



**MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL - 576 104**

A Constituent Institution of Manipal University, Manipal-
576 104



CERTIFICATE

This is to certify that the project work of

ISHDEEP PAL SINGH BHANDARI

on the project **HANDSHAKE - CRM EVENT MANAGEMENT APP**

is a record of the bonafide work carried out in partial fulfilment of the requirements for awarding the degree of **Bachelor of Technology in Computer Science and Engineering** discipline in **Manipal Institute of Technology** under **Manipal University, Manipal** during the academic year 2016.

Mr. DASHARATHRAJ K. SHETTY
Project Guide (Internal)

Dr. RENUKA A.
Head of the Department
Dept. of Computer Science & Engg.



SAP India Pvt. Ltd.
Wing A, 2nd Floor, Tower - B
Salarpuria Softzone
Sarjapur Outer Ring Road
Bellandur Post
Bangalore 560103
T: +91-80-6665 5555
F: +91-80-6665 5550

07 January 2016

Private & Confidential

Ishdeep Bhandari
F - 004, Ambience Lagoon Apartments
Ambience Islands, NH-8, Gurgaon
Haryana 122002 India

Dear Ishdeep,

Congratulations! I am pleased to inform you that your application for internship at SAP India Pvt. Ltd. has been accepted. Your internship will be for a period of 4 months with SAP India Pvt. Ltd., starting from 25 January 2016 to 30 May 2016.

Following are the terms and conditions and scope of your services:-

1. Your stipend will be amount in Indian Rupees equivalent to 20,000.00 Rupees per month for the above mentioned period, payable monthly by an account payee cheque.
2. The Company shall withhold taxes at the applicable rates. It will be your responsibility to meet all requirements under Indian tax laws, including tax compliance and the filing of tax returns, if applicable.
3. It is also your sole responsibility to meet all requirements (including visa/permit) as required by or under any applicable laws of India.
4. The Company would take care of your accommodation and transport arrangements to office during the time of your internship with us.
5. All expenses related to your travel/visa/permit to India will have to be borne by you personally.

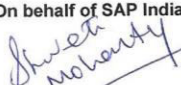
Kindly acknowledge the enclosed copy of this letter as a token of acceptance and return to us at the earliest.

We would be pleased to furnish you with any further information or clarify your queries.

Looking forward to a mutually rewarding association.

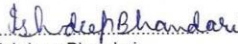
Best regards,

On behalf of SAP India Private Limited


Bhuvaneshwar Naik
Vice-President HR
SAP India Pvt Ltd


Sabish Kovath Bhaskar
HR Services Senior Consultant
HR Services Centre, SAP India Pvt Ltd

I accept the terms & conditions of service outlined above


Ishdeep Bhandari

25/01/16
Date



Project Completion Certificate

SAP India Pvt. Ltd.
6th Floor, Plot No A-2
MGF Corporate Park,
MGF Metropolitan Mall, Saket,
New Delhi-110017, India

Tel : +91 11 3090 7200 / 6603 7200
Fax : +91 11 3090 7300 / 6603 7300

www.sap.com

CIN : U72200KA1996PTC020063

Dated: 21 May 2016

This is to certify that **Mr. ISHDEEP PAL SINGH BHANDARI** (Reg. No.: 120905404) a Final Year (Eighth Semester) student of Manipal Institute of Technology, Manipal pursuing his B.Tech (CSE) has successfully completed his Four Month (16 weeks) Project in our Company.

During the above-mentioned period, from **25 January 2016 to 25 May 2016**, he worked under my guidance, as a **Software Development Trainee** and has successfully finished a project titled '**Handshake- CRM Event Management App**' which involved **developing an iOS application that would provide an outlook for the company CRM events.**

During the tenure of the project, he has been found to be sincere and hard working. His conduct was good and satisfactory. We wish him all the best for his future endeavors.



Yashdeep Bali

Integrating Marketing Senior Specialist – Marketing and Advertising

SAP India Pvt. Ltd.

ACKNOWLEDGEMENT

I take this opportunity to express my profound gratitude and deep regards to, my Internal Guide, Mr. Dasharathraj K. Shetty, Assistant Professor- Dept. of C.S.E, for his exemplary guidance, monitoring and constant encouragement throughout the course of this Project.

I also take this opportunity to express a deep sense of gratitude to, my External Guide, Mr. Yashdeep Bali, Integrating Marketing Senior Specialist, Marketing and Advertising, SAP India Pvt. Ltd., for giving me an opportunity to work in the company, for his cordial support, valuable guidance and encouragement throughout the period of this Project.

ABSTRACT

Keywords: CRM; Parse; Viewers; Owners; unified view

This project is aimed at making an app that provides easy access to internal users for managing company **CRM** events, so that all staff have a **unified view** of the internal marketing initiatives of the company, and to ensure that the marketing strategy is disseminated in an integrated, “*One SAP*” way. After installing the app, the first step is user registration. User registration data will be stored on a cloud server – the same is retrieved at the time of user login validation. There will be two types of users namely, **Viewers** and **Owners**. The **Viewers** cannot add any event into the app whereas **Owners** can. The app will have the functions for creating and editing **CRM** events, which will be described by the user. Access to the app is restricted to only internal staff of the organization (SAP). Whenever any user creates a new event he/she will add and names of the organizers of the event. All the mentioned organizers will be sent notifications to update their apps with the updated data. Every user can forward the event via mail and give feedback for any event. Every user registered on the app can sync any event with their calendars. All the data entered by the user in the app will be stored on a private cloud server (**Parse**). The app can access the **Parse** database whenever it wants to retrieve data.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	iv
ABSTRACT.....	v
LIST OF FIGURES.....	ix
ABBREVIATIONS.....	xi
CHAPTER 1 INTRODUCTION	
1.1 General.....	1
1.2 Organization.....	1
1.3 Area of Computer Science.....	2
1.4 Hardware Requirements	
1.4.1 MacBook/Mac.....	2
1.5 Software Requirements	
1.5.1 Xcode.....	2
1.5.2 OS version 10.11.3.....	2
1.5.3 Sketch.....	2
1.5.4 Parse Platform.....	3
1.6 Languages Used	
1.6.1 Swift.....	3
CHAPTER 2 PROBLEM DEFINITION AND OBJECTIVES	
2.1 Problem Definition.....	4
2.2 Objectives	
2.2.1 Easy access to events for the users.....	4
2.2.2 Unified view of all the CRM events for the staff.....	4

2.2.3	Access restricted to employees of SAP.....	5
2.2.4	Sync events to calendars.....	5
CHAPTER 3	BACKGROUND	6
CHAPTER 4	DESIGN	7
4.1	Model.....	7
4.2	View.....	7
4.3	Controller.....	8
4.4	Execution	
4.4.1	Model.....	8
4.4.2	View	8
4.4.3	Controller.....	8
CHAPTER 5	METHODOLOGY	
5.1	Mobile Development	
5.1.1	Requirement Analysis.....	9
5.1.2	Formulate Strategy.....	9
5.1.3	UI design and conceptualization.....	9
5.1.4	App Development.....	10
5.2	Parse	
5.2.1	Connecting to Parse.....	10
5.2.2	Parse code.....	10
5.2.3	Parse queries.....	11
5.3	Core Data	
5.3.1	Core Data queries.....	11

CHAPTER 6

IMPELMENTATION

6.1 Design and implementation of the app

6.1.1	Xcode.....	12
6.1.2	View Controller.....	13
6.1.3	Constraints.....	23
6.1.4	Segues.....	23
6.1.5	Class Functions.....	24
6.1.6	App Extensions.....	24
6.1.7	Styles.....	25
6.1.8	Text Fields and Labels.....	25
6.1.9	Images and Icon.....	26
6.1.10	Buttons.....	27
6.1.11	Activity Indicator.....	28
6.1.12	Alert.....	29
6.1.13	Search.....	29
6.1.14	Mailer.....	30
6.1.15	Picker View.....	31
6.1.16	Add to calendar.....	32
6.1.17	Feedback Control.....	33
6.1.18	Popover.....	34

