Egypt

TOUR GUIDE

EGYPT TOUR GUIDE

BY

Karem Hazem Ahmed
Clark Wagdy Gerges
Hossam Hassan Hathout
Sama Morsy Soltan
Mohamed Elshety

SUPERVISOR

Dr: Mohammed Nasef

Egypt

TOUR GUIDE

Supervisor
Dr: Mohammed Nasef

Contents

Contents

INTRODUCTION

Introduction

Tourism is vital for the success of many economies around the world. There are several benefits of tourism on host destinations. Tourism boosts the revenue of the economy, creates thousands of jobs, develops the infrastructures of a country, and plants a sense of cultural exchange between foreigners and citizens.The number of jobs created by tourism in many different areas is significant. These jobs are not only a part of the tourism sector but may also include the agricultural sector, communication sector, health sector, and the educational sector. Many tourists travel to experience the hosting destination's culture, different traditions, and gastronomy. This is very profitable to local restaurants, shopping centers, an stores. From here we got the idea of developing Egypt Tour Guide, a website and mobile application (android- IOS) to make it easy for tourists to see Egyptian cities, places, and hotels Also to reserve a hotel And provide some plans for tourists to encourage them to visit Egypt with attractive price.

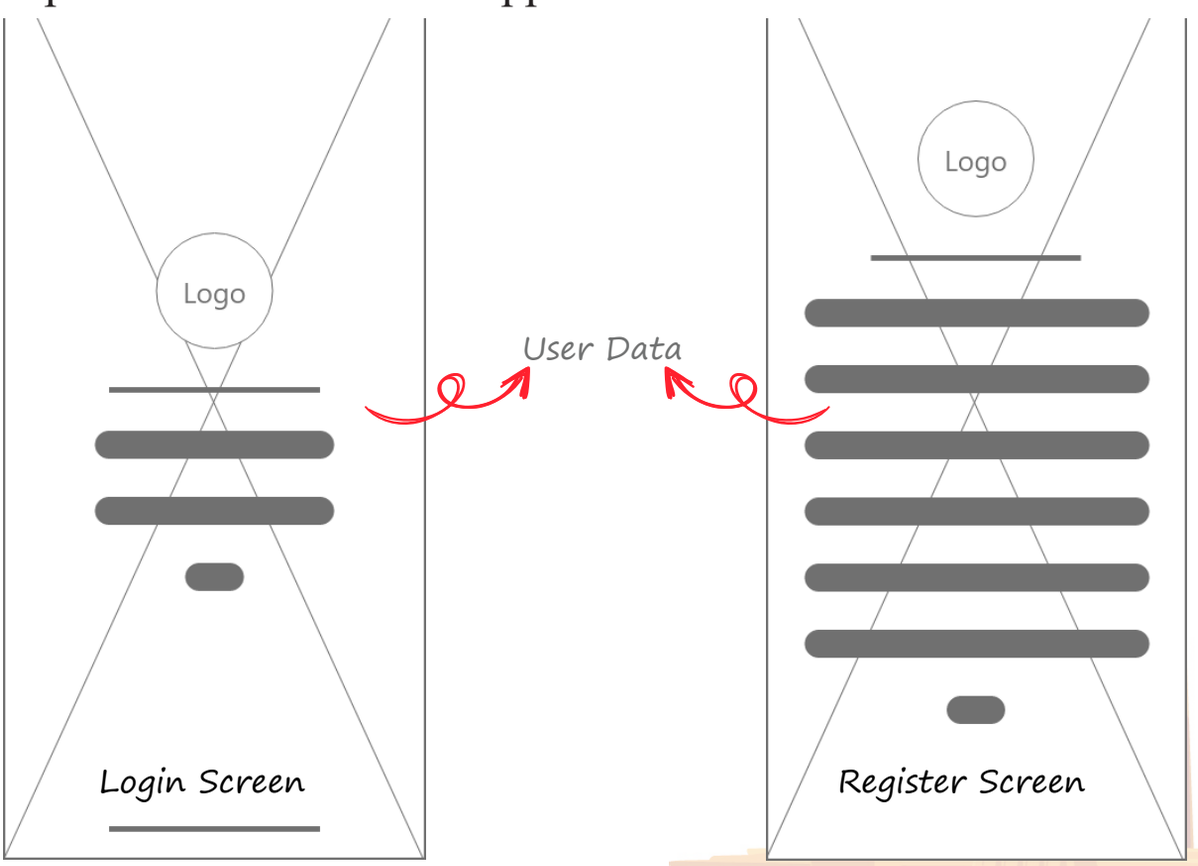
Objectives

- Showing Cities, Places, Hotels of Egypt in details
- Make it easy for tourists to reserve a hotel via mobile application or website
- Helping tourists to find the best places by rating and reviews
- Make some plans for tourists with transports and attractive price.

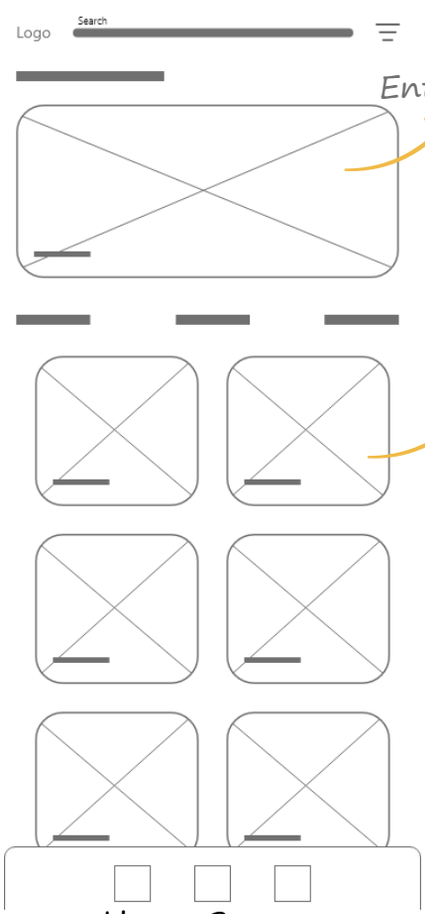
CHAPTER 1
UI/UX

UI/UX

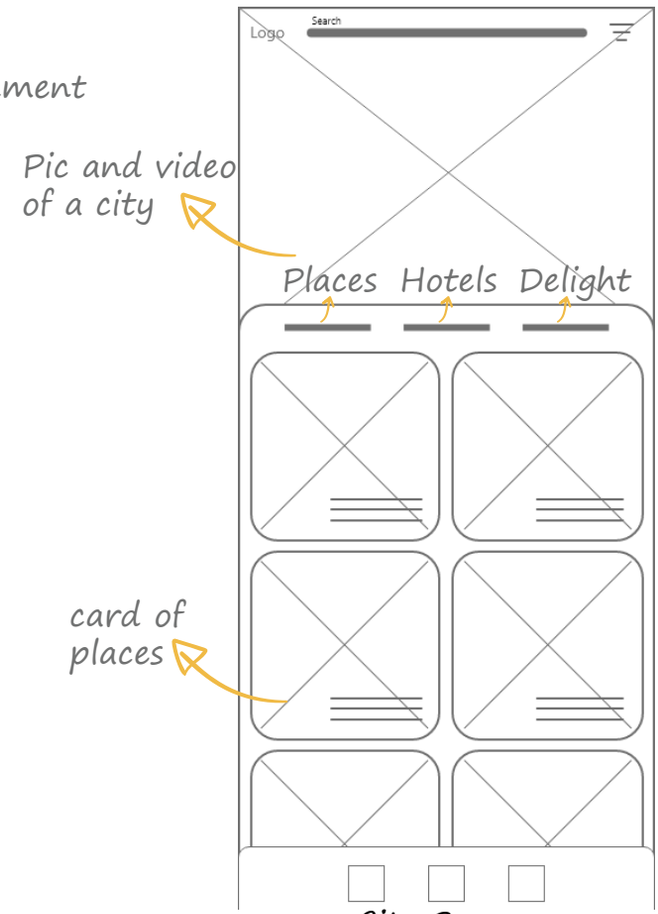
How we made the user experience easier and more comfortable? First, we started with gaining a deep understanding of users before we go on to create designs and test them with users like our friends and some people who specialize in the field of User experience. We have followed many important rules to create a site and application that is easier and more comfortable for the user such as Format the parts of each page so that the components of the page are easy to understand for the user dealing with the site or application is easier and We have taken into account the time it takes for the user to reach their goal from the site or app, and we have made sure that this time taken does not exceed seconds. Some illustrations for user experience in the mobile application.



