iPhone Software Engineering

Assignment 2

Joshua Orozco - s3485376
Jiahong He - s3526309 (postgraduate)

CINEGO Cinema Booking App

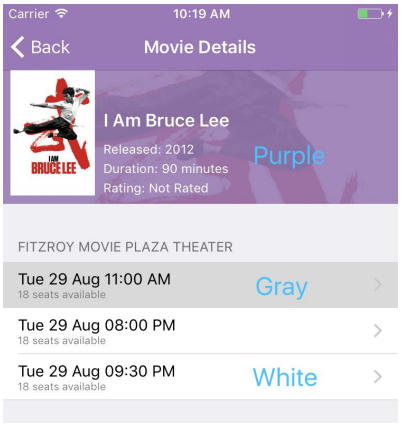
Cinego enables users to book into one of 4 cinemas in Melbourne in simplest possible way.

Table of Contents

Design Principles

Contrast

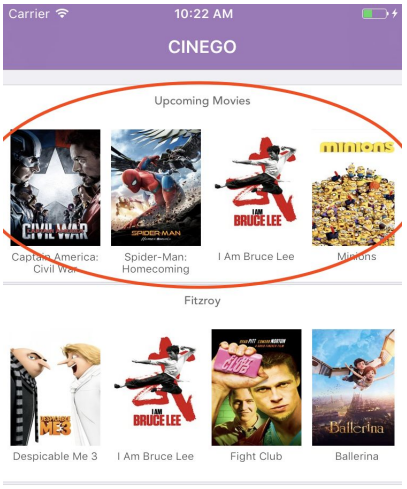
Purple vs white contrast helped give the app a ‘premium’ like feeling. Also contrast between light gray and white helps identify sections of the pages.



Repetition

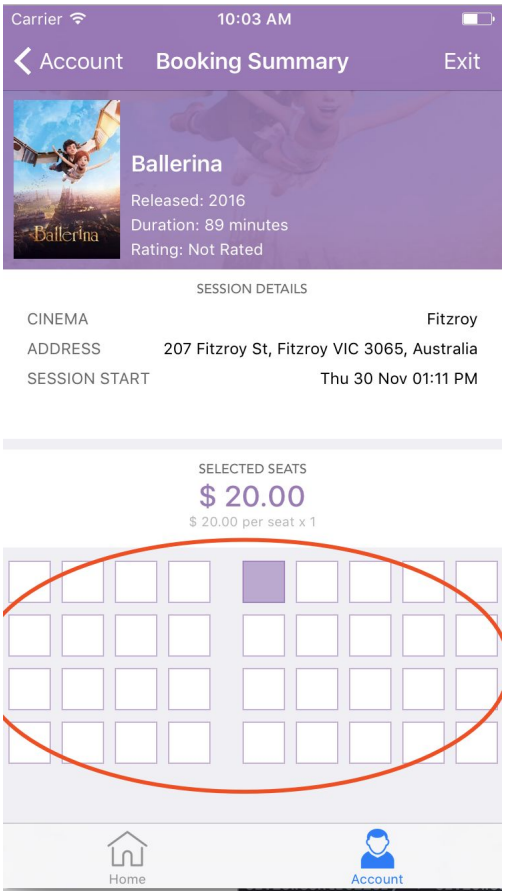
Many parts of the app share common information. The movies in the home page are organised into sections depending on the cinema
Additionally, the app contains repeating units of information. For each unit of information, we used a custom view to display its data

(eg. movie information is a unit of information. It is rendered in its own custom view called `MovieDetailsView`*)*