6.2 Database

6.2.1	Parse Connection.....	34
6.2.2	User Authentication.....	35
6.2.3	Parse queries.....	36
6.2.4	Core Data.....	38

CHAPTER 7	RESULT AND ANALYSIS	40
-----------	---------------------	----

CHAPTER 8	CONCLUSION	41
-----------	------------	----

REFERENCES.....	42
-----------------	----

PROJECT DETAILS.....	43
----------------------	----

LIST OF FIGURES

Figure	Title	Page
4.1	MVC Model	7
6.1	Xcode	12
6.2	Launch Screen	13
6.3	Login View Controller	14
6.4	Register View Controller	14
6.5	Reset View Controller	15
6.6	Support View Controller	15
6.7	Mail Controller	16
6.8	About View Controller	16
6.9	Feedback View Controller	17
6.10	Add Event View Controller	17
6.11	Tab Bar	18
6.12	Event Table View Controller	19
6.13	Event Table Mail Controller	19
6.14	MyEvent Table View Controller	20
6.15	SearchBy Table View Controller	20
6.16	Search Results Table View Controller	21
6.17	Detail Table View Controller	21
6.18	Popup View Controller	22
6.19	Dropdown Table View Controller	22
6.20	App Extension	24
6.21	Logo	26
6.22	Icon	27
6.23	Background Image	27
6.24	Button	28
6.35	Activity Indicator	29
6.26	Alert	29
6.27	Search Bar	30
6.28	Full Star	33

6.29	Empty Star	33
6.30	Parse Dashboard	35
6.31	Data Model	38

ABBREVIATIONS

CRM	Customer Relationship Management
iOS	iPhone Operating System
OSX	Macintosh Operating System
Inc.	Incorporated
sdk	Software Development Kit
MVC	Model View Controller
JSON	JavaScript Object Notation
UI	User Interface
ID	Identifier
NS	NeXTSTEP
IB	Interface Builder

1. INTRODUCTION

1.1 GENERAL

This app was developed to maintain a record of all the CRM events. iOS development was required in developing this app is not only because of its secure and strict platform but also because at SAP all the employees use iPhones which restricts the app to be built in iOS only. All the company apps are built in iOS only.

This project will include the design and implementation of the app done on the “Xcode” software using “Objective-c” and “Swift” programming languages. This app is aimed at maintaining a record and report of the CRM Events that the marketing team organizes. This is achieved by creating an interface that allows the user to add, delete and display events as per his/her needs. The user must be first registered with the app with his/her employee mail id and these details will be stored and retrieved from a private server on “Parse” when the user logs in to the app. All the basic features for a user authenticated event app have been added that are described below.

1.2 ORGANIZATION

As market leader in enterprise application software, SAP helps companies of all sizes and industries innovate through simplification. From the back office to the boardroom, warehouse to storefront, on premise to cloud, desktop to mobile device – SAP empowers people and organizations to work together more efficiently and use business insight more effectively to stay ahead of the competition. SAP applications and services enable customers to operate profitably, adapt continuously, and grow sustainably. SAP caters to a large number of customers in over 190 countries. SAP is further divided into 2 parts namely SAP India and SAP Labs. SAP India is responsible for the marketing management of SAP where SAP Labs focuses on the technical requirements of SAP. These 2 divisions coordinate in sync with each other to provide the best experience for its customers.

1.3 AREA OF COMPUTER SCIENCE

The area of computer science used in this project is mobile development. The mobile development technology used is Apple's iOS development. Apple has its own objective oriented languages namely "Objective-C" and "Swift" which are used to design the iOS apps. The software used to build custom apps is "Xcode" that provides a full fetched Interface builder where anyone can construct the UI and link it to the code to provide the functionality.

1.4 HARDWARE REQUIREMENTS

1.4.1 Macbook/Mac

Macbook was required for this project because it provided the latest OSX that is required for building the app.

1.5 SOFTWARE REQUIREMENTS

1.5.1 Xcode

Xcode is an "OSX" app provided by Apple, which you can download from the "iTunes" store. This app has a unique Interface Builder that caters to anyone who wants to make any kind of Apple OS app. The app can be programmed using two languages namely "Objective-C" and "Swift" which can communicate and use each other's functionality defined in their specific syntax. It also provides an iPhone simulator to test your code.

1.5.2 OS Version 10.11.3

The Mac OSX El Capitan is required to run the iOS sdk that is required for the development of the app.

1.5.3 Sketch

Due to the variety of versions available for iPhones in the current market that have different screen resolutions, images or icons added to the app have to be configured

according to the device requirements. Sketch is used to draw or convert any image to 3 different sizes of pixels i.e. 1x, 2x and 3x which are used as per the requirement.

1.5.4 Parse Platform

Parse is an online free cloud server that allows you to manage the database and user authentication services for your app. All you need is an account to link your app to this server for storing and retrieving data.

1.6 LANGUAGES USED

1.6.1 Swift

Swift is a multi-paradigm, compiled programming language created for iOS, OS X, watchOS, tvOS and Linux development by Apple Inc. Swift is designed to work with Apple's Cocoa and Cocoa Touch frameworks and the large body of existing Objective-C code written for Apple products. Swift is intended to be more resilient to erroneous code ("safer") than Objective-C and also more concise.

2. PROBLEM DEFINITION AND OBJECTIVES

2.1 PROBLEM DEFINITION

SAP is the world's leading ERP provider whose main goal is familiarize its customer to the use of ERP systems and how it would benefit them. SAP provides many services to its customers that help improve their experience in the management of their data. For all this to happen the customer needs to meet with the employees so they can tell them about what is bothering them and what they need. So the marketing team at SAP is responsible to set up CRM events that allow customer meetings with SAP employees. SAP already has an existing CRM software that stores all the details of the customers and any CRM event data. This data is then added to an excel sheet for the team to view. The main focus of SAP today is "Digital Transformation" in which one part is remove manual excel sheets. So the marketing team needed a provision that would allow them to view these event details in a unified form without the use of manually exchanging excel sheets.

2.2 OBJECTIVES

The project will deliver a functional, thoroughly tested, error-free app that will provide:

2.2.1 Easy access to events for the users

The UI of the app is designed to provide the user a very simple but sophisticated access to all the functions of the app. The events are stored in the form of a table view that allows the user to view all the registered events in the app at a quick go. The users will be able to add and view images corresponding to the required events, which would also be saved on the server.

2.2.2 Unified view of all CRM events to staff

The basic purpose of this app is to give a unified view of the all the CRM events organized by the marketing team. This app aims at updating all the users of the new events that are added to the app so that all the employees can be up to date with the current events.

2.2.3 Access restricted to employees of sap

Since this app will contain sensitive SAP data the access is restricted to only the SAP employees. Only those users who have an SAP corporate id will be allowed to register for the app and use its functionality. Using these validations on the app was required to avoid any misuse of the app if someone would get a hang of it.

2.2.4 Sync events to calendars

All the users can sync his/her calendar with any event as per requirements and updating any of these details in the app will be reflected in the calendar.

3. BACKGROUND

This project has been undertaken to create an event management app that would provide the user to see all the events that are being added by anyone using the app. This would keep all the team members up to date with any CRM event that is taking place. This project also has the feature of saving that event details to his/her calendar. This app will help keep a record of the all the CRM events organized by the marketing team. This will benefit the sales team to keep track of these events.

4. DESIGN

The design pattern followed for the development of this app was the MVC model (Fig. 4.1), which is described as follows:

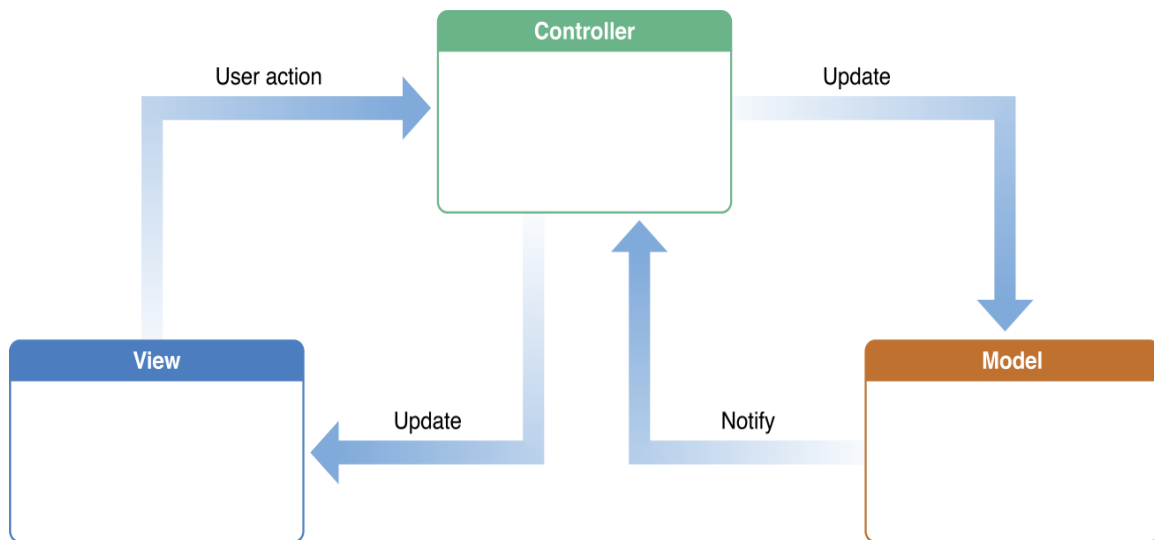


Fig. 4.1 MVC Model

4.1 MODEL

Model objects encapsulate the data specific to an application and define the logic and computation that manipulate and process that data.

