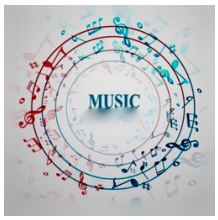


MUSIC GENIE

Name: Swati Maruthi Ram **Mobile Application Programming**

SUID: 563381781 **CIS 651**

FINAL PROJECT REPORT

Overview of the Application:

- Music Genie is the perfect app for music enthusiasts who are willing to learn instruments all in one app.
- It is inspired by well build apps on the Appstore like Piano Lessons and Great Guitar.
- By selecting the instrument, it navigates to the video tutorials of the instruments where one can watch videos in order of their expertise as beginners, intermediate and pro.
- One can like and dislike the video for not only his reference but also for others to view.
- It also has a progress bar which indicates the progress in each category the user has achieved.
- Profile of the user can have bio and can access gallery to update his picture.

Usefulness of the Application:

- Self-taught music application.
- Includes all categories of learners.
- Users can like dislike videos to portrait their opinion visible to others.
- All relevant videos at one spot.
- User can have his own profile description and safely logout of the app to maintain his progress which will not be disturbed.

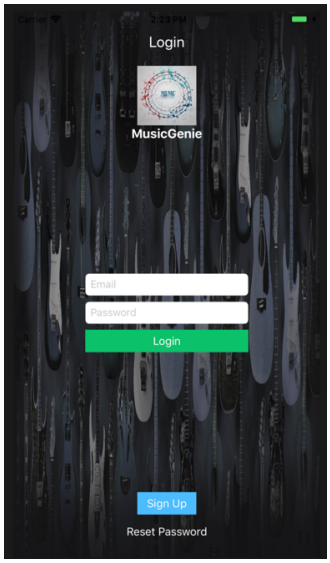
Uniqueness of the Application:

- Makes it easy for the user to browse more than 5 categories of instruments all at one app without having to face the hassle of searching the internet.
- Rating of videos and allowing all users to view it makes it easier for people to choose what videos to learn from
- Aimed at building music enthusiasts a useful app to encourage learning.

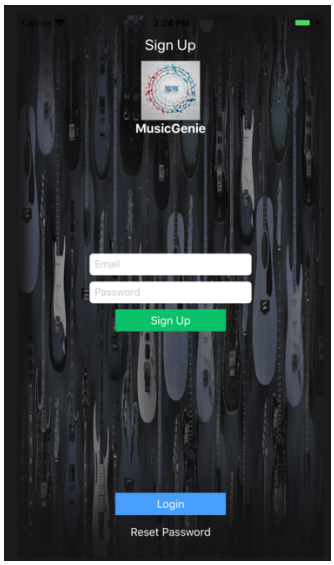
Features of the Application:

1. Login Feature

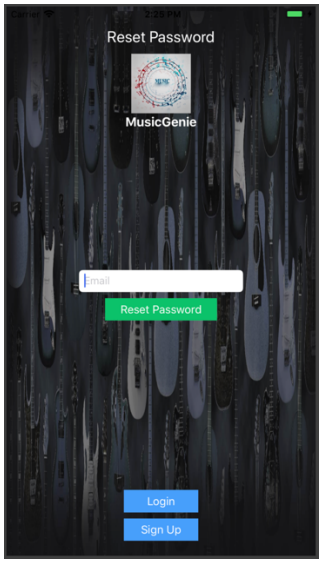
The first screen of the application gives an option for user to login or sign up as a new user using his email ID. Once the user logs in, the user information is saved in firebase database. Each time a user attempts to login, a connection is established, and details are authenticated through the Database.