UI/UX

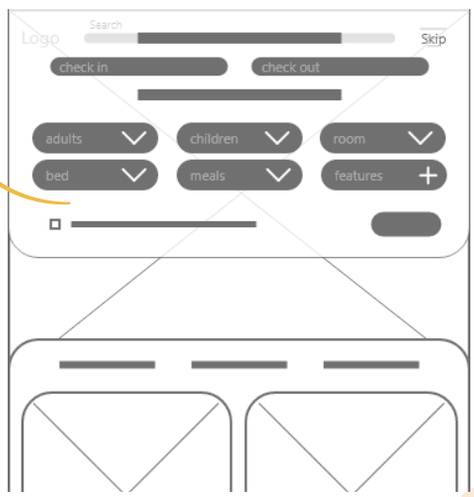


Home Screen



City Screen

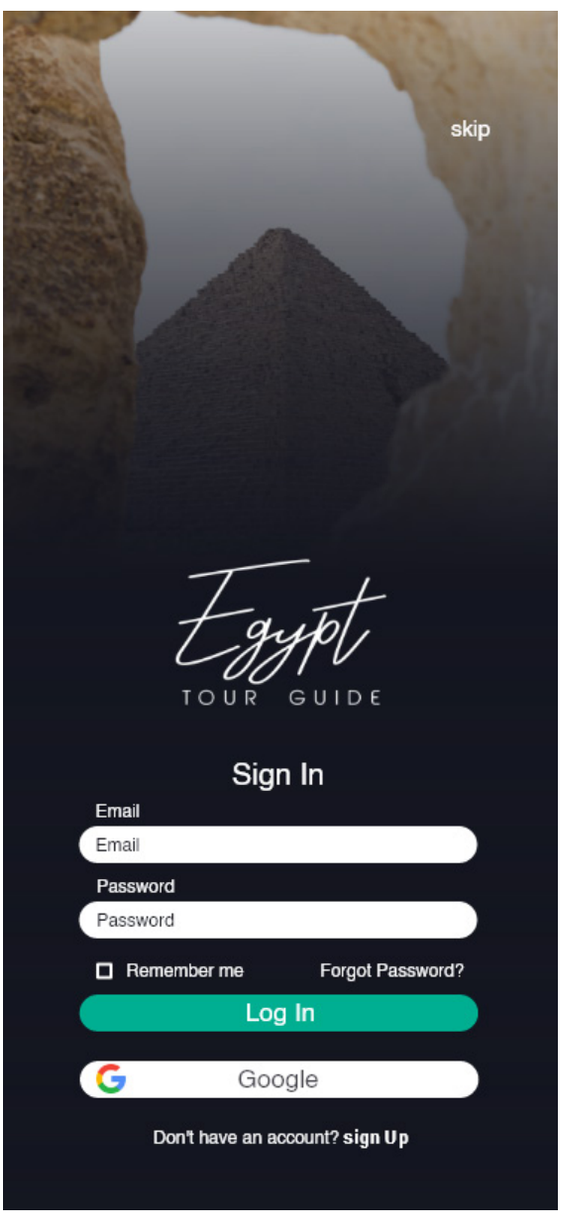
Data entered by the user to show everything that suits the user



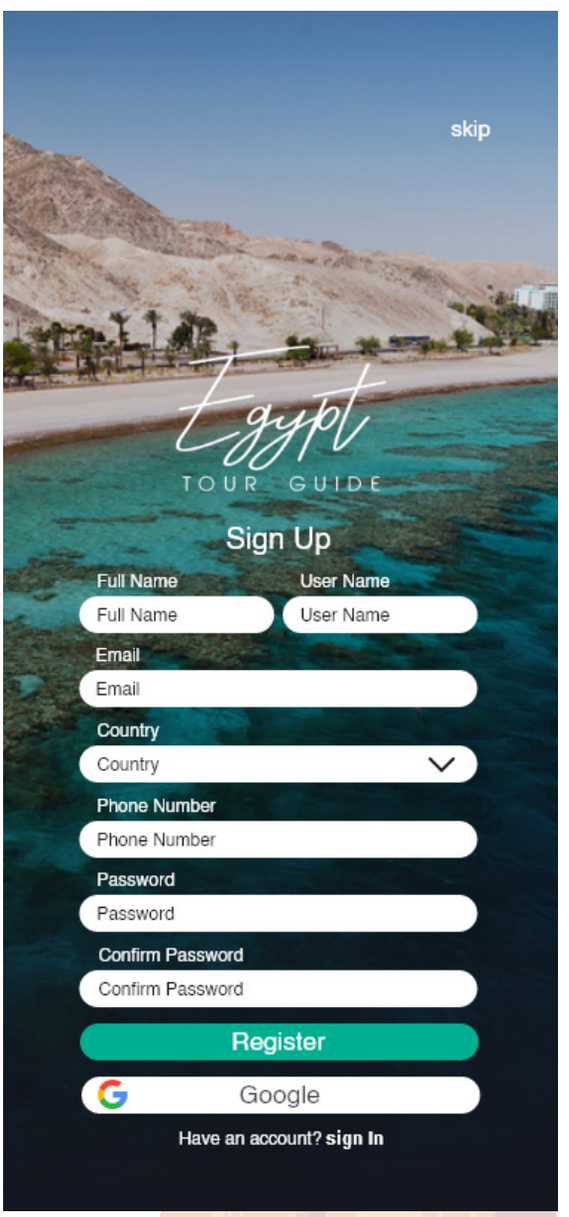
City Filter

UI/UX

How we made the user interface more comfortable?
Choose colors very carefully to be more comfortable for the user's eyes and we have been keen to choose the images with great care so that they are expressive to the user and through them, the user can be an imaginary image of the tourist area or recreational activities or about the hotel and hotel rooms. Some illustrations for user interface in the mobile application.