User actions in the view layer that create or modify data are communicated through a controller object and result in the creation or updating of a model object. When a model object changes (for example, new data is received over a network connection), it notifies a controller object, which updates the appropriate view objects.

4.2 VIEW

A view object is an object in an application that users can see. View objects learn about changes in model data through the application's controller objects and

communicate user-initiated changes—for example, text entered in a text field—through controller objects to an application’s model objects.

4.3 CONTROLLER

A controller object acts as an intermediary between one or more of an application’s view objects and one or more of its model objects. Controller objects are thus a conduit through which view objects learn about changes in model objects and vice versa.

A controller object interprets user actions made in view objects and communicates new or changed data to the model layer. When model objects change, a controller object communicates that new model data to the view objects so that they can display it.

4.4 EXECUTION

The Life cycle of a view controller follows the MVC model.

4.4.1 Model: When the project is built for the first time the Model loads all the data objects and documents that are accessed for loading the view.

4.4.2 Controller: After the documents are loaded the “AppDelegate.swift” is called which contains the main function that is executed first. This will call the **UIApplicationDelegate** to start loading all the controllers in the project.

4.4.3 View: After the “AppDelegate.swift” is called along with the view controllers all the UIobjects are also loaded i.e. buttons, labels etc.

5. METHODOLOGY

5.1 MOBILE DEVELOPMENT

5.1.1 Requirement Analysis

The first step was to analyze the requirement for the app. Who all will be the users, what type of events will be added to this app. How they can be provided with the best format to fill in the required details with ease and create a UI that they are familiar with using. Many such questions were asked during the Analysis Stage.

5.1.2 Formulate Strategy

The market trends were looked upon to ensure that the app should have the latest technology, components and elements. By assessing at how other apps at SAP are designed as well as their official webpages it helped to keep the UI similar to already existing apps. Also selecting the database required during the storing and retrieval of data.

5.1.3 UI and Design Conceptualization

This stage was one of the most crucial factors of the project. Justifying every single aspect of the design to ensure that each element is serving a specific need and the app is truly designed to enhance the end user-experience. Many thought processes and brainstorming sessions went into this process to create a design that is serving the existing needs as well as is sustainable for the future growth. Together with the input of my external guide, designs were formulated that would effectively communicate with all the users of the app.

5.1.4 App Development

UI design and adding functionality to the design using “Swift” and “Objective-C” in “Xcode” was the first step. Selecting what types of backgrounds to use along with the design of the buttons followed after. Adding validations on text fields and setting up alerts when the data is not retrieved or any wrong data is entered. Transitioning from one view to another via segues.

5.2 PARSE

The database used for this app is “Parse”, which is a mobile backend service provider that helps many developers to store their data in the cloud, manage identity log-ins, handle push notifications and run custom code on cloud. All data stored on parse is in the form of “JSON”.

5.2.1 Connecting to Parse

For using the parse database all that is required is to create a parse account and create a new app on the parse dashboard. Each new app created on parse will have its own unique key that is used to connect your project in “Xcode” to the app in the parse database. Installation of iOS parse sdk is required to import all the parse libraries to your project that will be accessed when saving or retrieving data from parse. Parse provides up to 1TB of free data storage that can be increased by purchasing more space if required.

5.2.2 Parse Code

Parse code can be used in a variety of languages, but since this project uses iOS development “Objective - C” or “Swift” languages were used to type the parse code. Parse provides its own docs, which provides code in the respective languages that needs to be used for saving or retrieving data. One example of parse code is registering users and logging them into the app. This gives a more secure approach to the app and can be accessed by registered users.

5.2.3 Parse Queries

Now that users are registered using the app, storage of any data entered by the users is required and displaying it to them. For these situations we use queries. Parse has provided a large amount of queries that can be used to create classes on the parse dashboard and store relevant data in them and retrieve from these classes whenever necessary. Every new entry creates a new object ID that can be used to retrieve specific data if required.

5.3 CORE DATA

Core Data is a framework that you use to manage the model layer objects in your application. It provides generalized and automated solutions to common tasks associated with object life cycle and object graph management, including persistence. It was used to store unique local event ids for all the events. These local event ids would help in retrieving the events that have been added to the user calendar whenever any changes are made. This would provide a unification of events in the app and the events synced with the calendar.

5.3.1 Core Data queries

Core Data although being a framework works like a database. So, to retrieve or store data there are a specific set of queries that helps with the functionality. “Xcode” provides a very easy interface to create “entities” and “fields” (stored as objects). Using the queries we can fetch these objects as per requirement.

6. IMPLEMENTATION

6.1 DESIGN AND IMPLEMENTATION OF THE APP

6.1.1 Xcode

The Xcode software provides a very user friendly Interface Builder that allows the developer to design the UI of the app with ease and link the UI to the code.

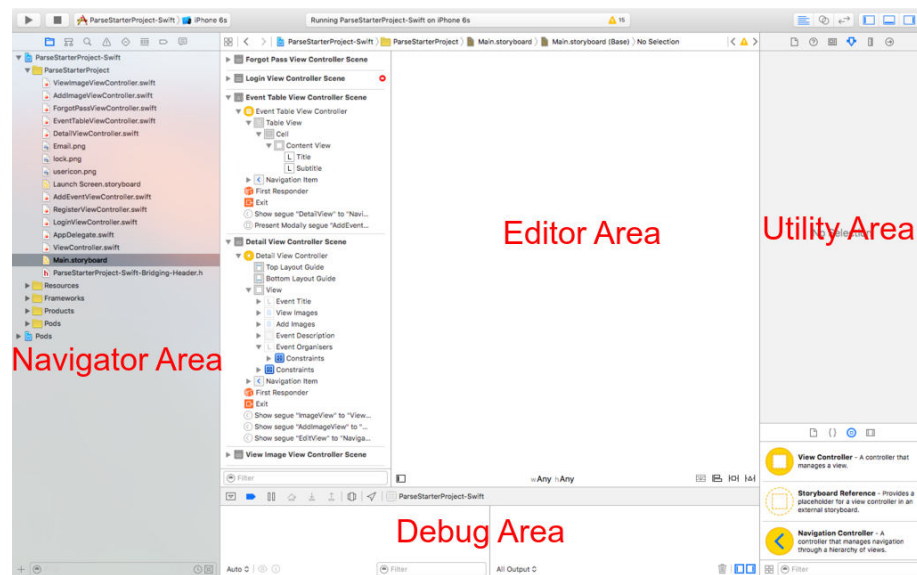


Fig. 6.1 Xcode

Navigator Area is where all the project files can be accessed which stores the functionality of the app. This is also where the design of the app i.e. “Main.Storyboard” and all the frameworks that are needed to implement specific classes in the project are stored.

Editor Area is where the designing of the UI takes place with the various controllers provided in xcode.

Debug Area is where all the errors are displayed and it is also used in the debug process when we are trying to catch the erroneous area.

Utility Area is where the design of the various labels, buttons, text fields etc. is specified and assigning the classes to the view controllers.

6.1.2 View Controller

This is where the UI is designed for the app. A view controller is like a page for an app that need to be linked to another view to generate a flow for the app structure. Whenever a new class is created a variety of options of already existing classes are provided to choose from as per requirement. These classes are assigned to the view controllers that add functionality to that view controller. Another type of view controller is the **navigation view controller** that is used to navigate from one view controller to another.

