker(options, (response) => {

      console.log('Response = ', response);

      if (response.didCancel) { console.log('User cancelled photo picker'); }

      else if (response.error) {

console.log('ImagePicker Error: ', response.error); }

      else {

        let source = response;

        this.setState({ filePath: source.uri, });

        AsyncStorage.setItem('Image\_id\_1', this.state.filePath);

      }

    });

  };

  componentDidMount() {

    AsyncStorage.getItem('Image\_id\_1').then((value) => {

      this.setState({ filePath: value });

    });

  }

  render() {

    var day = new Date().getDay();

    const dayNames = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];

    var hour = new Date().getHours();

    var minute = new Date().getMinutes();

    var timezone = new Date().toString().match(/\(([A-Za-z\s].\*)\)/)[1];

    var parts = timezone.split(' '); var tz = "";

parts.forEach(function (element) { tz += element.substring(0, 1); });

    return (

      <>

        {

          this.state.filePath

            ? (

              <ScrollView

                contentContainerStyle={styles.container}

                refreshControl={

                  <RefreshControl

                    refreshing={this.state.refreshing}

                    onRefresh={this.\_onRefresh}

                  />

                }

              >

                <View style={styles.timeContainer}>

                  <TouchableOpacity onPress={this.\_onRefresh.bind(this)}>

                    <Text

style={{ fontSize: 15, color: "#fff", paddingLeft: 15 }}>Refresh</Text>

                  </TouchableOpacity>

                  <View style={{ flex: 1, alignItems: "center" }}>

                    <Text style={styles.day}>{dayNames[day]}</Text>

                    <Text

style={styles.time}>{hour}{" : "}{minute}{" "}{timezone}</Text>

                  </View>

                  <TouchableOpacity onPress={this.chooseFile.bind(this)}>

                    <Text

style={{ fontSize: 15, color: "#fff", paddingRight: 15 }}>Update</Text>

                  </TouchableOpacity>

                </View>

                <PhotoView style={styles.image}

source={{ uri: this.state.filePath }} minimumZoomScale={1}

maximumZoomScale={2} />

              </ScrollView>

            ) : (

              <View style={styles.container} >

                <Button

                  title="Choose Time Table Photo"

                  onPress={this.chooseFile.bind(this)}

                  color="#24a0ed"

                />

              </View>

            )

        }

      </>

    );

  }

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    justifyContent: 'center',

    alignItems: 'center',

  },

  timeContainer: {

    backgroundColor: "#24a0ed",

    width: "100%",

    paddingTop: 10,

    paddingBottom: 10,

    flexDirection: "row",

    alignItems: "center",

  },

  day: {

    textAlign: 'center',

    fontSize: 22,

    fontWeight: "bold",

    color: "#fff",

  },

  time: {

    textAlign: 'center',

    fontSize: 18,

    fontWeight: "bold",

    color: "#fff",

  },

  image: {

    flex: 1,

    width: '100%',

    height: '100%',

  },

});

**6.6 TasksScreen.js**

import React, { Component } from 'react';

import { View, Text, Button, TextInput, StyleSheet, FlatList, Keyboard,

Platform } from 'react-native';

import AsyncStorage from '@react-native-community/async-storage';

export default class TasksScreen extends Component {

  state = {

    tasks: [],

    text: '',

  };

  changeTextHandler = (text) => {

    this.setState({ text: text });

  };

  addTask = () => {

    let notEmpty = this.state.text.trim().length > 0;

    if (notEmpty) {

      this.setState(

        (prevState) => {

          let { tasks, text } = prevState;

          return {

            tasks: tasks.concat({ key: tasks.length, text: text }),

            text: '',

          };

        },

        () => Tasks.save(this.state.tasks),

      );

    }

  };

  deleteTask = (i) => {

    this.setState(

      (prevState) => {

        let tasks = prevState.tasks.slice();

        tasks.splice(i, 1);

        return { tasks: tasks };

      },

      () => Tasks.save(this.state.tasks),

    );

  };

  componentDidMount() {

    Keyboard.addListener(

      isAndroid ? 'keyboardDidShow' : 'keyboardWillShow',

      (e) =>

        this.setState({ viewPadding: e.endCoordinates.height + viewPadding }),

    );

    Keyboard.addListener(

      isAndroid ? 'keyboardDidHide' : 'keyboardWillHide',

      () => this.setState({ viewPadding: viewPadding }),

    );