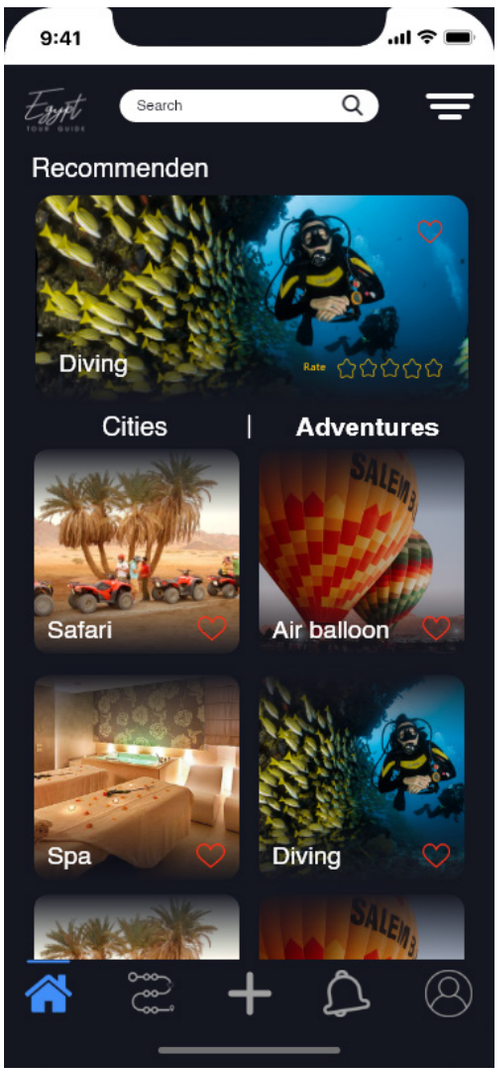


Login Screen

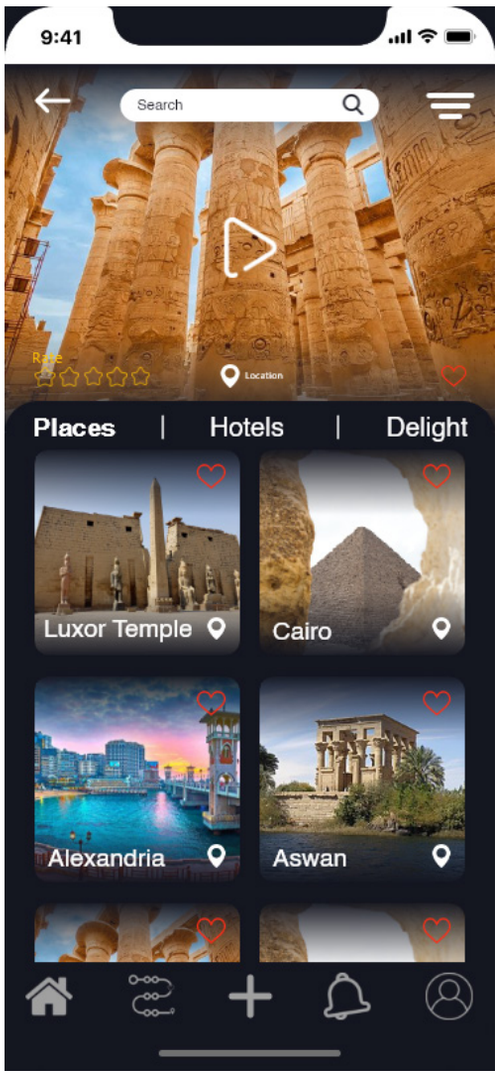


Register Screen

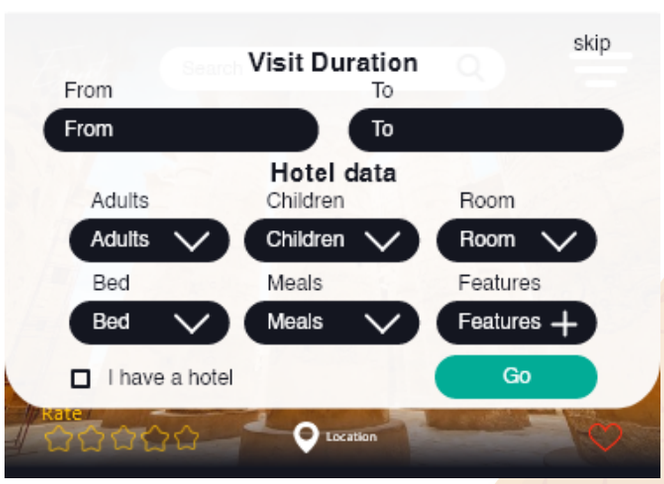
UI/UX



Home Screen



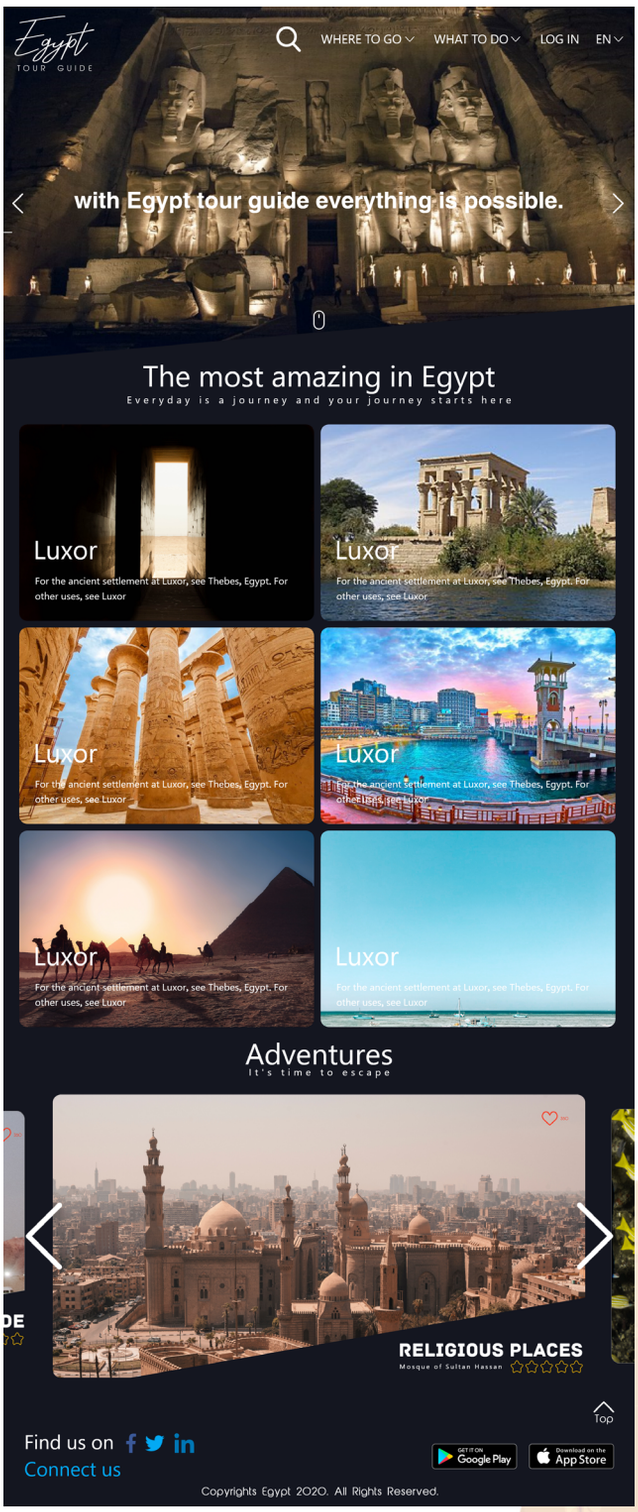
City Screen



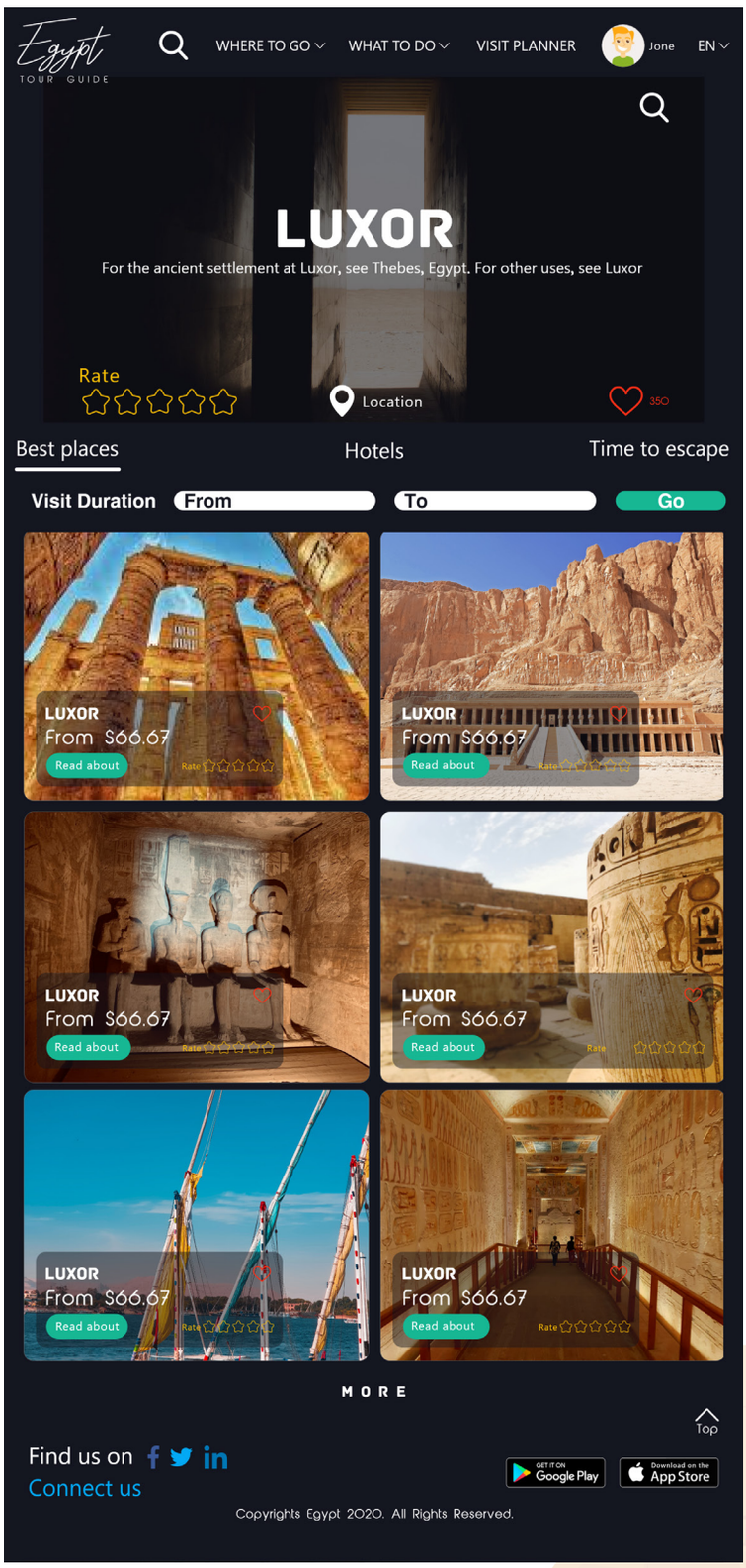
City Filter

UI/UX

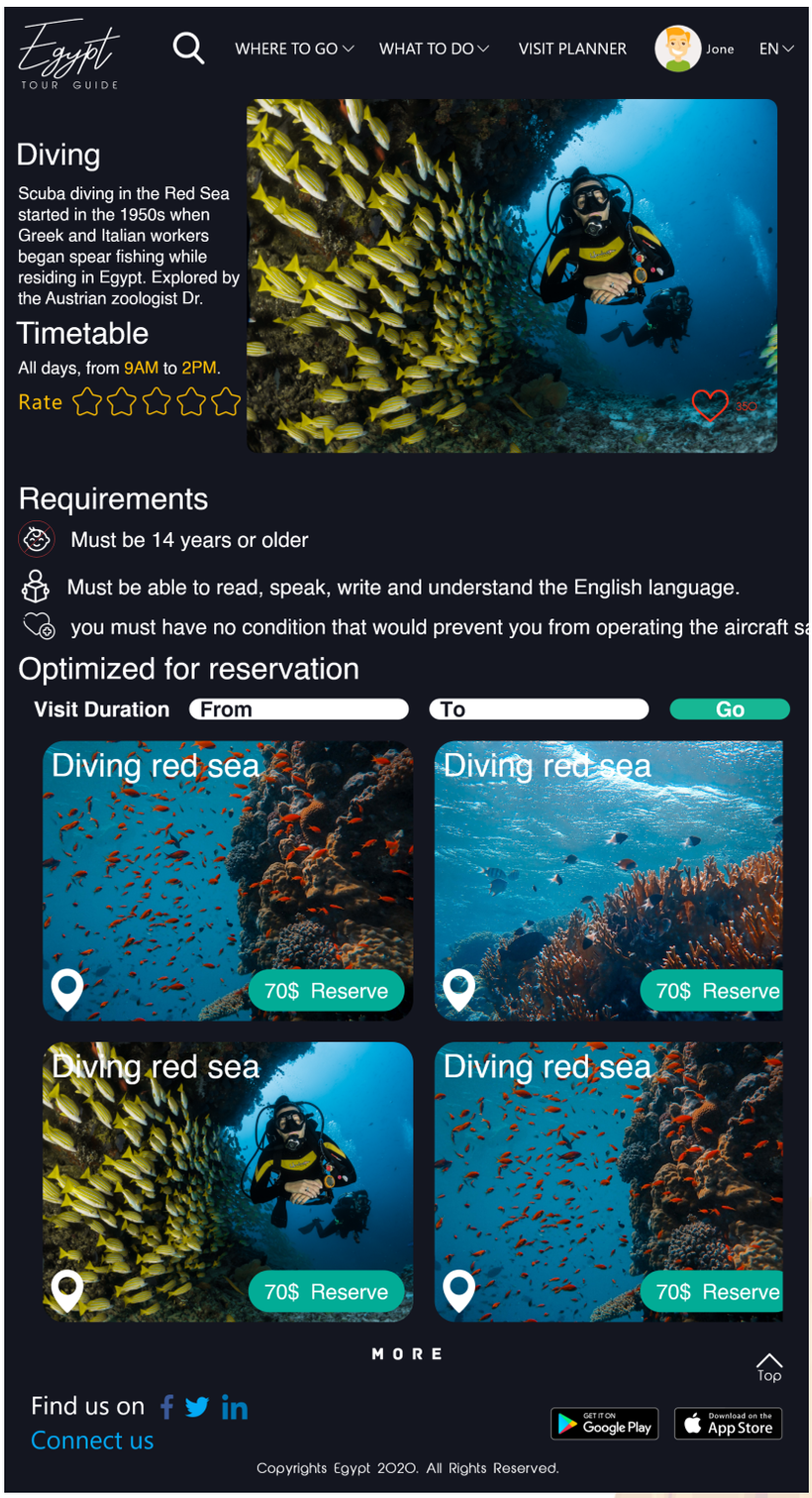
Some illustrations for user interface in the web application.



Home Screen


Egypt
TOUR GUIDE
WHERE TO GO
WHAT TO DO
VISIT PLANNER
Jone
EN
LUXOR
For the ancient settlement at Luxor, see Thebes, Egypt. For other uses, see Luxor
Rate
Location
350
Best places
Hotels
Time to escape
Visit Duration
From
To
Go
LUXOR
From $66.67
Read about
MORE
Top
Find us on
Connect us
GET IT ON Google Play
Download on the App Store
Copyrights Egypt 2020. All Rights Reserved.

City Screen



Delight Screen

CHAPTER 2
Back-end

Back-end

Egypt Tour Guide API.

It's a NodeJS API (a JavaScript runtime built on Chrome's V8 JavaScript engine) with express framework.





We will discuss in this Chapter:

About Project, Some Routes (How they designed, Admin routes and Pagination), Authentication and Authorization, Static Media Files, AWS S3 (file upload and local files), Data and Relations and Hotel Room Request and Tour Checking Availability Algorithm.

About Project

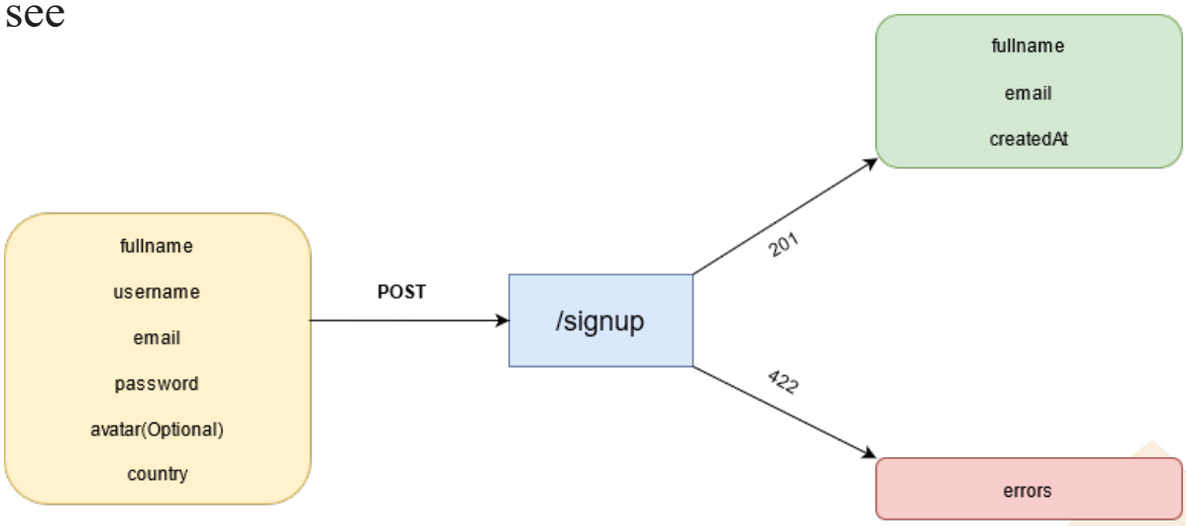
A small Egyptian Tourism Company playing a role in showing the Cities of Egypt and some places of each city also some hotels and it have its Transport and Tours for Tourists, Tourists could reserve a hotel or a plan (Tour or Tour with the hotel) via Egypt Tour Guide Website or mobile application.

Some Routes