Initial View Controllers:

When the app open for the first time the user is provided with 6 sets of standard view controller that were used which are; Launch Screen, Login View Controller, Register View Controller, Reset password View Controller, Support View Controller, About view Controller, Feedback View Controller, Add Event View Controller.

The **Launch Screen** (Fig. 6.2) also a normal view controller but it cannot be assigned with any custom class as it will be displayed when the app starts for the first time.



Fig. 6.2 Launch Screen

The **Login View Controller** (Fig. 6.3) allows the user to login into the app. This view also provides the functionality to register a new user (Sign Up?) or reset an existing password (Reset Password).

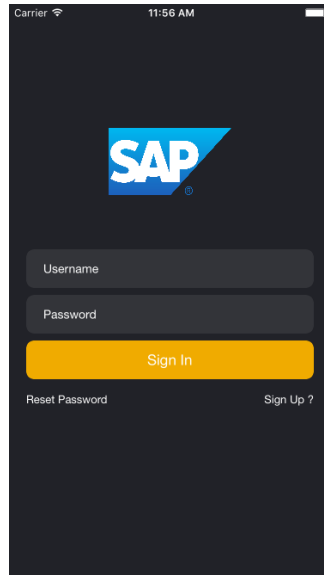


Fig. 6.3 Login View Controller

The **Register View Controller** (Fig. 6.4) allows any SAP employee to register with the above-mentioned details. Once the user signs up (Sign Up), he/she is taken back to the Login View to login with the registered details.

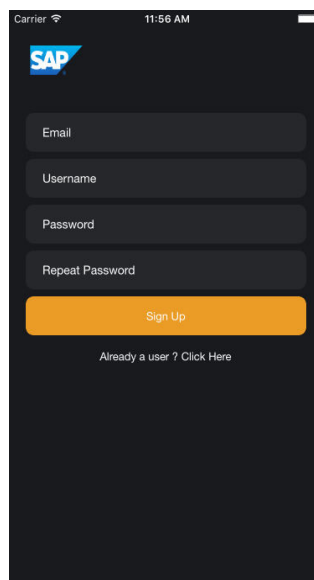


Fig. 6.4 Register View Controller

The **Register Password View Controller** (Fig. 6.5) view allows an already existing user to reset his/her password by sending a reset link to their registered main ids (Reset).

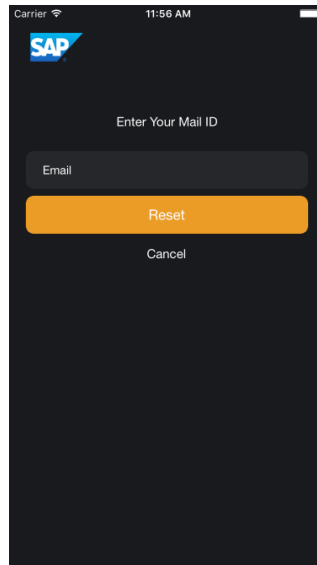


Fig 6.5 Reset Password View Controller

This **Support View Controller** (Fig. 6.6) allows the user to open up the default mail controller (Fig 6.7) of the phone. This will send a mail to the app developer describing the issue along with the issue subject.

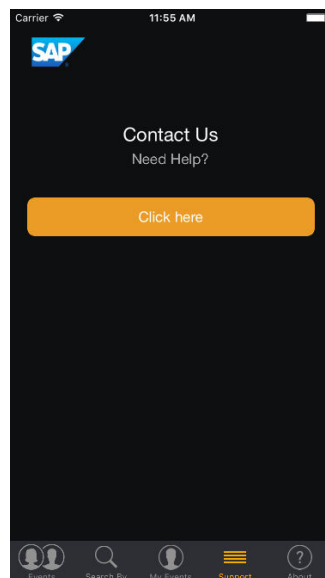


Fig 6.6 Support View Controller

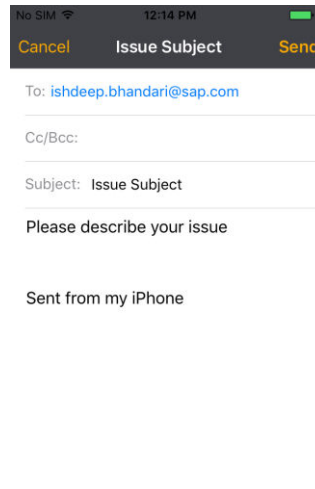


Fig 6.7 Mail Controller

The **About View Controller** (Fig. 6.8) is a standard view controller that informs the user about the copyrights for the app.

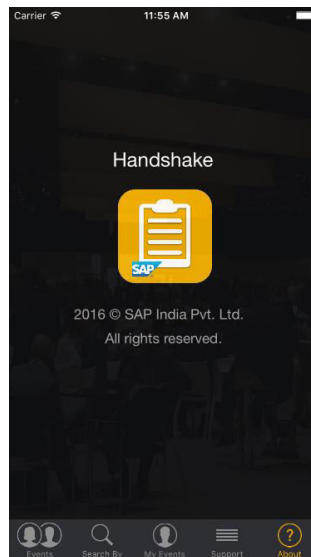


Fig. 6.8 About View Controller

The **Feedback View Controller** (Fig. 6.9) provides the feature to give feedback for any event (Rate). This is used to calculate the average rating with the number of feedbacks the event has received. The average for all ratings is then stored on the server.

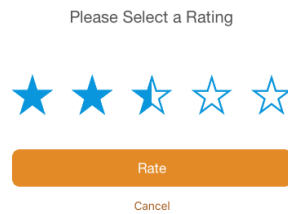


Fig. 6.9 Feedback View Controller

The **Add Event controller** (Fig. 6.10) contains a number of fields that have to be filled whenever a new event needs to be added to the app (Done) or if an already existing event needs updating.

The image shows a mobile app interface for adding a new event. At the top, there's a status bar with "Carrier", signal strength, "11:57 AM", and battery level. Below the status bar is a header with "Cancel" on the left, "New Event" in the center, and "Done" on the right. The main form contains several fields, each with a red asterisk indicating it's required: "Segment", "Audience", "Organiser", "CRM code", "Title", "Location", "Start", "End", "Number of attendees", "Short Description" (a larger text area), "MIP(,000 EUR)", and "MGQ(,000 EUR)".

Fig. 6.10 Add Event View Controller

Tab View Controller:

This is another type of view controller that provides a tab bar (Fig. 6.11) at the bottom of the screen. Then tab bar allows multiple views to run simultaneously side-by-side. A maximum of 5 tabs can be added to the tab bar as per iOS guidelines. The 5 tabs added were Events, Search By, My Events, Support and About.

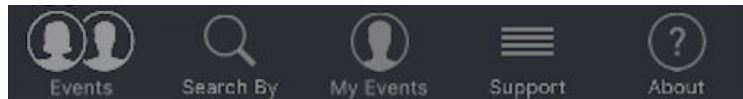


Fig. 6.11 Tab Bar

Table View Controllers:

This is the third type of view controller that was used in the project. According to iOS guidelines any data that needs to be displayed should be done using the table view controller. In this app 6 types of table view controllers were used which were; Event Table View Controller, MyEvent Table View Controller, SearchBy Table View Controller, SearchResults Table View Controller, Detail Table View Controller and Dropdown Table View Controller.

The **Event Table View Controller** (Fig. 6.12) is where all the events are stored in a table form. The design was chosen to look like a list that then expanded into a detail form. It will display the title of the event and the corresponding dates for the event. There is a detailed disclosure on each cell that will open into a new controller and display the details corresponding to the selected event. The “+” button opens the Add Event View Controller that allows the user to add a new event to the app.

On swiping to the left of the cell 3 fields are shown i.e. Forward, Feedback , Rating. These fields have specific roles, where on pressing the forward button the default mailer controller (Fig. 6.13) will open with the event details in the body. This feature will allow the user to send the event details to anyone via mail. The feedback feature opens the Feedback View Controller and the ratings displays the current average rating for the event.