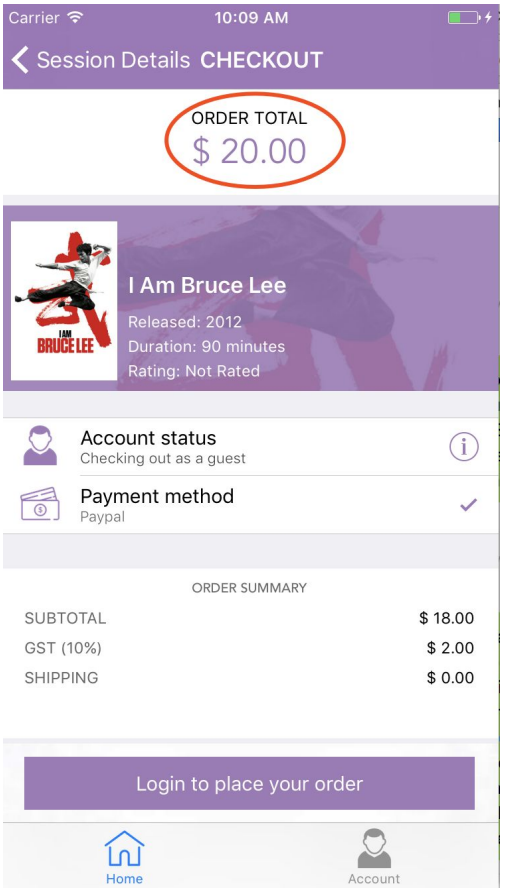
Proximity

Seating arrangement are arranged into sections. Spacing between sections indicates the isles of the theater.



Clarity

Price is considered to be an important information than other details. Therefore the total price label is given the most text size.



Depth

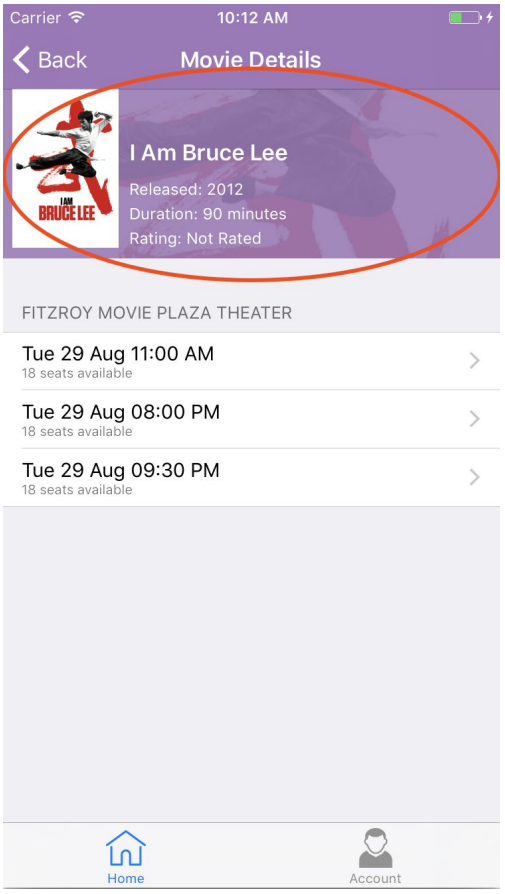
The translucency in the movie information section gives the users sense of depth. Without this translucency, the section will appear too plain.

Alignment

The app attempts to maintain constants amount of margins to help align sections and text content. Margins used are 8, 10 and 18 points

Deference

The app attempts to adopt a minimal design so that users will focus more on booking rather than its app's colour and decoration. Floating buttons at the bottom calls for action. On the home page, only all the upcoming movies and cinemas with movies are shown.