In the Beginning, we will see inside this middleware that we are logging each request IP and Requested URL to make it easy for us to find out what is going on

```
35  app.use((req,_,next)=>{
36      console.log(`-----------------------------------`)
37      console.log(`From: ${req.ip}`)
38      console.log(`URL: ${req.url}`)
39      console.log(`Method: ${req.method}`)
40      next()
41  })
42
```

First Route we will talk about it's the signup route as we can see



It takes firstname, username, email, password, Tourist picture as avatar and country Username is unique identifier same as email, Tourist can sign in with username or email with the password we validate the username to be as ^[a-z]{3,}[0-9]*$. Passwords hashed and salted before storing information into the database.

Back-end

The Login End-Point is

POST > /login

With username and password, if the information was correct will return back with the response some info contains Token and Refresh token, we will discuss the Authentication and Authorization in details, but for now we will walk across some other routes

GET > /cities

Will return all cities in the database but with query page we will have pagination, 4 cities as we can see

GET > /cities?page=1

To get a specific city

GET > /cities/id

Each city object contains id, name, description, media, location

GET > /places

Will return in response all places inside all cities

Back-end

GET > /places?city=id

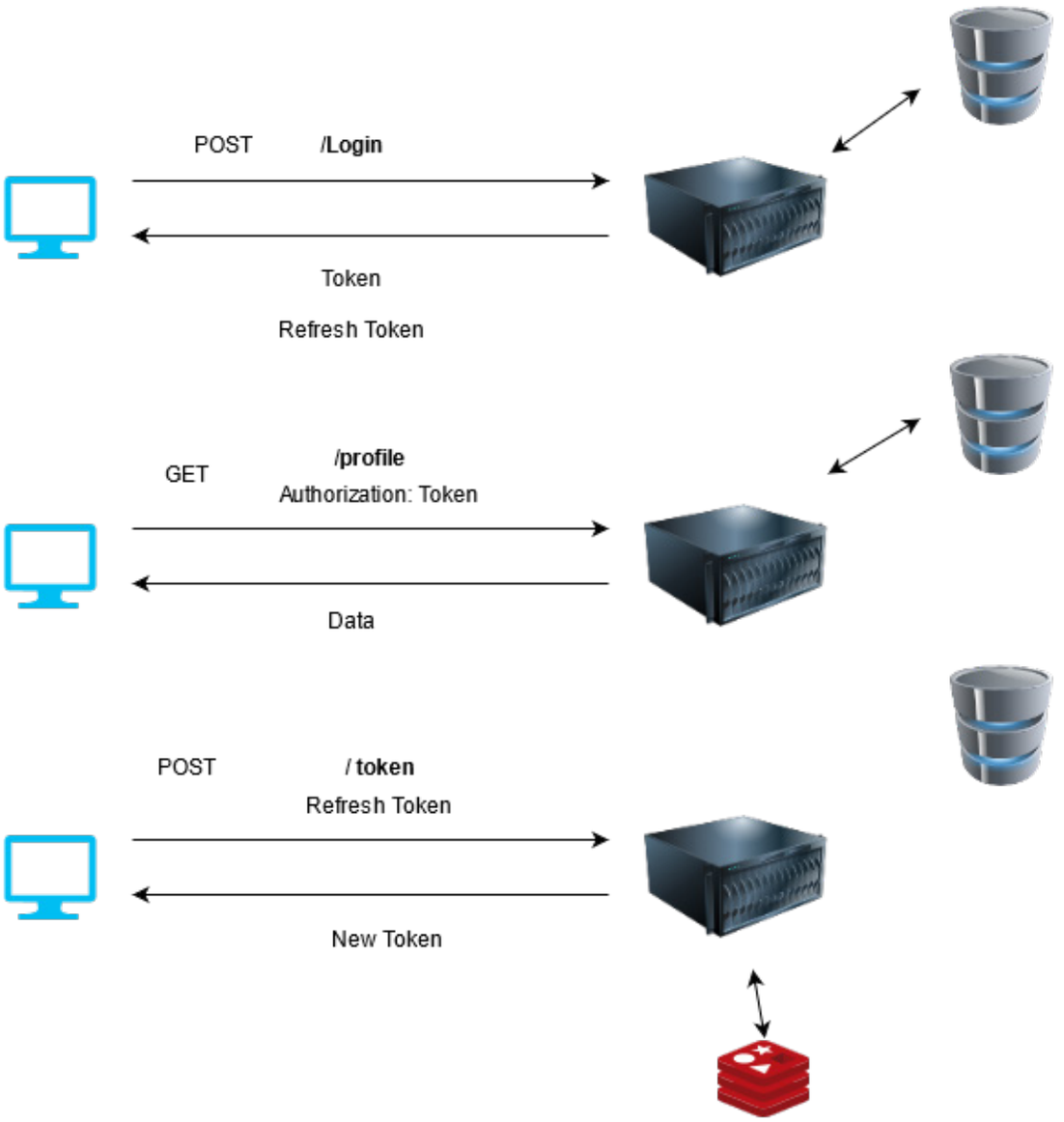
To show places inside specific city.
Admin have the role of adding, updating cities or places or hotels as we can see
admin uses this end point for adding a city