    Tasks.all((tasks) => this.setState({ tasks: tasks || [] }));

  }

  render() {

    let tick = '\u2713';

    return (

      <View style={styles.container}>

        <FlatList

          contentContainerStyle={{ paddingBottom: 20 }}

          style={styles.list}

          data={this.state.tasks}

          renderItem={({ item, index }) => (

            <View>

              <View style={styles.listItemCont}>

                <Text style={[styles.listItem, { paddingLeft: 5 }]}>

                  {' '}

                  {item.text}{' '}

                </Text>

                <View style={styles.buttonContainer}>

                  <View style={styles.button}>

                    <Button

                      title={tick}

                      onPress={() => this.deleteTask(index)}

                      color="#24a0ed"

                    />

                  </View>

                </View>

              </View>

            </View>

          )}

          keyExtractor={(item, index) => index.toString()}

        />

        <TextInput

          style={styles.textInput}

          textAlign={'center'}

          onChangeText={this.changeTextHandler}

          onSubmitEditing={this.addTask}

          value={this.state.text}

          placeholder="Add Task"

          returnKeyType="done"

          returnKeyLabel="done"

        />

      </View>

    );

  }

}

let Tasks = {

  convertToArrayOfObject(tasks, callback) {

    return callback(

      tasks ? tasks.split('||').map((task, i) => ({ key: i, text: task }))

: [],

    );

  },

  convertToStringWithSeparators(tasks) {

    return tasks.map((task) => task.text).join('||');

  },

  all(callback) {

    return AsyncStorage.getItem('TASKS', (err, tasks) =>

      this.convertToArrayOfObject(tasks, callback),

    );

  },

  save(tasks) {

    AsyncStorage.setItem('TASKS', this.convertToStringWithSeparators(tasks));

  },

};

const isAndroid = Platform.OS == 'android';

const viewPadding = 10;

const styles = StyleSheet.create({

  container: {

    flex: 1,

    justifyContent: 'center',

    alignItems: 'center',

    backgroundColor: 'white',

  },

  list: {

    flex: 1,

    width: '95%',

  },

  listItemCont: {

    marginRight: 5,

    marginLeft: 5,

    flexDirection: 'row',

    alignItems: 'center',

    backgroundColor: 'white',

    borderRadius: 10,

    marginTop: 15,

    shadowColor: '#000',

    shadowOffset: { width: 0, height: 2 },

    shadowOpacity: 0.25,

    shadowRadius: 4,

    elevation: 5,

  },

  listItem: {

    paddingTop: 15,

    paddingBottom: 15,

    paddingLeft: 5,

    fontSize: 20,

  },

  buttonContainer: {

    flex: 1,

    flexDirection: 'row',

    justifyContent: 'flex-end',

    marginRight: 10,

  },

  button: {

    width: 40,

  },

  textInput: {

    fontSize: 25,

    height: 50,

    width: '90%',

    paddingRight: 10,

    paddingLeft: 10,

    borderColor: 'gray',

    borderWidth: isAndroid ? 0 : 1,

    justifyContent: 'flex-end',

  },

});

**6.7 ConcentrateScreen.js**

import React, { Component } from 'react';

import Player from './music\_player\_components/Player';

export const TRACKS = [

  {

    title: 'Concentration Music 1',

    artist: 'Gravity Music',

    albumArtUrl: "https://i.imgur.com/bDutmFm.jpg",

    audioUrl: "https://www.mboxdrive.com/concentration\_music\_01.mp3",

  },

  {

    title: 'Concentration Music 2',

    artist: 'Zen Meditation Planet',

    albumArtUrl: "https://i.imgur.com/zx8UNDk.jpg",

    audioUrl: 'https://www.mboxdrive.com/concentration\_music\_2.mp3',

  },

  {

    title: 'Concentration Music 3',

    artist: 'ChilledCow',

    albumArtUrl: 'https://i.imgur.com/Vrysh1E.jpg',

    audioUrl: 'https://www.mboxdrive.com/concentration\_music\_3.mp3',

  },

];

export default class ConcentrateScreen extends Component {

  render() {

    return <Player tracks={TRACKS} />

  }

}

**6.7.1 Player.js**

import React, { Component } from 'react';

import { View, StatusBar } from 'react-native';

import AlbumArt from './AlbumArt';

import TrackDetails from './TrackDetails';

import SeekBar from './SeekBar';

import Controls from './Controls';

import Video from 'react-native-video';

export default class Player extends Component {

  constructor(props) {

    super(props);

    this.state = {

      paused: true,

      totalLength: 1,

      currentPosition: 0,

      selectedTrack: 0,

    };