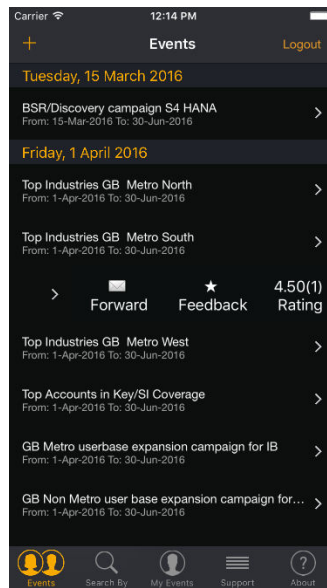


Fig. 6.12 Event Table View Controller

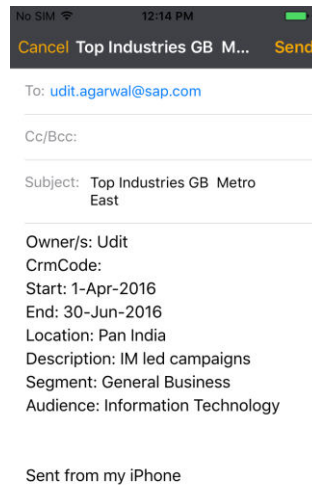


Fig. 6.13 Event Table Mail Controller

The **MyEvent view controller** (Fig. 6.14) only contains those events that are linked to the current user. All the functionalities mentioned above for the Event View Controller also apply to this table view controller.



Fig. 6.14 MyEvent Table View Controller

The **SearchBy Table View Controller** (Fig. 6.15) contains 4 cells which are; Audience, Date, Location, Segment.

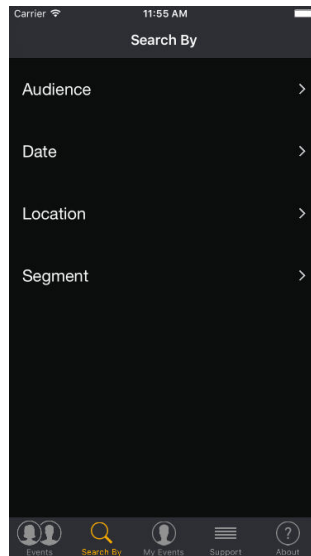


Fig. 6.15 SearchBy Table View Controller

On clicking one of the fields in the SearchBy Table View Controller, it will open the **Search Results View Controller** (Fig. 6.16). This view will contain a Search Controller whose input will be given from a Picker View. The data in the picker view will be decided based on the cell selected in the previous view controller. The search controller will filter details based on the selection from the picker view entered in the search bar.

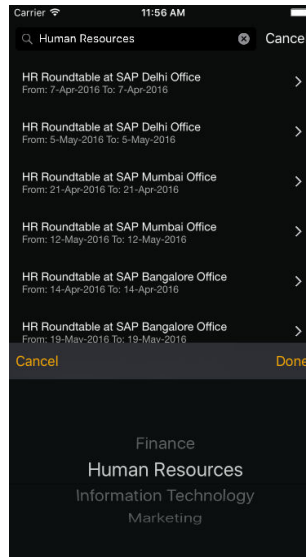


Fig. 6.16 Search Results Table View Controller

On clicking on the disclosure button the **Detail Table View Controller** (Fig. 6.17) will open up which will display the details of the corresponding event. It will contain all the fields that were shown in the Add Event View Controller. A detail button is added to the top right that will bring up a Popup View Controller (Fig. 6.18). This controller will contain 3 buttons i.e. Edit, Delete, Sync. The edit button will open the Add Event View Controller with the details from the Detail Table View Controller. The Delete button will remove the event from the app permanently. The Sync button allows the user to sync that particular event to the user's personal calendar. The cell was custom designed with 2 labels (one below another) that would adjust according to the text displayed in them.

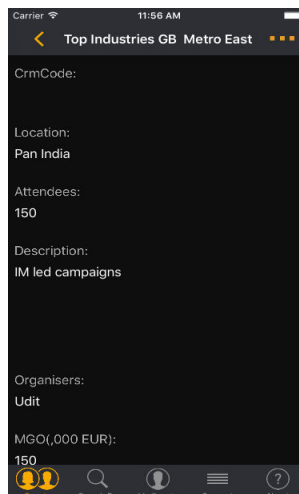


Fig. 6.17 Detail Table View Controller

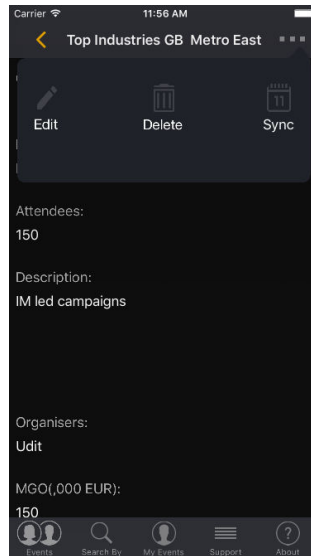


Fig. 6.18 Popup View Controller

The **Dropdown Table View Controller** (Fig. 6.19) will be called whenever the Segment, Audience or Organizer fields need to be filled. It will contain the corresponding data for the fields. This allows multiple selections without any duplicate entries.

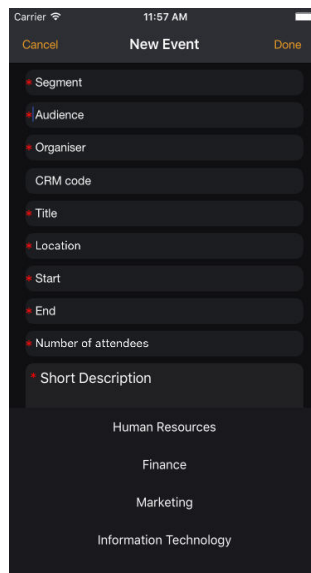


Fig.6.19 Dropdown Table View Controller

6.1.3 Constraints

To make sure that all the different elements added to the view controllers stay fixed and don't move about the view. These constraints also with the help of auto layout help in keeping the elements aligned with the rest of the view controllers. This helps in making the app more versatile and usable on multiple devices. The constraints are set in the "Editor View" and need to be set for each element on the view.

6.1.4 Segues

These are the elements that are used to connect different view controllers when there is transition between view controllers. Each segue has its own identifier which is called when a transition to the next view controller takes place. There were 2 kinds of segues that were used to transition between view controllers namely; Present Modally (default/system segue) and Custom (custom segue).

Pseudo code:

```
self.performSegueWithIdentifier("identifier", sender: self)
self.navigationController!.pushViewControllerAnimated(true)
```

Custom segue code:

```
override func perform(){
let src: UIViewController = self.sourceViewController
let dst: UIViewController = self.destinationViewController
let transition: CATransition = CATransition()
let timeFunc : CAMediaTimingFunction = CAMediaTimingFunction(name:
kCAMediaTimingFunctionEaseInEaseOut)
transition.duration = 0.25
transition.timingFunction = timeFunc
transition.type = kCATransitionPush
transition.subtype = kCATransitionFromRight
src.navigationController!.view.layer.addAnimation(transition, forKey: kCATransition)
src.navigationController!.pushViewController(dst, animated: false)}
```

6.1.5 Class Functions

This is where all the functionality of each view controller is defined in the corresponding classes they are linked too. There are 3 main functions that must be there and overridden in every class which are:

viewDidLoad() : This is responsible for loading the view controller and presenting it to the user. Any code added in this function will be launched when the view loads.

viewDidAppear() : This is called when an already existing view controller is viewed again after being loaded once. Any code added in this function will be launched when the view appears again.

viewDidReceiveMemoryWarning() : This disposes of any resources that can be recreated again when the view is called.

6.1.6 App Extensions

An app extension (Fig. 6.20) is not an app. It implements a specific, well scoped task that adheres to the policies defined by a particular extension point. App extension was used to add global alert functionality, delay function and date comparable functions.

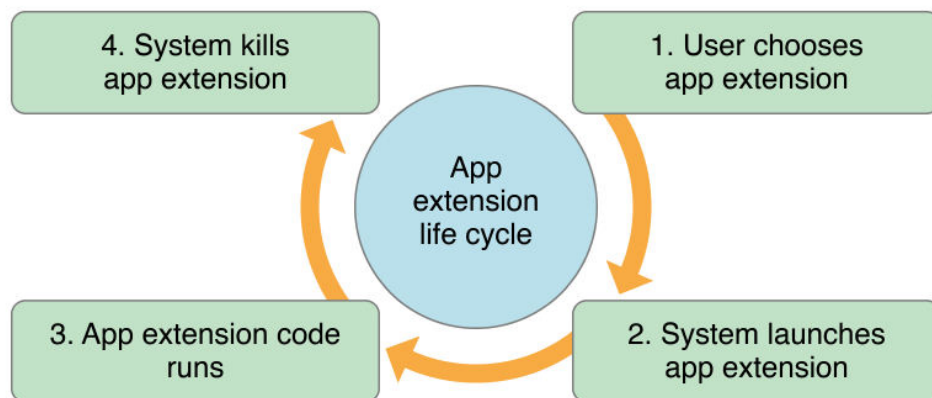


Fig. 6.20 App Extension

Pseudo code:

```
extension UIViewController{ }
```

6.1.7 Styles

The Font used on all the elements is “Helvetica Neue”. For the background of the app the color used was black. The color used for buttons was similar to medium yellow and the text size was 13 pts. The color used for all the text was white. All individual buttons were yellow.