POST > /cities

{ name, description, media, location }
Same as places, hotels but with more data.

Authentication and Authorization
For authentication we covered the login end-point and we said that in the response if the data was correct will return back some information contains JWT token and refresh token.So what is What is JSON Web Token? JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

Back-end


POST
/Login
Token
Refresh Token
GET
/profile
Authorization: Token
Data
POST
/ token
Refresh Token
New Token
POST
/logout
Refresh Token

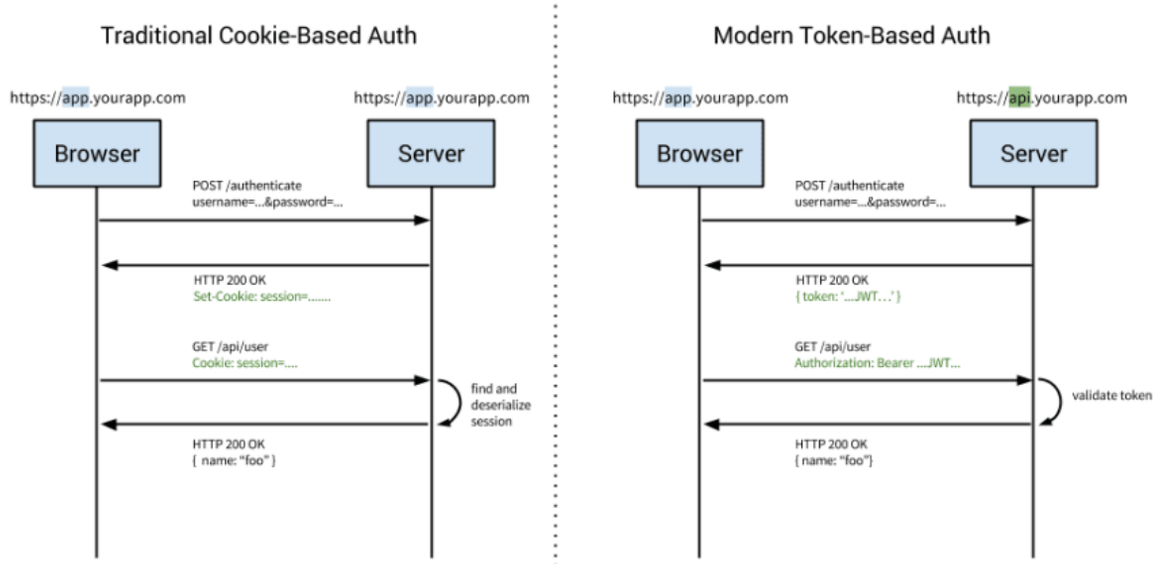
Back-end

controllers > AuthController > JS AdminAuth.js > ...

```
const jwt = require("jsonwebtoken")
const { ADMIN } = require('../../utility/userTypes')
const AdminAuth = (req,res,next)=>{

    //Get token from authorization header
    const token = req.headers['authorization']

    if(!token) return res.sendStatus(403)

    // Verify Token and Check for userType
    jwt.verify(token, process.env.TOKEN_SCERET, (err, user)=>{
        if(err) return res.sendStatus(403)
        else if(user.userType !== ADMIN) return res.sendStatus(403)
        else{
            req.user = user
            next()
        }
    })

}

```

Authorization middleware check for the jwt inside the authorization header and the jwt contains the id of the tourist or admin and userType if admin or tourist, so some routes check if the user has the permission to access this route or not.As we see inside this image the route /profile is protected and you should send inside authorization header a valid token, if the token expired you can use refresh token to get another valid token.So what is the difference between jwt and session? The biggest difference here is that the user's state is not stored on the server, as the state is stored inside the token on the client side instead. Most of the modern web applications use JWT for authentication for reasons including scalability and mobile device authentication that's why a single token can be used with multiple backends

Back-end



Static Media Files, AWS S3

When we create a new city, place, .. or when user signup with his picture, server store this media offline on the hard disk on the application files /data/media, And store the profile picture under /data/avatar, Filter the files to validate type of the files and the allowed file types are png, jpg, jpeg, mp4 for media only.

```
// Maximum File Size (50 MB)
const maxSize = 50 * 1024 * 1024

//Validating file type ( jpg, png, jpeg, mp4 )
const fileFilter = (req, file, callback)=>{
    if( file.mimetype === 'image/png' || file.mimetype === 'image/jpg' || file.mimetype === 'image/jpeg'
        callback(null, true)
    else
        callback(new Error('Only (png, jpg, jpeg, mp4) files are allowed!'), false)
}

const storage = multer.diskStorage({
    //where to store
    destination: (req, file, callback)=>{
        //Global dir
        let dir = path.join(__dirname , '../', '../', 'data', 'media')
        if(file.mimetype === 'video/mp4'){
            //Dir for videos
            dir += path.join('/videos')
            if(!fs.existsSync(dir)){
                fs.mkdirSync(dir, { recursive: true }, function(err) {
                    if (err) {
                      console.log(err)
                    }
                  })
            }
```

Back-end

```
// Maximum Image Size (3 MB)
const maxSize = 3 * 1024 * 1024

//Validating file type ( jpg, png, jpeg )
const fileFilter = (req, file, callback)=>{
    if( file.mimetype === 'image/png' || file.mimetype === 'image/jpg' || file.mimetype === 'image/jpeg')
        callback(null, true)
    else
        callback(new Error('Only image files are allowed!'), false)
}

const storage = multer.diskStorage({
    //where to store images
    destination: (req, file, callback)=>{
        callback(null, 'data/avatar')
    },
    //generate image name with date and file type
    filename: (req, file, callback)=>{
        callback(null, Date.now() + '-' + file.mimetype.replace('image/','.'))
    },
})

const upload = multer({storage, limits: { fileSize: maxSize }, fileFilter}).single('avatar')

```

And this for the avatar, but when we become online it will not be a good way to store this files on the host storage
So we decided to use AWS S3 Bucket and uploading this images and media files on the cloud

```
const uploadAvatarS3 = ({filename, filepath})=>{
    return new Promise((resolve, reject) => {

        const params = {
            Bucket: process.env.BUCKET,
            Key: `public/avatar/${filename}`,
            Body: fs.createReadStream(filepath)
        }

        s3.upload(params, (error, data)=>{
            if(error) reject(error)
            else
                resolve(data)
        })

    })
}
```

Server is Online at: https://egypttourguide.herokuapp.com/

Data and Relations

User	
PK	**UserID**
	Name
	Username
	Email
	Password
	lastLogin
	Picture
	userType

Tourist	
PK	**TouristID**
FK	TourID
FK	NotificationID
	Country
	Rate
	phone
	warns
	isActive
	favourites

Notification	
PK	**NotificationID**
	Title
	Content
FK	TouritsID

Admin	
PK	**AdminID**
FK	UserID

TouristPlan	
PK	**TouristPlanID**
FK	PlanID
	startDate
	endDate
	Person
FK	HotelID
FK	TouristID
FK	TransportID
	price

Plan	
PK	**PlanID**
FK	CityID
	Title
	Features
	Tour
	Ticket
	Duration
	media
	Rate
FK	TouristPlanID
FK	RateFormID