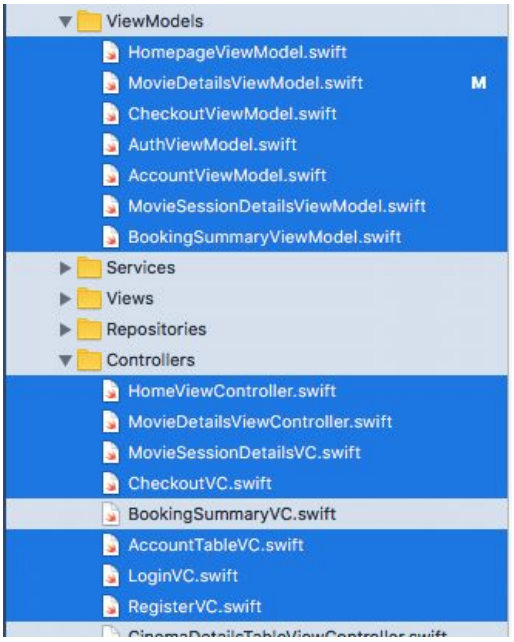
Design Patterns

MVVM (model-view-viewmodel)

The **view layer** will handle how data and state changes will be displayed on the screen. All storyboard, custom XIB files and view controllers belong to this layer.

The **model layer** represents the data source. All business entities, service files, services and repositories belong to this layer.

All data used by the view controller will be referenced on **ViewModels**. The ViewModels then get its data from the model layer.

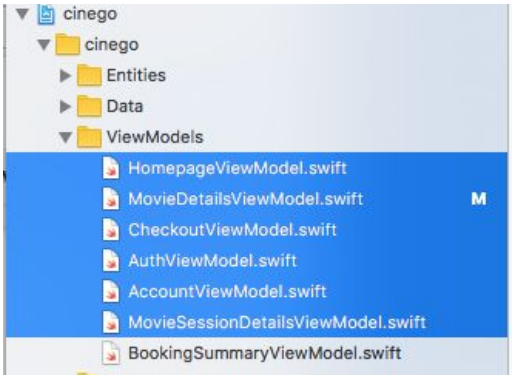



```
class LoginVC: UIViewController {

    var authViewModel: AuthViewModel! {
        didSet { self.authViewModel.delegate = sel
    }
```

```
class AuthViewModel {
    static var isSigninUp = false

    weak var delegate: AuthViewModelDelegate?
    var currentUser: User?

    var userService: IUserService
    init(userService: IUserService){
        self.userService = userService
    }

    // Checks login.
    func checkAuth() {
        userService.getCurrentUser().then {
            self.currentUser = $0
        }.always {}
    }

    // Login with email and password.
    // On successful login, userLoggedIn() me
```

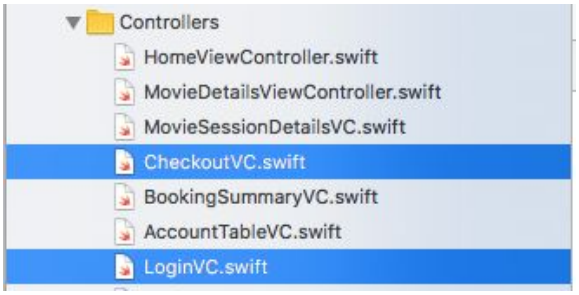
Delegation

Our project depends heavily on asynchronous operations. We used **delegates** in order to assign responsibility once an operation (eg logging in, getting data from network, or when some error occurs) completes/fails.



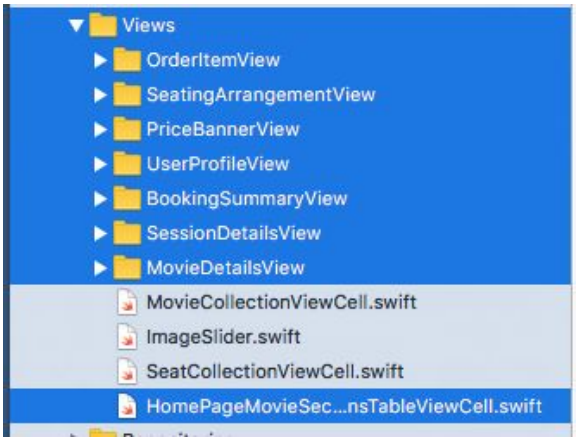
Delegation used frequently within ViewModels


```
protocol SeatingArrangementViewDelegate: class {
    func didUpdateSeats(_ selectedSeats: [Seat])
}

@IBDesignable
class SeatingArrangementView: UIView {

    weak var delegate: SeatingArrangementViewDelegate?
```