6.1.8 Text Fields and Labels

Textfields are elements that are used to enter the text that is then retrieved in the class file so that you make changes to it or use it as per requirement. Labels are used to display text on the device and have to be similarly linked to the code.

Pseudo code:

```
@IBOutlet weak var EmailTextField: UITextField!
```

```
@IBOutlet weak var EventTitle: UILabel!
```

UITextField and UILabel are built in classes that allow use to access the various properties of textfields and labels.

Outlet is required to link your object to your code so that you can use its properties whenever required.

Weak is used when the user doesn’t want to have any control over the lifetime of the object. The object you are referencing weakly only lives on because at least one other object holds a strong reference to it. Once that is no longer the case, the object gets destroyed and your weak property will automatically get set to nil.

Strong is used when you will want to own the object you are referencing. The compiler will take care that any object that you assign to this property will not be destroyed as long

as you (or any other object) points to it with a strong reference. Only once you set the property to nil will the object get destroyed.

6.1.9 Images and Icon

Whenever an image is used in the project, the image should have 3 sizes i.e. 1x, 2x and 3x that represents the different resolution for the iPhones, similarly for the icons. Apple guidelines provide the required resolution sizes that each image should have. All these images are stored in the assets folder already added to every “Xcode” project in JSON format.

Sketch:

Sketch is a design tool that is very similar to Photoshop but provides simple usage. All the images used for buttons, tabs etc were designed in Sketch. Sketch has a repository of resources [13] that contains already existing designs that can be modified and used later. The company Logo (Fig. 6.21) and the background image (Fig. 6.23) were pulled from the Brand assets library of SAP. The icon (Fig. 6.22) was first designed in Sketch then converted to the required resolution and sizes with MakeAppIcon [14].



Fig. 6.21 Logo



Fig 6.22 Icon



Fig. 6.23 Background Image

6.1.10 Buttons

These are used when some action needs to be performed upon interaction, for example transition to new view controller or cause the saving of data. The buttons used were designed using the features provided in Sketch. The general button design is shown below (Fig. 6.24). Buttons for which images were designed were; Add, Logout, Edit, Delete, Sync, Detail

Pseudo code:

```
@IBAction func AddImageButtonTapped(sender: AnyObject) {  
}
```

Action: This is used when you some code needs to be executed whenever the object is selected or tapped. It creates a function in which you can add anything that requires execution whenever the user interacts with it.

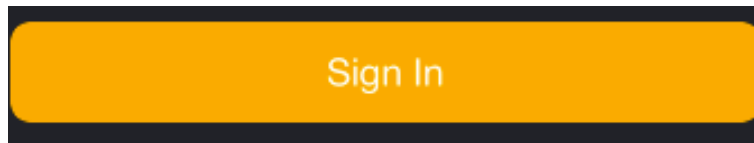


Fig. 6.24 Button

6.1.11 Activity Indicator

The Activity Indicator (Fig. 6.25) is used to let the user know that the app is performing some background functions, so it doesn't let the user mess with the UI until the process is completed.

Pseudo code:

```
Var activityindicator: UIActivityIndicatorView = UIActivityIndicatorView()  
self.activityindicator = UIActivityIndicatorView(frame: self.view.frame)  
activityindicator.backgroundColor = UIColor(white: 1.0, alpha: 0.5)  
self.activityindicator.center = self.view.center  
self.activityindicator.hidesWhenStopped = true  
self.activityindicator.activityIndicatorViewStyle = UIActivityIndicatorViewStyle.Gray  
view.addSubview(self.activityindicator)  
self.activityindicator.startAnimating()  
UIApplication.sharedApplication().beginIgnoringInteractionEvents()  
self.activityindicator.stopAnimating()  
UIApplication.sharedApplication().endIgnoringInteractionEvents()
```

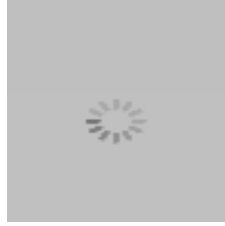


Fig 6.25 Activity Indicator

6.1.12 Alert

The Alerts (Fig. 6.26) are called whenever the user either leaves something blank or enters an erroneous input. Also when there was some problem in storing or retrieving data.

Pseudo code:

```
let MyAlert = UIAlertController(title: title, message: errmessage, preferredStyle:
UIAlertControllerStyle.Alert)
let okAction = UIAlertAction(title: "Ok", style: UIAlertActionStyle.Default, handler: nil)
MyAlert.addAction(okAction)
self.presentViewController(MyAlert, animated: true, completion: nil)
```

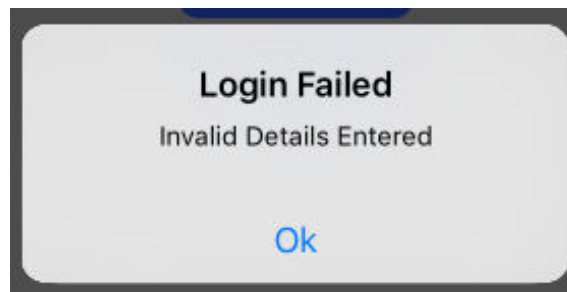


Fig 6.26 Alert

6.1.13 Search

The search feature was added to the Search Results View Controller that allows you sort the data based on the text entered in the Search bar Field (Fig. 6.27).



Fig. 6.27 Search Bar

Pseudo code:

```
var resultSearchController = UISearchController(searchResultsController: nil)
func filterContentForSearchText(searchText: String, scope: String = "All"){
    return data.audience.lowercaseString.containsString(searchText.lowercaseString)
    tableView.reloadData() }
func updateSearchResultsForSearchController(searchController: UISearchController) { }
```

6.1.14 Mailer

The Mail controller was used to call the default mailer in the phone. Predefined data can be entered to through the code recipients, subject and body text. This was used in Support View Controller to seek help via mail and in Event View Controller to forward event details via mail.

Pseudo code:

```
func configuredMailComposeViewController() -> MFMailComposeViewController {
    let mailComposerVC = MFMailComposeViewController()
    mailComposerVC.mailComposeDelegate = self
    mailComposerVC.setToRecipients(["ishdeep.bhandari@sap.com"])
    mailComposerVC.setSubject("Issue Subject")
    mailComposerVC.setMessageBody("Please describe your issue", isHTML: false)
    return mailComposerVC
}
if MFMailComposeViewController.canSendMail() {
    self.presentViewController(mailComposeViewController, animated: true, completion:
    nil)}
```

```

else {
    self.showSendMailErrorAlert()
}

```

6.1.15 Picker View

The Picker View allows a single choice pick where the user can choose one of the many values. It was used in Search Results View Controller to search through different options of every field. The selection would be entered into the search bar that would then sort the data accordingly. Another type of picker is the Date picker that allows the user to choose dates.