  }

  setDuration(data) {

this.setState({ totalLength: Math.floor(data.duration) }); }

  setTime(data) {

this.setState({ currentPosition: Math.floor(data.currentTime) }); }

  seek(time) {

    time = Math.round(time);

    this.refs.audioElement && this.refs.audioElement.seek(time);

    this.setState({ currentPosition: time, paused: false, });

  }

  onBack() {

    if (this.state.currentPosition < 10 && this.state.selectedTrack > 0) {

      this.refs.audioElement && this.refs.audioElement.seek(0);

      this.setState({ isChanging: true });

      setTimeout(() => this.setState({

        currentPosition: 0,

        paused: false,

        totalLength: 1,

        isChanging: false,

        selectedTrack: this.state.selectedTrack - 1,

      }), 0);

    } else {

      this.refs.audioElement.seek(0);

      this.setState({

        currentPosition: 0,

      });

    }

  }

  onForward() {

    if (this.state.selectedTrack < this.props.tracks.length - 1) {

      this.refs.audioElement && this.refs.audioElement.seek(0);

      this.setState({ isChanging: true });

      setTimeout(() => this.setState({

        currentPosition: 0,

        totalLength: 1,

        paused: false,

        isChanging: false,

        selectedTrack: this.state.selectedTrack + 1,

      }), 0);

    }

  }

  render() {

    const track = this.props.tracks[this.state.selectedTrack];

    const video = this.state.isChanging ? null : (

      <Video source={{ uri: track.audioUrl }}

        ref="audioElement"

        paused={this.state.paused}

        resizeMode="cover"

        onLoadStart={this.loadStart}

        onLoad={this.setDuration.bind(this)}

        onProgress={this.setTime.bind(this)}

        onEnd={this.onEnd}

        onError={this.videoError}

        style={styles.audioElement}

      />

    );

    return (

      <View style={styles.container}>

        <StatusBar hidden={true} />

        <AlbumArt url={track.albumArtUrl} />

        <TrackDetails title={track.title} artist={track.artist} />

        <SeekBar

          onSeek={this.seek.bind(this)}

          trackLength={this.state.totalLength}

          onSlidingStart={() => this.setState({ paused: true })}

          currentPosition={this.state.currentPosition}

        />

        <Controls

          forwardDisabled=

{this.state.selectedTrack === this.props.tracks.length - 1}

          onPressPlay={() => this.setState({ paused: false })}

          onPressPause={() => this.setState({ paused: true })}

          onBack={this.onBack.bind(this)}

          onForward={this.onForward.bind(this)}

          paused={this.state.paused}

        />

        {video}

      </View>

    );

  }

}

const styles = {

  container: {

    flex: 1,

    backgroundColor: '#24a0ed',

  },

  audioElement: {

    height: 0,

    width: 0,

  }

};

**6.7.2 AlbumArt.js**

import React from 'react';

import { View, StyleSheet, Image, TouchableOpacity,

Dimensions } from 'react-native';

const AlbumArt = ({ url, onPress }) => (

  <View style={styles.container}>

    <TouchableOpacity onPress={onPress}>

      <Image

        style={styles.image}

        source={{ uri: url }}

      />

    </TouchableOpacity>

  </View>

);

export default AlbumArt;

const { width } = Dimensions.get('window');

const imageSize = width - 48;

const styles = StyleSheet.create({

  container: {

    paddingLeft: 24,

    paddingRight: 24,

  },

  image: {

    marginTop: 20,

    borderRadius: 10,

    width: imageSize,

    height: imageSize,

  },

})