Observer

Our application states always changes. The UI has to respond to every state changes in our app. Our project uses **property observers `(didSet)`** to simplify implementation of observer pattern. Property observes are widely used within CustomViews and ViewModels



Property observers implemented in these CustomViews

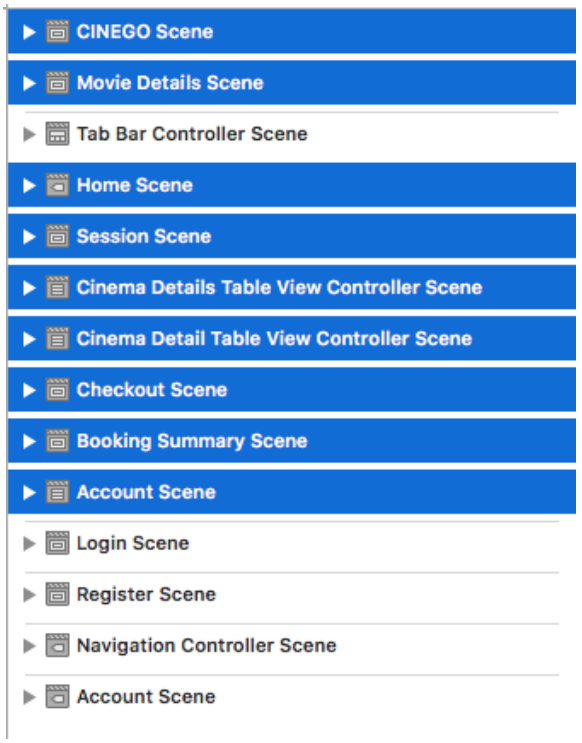
```
var booking: Booking! {
    didSet {
        let url = URL(string: booking.movieSession.movie.images.first!)!
        movieBanner.hnk_setImageFromURL(url)
        totalPriceLabel.text = String(format: "$ %.02f", self.booking.price)
        numSeatsLabel.text = String(format: "%d seats >",
            self.booking.seats.count)
        cinemaLocationLabel.text = self.booking.movieSession.cinema.name
        cinemaAddressLabel.text = self.booking.movieSession.cinema.address

        let formatter = DateFormatter()
        formatter.dateFormat = "EEE dd-MMM hh:mm aa"
        sessionStartLabel.text = formatter.string(from: self.booking.movieSession.
            startTime)
        movieTitleLabel.text = self.booking.movieSession.movie.title
    }
}
```

Property Observer

CRUD

In order to reduce network calls, new images, **cinema** and **movie** information are **INSERT**ed locally upon **exiting** the app. If the movie or cinema exist, the existing information is **UPDATE**d instead. On application **startup**, these information are **READ** from Core Data and loaded into cache.

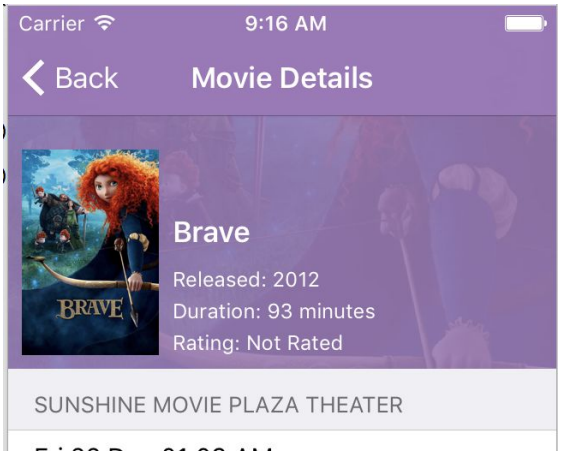
CRUD operations happen inside `AppDelegate.swift`. Scenes that make use of movie and cinema information gets their data from cache, which is loaded originally from CoreData



REST Implementation

To fetch movie information, we used **TMDB service** to implement REST in our app. Movie session and booking data are supplied by **Firebase**

All network calls to TMDB API and images are cached with the help of the library "`Haneke Swift`".