General	
PK	**GeneralID**
	Name
	Location
	Description
	Media

City	
PK	**CityID**
FK	GeneralID
FK	HotelID
FK	PlaceID
FK	PlanID
FK	TransportID

Place	
PK	**PlaceID**
FK	GeneralID
	Ticket
	Rate
	Available
	Hours
FK	RateFormID

Transport	
PK	**TransportID**
	Type
	Seats
	DriverName
	DriverPhone
	Hours

Back-end

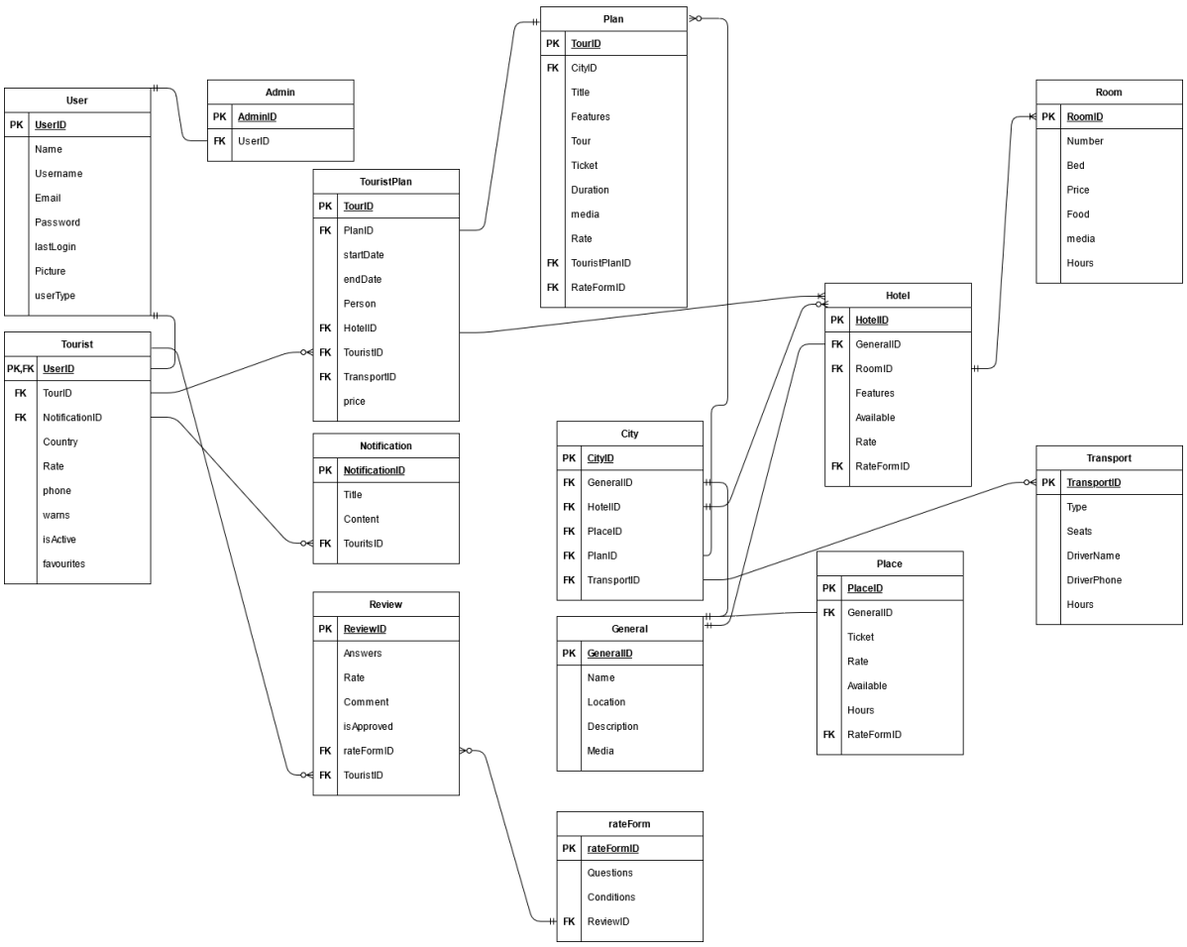
Hotel	
PK	**HotelID**
FK	GeneralID
FK	RoomID
	Features
	Available
	Rate
FK	RateFormID
	Requests

Room	
PK	**RoomID**
	Number
	Bed
	Price
	Food
	media
	Hours

rateForm	
PK	**rateFormID**
	Questions
	Conditions
FK	ReviewID

Review	
PK	**ReviewID**
	Answers
	Rate
	Comment
	isApproved
FK	rateFormID
FK	TouristID

Back-end



Hotel Room Request and Tour Checking Availability Algorithm

When we request hotels inside a city which contains available rooms we use this e nd point

GET > /hotels?city=id&from=date&to=another date

We can add query bed=1 to get all available hotels from date to another date that contains at least one room with one bed.

From the data and relations we saw that room has a hours attribute, Hours is an array of objects each object contains from, to, touristId variables.

Back-end

So when we want to find if the room is available from & to a specific date, we can simply check if the date we want to reserve is available for every object in hours

```
const validRoomsWithDate = (rooms, from, to)=>rooms.filter(room=>{
    function isValidRoom(el,index,arr) {
        let roomFrom = new Date(el.from)
        let roomTo = new Date(el.to)
        let myDateFrom = new Date(from)
        let myDateTo = new Date(to)

        if ((roomFrom.getTime() > myDateTo.getTime()) || (roomTo.getTime() < myDateFrom.getTime())){
          return true
        }
        else {
          return false
        }
      }
    if(room.hours.length === 0 || room.hours.every(isValidRoom))
        return true
    else
        return false
})
```

If we found available hotel room and we want to send a request to reserve POST > /hotels/:id/request {from, to, roomId} From data and relations Plan has a Tour attribute, Tour is an array of objects each object contains day, from, to, placeId variables also each place has its own hours array, the problem here is when tourist request this plan on a specific day and the tour inside the plan has place and place contains it's opening hours so we need to check for the date (tourist choose to start his tour) whether if places are available in that day in this specific hour at the tour or not, also the transport has hours attribute we need to find the available transport and if the plan duration more that one day we could reserve a hotel for this tourist we need to find the well-rated hotel available.

Back-end

So first if the plan duration was for one day only we won't need a hotel and just need to check for availability of the places at this time with available transport. End-point for checking availability of the plan is.
POST > /plans/id/check {start (Ex: 2021-07-13), persons}

```
//Duration for one day
if(plan.duration.days === 0){
    //Check Places Hours
    let dayTime = plan.tour.map(t=>({from: t.from, to: t.to, placeHour: t.placeId.hours.find(h=>h.day === startDayName)}))
    if(dayTime.filter(t=>t.placeHour === undefined).length > 0){
        return res.status(200).json({available: false, msg: `Some places are not available at ${startDayName}`})
    }
    if(!dayTime.every(t=>t.from >= t.placeHour.from && t.to <= t.placeHour.to)){
        return res.status(200).json({available: false, msg: `Some places are not available at ${startDayName}`})
    }
    //Check Transport
    let transports = await Transport.find().sort({"seats": 1})
    let validTransport = transports.filter(t=>{
        return t.history.every(h=>new Date(h.from).getTime() > startDate.getTime() || new Date(h.to).getTime() < startDate
    })
    let pTransport = validTransport.find(t=>t.seats === parseInt(persons))
    if(!pTransport){
        pTransport = validTransport.find(t=>t.seats > parseInt(persons))
    }
    if(!pTransport) return res.status(200).json({available: false, msg: `Cannot find Transport for This time!`})
    if(tourist.country){
        let egyptian = parseInt(plan.ticket.egyptian) * parseInt(persons)
        let foreign = parseInt(plan.ticket.foreign) * parseInt(persons)
        TotalPrice = tourist.country.trim().toLowerCase() === 'egypt' ? egyptian+' EGP' : foreign + ' USD'
    }else{
        TotalPrice = 'Please Set Your Country first!'
```

As we can see first we will check for the start day name inside hours in every place inside the tour so for example:If we want to start the plan at 2021-07-13 which is a Tuesday We should check every place inside the tour if it will be available at Tuesday at the time which in the tour If we have a tour like: place: 1, from: 08:30AM to 10:30AM Place: 2 from: 10:30AM to 12:00PM the place number 1 should be available at Tuesday from 08:30AM to 10:30AM Second we should find suitable transport from the image above at line 52

Back-end

We get a sorted transport by seats because if the persons was 5 and we have 2 cars are available first is 4 seats and second is 6 seats we will take the second one And of course if we have a 5 seats available car we will take it. If the duration is more than one day and the tourist want us to reserve a hotel for him, we will find an available room takes the 5 persons.