**6.7.3 TrackDetails.js**

import React from 'react';

import { View, Text, StyleSheet } from 'react-native';

const TrackDetails = ({ title, artist, onTitlePress, onArtistPress }) => (

  <View style={styles.container}>

    <View style={styles.detailsWrapper}>

      <Text style={styles.title} onPress={onTitlePress}>{title}</Text>

      <Text style={styles.artist} onPress={onArtistPress}>{artist}</Text>

    </View>

  </View>

);

export default TrackDetails;

const styles = StyleSheet.create({

  container: {

    paddingTop: 24,

    flexDirection: 'row',

    paddingLeft: 20,

    alignItems: 'center',

    paddingRight: 20,

  },

  detailsWrapper: {

    justifyContent: 'center',

    alignItems: 'center',

    flex: 1,

  },

  title: {

    fontSize: 16,

    fontWeight: 'bold',

    color: 'black',

    textAlign: 'center',

  },

  artist: {

    color: 'black',

    fontSize: 12,

    marginTop: 4,

  },

});

**6.7.4 SeekBar.js**

import React from 'react';

import { View, Text, StyleSheet } from 'react-native';

import Slider from '@react-native-community/slider';

function pad(n, width, z = 0) {

  n = n + '';

  return n.length >= width ? n : new Array(width - n.length + 1).join(z) + n;

}

const minutesAndSeconds = (position) => ([

  pad(Math.floor(position / 60), 2),

  pad(position % 60, 2),

]);

const SeekBar = ({ trackLength, currentPosition, onSeek, onSlidingStart }) => {

  const elapsed = minutesAndSeconds(currentPosition);

  const remaining = minutesAndSeconds(trackLength - currentPosition);

  return (

    <View style={styles.container}>

      <View style={{ flexDirection: 'row' }}>

        <Text style={styles.text}>

          {elapsed[0] + ":" + elapsed[1]}

        </Text>

        <View style={{ flex: 1 }} />

        <Text style={[styles.text, { width: 40 }]}>

          {(trackLength > 1) ?

"-" + remaining[0] + ":" + remaining[1] : "loading"}

        </Text>

      </View>

      <Slider

        maximumValue={Math.max(trackLength, 1, currentPosition + 1)}

        onSlidingStart={onSlidingStart}

        onSlidingComplete={onSeek}

        value={currentPosition}

        style={styles.slider}

        minimumTrackTintColor='#fff'

        maximumTrackTintColor='#fff'

        thumbStyle={styles.thumb}

        trackStyle={styles.track}

      />

    </View>

  );

};

export default SeekBar;

const styles = StyleSheet.create({

  slider: {

    marginTop: 10,

  },

  container: {

    paddingLeft: 16,

    paddingRight: 16,

    paddingTop: 16,

  },

  track: {

    height: 2,

    borderRadius: 1,

  },

  thumb: {

    width: 10,

    height: 10,

    borderRadius: 5,

    backgroundColor: 'black',

  },

  text: {

    color: 'black',

    fontSize: 12,

    textAlign: 'center',

  }

});

**6.7.5 Controls.js**

import React from 'react';

import { View, StyleSheet, Image, TouchableOpacity } from 'react-native';

const Controls = ({ paused, onPressPlay, onPressPause, onBack, onForward,

forwardDisabled }) => (

  <View style={styles.container}>

    <View style={{ width: 40 }} />

    <TouchableOpacity onPress={onBack}>

      <Image source=

{require('../music\_player\_images/ic\_skip\_previous\_white\_36pt.png')} />

    </TouchableOpacity>

    <View style={{ width: 20 }} />

    {!paused ?

      <TouchableOpacity onPress={onPressPause}>

        <View style={styles.playButton}>

          <Image source=

{require('../music\_player\_images/ic\_pause\_white\_48pt.png')} />

        </View>

      </TouchableOpacity> :

      <TouchableOpacity onPress={onPressPlay}>

        <View style={styles.playButton}>

          <Image source=

{require('../music\_player\_images/ic\_play\_arrow\_white\_48pt.png')} />

        </View>

      </TouchableOpacity>

    }

    <View style={{ width: 20 }} />

    <TouchableOpacity onPress={onForward}

      disabled={forwardDisabled}>

      <Image style={[forwardDisabled && { opacity: 0.3 }]}

        source=

{require('../music\_player\_images/ic\_skip\_next\_white\_36pt.png')} />

    </TouchableOpacity>

    <View style={{ width: 40 }} />

  </View>

);

export default Controls;

const styles = StyleSheet.create({

  container: {

    flexDirection: 'row',

    alignItems: 'center',

    justifyContent: 'center',

    paddingTop: 8,

  },

  playButton: {

    height: 72,

    width: 72,

    borderWidth: 1,

    borderColor: 'white',

    borderRadius: 72 / 2,

    alignItems: 'center',

    justifyContent: 'center',

  }

})