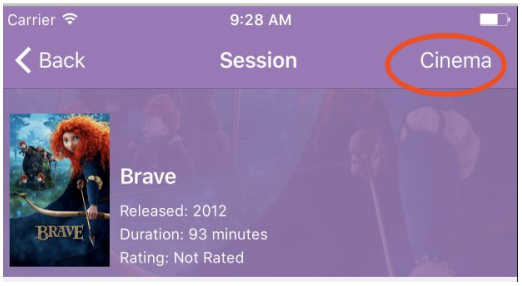

```
class TMDBMovieService: ITMDBMovieService {

    let tmdb_apikey = "8e91ab723e730b59175061f4aa1ed37c"
    let tdmb_movieUrl = "https://api.themoviedb.org/3/movie/"
    let tmdb_imageUrl = "https://image.tmdb.org/t/p/w500"

    private func movieUrl(_ id: Int) -> String {
        return "\(tdmb_movieUrl)\(String(id))?api_key=\(tmdb_apikey)"
    }

    private func posterUrl(_ posterFilename: String) -> String {
        return "\(tmdb_imageUrl)\(posterFilename)?api_key=\(tmdb_apikey)"
    }
```

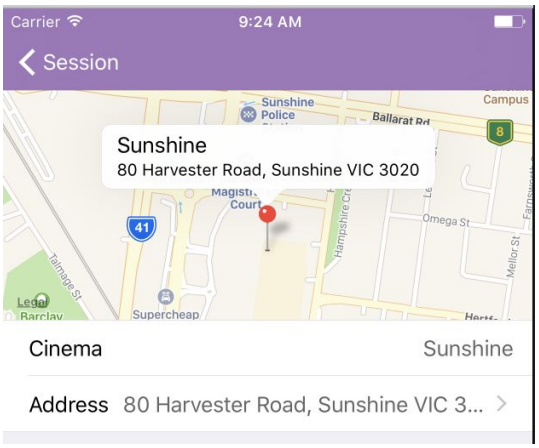
Cocoa Framework

This app implements the map for cinema. In the session page, when user clicks the “Cinema” button, he/she will be sent to “Cinema” page, showing the map of the cinema location

```
// Flinders Street as default
let coordinates = CLLocationCoordinate2D(
    latitude: cinema.latitude,
    longitude: cinema.longitude
)
let area = 500.00
let region = MKCoordinateRegionMakeWithDistance(
    coordinates,area, area)
mapView.setRegion(region, animated: false)
let annotation = CinemaLocationAnnotation(
    coordinates, title: cinema.name,
    subtitle: cinema.address)
mapView.addAnnotation(annotation)
```