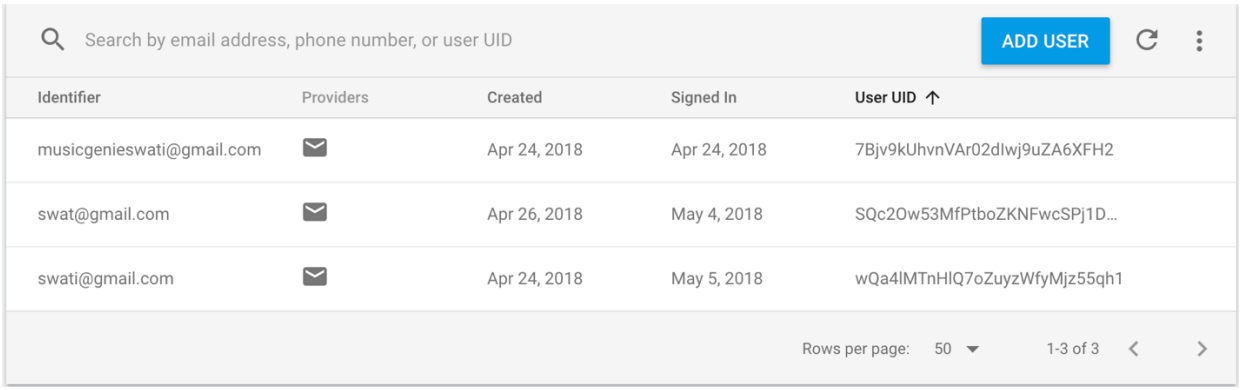
Login Screen



Sign Up Screen



Reset Password



Registered Users shown in Firebase Database

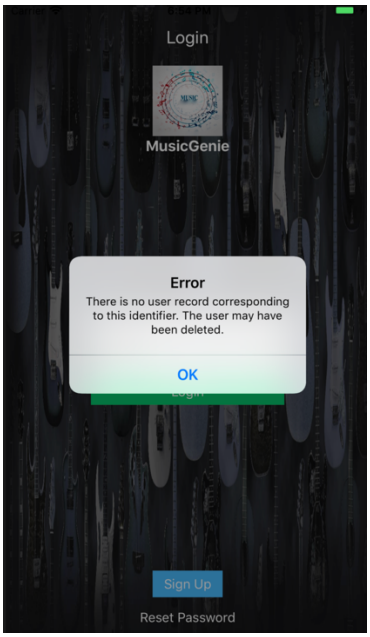
The login screen also has validations for incorrect password and unregistered email ID. The signup screen has validations for correct syntax of email ID and minimum 6 characters for password. The Password reset screen also has the validations for unregistered email ID and invalid password. The code snippet for login and validations is as shown below:

```
Auth.auth().signIn(withEmail: self.emailTextField.text!, password: self.passwordTextField.text!) { (user, error) in

    if error == nil {

        //Print into the console if successfully logged in
        print("You have successfully logged in")

        //Go to the HomeViewController if the login is sucessful
        DispatchQueue.main.async {
        let vc = self.storyboard?.instantiateViewController(withIdentifier: "TabController")
        self.present(vc!, animated: true, completion: nil)
        }
```

Login Authentication from Firebase

```
else {

    //Tells the user that there is an error and then gets firebase to tell them the error
    let alertController = UIAlertController(title: "Error", message: error?.localizedDescription,
        preferredStyle: .alert)

    let defaultAction = UIAlertAction(title: "OK", style: .cancel, handler: nil)
    alertController.addAction(defaultAction)

    self.present(alertController, animated: true, completion: nil)
}
```

Error handling of invalid user from Firebase



Error Screen retrieved from Firebase

2. Home Screen Collection View

The launch screen as soon as login is the successful is the Collection view of all different instruments. It has a tab bar view for toggling between Instruments screen and User Profile Screen.