**6.8 ProfileScreen.js**

import React, { Component } from 'react';

import { StyleSheet, Text, View, Image, TextInput, TouchableOpacity,

ScrollView, RefreshControl } from 'react-native';

import AsyncStorage from '@react-native-community/async-storage';

import ImagePicker from 'react-native-image-picker';

function fetchData(timeout) {

  return new Promise((resolve) => {

    setTimeout(resolve, timeout);

  });

}

export default class Profile extends Component {

  constructor() {

    super();

    this.state = {

      present: 0,

      total: 0,

      dpPath: 'path',

      username: 'username',

      bio: 'bio',

      refreshing: false,

    };

  };

  \_onRefresh = () => {

    this.setState({ refreshing: true }, () => {

      this.componentDidMount();

    });

    fetchData().then(() => {

      this.setState({ refreshing: false });

    });

  };

  saveUsername = () => {

AsyncStorage.setItem('USERNAME', this.state.username); };

  saveBio = () => { AsyncStorage.setItem('BIO', this.state.bio); };

  chooseFile = () => {

    var options = {

      title: 'Select Profile Photo',

      quality: 1.0,

      storageOptions: { skipBackup: true, path: 'images', },

    };

    ImagePicker.showImagePicker(options, (response) => {

      console.log('Response = ', response);

      if (response.didCancel) {

        console.log('User cancelled photo picker');

      } else if (response.error) {

        console.log('ImagePicker Error: ', response.error);

      } else {

        let source = response;

        this.setState({ dpPath: source.uri, });

        AsyncStorage.setItem('Image\_id\_2', this.state.dpPath);

      }

    });

  };

  componentDidMount() {

    AsyncStorage.getItem('Image\_id\_2').then((value) => {

      this.setState({ dpPath: value });

    });

    AsyncStorage.getItem('USERNAME').then((value) => {

      if (value) { this.setState({

username: (value || this.state.username).toString() }); }

    });

    AsyncStorage.getItem('BIO').then((value) => {

      if (value) { this.setState({

bio: (value || this.state.bio).toString() }); }

    });

    AsyncStorage.getItem('PRESENT').then((value) => {

      if (value) { this.setState({

present: JSON.parse(value || this.state.present) }); }

    });

    AsyncStorage.getItem('TOTAL').then((value) => {

      if (value) { this.setState({

total: JSON.parse(value || this.state.total) }); }

    });

  };

  render() {

    var overall\_attendance =

((this.state.present / this.state.total) \* 100).toFixed(2);

    return (

      <>

        <ScrollView

          refreshControl={

            <RefreshControl

              refreshing={this.state.refreshing}

              onRefresh={this.\_onRefresh}

            />

          }>

          <View style={styles.header}></View>

          <>

            {this.state.dpPath ? (

              <Image source={{ uri: this.state.dpPath }}

style={styles.avatar} />

            ) : (

                <Image style={styles.avatar}

source={require('./avatar.png')} />

              )}

          </>

          <View style={styles.body}>

            <View style={styles.bodyContent}>

              <Text style={styles.name}>{this.state.username}</Text>

              <Text style={styles.bio}>{this.state.bio}</Text>

              <View style={styles.buttonContainer}>

                <TouchableOpacity

                  style={styles.button}

                  onPress={this.chooseFile.bind(this)}>

                  <Text style={{ color: 'grey', padding: 5 }}>

Update Photo</Text>

                </TouchableOpacity>

                <TouchableOpacity style={styles.button}>

                  <TextInput

                    style={{ padding: 5, color: "grey" }}

                    alignItems="center"

                    placeholder="Update Username"

                    returnKeyLabel={'next'}

                    onChangeText={(text) => this.setState({ username: text })}

                    onSubmitEditing={this.saveUsername}

                  />

                </TouchableOpacity>

                <TouchableOpacity style={styles.button}>

                  <TextInput

                    style={{ padding: 5, color: "grey" }}

                    alignItems="center"

                    placeholder="Update Bio"

                    returnKeyLabel={'next'}

                    onChangeText={(text) => this.setState({ bio: text })}

                    onSubmitEditing={this.saveBio}

                  />

                </TouchableOpacity>

              </View>

              <View style={styles.listItemView}>

                <Text style={styles.listItemText}>Overall attendance :