Pseudo code:

```

var pickerView: UIPickerView?

PickerView = UIPickerView()
PickerView?.delegate = self
let toolBar = UIToolbar()
toolBar.barStyle = UIBarStyle.Default
toolBar.sizeToFit()

let doneButton = UIBarButtonItem(title: "Done", style: UIBarButtonItemStyle.Plain,
target: self, action: #selector(SearchResultsController.donePicker))
let spaceButton = UIBarButtonItem(barButtonSystemItem:
UIBarButtonItem.FlexibleSpace, target: nil, action: nil)
let cancelButton = UIBarButtonItem(title: "Cancel", style: UIBarButtonItemStyle.Plain,
target: self, action: #selector(SearchResultsController.cancelPicker))
toolBar.setItems([cancelButton, spaceButton, doneButton], animated: false)
toolBar.userInteractionEnabled = true

func numberOfComponentsInPickerView(colorPicker: UIPickerView!) -> Int {
    return 1
}

```

```

func pickerView(pickerView: UIPickerView!, numberOfRowsInComponent component:
Int) -> Int {
    return PickerData.count
}

func pickerView(pickerView: UIPickerView, attributedTitleForRow row: Int,
forComponent component: Int) -> NSAttributedString? {
    let attributedString = NSAttributedString(string: PickerData[row], attributes:
[NSForegroundColorAttributeName : UIColor.whiteColor()])
    return attributedString
}

func pickerView(pickerView: UIPickerView, didSelectRow row: Int, inComponent
component: Int) {
    let searchtextfield = resultSearchController.searchBar.valueForKey("searchField") as?
UITextField
    searchtextfield?.text = PickerData[row]}

```

6.1.16 Add to Calendar

The sync button allowed the user to sync that event with his/her calendar. The user was prompted if he/she wanted to sync the event with the calendar.

Pseudo code:

```

let calstore = EKEventStore()
calstore.requestAccessToEntityType(.Event) {(granted, error) in
    if !granted { return }
    let appevent = EKEvent(eventStore: self.calstore)
    appevent.title = self.eventtitle
    appevent.startDate = convertedstart!
    appevent.endDate = convertedstop!
    appevent.location = self.eventlocation
    appevent.calendar = self.calstore.defaultCalendarForNewEvents
    appevent.notes = self.eventdescription
    let eventalarm1 = EKAlarm()

```

```
let eventalarm2 = EKAlarm()
eventalarm1.relativeOffset = -86400
eventalarm2.relativeOffset = -3600
appevent.addAlarm(eventalarm1)
appevent.addAlarm(eventalarm2)
do {
    try self.calstore.saveEvent(appevent, span: .ThisEvent, commit: true)}
    catch { // Display error to user}
```

6.1.17 Feedback Control

A rating system was designed by using a star-rating concept. A rating view was designed using @IBInspectable. This allows assigning the rating class to any UIView. There were 2 images that were used to represent the stars; Full star (Fig. 6.28) and Empty star (Fig. 6.29). The user could fill these stars by tracing the finger on them. The user has a choice to fill either a full start or half a star.



Fig. 6.28 Full Star



Fig. 6.29 Empty Star

6.1.18 Popover

Popover was added to the detail button on the Detail View Controller. It was used to provide easy access to the Edit, Delete and Sync buttons. A popover is a standard view controller but it is called as a popup on button press.

Pseudo code:

```
let storyboard : UIStoryboard = UIStoryboard(name: "Main", bundle: nil)
let vc = storyboard.instantiateViewControllerWithIdentifier("PopoverMenuController")
as! PopoverMenuViewController
vc.delegate = self
vc.eventtitle = titleevent
vc.modalPresentationStyle = UIModalPresentationStyle.Popover
vc.preferredContentSize = CGSizeMake(400, 150)
let popover: UIPopoverPresentationController? = vc.popoverPresentationController
if popover == vc.popoverPresentationController!{
    popover!.backgroundColor = vc.view.backgroundColor
    popover!.barButtonItem = sender as? UIBarButtonItem
    popover!.delegate = self
    presentViewController(vc, animated: true, completion:nil)}
delay(0.1){popover?.passthroughViews = nil }
```

6.2 DATABASE

6.2.1 Parse Connection

As mentioned above parse (Fig. 6.30) needs to be connected to your app with the following code which is added to the “AppDelegate.swift” file.

Pseudo code:

```
func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
    Parse.enableLocalDatastore()
```

```

Parse.setApplicationId("App Application ID ",clientKey:"App Client Key")
PFAnalytics.trackAppOpenedWithLaunchOptions(launchOptions)
return true }

```

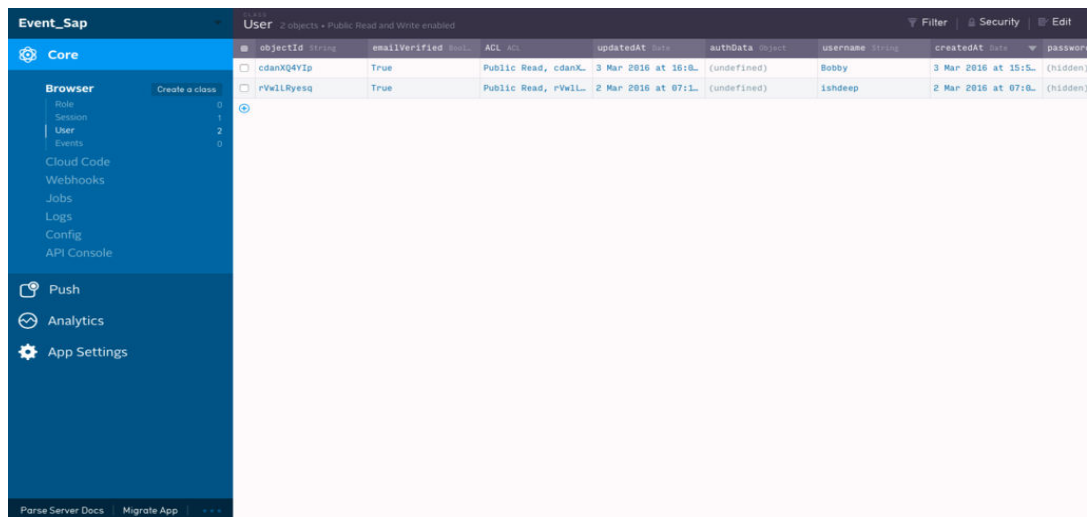


Fig 6.30 Parse Dashboard

6.2.2 User Authentication

As mentioned above parse has its own syntax for creating new users that is provided in the parse docs [5].

Pseudo code:

Register:

```

let user = PFUser()
user.signUpInBackgroundWithBlock ({(success , error) -> Void in
    if error == nil {
        print("Successful Sign Up")
        self.performSegueWithIdentifier("RegisterNew", sender: self)
    } else
    {
        if let errorString = error!.userInfo["error"] as? String{
            errorMsg = errorString
        }
        self.displayAlertMessage("Failed Sign up", errmessage: errorMsg) }}})

```

Login:

```

PFUser.logInWithUsernameInBackground(usrname!, password:pass!) {
    (user: PFUser?, error: NSError?) -> Void in
    if user != nil{
        self.performSegueWithIdentifier("Loggedin", sender: self)
    }
    else {
        print(error)
        self.displayAlertMessage("Login Failed", errmessage: "Invalid Details Entered")
    }
}

```

A PFUser object is created whenever parse code needs to be used for the required functionality. The object's functionality is defined through the imported parse frameworks.

6.2.3 Parse Queries

Parse Queries were used to save and retrieve data to and fro from the parse server.

Pseudo code:**Saving:**

```

let event = PFOBJECT(className: "Events")
event["Name"] = EventTitle.text
event["Description"] = EventDescription.text
event["Start"] = EventStartText.text
event["Stop"] = EventStopText.text
event["Organisers"] = EventOrganisers.text
event["CrmCode"] = EventCrmCode.text
event["Location"] = EventLocation.text
event["Attendees"] = EventAttendee.text
event["Mgo"] = EventMgo.text
event["Mip"] = EventMip.text

```

```

event["UserId"] = PFUser.currentUser().!.username
event.saveInBackgroundWithBlock{(success, error) -> Void in
    if (success == true){
        print("successfull")
        self.dismissViewControllerAnimated(false, completion: nil)
    }
    else
    {
        self.displayAlertMessage("Error in creating entry", errmessage: "Please try
again")}}}]

```

Here a class is created that will store the corresponding data from text fields entered by the users.

Retrieving:

```

let query = PFQuery(className: "Events")
query.whereKey("objectId", equalTo: objectidret)
query.getFirstObjectInBackgroundWithBlock {
    (object: PFObject?, error: NSError?) -> Void in

    self.activityindicator.stopAnimating()
    UIApplication.sharedApplication().endIgnoringInteractionEvents()

    if error != nil || object == nil {
        self.displayAlertMessage("Error Retrieving Information", errmessage: "There
was an error retrieving data. Please try again")
    } else {
self.EventTitle.text = object?.valueForKey("Name") as? String
self.EventOrganisers.text = object?.valueForKey("Organisers") as? String
self.EventDescription.text = object?.valueForKey("Description") as? String
self.EventUser.text = object?.valueForKey("UserId") as? String
self.EventLocation.text = object?.valueForKey("Location") as? String
self.EventAttendee.text = object?.valueForKey("Attendees") as? String
self.EventCrmcode.text = object?.valueForKey("CrmCode") as? String

```



```
self.EventMip.text = object?.valueForKey("Mip") as? String
self.EventMgo.text = object?.valueForKey("Mgo") as? String}
```

Since all the events are stored in the form of objects, they are retrieved using their object id that is being passed from the “Event Table View Controller” to the “Detail View Controller” and stored in the labels to display them.