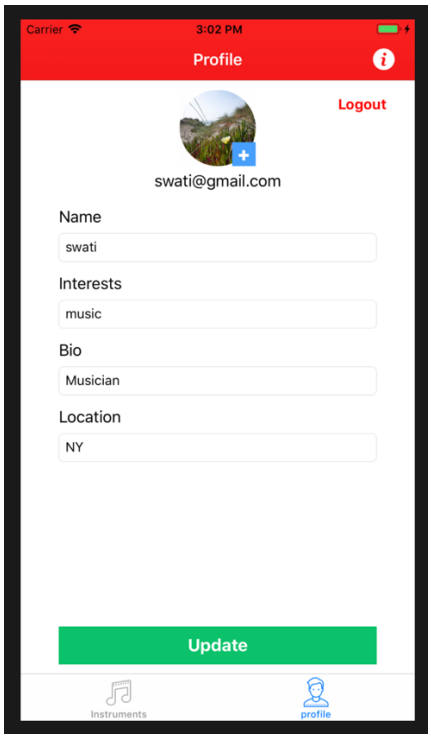
Home Screen

The collection view implementation:

```
class InstrumentCollectionViewCell: UICollectionViewCell {

    @IBOutlet weak var instrumentImageView : UIImageView?
    @IBOutlet weak var instrumentNameLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        instrumentImageView?.layer.cornerRadius = self.frame.height / 12.0
        instrumentImageView?.layer.masksToBounds = true
        // Initialization code
    }

}
```

3. Tab Bar View for User profile

The user profile has fields to fill in the description and it also has a feature for including profile images accessing the camera. And when screen is updated, alert shows the updating of data and goes back to home screen.

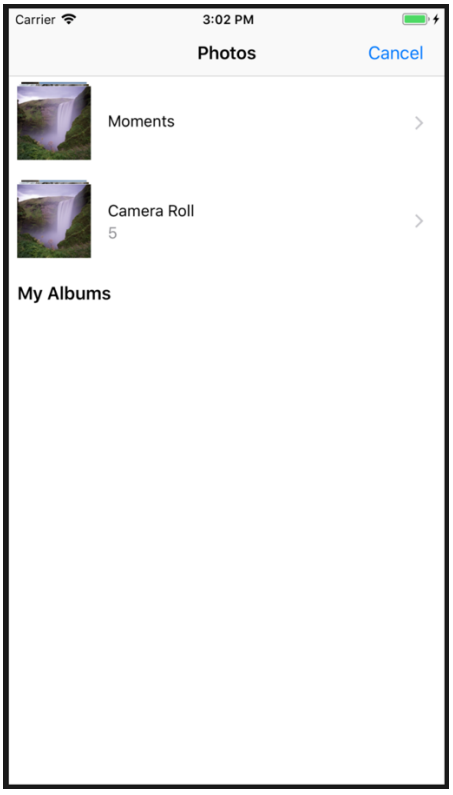


User profile Screen

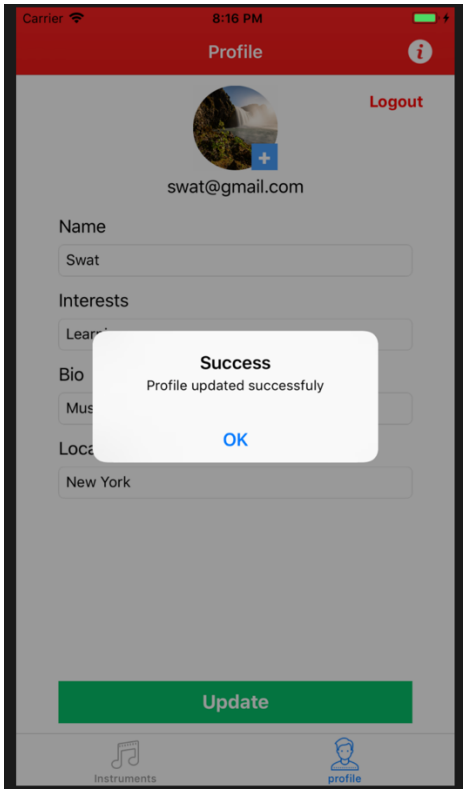
User can also logout in the profile page and access images from camera roll to update his profile picture.