{overall\_attendance >= 0 ? overall\_attendance : 0} % </Text>

                <TouchableOpacity onPress={this.\_onRefresh.bind(this)}>

                  <Text style={styles.listItemTouch}>🔄</Text>

                </TouchableOpacity>

              </View>

              <>

                {overall\_attendance >= 60 ? (

                  <Text style={[styles.listItemCont, { color: 'limegreen' }]}>

                    Well done! Keep it up! 👍

                  </Text>

                ) : (

                    <Text style={[styles.listItemCont, { color: 'red' }]}>

                      Don't miss the next class! 🙄

                    </Text>

                  )}

              </>

            </View>

          </View>

        </ScrollView>

      </>

    );

  }

}

const styles = StyleSheet.create({

  header: {

    backgroundColor: '#24a0ed',

    height: 150,

  },

  avatar: {

    width: 130,

    height: 130,

    borderRadius: 63,

    borderWidth: 4,

    borderColor: 'white',

    marginBottom: 10,

    alignSelf: 'center',

    position: 'absolute',

    marginTop: 80,

  },

  body: {

    marginTop: 40,

  },

  bodyContent: {

    flex: 1,

    alignItems: 'center',

    padding: 30,

  },

  name: {

    fontSize: 28,

    color: '#696969',

    fontWeight: '600',

  },

  bio: {

    fontSize: 16,

    color: '#24a0ed',

    marginTop: 10,

  },

  buttonContainer: {

    marginTop: 20,

    marginBottom: 10,

    flex: 1,

    flexDirection: 'row',

    justifyContent: 'center',

    alignItems: 'center',

  },

  button: {

    marginLeft: 10,

    height: 30,

    flexDirection: 'row',

    justifyContent: 'center',

    alignItems: 'center',

    borderRadius: 5,

    backgroundColor: '#fff',

  },

  listItemView: {

    marginRight: 5,

    marginLeft: 5,

    marginBottom: 5,

    marginTop: 50,

    width: 315,

    flexDirection: 'row',

    alignItems: 'center',

    backgroundColor: 'white',

    padding: 15,

    flexDirection: "row",

    borderRadius: 5,

    marginTop: 15,

    shadowColor: '#000',

    shadowOffset: { width: 0, height: 2 },

    shadowOpacity: 0.25,

    shadowRadius: 4,

    elevation: 5,

  },

  listItemText: {

    flex: 1,

    alignItems: "flex-start",

    fontSize: 18,

    fontWeight: 'bold',

    color: '#24a0ed',

  },

  listItemTouch: {

    flex: 1,

    fontSize: 18,

    alignItems: "flex-end",

  },

  listItemCont: {

    marginRight: 5,

    marginLeft: 5,

    marginBottom: 5,

    marginTop: 50,

    width: 315,

    flexDirection: 'row',

    alignItems: 'center',

    backgroundColor: 'white',

    padding: 15,

    fontSize: 18,

    borderRadius: 5,

    marginTop: 15,

    shadowColor: '#000',

    shadowOffset: { width: 0, height: 2 },

    shadowOpacity: 0.25,

    shadowRadius: 4,

    elevation: 5,

    fontWeight: 'bold',

    color: '#24a0ed',

  },

});

**Test Plan**

A test plan documents the strategy that will be used to verify and ensure that a product or system meets its design specifications and other requirements. A test plan is usually prepared by or with significant input from test engineers.

Depending on the product and the responsibility of the organization to which the test plan applies, a test plan may include a strategy for one or more of the following:

* Design Verification or Compliance test – to be performed during the development or approval stages of the product, typically on a small sample of units.
* Manufacturing or Production test – to be performed during preparation or assembly of the product in an ongoing manner for purposes of performance verification and quality control.
* Acceptance or Commissioning test – to be performed at the time of delivery or installation of the product.
* Service and Repair test – to be performed as required over the service life of the product.
* Regression test – to be performed on an existing operational product, to verify that existing functionality was not negatively affected when other aspects of the environment were changed (e.g., upgrading the platform on which an existing application runs).

A complex system may have a high-level test plan to address the overall requirements and supporting test plans to address the design details of subsystems and components.

Test plan document formats can be as varied as the products and organizations to which they apply. There are three major elements that should be described in the test plan: Test Coverage, Test Methods, and Test Responsibilities. These are also used in a formal test strategy.