6.2.4 Core Data

Core Data requires a set of code to integrate it with the app. This code is added into the “AppDelegate.swift” file. The code is added automatically when you create your project in “Xcode”. The code basically manages each object added to the model and how to save and retrieve it when it is called during execution. All the entities and relationships are added to the model file (Fig. 6.31) that is automatically generated. It was used to store the global and local event ids so that when any changes are made to the events they can also be reflected in the calendar if the user has synced them

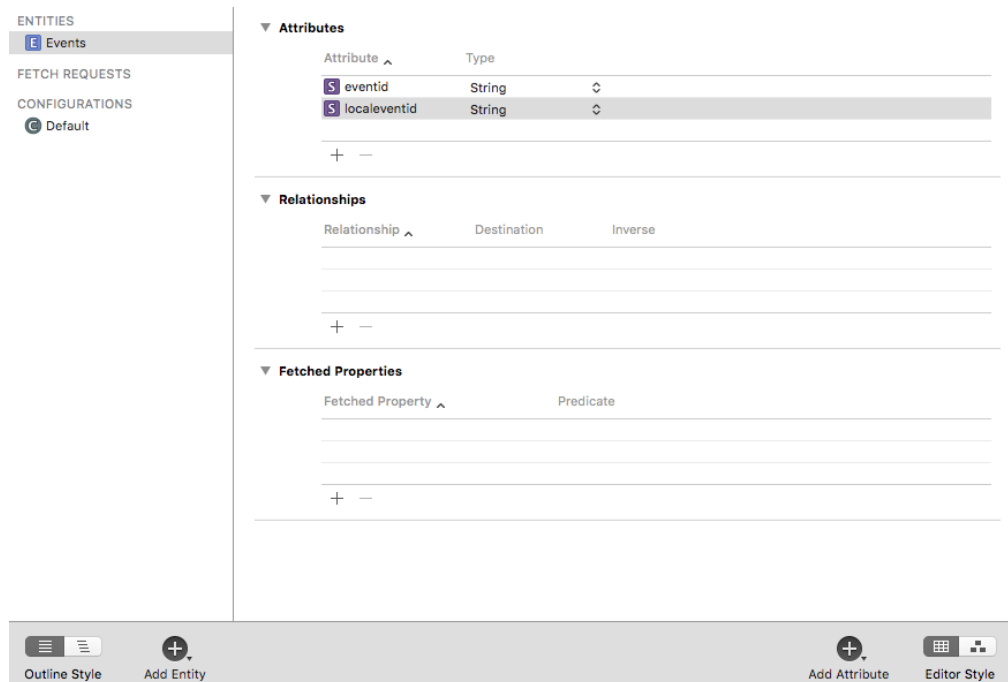


Fig. 6.31 Data Model

Pseudo code:

```
let appDel: AppDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
let context: NSManagedObjectContext = appDel.managedObjectContext
```

Saving:

```
let newUser = NSEntityDescription.insertNewObjectForEntityForName("Events",
inManagedObjectContext: context)
newUser.setValue(newdata.objectid, forKey: "eventid")
do {try context.save()}
catch {print("There was a problem!")}
```

Retrieving:

```
let request = NSFetchRequest(entityName: "Events")
request.predicate = NSPredicate(format: "eventid = %@", newdata.objectid)
request.returnsObjectsAsFaults = false

do {
let results = try context.executeFetchRequest(request)
if results.count > 0 {
for result in results as! [NSManagedObject] {
localeventid = result.valueForKey("localeventid") as? String
}
}
catch {print("Fetch Failed")}
```

7. RESULT AND ANALYSIS

The testing for the app was done on an “iPhone 5C” model. All the above-mentioned functionalities were individually checked. The errors and probable crash issues were solved using the debug feature provided in “Xcode”. This debug feature traverses the code line by line that helped in identifying the errors easily. Suspected crash breakpoints were added to check which part of the code was not running as intended. These breakpoints helped in identifying the issues that would cause the app to crash and whenever a nil value was passed or retrieved. Any runtime errors were handled by generating alerts to the user.

After a series of tests the app was fully functional on all iPhone devices. It ran smoothly and efficiently as was intended and without any issues. The main reason for the development of the app was “Digital Transformation”. This app will be a small part of the broader digital transformation concept that SAP needs to achieve i.e. replacing manual excel sheets with remote access to CRM events.

8. CONCLUSION

The main goal of this project was to emphasize the importance of designing, implementing and maintaining an efficient mobile app. This was done by implementing a responsive mobile design methodology and seeking future friendly solutions. These factors were taken into consideration regardless of the technology or content management system used for mobile implementation.

During this process, it also became clear that change is the only constant thing in mobile development. A very important part of mobile development was capturing the trends, focusing on what is important for the company and developing successfully on these requirements. For me, mobile development is a never-ending learning experience since change is an inevitable and continuous process in this field.

Based on the final implementation of the project, I consider that all the objectives were met very well. I was able to respond, adapt and deliver, the changes the company wanted, and the best way to achieve this is was to be agile and implement development in small iterations. The whole process of this Project was a process of self-assessment for me, because I got the opportunity to reflect on my work and consider what should be done differently in order to aim and achieve the best possible result.

This app was able to achieve a remote access to the CRM software that helped the marketing and sales team to collaborate with each other regarding the CRM events. This app could also be further developed to add/view an image feature that would allow a unified view of all the images tied to one event.

REFERENCES

1. **Lynda Tutorials**
<http://www.lynda.com/member>
2. **Udemy**
Tutorials <https://www.udemy.com/the-complete-iOS-9-developer-course/learn/>
3. **Stackoverflow**
<http://stackoverflow.com>
4. **Parse**
<https://dashboard.parse.com/apps/eventsap/browser/Events>
5. **Parse docs**
<https://parse.com/docs/iOS/guide#queries>
6. **Apple iOS docs**
<https://developer.apple.com/library/iOS>
7. **Youtube**
<https://www.youtube.com>
8. **Evanto**
<http://tutsplus.com>
9. **Tutorials Point**
<http://www.tutorialspoint.com>
10. **Github**
<https://github.com>
11. **Appcoda**
<http://www.appcoda.com/tutorials/iOS/>
12. **Raywenderlich Tutorials**
<https://www.raywenderlich.com>
13. **Sketch Resources**
<http://www.sketchappsources.com>
14. **MakeAppIcon**
<https://makeappicon.com/ios9icon>

PROJECT DETAILS

Student Details

Student Name	Ishdeep Pal Singh Bhandari		
Register Number	120905404	Branch / Roll No	CSE/38
Email Address	ishdeepbhandari@gmail.com	Phone No (M)	9740547038

Project Details

Project Title	Handshake - CRM Event Management App		
Project Duration	4 months	Date of reporting	25 th January

Organization Details

Organization Name	SAP India Pvt. Ltd.		
Full postal address with pin code, telephone number, Fax and email	6th Floor, Plot No A-2, MGF Corporate Park, MGF Metropolitan Mall, Late Shaheed Pankaj Wal Marg, Saket, New Delhi, Delhi 110017 011 3090 7200		
Website address	http://go.sap.com/india/index.html/		

External Guide Details

Supervisor Name	Mr. Yashdeep Bali		
Designation	Integrating Marketing Senior Specialist		
Full contact address with pin code	6th Floor, Plot No A-2, MGF Corporate Park, MGF Metropolitan Mall, Late Shaheed Pankaj Wal Marg, Saket, New Delhi, Delhi 110017		
Email address	yashdeep.bali@sap.com	Phone No (M)	8588803880

Internal Guide Details

Faculty Name	Mr. Dasharathraj Shetty		
Full contact address with pin code	Department of Computer Science and Engineering Manipal Institute of Technology, Manipal , 576104		
Email address	raja.shetty@manipal.edu		

Cheney
24/8/16