The code snippet for camera access is:

```
extension ProfileViewController: UIImagePickerControllerDelegate,UINavigationControllerDelegate {

    func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [String : Any]) {
        let image = info[UIImagePickerControllerOriginalImage] as? UIImage
        self.profileImage.image = image
        picker.dismiss(animated: true, completion: nil)
    }
}
```



Camera Access for Profile Image



Profile Info Updating Alert

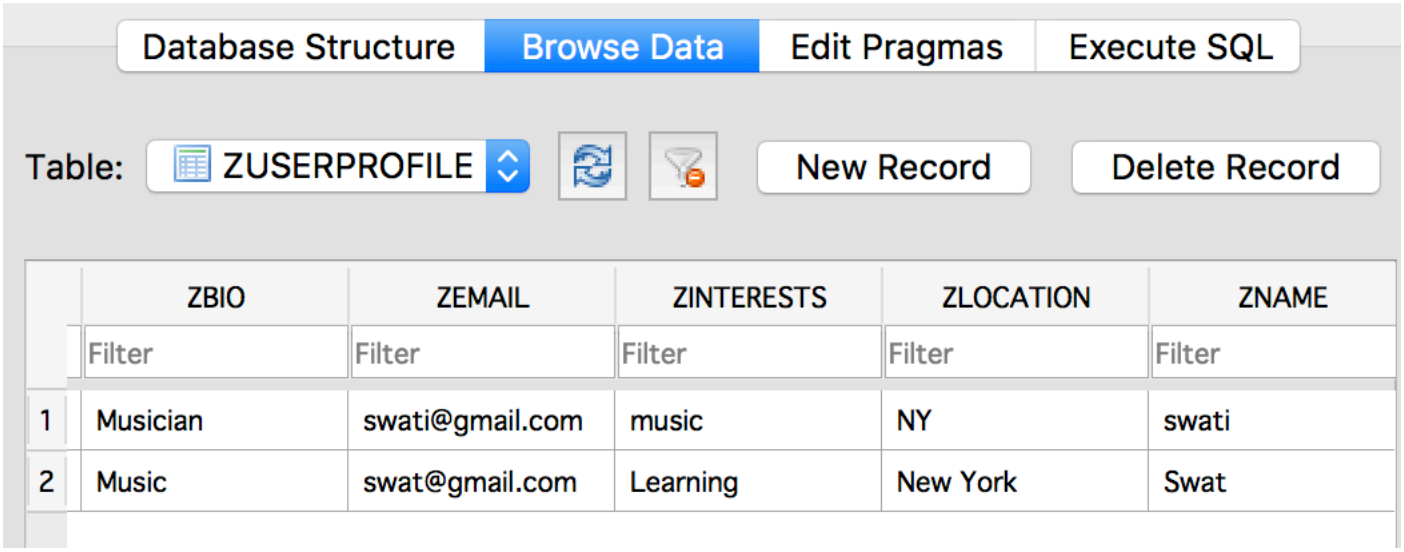
```swift
@IBAction func logout(_ sender: Any) {
    if Auth.auth().currentUser != nil {
        do {
            try Auth.auth().signOut()
            let vc = UIStoryboard(name: "Main", bundle: nil).instantiateViewController(withIdentifier:
                "LoginViewController")
            present(vc, animated: true, completion: nil)

        } catch let error as NSError {
            print(error.localizedDescription)
        }
    }

    UserModel.clearData()
}
```

Code Snippet for User Logout

```swift
let alertController = UIAlertController(title: "Success", message: "Profile updated successfuly",
    preferredStyle: .alert)
// let defaultAction = UIAlertAction(title: "OK", style: .cancel, handler: nil)
let defaultAction = UIAlertAction(title: "OK", style: .cancel) { (_) in
    self.tabBarController?.selectedIndex = 0
}
alertController.addAction(defaultAction)
self.present(alertController, animated: true, completion: nil)
```