**Test coverage**

Test coverage in the test plan states what requirements will be verified during what stages of the product life. Test coverage is derived from design specifications and other requirements, such as safety standards or regulatory codes, where each requirement or specification of the design ideally will have one or more corresponding means of verification. Test coverage for different product life stages may overlap, but will not necessarily be exactly the same for all stages. For example, some requirements may be verified during Design Verification test, but not repeated during Acceptance test. Test coverage also feeds back into the design process, since the product may have to be designed to allow test access.

**Test methods**

Test methods in the test plan state how test coverage will be implemented. Test methods may be determined by standards, regulatory agencies, or contractual agreement, or may have to be created new. Test methods also specify test equipment to be used in the performance of the tests and establish pass/fail criteria. Test methods used to verify hardware design requirements can range from very simple steps, such as visual inspection, to elaborate test procedures that are documented separately.

**Test responsibilities**

Test responsibilities include what organizations will perform the test methods and at each stage of the product life. This allows test organizations to plan, acquire or develop test equipment and other resources necessary to implement the test methods for which they are responsible. Test responsibilities also include what data will be collected and how that data will be stored and reported (often referred to as "deliverables"). One outcome of a successful test plan should be a record or report of the verification of all design specifications and requirements as agreed upon by all parties.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Plan** | **Screen** | **Test Name** | **Purpose of Test** | **Test Data/ Situation** | **Expected Result** | **Outcomes and Actions Required** |
| 1 | Attendance | Rendering of subjects FlatList with buttons. | Test that screen renders as expected. | 09-10-20, 02:07  Android 7.1 | FlatList containing all the subjects with respective buttons to mark attendance. | Screen rendered as expected. No action required. |
| 2 | Subjects | Rendering of subjects FlatList. | Test that screen renders as expected. | 09-10-20, 02:07  Android 7.1 | FlatList containing all the subjects with one delete button. | Screen rendered as expected. No action required. |
| 3 | Time Table | Rendering of time table photo. | Test that screen renders as expected. | 09-10-20, 02:07  Android 7.1 | Photo retrieval from AsyncStorage. | Screen rendered as expected. No action required. |
| 4 | Tasks | Rendering of tasks FlatList. | Test that screen renders as expected. | 09-10-20, 02:07  Android 7.1 | FlatList containing all the tasks with a remove button. | Screen rendered as expected. No action required. |
| 6 | Concentrate | Rendering of concentration music player | Test that screen renders as expected. | 09-10-20, 02:07  Android 7.1 | A music player with navigation controls. | Screen rendered as expected. No action required. |
| 6 | Profile | Rendering of profile details and overall attendance. | Test that screen renders as expected. | 09-10-20, 02:07  Android 7.1 | Profile photo with username, bio and overall attendance. | Screen rendered as expected. No action required. |

**Conclusion**

**7.1 Current Status of Project:**

Report name: 09th Oct 2020, Studify v3.0

Project status: On Track.

Milestone 1: Rendering of subjects FlatList successful.

Milestone 2: Rendering of subjects FlatList with buttons successful.

Milestone 3: Rendering of time table photo successful.

Milestone 4: Rendering of tasks FlatList successful.

Milestone 5: Rendering of concentration music player successful.

Milestone 6: Rendering of profile details and overall attendance successful.

Issues/challenges: All issues have been taken care of as of now. No challenges.

**7.2 Future Scope:**

Although, this app has many features which resolve many problems of many students, still I believe there’s a scope of improvement in these features, which are listed below:

1. Users will be able to store their data in an online database (such as Firebase).

2. Users will be able to generate a backup of their data and restore it, as and when needed.

3. Data will be automatically refreshed (on navigation), no need to press the refresh button.

4. Users will be able to choose which concentration music to listen among dozens.

5. Users will be able to share their data to their socials.

**7.3 Limitations:**

While building this app, I wanted to include as many features as possible, but there were some technical restrictions and lack of knowledge, due to which they couldn’t be implemented, and are listed below:

1. Online data storage.

2. Backup & Restore.

3. Auto Refresh.

4. Variety of music.

5. Stats Sharing.

**7.4 Bibliography:**

Wikipedia: <https://en.wikipedia.org/>

ReactJS official docs: <https://reactjs.org/docs/getting-started.html>

React Native official docs: <https://reactjs.org/docs/getting-started.html>

React Navigation official docs: <https://reactnavigation.org/docs/getting-started/>