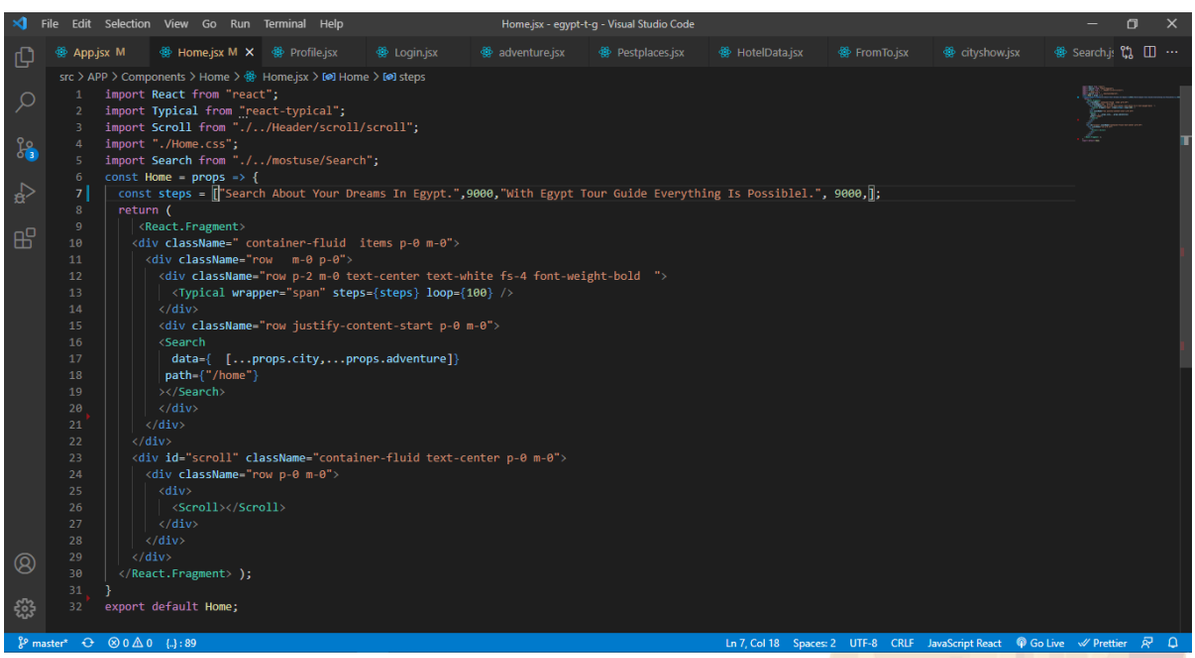
```
let result = await City.findById(plan.cityId).select({"hotels": 1, "_id": 0}).populate({
    path: 'hotels',
    module: 'hotel',
    options: { sort: {'rate': -1} },
    populate: [
            {
                path: 'generalId', module: 'general'
            },
            {
                path: 'roomId',
                module: 'room',
                match: {bed: persons},
                options: { sort: { 'price': 1 } }
            }
    ]
})
hotels = result.hotels.filter(h=>h.roomId.length > 0)
if(!(hotels.length > 0)) return res.status(200).json({available: false, msg: 'Sorry, We Did\'t find a hotel for
foundHotels = hotels.map(hotel=>{
    return ({
    id: hotel._id,
    name: hotel.generalId.name,
    media: hotel.generalId.media,
```

First we sorting hotels by rate and find a room could take the 5 persons and the last thing to check if room hours are available from the start date to the start date plus the duration of the plan.

CHAPTER 3

Front-End

In the front-End molecule, we took into account creating a website with the latest technology to create an enjoyable user experience characterized by speed and the best possible performance. In the front end molecule, we took into account creating a website with the latest technology to create an enjoyable user experience characterized by speed and the best possible performance. We have used many modern technologies in order to create a high-quality website, for example, we use Reat.js in order to create a single page application that is



CHAPTER 4
ANDROID APP

Android Application

Why React Native ?? Programming mobile applications using web technologies is possible through one of the following three ways: A web application inside a WebView or mini browser by using: PhoneGap/Cordova or the famous Ionic framework. Or by translating the application naturally through techniques such as: Haxe, Xamarin or RubyMotion. Or by using the platform's javascript engine, for example: Titanum, NativeScript or React Native that fall within this category React Native differs from other technologies in terms of executing the code in bridge, to avoid the user interface getting stuck. This feature is possible thanks to the asynchronous communication between the natural code and the JavaScript code. React Native is a framework for programming natural applications using JavaScript that reuses the natural components of the platform. This framework enables the creation of a high-level user experience through software tools based on javascript and the React library. The goal of using React Native is to make it easier for developers across all mobile platforms with the motto Learn Once, Write Everywhere. It is noteworthy that Facebook uses React Native in many applications and is still using it for one now. Is React Native a Hybrid Platform? React Native is not a hybrid platform. Therefore, it is not considered as an application within a WebView or as a framework based on HTML5. And it's not even a rewrite of the iOS or Android components as it is for the Ionic framework. React Native enables you to write native code. That is why the programming process is very similar to that used in programming web applications

Android Application

Home Page(1) 1-Stack-Navigator Provides a way for your app to transition between screens where each new screen is placed on top of a stack. By default the stack navigator is configured to have the familiar iOS and Android look & feel: new screens slide in from the right on iOS, fade in from the bottom onAndroid. On iOS the stack navigator can also be configured to amodal style where screens slide in from the bottom.

```
const Homes = (props) => {
  return (
    <Stack.Navigator screenOptions={{ headerShown: true }}>
      <Stack.Screen name='Home' component={Home} />
      <Stack.Screen name='City' component={City} />
      <Stack.Screen name='Place' component={Place} />
      <Stack.Screen name='Review' component={Review} />
      <Stack.Screen name='ReviewHotel' component={ReviewHotel} /
      <Stack.Screen name='Hotel' component={Hotel} />
      <Stack.Screen name='Room' component={Room} />
      <Stack.Screen name='Ticket' component={Ticket} />
    </Stack.Navigator>
  )
```

2-Material Top Tabs Navigator A material-design themed tab bar on the top of the screen that lets you switch between different routes by tapping the tabs or swiping horizontally. Transitions are animated by default. Screen components for each route are mounted immediately.

```
const Tabs = (props) => (
  <Tab.Navigator
    swipeVelocityImpact={0.5}
    swipeEnabled={true}
    tabBarOptions={{
      activeTintColor: "#fff",
      inactiveTintColor: 'gray',
      style: { backgroundColor: 'rgba(0,0,0,0.0)' }
    }}
  >
    <Tab.Screen name="Cities" component={Cities} />
    <Tab.Screen name="Advanture" component={Advanture} />
  </Tab.Navigator>
)
```

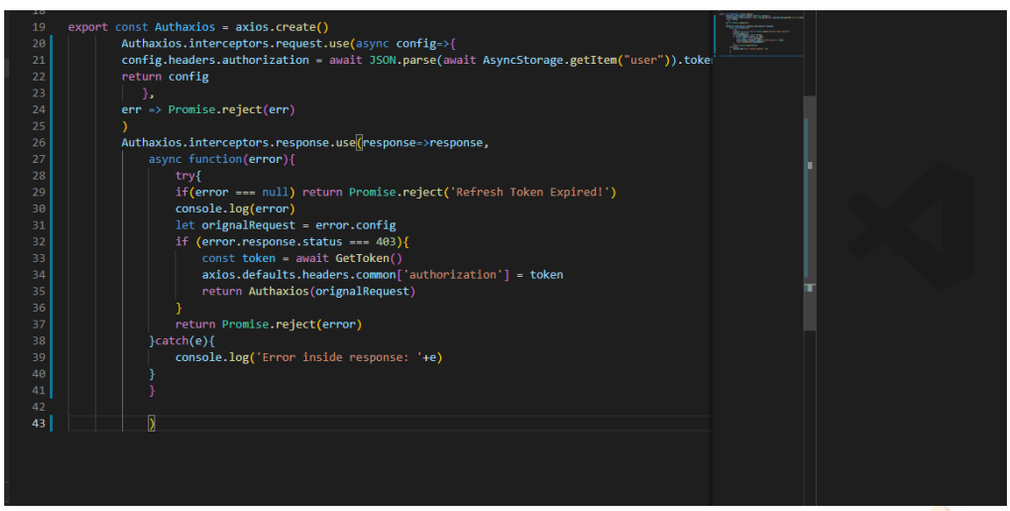
3-ApI get token & Axois

Axios is a promise-based HTTP Client for node.js and the browser. It is isomorphic (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js http module, while on the client (browser) it uses XMLHttpRequests.

Features - Make XMLHttpRequests from the browser -Make http requests from node.js -Supports the Promise API -Intercept request and response -Transform request and response data -Cancel requests -Automatic transforms for JSON data

Client side support for protecting against XSRF

Authaxios



```
export const Authaxios = axios.create()
        Authaxios.interceptors.request.use(async config=>{
        config.headers.authorization = await JSON.parse(await AsyncStorage.getItem("user")).toke
        return config
           },
        err => Promise.reject(err)
        )
        Authaxios.interceptors.response.use(response=>response,
            async function(error){
                try{
                if(error === null) return Promise.reject('Refresh Token Expired!')
                console.log(error)
                let orignalRequest = error.config
                if (error.response.status === 403){
                    const token = await GetToken()
                    axios.defaults.headers.common['authorization'] = token
                    return Authaxios(orignalRequest)
                }
                return Promise.reject(error)
            }catch(e){
                console.log('Error inside response: '+e)
            }
            }

            )
```

login & logout

```
login: (newUser)=>{
            console.warn(newUser)
            this.setState({user: newUser})
        },
        logout: async()=>{
            try{
                await AsyncStorage.removeItem('user')
                this.setState({user: null})
            }catch(e){
                this.setState({user: null})
                console.log(e)
            }
        }
        }}
```

4-context Api

Context API is a Re can solve a lot of problems that modern applications face related to state management and how they're passing state to their components. Instead of installing a state management library in your project that will eventually cost your project performance and increase your bundle size, you can easily go with Context API and be fine with it.

Why Context API?