Code snippet for updating Profile

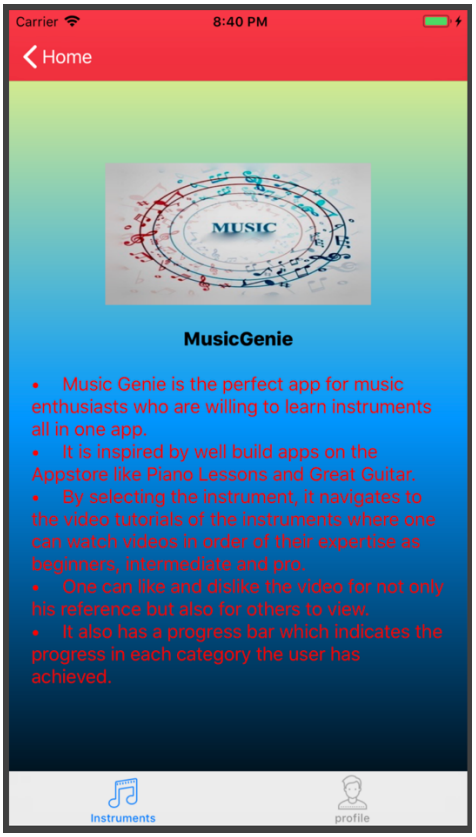
The User information is saved in the database and can be modified from there



	ZBIO	ZEMAIL	ZINTERESTS	ZLOCATION	ZNAME
	Filter	Filter	Filter	Filter	Filter
1	Musician	swati@gmail.com	music	NY	swati
2	Music	swat@gmail.com	Learning	New York	Swat

4. App Information with Navigation Controller

The information screen on top is a common feature in the navigation bar which gives the app information. It is accessible from every screen in the app and it goes back to the screen from which it was called. This is a simple UILabel and UIImage View that has been used and

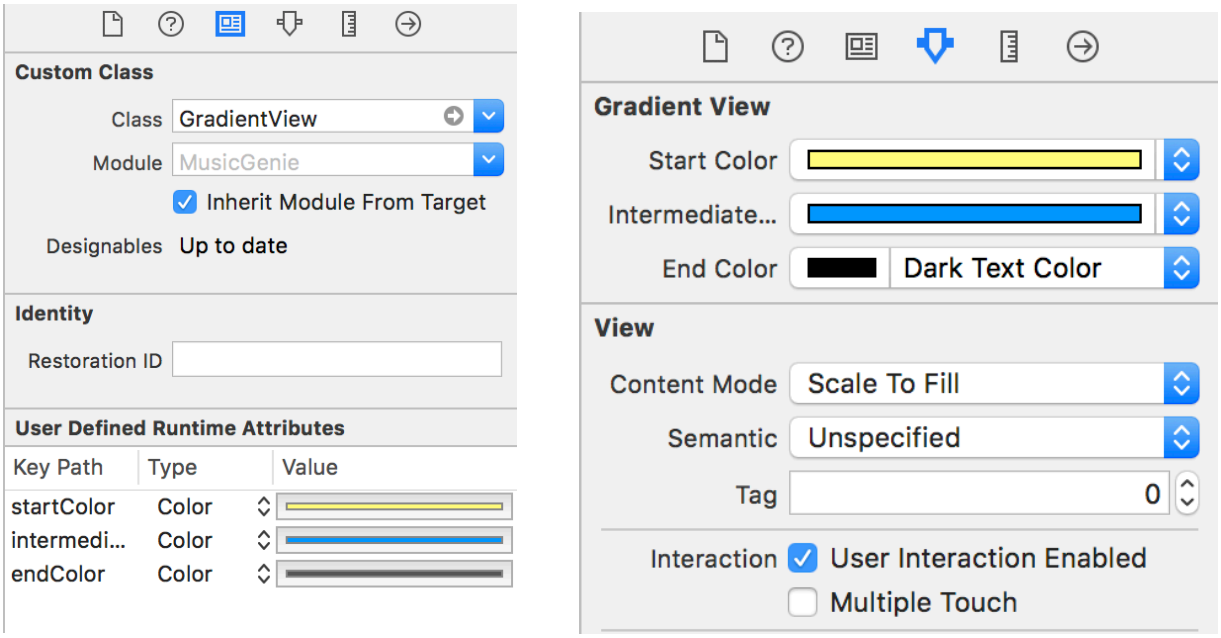


App Info selected from Home

App Info selected from Keyboard

```
@IBDesignable  class GradientView: UIView {

    @IBInspectable var startColor: UIColor = UIColor.clear
    @IBInspectable var intermediateColor: UIColor = UIColor.clear
    @IBInspectable var endColor: UIColor = UIColor.clear

    override func draw(_ rect: CGRect) {
        let gradient: CAGradientLayer = CAGradientLayer()
        gradient.frame =  self.bounds
        gradient.colors = [startColor.cgColor,intermediateColor.cgColor, endColor.cgColor]
        gradient.locations = [ 0.0, 0.5 , 1.0]
        layer.insertSublayer(gradient, at: 0)
    }
}
```

The background given to the app info screen is a gradient tint which is coded to give combination of three colors when the class is included as a custom class. The values can be varied and the background tint changes accordingly.