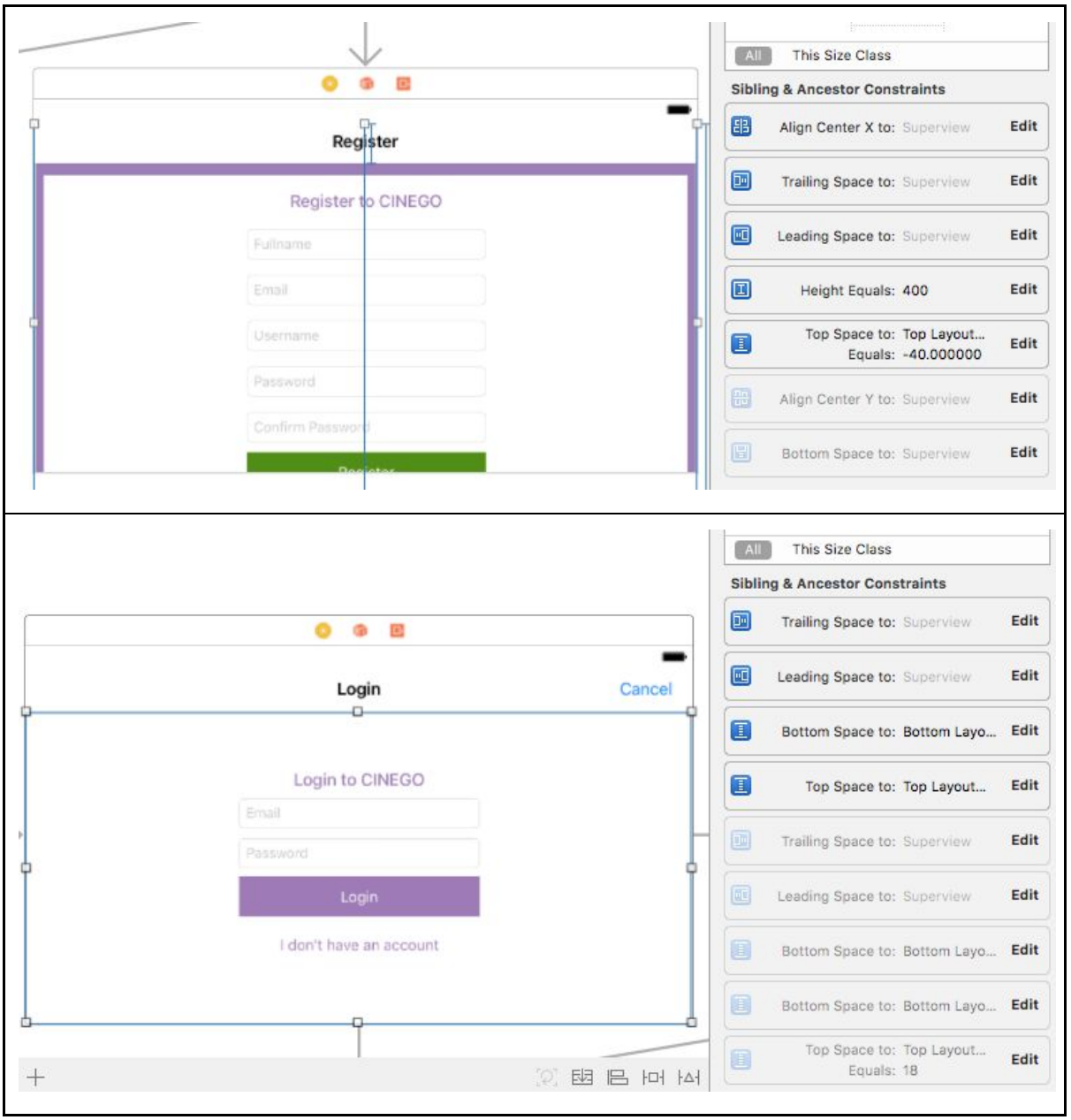
Using `MapKit`





Size classes

The login and registration page uses size classes. Other views uses mostly `TableViews` to adopt landscape format.



References

A Size Class Reference Guide by Harrison K.
https://useyourloaf.com/blog/size-classes/

Dabbling with MVVM in Swift 3 by Erica Millado
https://medium.com/yay-its-erica/dabbling-with-mvvm-in-swift-3-3bbeba61b45b

Three ways to pass data from Model to Controller by Stan Ostrovskiy
https://medium.com/ios-os-x-development/ios-three-ways-to-pass-data-from-model-to-controller-b47cc72a4336

Pure Swift MVVM by Scott Robbins
https://www.mobiledefense.com/blog/2016/02/07/pure-swift-mvvm/

iOS Design Themes from Apple
https://developer.apple.com/ios/human-interface-guidelines/overview/themes/

Dependency Injection from Wikipedia
https://en.wikipedia.org/wiki/Dependency_injection