One of the concepts of React is to break your application into components, for reusability purposes. So in a simple React application, we have a few different components. As our application grows, these components can become huge and unmaintainable, so we break them into smaller components. That's one of the best concepts about React—you can create a bunch of components and have a fully maintainable and concise application, without having to create a super huge component to deal with your whole application. After breaking components into smaller components for maintainability purposes, these small components might now need some data to work properly. If these small components need data to work with, you will have to pass data through props from the parent component to

save user.

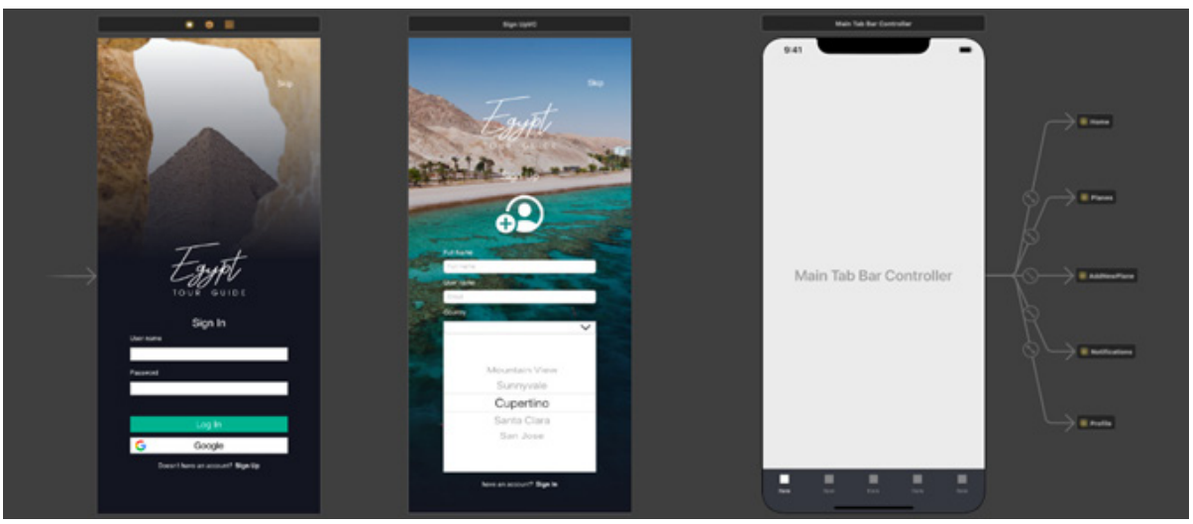
Auth > AuthProvider.js > AuthProvider > render

```
import React, { createContext, useState } from 'react'
import AsyncStorage  from '@react-native-async-storage/async-storage'
export const AuthContext = createContext({
    user: null,
    login: ()=>{}, logout: ()=>{}, saveUser: async()=>{}
})
export class AuthProvider extends React.Component{
    state = { user: null }
    render(){
    return(
        <AuthContext.Provider
        value={{
            user: this.state.user,
            saveUser: async(newUser)=>{
                try{
                    console.warn(newUser)
                    await AsyncStorage.setItem('user', JSON.stringify(newUser))
                    this.setState({user: newUser})
                }catch(e){
                    console.log(e)
                }
            },
            login: (newUser)=>{
                console.warn(newUser)
                this.setState({user: newUser})
            },
```


CHAPTER 5
IOS APP

IOS Application

-For developing the application I used UIKit. UIKit is a framework that provides the view architecture for implementing the interface. It includes the UIComponents for the design and also foundation library. -For the design I used storyboard and coding. I mainly relied on the storyboeard and the UIComponent for the design.



-As a design pattern I used MVC.

-For networking and API calls I used Alamofire and SwiftyJson.

```
func post_signin() {

    startLoading()

    guard let url = URL(string: "https://egypttourguide.herokuapp.com/login") else {return}

    let header = ["Content-Type":"application/json; charset=utf-8"]
    let AlamoHeader = HTTPHeaders(header)

    let params: [String: AnyObject] = ["username": userNameTF.text,
                                       "password": passwordTF.text] as [String: AnyObject]


    AF.request(url, method: .post, parameters: params, encoding: JSONEncoding.default, headers:
        AlamoHeader).responseJSON { (response) in

        self.dismiss(animated: true, completion: nil)

        switch response.result {

        case .success(_):

            let responseJSON = JSON(response.value as Any)
            //print(responseJSON)

            if responseJSON["id"].intValue != 0 {
```

IOS Application

```
func get_cities() {

    guard let url = URL(string: "https://egypttourguide.herokuapp.com/cities") else {return}

    let header = Constants().defaultHeader
    let alamoHeader = HTTPHeaders(header)

    AF.request(url, method: .get, parameters: nil, encoding: URLEncoding.default, headers:
        alamoHeader).responseJSON { (response) in

        switch response.result {

        case .success(_):

            let repValue = response.value as? Dictionary<String,AnyObject>
            let cities = repValue!["cities"] as? [Dictionary<String,AnyObject>]
            //print(cities)

            for city in cities! {

                let getCity = GetCity(cities: city)
                self.getCityArr.append(getCity)
                self.cityImagesArr.append(getCity.media![0])

            }
```

This is a custom tap. I used a collection view to create a custom tap by using the collection's cell and its property "isSelected" which is a boolean variable.The cell contains content view, label and view -the underline view-.If the property "isSelected" is true, then change the label's text color, show the line, and change the line's background color.

```
import UIKit

class ChooseProFavCVCell: UICollectionViewCell {

    @IBOutlet weak var backView: UIView!
    @IBOutlet weak var cellLabel: UILabel!
    @IBOutlet weak var underLineView: UIView! {
        didSet {
            underLineView.isHidden = true
        }
    }

    override var isSelected: Bool {
        didSet {
            if isSelected {
                cellLabel.textColor =
                underLineView.isHidden = false
                underLineView.backgroundColor =

            } else {
                cellLabel.textColor = .white
                underLineView.isHidden = true
                underLineView.backgroundColor = .white
            }
        }
```

IOS Application

This is a UIdatePickerView to pick the date and to get the right format of the date and to know the duration of the room reservation.In the function setUpDatePicker, I'm intializing the picker and choose its mode, then I add an action to the datePicker. when the user select a date, the action is formatting the date as "yyyy-MM-dd" and put the date ih the text field.Also I created a tapGesture, so when the user select a date and tap on the super view, the datePicker will disappear.

```
172    func setUpFromDatePicker() {
173
174        datePicker = UIDatePicker()
175        datePicker?.datePickerMode = .date
176        datePicker?.addTarget(self, action: #selector(fromTFdatePicked(datePicker:)), for: .valueChanged)
177
178        let tapGesture = UITapGestureRecognizer(target: self, action:
               #selector(endPicking(tapGestureRecognizer:)))
179
180        view.addGestureRecognizer(tapGesture)
181        fromTF.inputView = datePicker
182    }
```

```
63      }
64
65      @objc func fromTFdatePicked(datePicker: UIDatePicker) {
66
67          let dateFormatter = DateFormatter()
68          dateFormatter.dateFormat = "yyyy-MM-dd"
69          fromTF.text = dateFormatter.string(from: datePicker.date)
70          //view.endEditing(true)
71      }
```

This is a custom interceptor which I used to handle token and refresh token, these functions are built-in functions in alamofire, I just customized the retry function as I needed.
First, when the user sign in I store the token and the refreshToken using UserDefault, then I use the token for the API calls which need token to be done and if the token is expired I refresh the token using the refreshToken and change the value of the stored token to the new one.

IOS Application

```
UserDefaults.standard.set(self.token, forKey: "APIToken")
UserDefaults.standard.set(self.refreshToken, forKey: "refreshToken")
```

```
func retry(_ request: Request, for session: Session, dueTo error: Error, completion: @escaping (RetryResult)
    -> Void) {

    if let urlString = request.request?.url, urlString.absoluteString.contains("/review"),
        request.response?.statusCode == 403 {

        UserDefaults.standard.removeObject(forKey: "APIToken")
        postRefresh()
        print("Retry: \(APIToken)")
        UserDefaults.standard.set(APIToken, forKey: "APIToken")
        completion(.retry)
    } else {
        completion(.doNotRetry)
    }
}
```

Custom loading indicator using UIActivityIndicatorView

```
229
230     func startLoading() {
231
232         let alert = UIAlertController(title: nil, message: "Please wait...", preferredStyle: .alert)
233
234         alert.view.tintColor = UIColor.black
235         let loadingIndicator: UIActivityIndicatorView = UIActivityIndicatorView(frame: CGRect(x: 10, y: 5,
                width: 50, height: 50)) as UIActivityIndicatorView
236         loadingIndicator.hidesWhenStopped = true
237         loadingIndicator.style = UIActivityIndicatorView.Style.medium
238         loadingIndicator.startAnimating();
239
240         alert.view.addSubview(loadingIndicator)
241         present(alert, animated: true, completion: nil)
242     }
243
244 }
245
```

Custom alert using UIAlertController

```
221
222     func showAlert(message: String) {
223
224         let alert = UIAlertController(title: "Alert", message: message , preferredStyle:
                UIAlertController.Style.alert)
225         alert.addAction(UIAlertAction(title: "Ok", style: UIAlertAction.Style.default, handler: nil))
226         present(alert, animated: true, completion: nil)
227     }
228
```

Scan me

Egypt

TOUR GUIDE