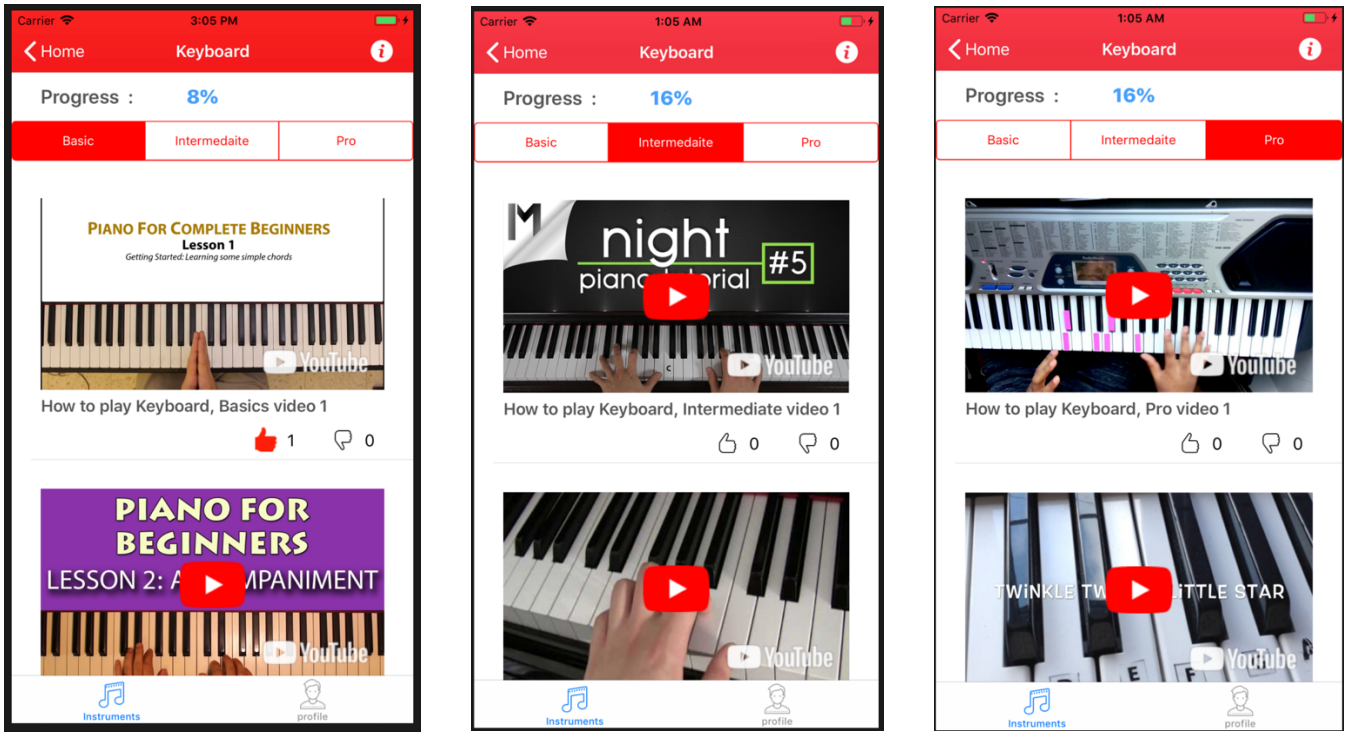
Custom Class enabling multicolor background for UIView Controller

5. Instruments screen as Table View and Segmented Control for levels of learning

As soon as the user selects the particular instrument he wants to learn, it navigates to a table view screen where the videos are displayed according to the choice of level by user.

There is a segmented control which has been used for the bifurcation of the levels of learning as beginning, intermediate and expert. User can start viewing the videos and learning from them starting from which level he is comfortable with. Toggling between the segments is also possible.

There are two other features included in this table view which are the like and dislike button and the progress bar which will be explained in a bit.



The video playing screen showing the segmented view.

As we can see in the video screen, like and dislike buttons are given. This is same across users and they can view the opinions of other users and also the progress of themselves.

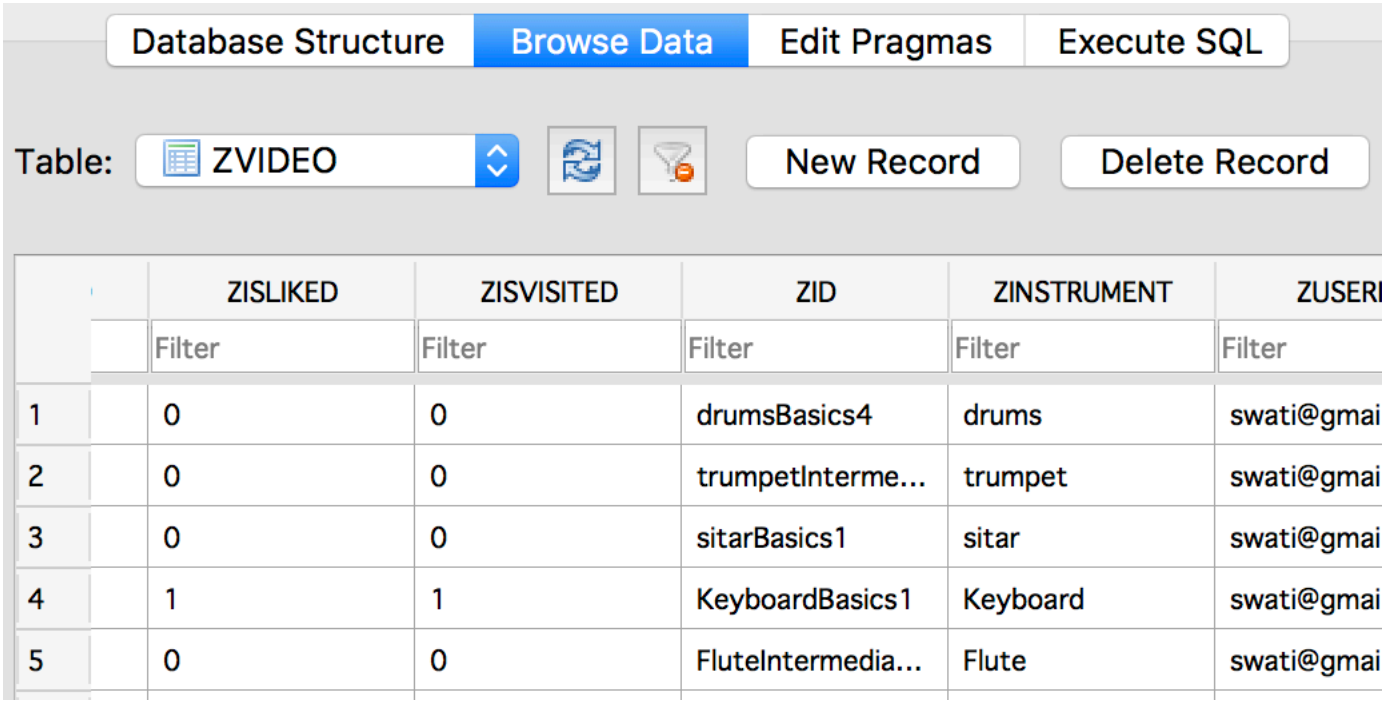
The progress bar works on the logic of dividing the number of videos and assigning percentage to each video when viewed which can be shown below:

```
// to track progress of course
public class func fetchVideoObjectForInstrument(withRequest request: NSFetchRequest<Video>,
    userID:String,instrument:String)-> [Video]? {
    let fetchRequest: NSFetchRequest<Video> = request
    let predicate = NSPredicate(format: "userId = %@ && instrument = %@ ", userID as String,instrument as String)
    fetchRequest.predicate = predicate
    do {
        //go get the results
            return  try MusicGenieCoreData.sharedInstance().getContext().fetch(fetchRequest)
        }
     catch {
        print("error")
    }
    return nil
}
```

The like and dislike count is maintained in the sqlite and is handled in the code as follows:

```
// like and dislike count
public class func fetchVideoObjectBasedOnID(withRequest request: NSFetchRequest<Video>, videoID:String)-> [Video]? {
    let fetchRequest: NSFetchRequest<Video> = request
    let predicate = NSPredicate(format: "id = %@ ", videoID as String)
    fetchRequest.predicate = predicate
    do {
        //go get the results
        return  try MusicGenieCoreData.sharedInstance().getContext().fetch(fetchRequest)
    }
    catch {
        print("error")
    }
    return nil
}
```

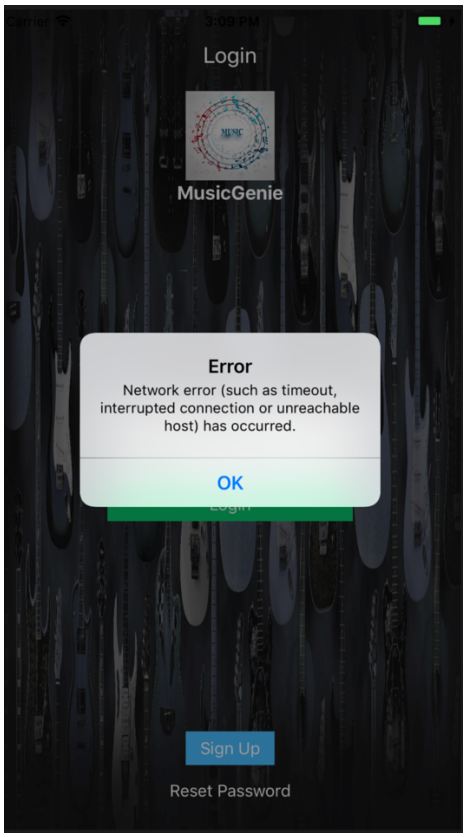
The database also handles this by making the like column 1 and dislike remains 0. Only one of like and dislike can be 1 at a time and that is how toggling between the buttons is possible.



	ZISLIKED	ZISVISITED	ZID	ZINSTRUMENT	ZUSERI
	Filter	Filter	Filter	Filter	Filter
1	0	0	drumsBasics4	drums	swati@gmai
2	0	0	trumpetInterme...	trumpet	swati@gmai
3	0	0	sitarBasics1	sitar	swati@gmai
4	1	1	KeyboardBasics1	Keyboard	swati@gmai
5	0	0	FluteIntermedia...	Flute	swati@gmai

6. Network Detection

Another feature included in the application is detection of network signals and showing an error if the app doesn't get network access. The code and the screen shot for the above feature is as mentioned below:




```
class NetworkReachability: NSObject {
    open static var sharedManager: NetworkReachabilityManager = {
        let reachabilityManager = NetworkReachabilityManager()
        reachabilityManager?.listener = { (status) in
            switch status {
            case .notReachable:
                print("The network is not reachable")
                NotificationCenter.default.post(name: NSNotification.Name(rawValue: "unsuccessful"), object: nil)
            case .unknown:
                print("It is unknown wether the network is reachable")
            case .reachable(.ethernetOrWiFi):
                print("The network is reachable over the WiFi connection")
                NotificationCenter.default.post(name: NSNotification.Name(rawValue: "successful"), object: nil)
            case .reachable(.wwan):
                print("The network is reachable over the WWAN connection")
                NotificationCenter.default.post(name: NSNotification.Name(rawValue: "successful"), object: nil)
            }
        }
        reachabilityManager?.startListening()
        return reachabilityManager!
    }()
}
```

7. Model View Controller Architecture

The architecture which is followed while building the application is MVC. The folder structure can be seen as below: