Programmer's Guide

By Federico R, Brett W, Josh B, Jacob V, Mike B

Requirements and Planning

1. **Statement**

   The software engineering team Federico Read Grullon, Brett Weicht, Joshua Blaz, Jacob von der Lippe, and Michael Bonderud agrees to develop a mobile application that allows students to quickly and easily access dining hall menus, which will be updated daily to display available meal options. Hours of operation for each dining hall will be provided along with advertised events and closings throughout the school year. Students will have the option to rate and comment (180 characters or less) on meals or specific food items, enabling the students to submit and receive feedback on food options and know what others enjoyed. We will provide this application on both apple's app store and google play to provide this service to as many students as possible. Students/clients will have access to the application gradually over time. Notoriety for the final distribution of our application will be done through word-of-mouth only as to allow initial corrections by the software engineering team to take place without major disturbance.

2. **Timeline and Milestones**

   The project must be completed by 12/11/2017. Our major milestone ideals are and will be completed as follows:

   - Acquiring the Bon Appetit Menu: Oct. 1st
   - Regular access/updating: Oct. 14th
   - Rating and Commenting System: Oct. 21st
   - User Interface: Oct. 7th
   - Hours of operation: Oct. 14th.
   - A functioning application: Nov. 1st
   - A functioning application (available for download): Dec. 1st

3. **Cost Estimate**

   Our team, with all of our schedules kept in mind, are willing to commit a maximum of 2 hours a day, 5 days a week. This will provide a suitable amount of

time to complete the project. There will be a cost of $125 for putting the application onto both google play and apple's app store. This is the only foreseeable monetary cost.

4. **Equipment**

   The hardware we will be using while creating our application are our own personal computers as well as the Olin servers with Dr. Bressoud's consent for data storage. For software, we will be using Xcode, which will allow our application to be downloaded on apple phones.

5. **Non-Functional Requirements**

   We will implement a login system to increase accountability and deter users from writing irrelevant comments. We will integrate a minimal response time for retrieving menus and hours of operation (less than 10 seconds.) Also a major part of our application is to promote user friendliness with accessing this information for students. The transition from opening the app to retrieving desired information will be as efficient as possible.

6. **Communication**

   Communication between members of the development team will be done through the use of the GroupMe app and weekly in-person meetings. This will allow for thorough sharing of ideas and expectations for the project. Communication between our clients will be done in person as well as email. Once an initial meeting has been completed with Denison food services and Denison tech support; weekly check-ins will follow.

7. **Possible Risks**

   General Deterrents: Scheduling conflicts between team members, expectation conflict with clients, base website query is inadequate, etc.

Below will feature a running list of time commitments among team members:

Josh - Out of town overnight for football 9/15-9/16 and 10/6-10/7.
Jake/Brett - Nov. 30th - Dec. 2nd swim Meet.

We will be utilizing Dropbox to store and easily transfer parts of the project between team members. This will allow us to easily access backup files in the case of something going wrong. We will also be working on all parts of the project in groups of two or more to be sure that if someone is unable to complete a part by the deadline, there is at least one other team member prepared to continue working. We will also be sure to have constant communication between all members of the group such that no part of the project will be foreign to another member, even if they aren't assigned that part of the project.

8. Assessment

The project will be assessed by our clients on delivery. Our project will be assessed on basic functions including: fundamental UI + navigation, listing of pertinent information (dining hall hours, menus), and updates to above, all of which are described in our Statement. Assessment from our clients will be done through our weekly meetings through an agile project scheme. If the assessment is a success, then we have met all of the requirements agreed upon between the Software Engineering team and our clients.

9. Training

This application is being developed with a no client training mentality. We will develop the app with an extremely intuitive UI that uses many iPhone specific functional tools. We will not require administering any sort of training in order to use or maintain our application.

10. Documentation

Our documentation will mostly be in the form of our weekly reports. These reports will keep record of our progress and features developed. Each team member will be responsible for creating and storing a weekly report once every five weeks. The order for reports by week will be as follows: Jacob, Brett, Federico, Michael, Josh. If any member is unable to complete their report during the week they have been assigned, it will be up to that member to notify the group and find a replacement member to cover for them. We will also have a help section within the application that will have FAQ's as well as guides for using functions within the application.

11. Maintenance

We will create a standalone app that can function without constant maintenance from our team or the client. If bugs are found and/or changes are needed to coincide with updates to feeder websites. We will rely on reviews through the application stores (Google play and Apple App store) to notify us of changes we need to implement, and sections to maintain. Decision on who will perform the maintenance will be discussed and decided upon by the software engineering team after the reception of a post development issue.

12. Changes to Requirements

1. Our application will be supported on only IOS devices as this will minimize production time using only Xcode and Swift rather than Xamarin. This will also reduce costs to $100. Our training will also be shortened to IOS specific only.
2. Our application will not support ratings but instead focus on user comments and a login system for students. This will enforce user-friendliness by opening up more space on the User Interface.
3. We will be using Google Firebase instead of Dr. Bressoud's database in order to alleviate necessary communication with the professor.
4. The commenting system will not check content of user inputs (although this is a function we may implement at a later date) but instead the database will allow for regular cleanups. A function will be used through Firebase Functions and Javascript to clean all data in the database that is older than 6 hours.

New Milestones:

The project must be completed by 12/11/2017.

- Acquiring the Bon Appetit Menu: Nov. 10th
- Regular access/updating: Nov. 10th
- Hours of operation: Nov. 10th
- Rating and Commenting System: Nov. 17th
- User Interface: Oct. 30
- Testing week: Nov. 27th - Dec. 1st
- A functioning application: Dec. 1st
- A functioning application (available for download): Dec. 4th

Design

See Attached PDF

Documented Software

See Attached PDF

Testing Document

Below are the individual tests that we performed.

Name of item being tested: Application Data Import (Menus/Hours)

Team Contact: Mike, Brett, Federico

Team Tester: Mike

Type of test (white box or black box,? unit, integration, validation, or system?): Integration

Purpose of test: Testing for failure of importing dining data into *meal hall* app page.

If white box, what paths are being covered?: Direct contact pathways between online -> database -> app

Calls the scrape functions which harvests the data from BONAPPETIT for import.

If black box, what specifications are being used?: Whether data is displayed correctly in the UI.

Precondition: UI is bare, data is online (Bonappetit.com)

Postcondition: UI is populated with dining hours and hall menu

Test inputs: Imported data from the BonAppetit website

Inputs are within the system not inputs from a user. The input will come from the Bon Appetit website, the data will be brought to our app and displayed on the dining hall pages.

Expected outcomes from each input: Proper presentation of data, in application (ui)

Results from each input:

INPUT	EXPECTED RESULTS	ACTUAL RESULTS
Online Source (Curtis)	Complete copy of Online Source information(Hours and Menu)	Complete copy of Online Source information(Hours and Menu)
Online Source (Huffman)	Complete copy of Online Source information(Hours and Menu)	Complete copy of Online Source information(Hours and Menu)

Errors discovered from each input: None

Proposed changes: Small changes to UI appearance to make the information easier to read.

Other items affected: This function was self contained and did not bring forward errors in other sections.

Contact plan: Communicate with Team Contacts listed above if errors are found. Contact information will be documented in the help section Documentation of the app.

Retest plan for this item and for affected items: After the completion of testing for other sections, will make sure that any changes made will not impact the results from this test. This will be done through a retest within the next week.

Follow-up: Same tests were run as above with similar results.

Name of item being tested: Sending Ratings and Comments (Mobile to Database)

Team Contact: Jake and Josh

Team Tester: Brett

Type of test (white box or black box,? unit, integration, validation, or system?): Black box unit testing
Purpose of test: To test whether the app is sending information to the database.

If white box, what paths are being covered? N/A

If black box, what specifications are being used? The preconditions are that we linked to the database.

Test inputs: Each test input consists of a fixed string containing the name of a dining hall, and the comment itself. The comments tested were String of characters, String of numbers, string with numbers and characters, an empty string and a string of special characters.

Expected outcomes from each input: We would expect that all strings are eligible to be stored in the database. However, an empty string should not be stored as a comment. All strings should be sectioned off based on the dining hall subsection they are identified with and the database should only contain strings of length 1 or more. An example could be, if a comment "I loved it!" was sent from the Curtis comment section, it should appear in the Database under the subsection "Curtis" as input by the user. An example of a non-allowed comment would be an empty entry. In this case, if the user does not enter any text into the comment box but returns anyway, the database should not create a new entry anywhere.

Results from each input:

INPUT (all preceded by a dining hall name)	EXPECTED RESULTS	ACTUAL RESULTS
Empty string - ""	No information is sent to the database	On Firebase an empty string did appear. This is a failure
Special characters - "!@* %$#^&"	"!@* %$#^&" appears under the subsection "Slayter" on Firebase	"!@* %$#^&" appears under the subsection "Slayter" on Firebase
Regular string - "Tonights Dinner was fantastic"	"Tonights Dinner was fantastic" appears under the subsection "Curtis" on Firebase	"Tonights Dinner was fantastic" appears under the subsection "Curtis" on Firebase

Numbers - “651098234”	“651098234” appears under the subsection “Huffman” on Firebase	“651098234” appears under the subsection “Huffman” on Firebase
Numbers and characters - “asTSDFG 8734 8cE4 k5JD873”	“asTSDFG 8734 8cE4 k5JD873” appears under the subsection “Curtis” on Firebase	“asTSDFG 8734 8cE4 k5JD873” appears under the subsection “Curtis” on Firebase

Errors discovered from each input: We need to correct our issue with accepting an empty string as a comment.

Proposed changes: I propose that we check the length of the comments and also look for a string containing only spaces/tabs.

Other items affected: This function was self contained and did not bring forward errors in other sections.

Contact plan: Communicate with Team Contacts listed above if errors are found. Contact information will be documented in the help section Documentation of the app.

Retest plan for this item and for affected items: Josh and Jake will attempt to correct the issue in the next few days and I will retest their new code.

Follow-up: The issue has been resolved and the empty string is no longer accepted as a comment.

Name of item being tested: Receiving Ratings and Comments (Database to Mobile)

Team Contact: Jake and Josh

Team Tester: Brett

Type of test (white box or black box,? unit, integration, validation, or system?): Black box unit test

Purpose of test: To test whether the app is retrieving information from the database.

If white box, what paths are being covered? N/A

If black box, what specifications are being used? The preconditions are that there are comments and ratings for current meal options stored in the database. The postconditions are that the ratings and comments are displayed on the correct dining hall page.

Test inputs: Long strings, short strings, empty strings, strings of characters, strings of numbers, symbols, proper database format, improper database format

Expected outcomes from each input: Long strings: code will reject the string and not delegate space on the app if it exceeds 180 characters, otherwise it will allow it to be printed and scale the comment to the screen accordingly. For example, the comment "I love to type a million million million million million million million million million million million million million things!" will be compressed by the app to fill the same amount of space on the screen as a comment like "Short but Sweet." Short strings: short strings will be printed and scaled similarly to long strings above. Empty string: Code will reject like above if an empty string is found. Strings of chars/numbers/symbols: the code should be able to recognize all ASCII characters and print them without any changes. Improper database format: If the database is unreadable, the code should send a message to the user saying that data is not retrievable at this time. Code should continue working without drawing data from the database.

Results from each input:

INPUT (all preceded by a dining hall location)	EXPECTED RESULTS	ACTUAL RESULTS
Empty string - ""	No information is sent to the phone	The phone did portray an empty space. This is a failure
Special characters - "!@* %$#^&"	"!@* %$#^&" appears in the "Slayter" comment section.	"!@* %$#^&" appears in the "Slayter" comment section.
Regular string - "Tonights Dinner was fantastic"	"Tonights Dinner was fantastic" appears in the "Curtis" comment section.	"Tonights Dinner was fantastic" appears in the "Curtis" comment section.
Numbers - "651098234"	"651098234" appears under the subsection "Huffman" on	"651098234" appears under the subsection "Huffman" on

	Firebase	Firebase
Numbers and characters - “asTSDFG 8734 8cE4 k5JD873”	“asTSDFG 8734 8cE4 k5JD873” appears under the subsection “Curtis” on Firebase	“asTSDFG 8734 8cE4 k5JD873” appears under the subsection “Curtis” on Firebase

Errors discovered from each input: N/A

Proposed changes: N/A

Other items affected: N/A

Contact plan: Communicate with Team Contacts listed above if errors are found. Contact information will be documented in the help section Documentation of the app.

Retest plan for this item and for affected items: N/A

Follow-up: N/A

Name of item being tested: Storing Comments/ Ratings

Team Contact: Josh and Jake

Team Tester: Mike, Brett, Federico

Type of test (white box or black box,? unit, integration, validation, or system?): Integration

Purpose of test: To send and store comment/ratings data. (includes day, meal hall, meal)

If white box, what paths are being covered? Comments and rating section in ui -> Database -> sub_directory

Calls “send” from the mobile app and “store” on the database side which will export the data to our database for storage.

If black box, what specifications are being used? Permits data retrieval functionality

Pre: the section of memory in the database for that day is not full.
Post: the database folder has been populated with comments/ ratings data.

Test inputs: Example ratings/comments data.

Expected outcomes from each input: Encoded data properly stored in database.

Results from each input: Tested concurrently with Sending ratings and Comments, so the tables are similar.

INPUT (all preceded by a dining hall name)	EXPECTED RESULTS	ACTUAL RESULTS
Special characters - "Slayter" "!@* %$#^&"	"!@* %$#^&" appears under the subsection "Slayter" on Firebase	"!@* %$#^&" appears under the subsection "Slayter" on Firebase
Regular string - "Curtis" "Tonights Dinner was fantastic"	"Tonights Dinner was fantastic" appears under the subsection "Curtis" on Firebase	"Tonights Dinner was fantastic" appears under the subsection "Curtis" on Firebase
Numbers - "Huffman" "651098234"	"651098234" appears under the subsection "Huffman" on Firebase	"651098234" appears under the subsection "Huffman" on Firebase
Numbers and characters - "Curtis" "asTSDFG 8734 8cE4 k5JD873"	"asTSDFG 8734 8cE4 k5JD873" appears under the subsection "Curtis" on Firebase	"asTSDFG 8734 8cE4 k5JD873" appears under the subsection "Curtis" on Firebase

Errors discovered from each input: No errors were discovered. In each case the database was updated and displayed the correct information.

Proposed changes: None.

Other items affected: None. This item has little interaction with the rest of our app.

Contact plan: Communicate with Team Contacts listed above if errors are found. Contact information will be documented in the help section Documentation of the app.

Retest plan for this item and for affected items: none.

Follow-up: none.

Name of item being tested: User Interface (Transition buttons)

Team Contact: Federico and Brett

Team Tester: Jake

Type of test (white box or black box,? unit, integration, validation, or system?): White Box and validation

Purpose of test: Ensure all buttons and pages have functioning transitions to the correct page.

If white box, what paths are being covered? We will be testing all the transitions, or segues, of our UI. We will indicate the location of this code when we combine documents.

If black box, what specifications are being used? N/A

Test inputs: Selection of each button on our Home UI. Buttons include “Back”, “Curtis”, “Huffman”, “Slayter” and the help button. We will also test for combinations of selections.

Expected outcomes from each input: Correct transition is made and pages are loaded.
Back: Returns to the home page from any of the three dining pages.
Curtis: navigates from the home page to the Curtis dining hall page.
Huffman: navigates from the home page to the Huffman dining hall page.
Slayter: navigates from the home page to the Slayter dining hall page.
The help button is located on the home screen and if selected will bring up the help screen.
Combinations: Any combination of the selections above should not result in any unexpected changes. The functions above should remain unchanged regardless of how many prior selections were made.

Results from each input:

Input (path of transitions made)	Success / Failure
Help button	Success, Our help page pops up
Slayter	Success, The app transitioned to the page for Slayter
Curtis	Success, the page for Curtis was brought up
Huffman	Success, the page for Huffman was brought up
Slayter->Back	Success, transition to Slayter then back to the Home page
Curtis->Back->Huffman-> Back->Help	Success, all transitions followed expectations and finished at the Help page
Huffman->Back->Slayter->Back->Help->Back->Slayter->Back->Curtis	Success, All transitions worked

Errors discovered from each input: No errors were discovered

Proposed changes: No changes necessary

Other items affected: no items seriously impacted

Contact plan: Communicate with Team Contacts listed above if errors are found. Contact information will be documented in the help section Documentation of the app.

Retest plan for this item and for affected items: N/A

Follow-up: N/A

Name of item being tested: User Interface (Transitions between dining halls)

Team Contact: Federico and Brett

Team Tester: Jake

Type of test (white box or black box,? unit, integration, validation, or system?): Black Box validation testing.

Purpose of test: Ensure transitions between dining hall pages are all working.

If white box, what paths are being covered? N/A

If black box, what specifications are being used? We are testing the functionality of the transitions. Pre: Beginning on one page, and app is able to register a "swipe" across the screen. Post: The app displays a dining option.

Test inputs: On each dining hall page the user has the opportunity to swipe left or swipe right.

Expected outcomes from each input: Correct transition is made and pages are loaded.

Curtis Page: Swipe left- transitions to Slayter
Swipe right- transitions to Huffman
Huffman Page: Swipe left- transitions to Curtis
Swipe right- transitions to Slayter
Slayter Page: Swipe left- transitions to Huffman
Swipe right- transitions to Curtis

Results from each input:

Input (path of transitions made)	Success / Failure
Slayter -> Huffman	Success
Slayter->Curtis	Success
Huffman->Curtis->Huffman	Success
Slayter->Curtis->Slayter->Huffman	Success
Curtis->Huffman->Slayter->Curtis	Success
Huffman->Curtis->Slayter->Huffman	Success

Errors discovered from each input: No errors were found. All combinations tested worked and the final page had full functionality.

Proposed changes: No proposed changes

Other items affected: All other buttons and and functions still work as expected so no items were negatively affected.

Contact plan: Communicate with Team Contacts listed above if errors are found. Contact information will be documented in the help section Documentation of the app.

Retest plan for this item and for affected items: No retest necessary unless new code is added affecting our transitions.

Follow-up: No retests were performed.

Verification & Validation

Validation: Our initial requirements and how we fulfilled them.

Our requirements for the mobile application changed dramatically throughout the project process but we were able to validate many of the original dependencies of our requirements statement. Among these were the IOS platform application, dining hall information access, hours of operation access, student commenting system, and easy-to-use Interface.

Our original plan was to create the application for both IOS and Android platforms using Xamarin. This was changed to just IOS platforms after realizing the difficulties of Xamarin as well as our time restraints. Our final application is only supported on IOS devices although this allowed us to implement one of the most important goals much more smoothly: a cohesive and functioning user interface. Because we only had to focus on a single platform, the development of our user interface went much more smoothly.

A basic requirement of our application was the viewership or menu options in each dining hall. Functionality for each menu was fulfilled through the use a of a webview window. Initially we attempted to utilize a parsing function thats native scope was in HTML. Through some trial and error we discovered that HTML parsing was impractical due to the amount of pruning that was necessary to properly display the data. This was especially true as the basic format of each menu was subject to change with weekly changes to page structure. From this point our team pivoted toward a Javascript implementation but this proved to have the same

problems. Lastly, we tackled the importation problem by using a in app webview window. Once insecure websites were enabled for viewing the webview implementation worked as expected. This reclassified our mobile app as a hybrid application, as the webview allowed for a mobile web browser to be used in our app via a Javascript framework.

The original document also called for our application to allow users to comment and rate dining halls throughout the day and view each others' comments. We implemented a commenting system using Google Firebase and a Cocoa Pods implementation in Xcode although we did not implement the rating system since this would have created problems with the database structure. The commenting system, however, works very much as expected in our requirements document. A few changes we included are: User's can enter more than 180 characters, and instead of a "word checker" to search for profanity, we implemented a database updater which clears out old information throughout the day to allow the database to keep from getting cluttered.

Our plan document outlined basic navigation functionality. This was implemented through two strategies, a home screen and a swiping function. Our home screen opens upon app launch, unless there are app loading errors, which portrays tap boxes labeled with the name of each dining hall. With the selection of any of these options, the app loads into the corresponding hall page. This was done via a redirect app flow map in Xcode. Next, swiping functionality was enabled via Xcode. Once in a dining hall page you can navigate from hall to hall with a swipe right function. For example, if you are currently viewing the Huffman page by swiping right on the screen you would be taken to the Curtis menu page.

Verification: Does our code work according to our tests? What corrections were made based on our tests?

We have shown that our code works from our previously included Test Document. Please refer to that for specifics on how we verified that our code performed each function correctly.

As is shown on our test plan our tests were performed with few to no errors. In fact the only error that was discovered upon our testing was that an empty string could be accepted by our database as a comment. We planned to prevent this from happening in order to eliminate irrelevant comments to minimize data usage. This was an easy fix and completed within an hour and we have since successfully retested our application.

User's Guide

Below are the different options on each page.

Home page:

- Buttons:
  - Curtis - opens Curtis Dining Hall's menu
  - Huffman - opens Huffman Dining Hall's menu
  - Slayter - opens Slayter Markets's menu
  - Gear - help & documentation

Curtis page:

- Buttons:
  - Leave a comment!- opens Curtis comment page
  - Back - opens Home page
- Swipe:
  - Swipe right- opens Huffman page

Huffman page:

- Buttons:
  - Leave a comment!: Opens Huffman comment page
  - Back - opens Home page
- Swipe:
  - Swipe right- opens Slayter page

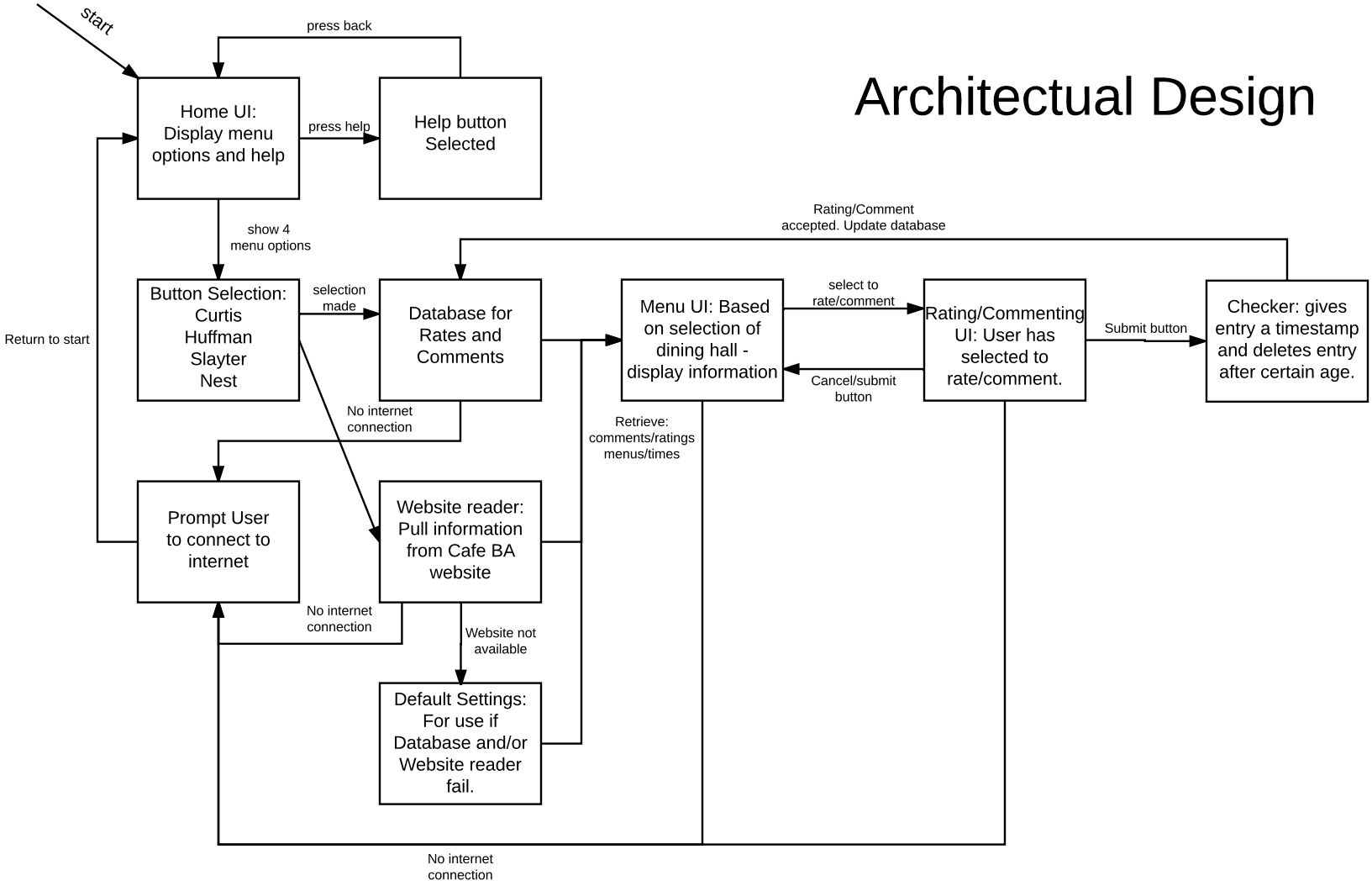
Slayter page:

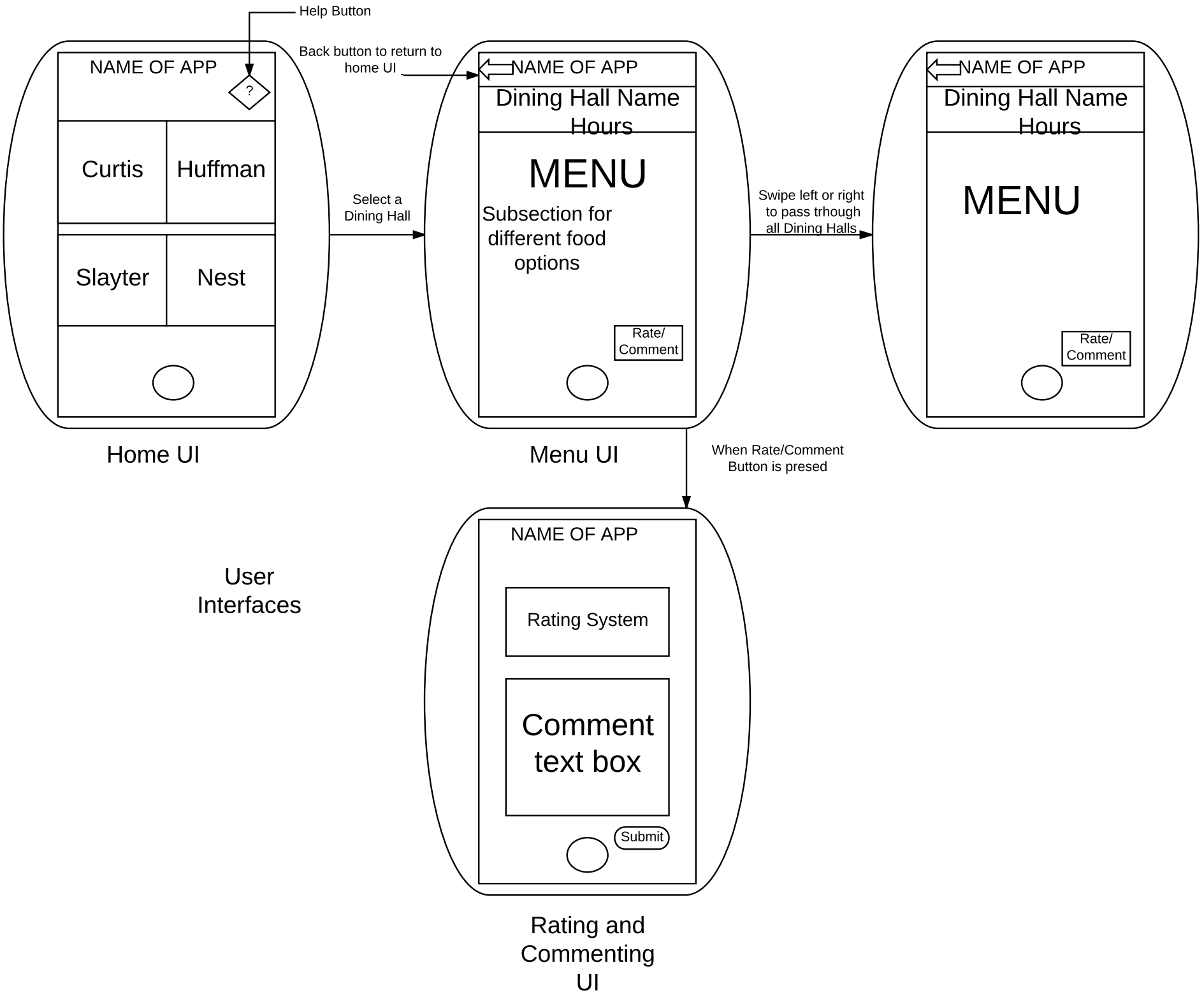
- Buttons:
  - Leave a comment!: Opens Slayter comment page
  - Back - opens Home page
- Swipe:
  - Swipe right- opens Curtis page

Comment pages:

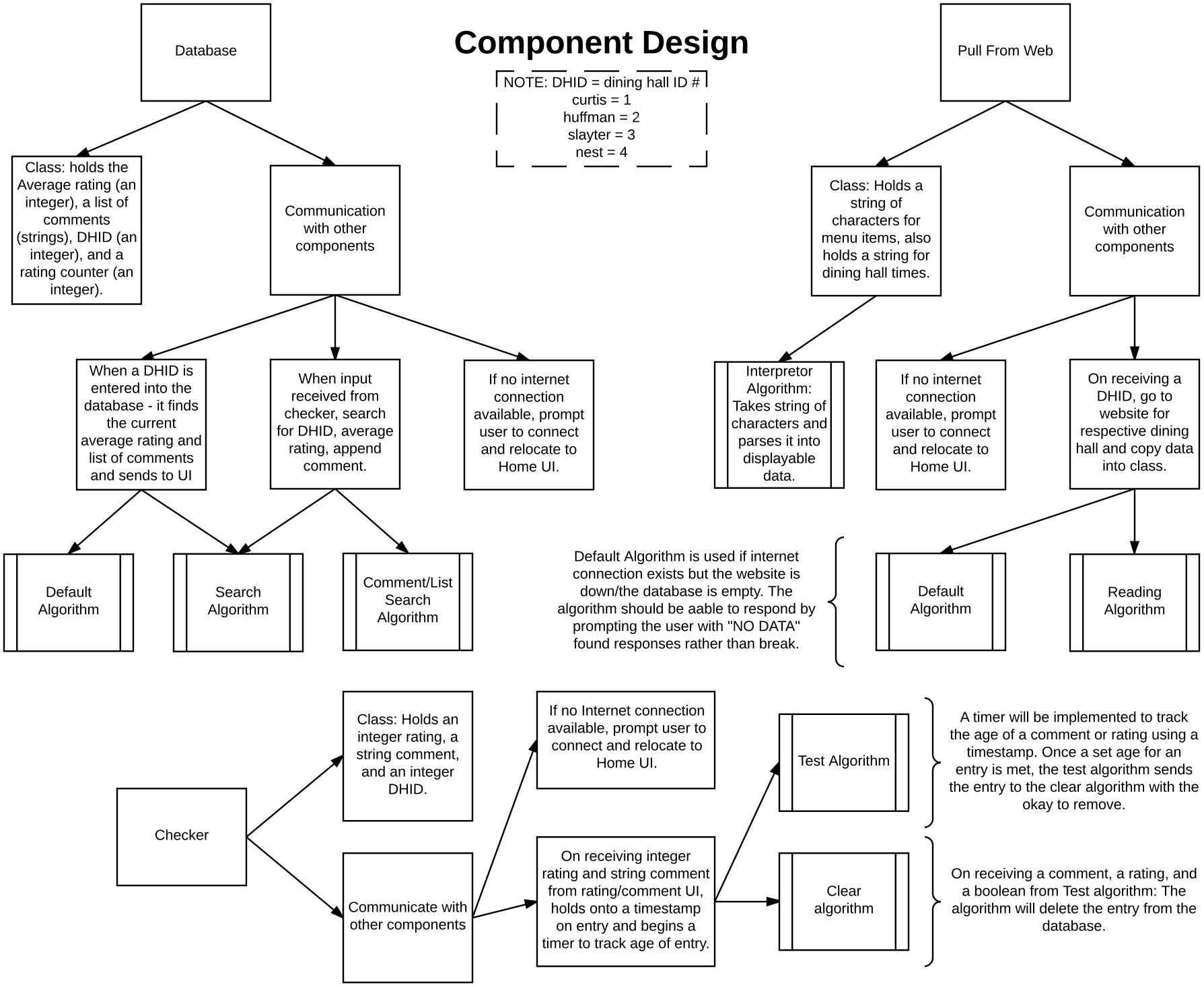
- Buttons:
  - Plus button - add comment, leave name

Architectual Design

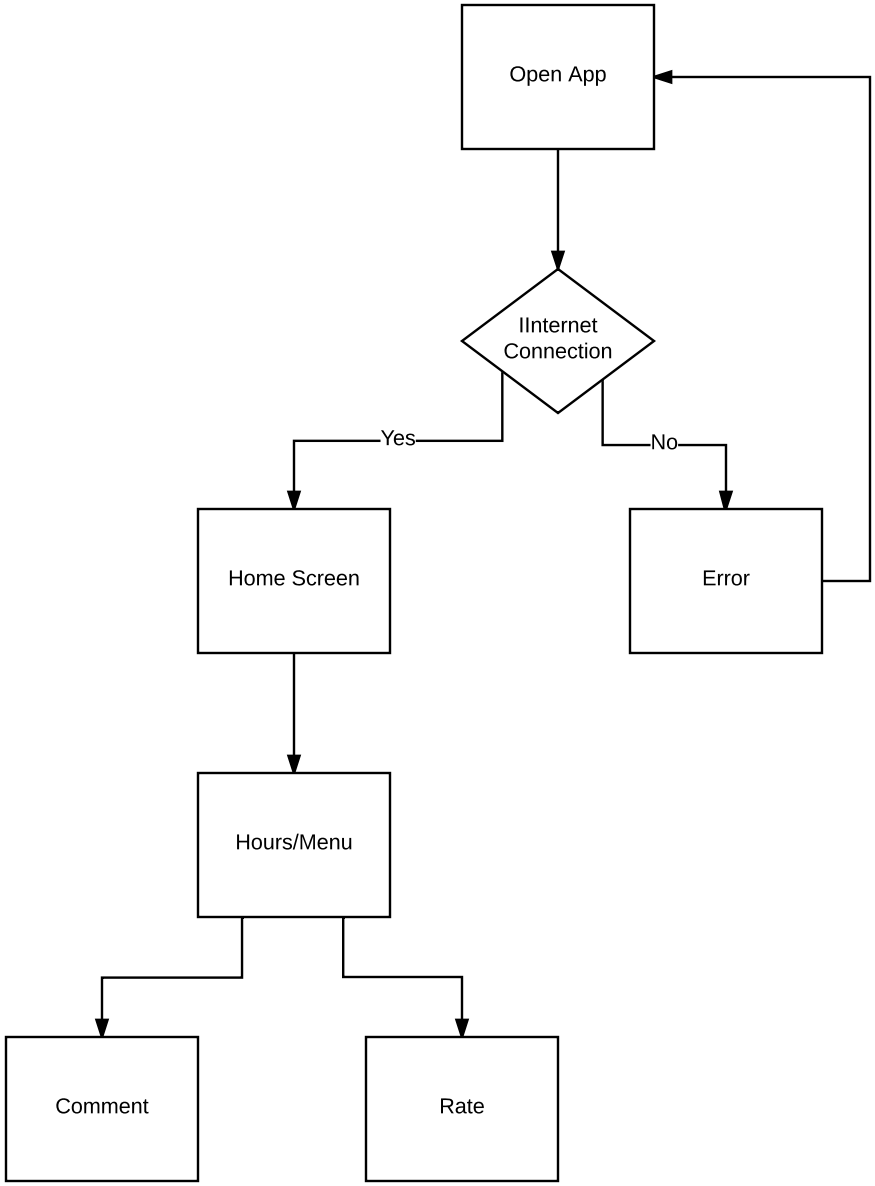



Help Button
Back button to return to home UI
NAME OF APP
?
Curtis
Huffman
Slayter
Nest
Select a Dining Hall
NAME OF APP
Dining Hall Name Hours
MENU
Subsection for different food options
Rate/ Comment
Swipe left or right to pass trhough all Dining Halls
NAME OF APP
Dining Hall Name Hours
MENU
Rate/ Comment
Home UI
Menu UI
When Rate/Comment Button is presed
User Interfaces
NAME OF APP
Rating System
Comment text box
Submit
Rating and Commenting UI

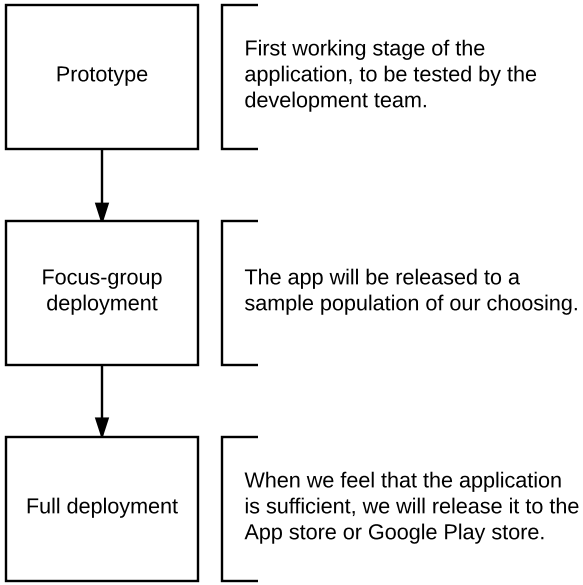
Component Design



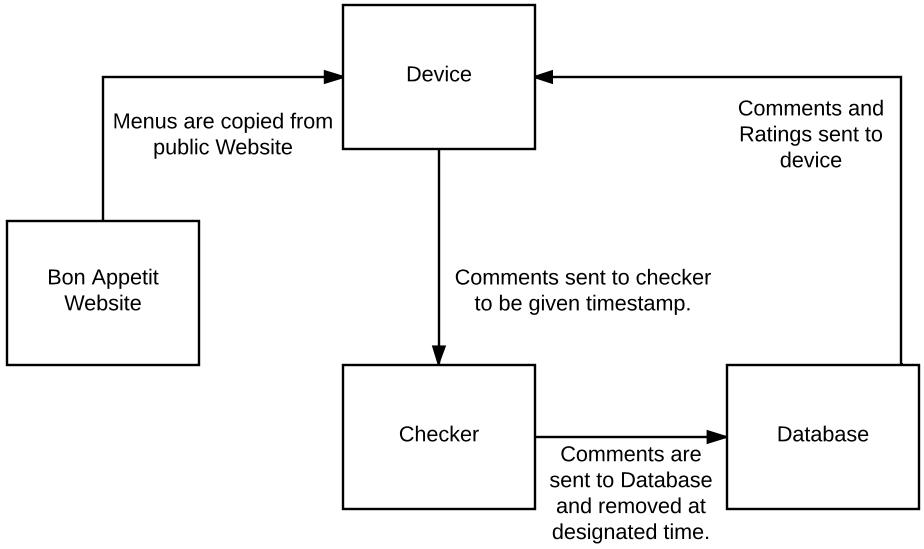
Acitivity Diagram



Deployment Design



Data Design



```swift
//
//  AppDelegate.swift
//  DatabaseDine
//
//  Created by Jacob on 12/3/17.
//  Copyright © 2017 Jacob. All rights reserved.
//

import UIKit
import CoreData
import Firebase

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?   //Initialize application


    internal func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.

        FirebaseApp.configure() //Creates the connection to Firebase
for commenting


        return true
    }

    func applicationWillResignActive(_ application: UIApplication) {
        // Sent when the application is about to move from active to
inactive state. This can occur for certain types of temporary
interruptions (such as an incoming phone call or SMS message) or when
the user quits the application and it begins the transition to the
background state.
        // Use this method to pause ongoing tasks, disable timers, and
invalidate graphics rendering callbacks. Games should use this method
to pause the game.
    }

    func applicationDidEnterBackground(_ application: UIApplication) {
        // Use this method to release shared resources, save user
data, invalidate timers, and store enough application state
information to restore your application to its current state in case
it is terminated later.
        // If your application supports background execution, this
method is called instead of applicationWillTerminate: when the user
quits.
    }
```

```
    func applicationWillEnterForeground(_ application: UIApplication)
{
        // Called as part of the transition from the background to the
active state; here you can undo many of the changes made on entering
the background.
    }

    func applicationDidBecomeActive(_ application: UIApplication) {
        // Restart any tasks that were paused (or not yet started)
while the application was inactive. If the application was previously
in the background, optionally refresh the user interface.
    }

    func applicationWillTerminate(_ application: UIApplication) {
        // Called when the application is about to terminate. Save
data if appropriate. See also applicationDidEnterBackground:.
    }


}

//
//  SlayterClass.swift
//  Dineson
//
//  Created by Federico Read on 11/10/17.
//  Copyright © 2017 testing. All rights reserved.
//

import UIKit
import WebKit
class SlayterClass: UIViewController {

    @IBOutlet weak var slayterWebView: WKWebView!

    @IBOutlet weak var Slayter_status: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
        let url = URL(string:"http://denison.cafebonappetit.com/cafe/
slayter-market/") //webview link
        let request = URLRequest(url:url!)

        slayterWebView.load(request)
        //segues from Slayter to curtis pages
        let RightSwipe = UISwipeGestureRecognizer(target: self,
action: #selector(SlayswipeAction(swipe:)))
```

```
        RightSwipe.direction = UISwipeGestureRecognizerDirection.left
        self.view.addGestureRecognizer(RightSwipe)

        self.dateCheck()
    }

    func dateCheck()
    {
        let today = Date().dayOfWeek()

        if today == 1 || today == 7
        {
            //print("Sunday")
            self.weekend()
        }

        else
        {
            self.weekDay()
        }
    }

    func getTime() -> (hour:Int, minute:Int, second:Int) {
        let currentDateTime = Date()
        let calendar = NSCalendar.current
        let component =
calendar.dateComponents([.hour,.minute,.second], from:
currentDateTime)
        let hour = component.hour
        let minute = component.minute
        let second = component.second
        return (hour!,minute!,second!)
    }


    func closed(endTime:String)
    {
        Slayter_status.text = "CLOSED until"+endTime
        Slayter_status.backgroundColor = UIColor.red
        Slayter_status.textColor = UIColor.white

    }

    func opened(endTime:String)
    {
        Slayter_status.text = "Open! Declining NOT accepted until
"+endTime
        Slayter_status.backgroundColor = UIColor.green
        Slayter_status.textColor = UIColor.white
    }
```

```
    func openedDeclining(endTime:String)
    {
        Slayter_status.text = "Open and accepting Declining until
"+endTime
        Slayter_status.backgroundColor = UIColor.green
        Slayter_status.textColor = UIColor.white
    }

    func weekDay()
    {
        let time = getTime().hour

        switch time
        {
        case 0: openedDeclining(endTime: " 11")

        case 7...10: openedDeclining(endTime: " 10:45")
        case 11...12: opened(endTime: " 1")
        case 13...23: openedDeclining(endTime: " 1")
        default:closed(endTime: " 7")
        }
    }

    func weekend()
    {
        let time = getTime().hour

        switch time
        {
        case 12...23: openedDeclining(endTime: " 2:30")

        default:closed(endTime: " 12")
        }
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to do a
little preparation before navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?)
{
        // Get the new view controller using
```

```
segue.destinationViewController.
        // Pass the selected object to the new view controller.
    }
    */

}

extension UIViewController
{
    @objc func SlayswipeAction(swipe:UISwipeGestureRecognizer)
    {
        switch swipe.direction.rawValue {
        case 2:
            performSegue(withIdentifier: "Slay2Curt", sender: self) //
segues name and location
        default:
            break
        }
    }
}
//
//  HuffmanClass.swift
//  Dineson
//
//  Created by Federico Read on 11/10/17.
//  Copyright © 2017 testing. All rights reserved.
//

import UIKit
import WebKit


class HuffmanClass: UIViewController {

    @IBOutlet weak var huffmanWebView: WKWebView!

    @IBOutlet weak var HuffStatus: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.

        let url = URL(string:"http://denison.cafebonappetit.com/cafe/
huffman-cafe/") //webview link
        let request = URLRequest(url:url!)

        huffmanWebView.load(request)
        //segues from Huffman to slayter pages
        let RightSwipe = UISwipeGestureRecognizer(target: self,
```

```
action: #selector(HuffswipeAction(swipe:)))
        RightSwipe.direction = UISwipeGestureRecognizerDirection.left
        self.view.addGestureRecognizer(RightSwipe)
        self.dateCheck()
    }

    func dateCheck()
    {
        let today = Date().dayOfWeek()

        if today == 1 || today ==  7
        {
            //print("Sunday")
            self.weekend()
        }
        else
        {
            //print("Saturday")
            self.weekDay()
        }
    }

    func getTime() -> (hour:Int, minute:Int, second:Int) {
        let currentDateTime = Date()
        let calendar = NSCalendar.current
        let component =
calendar.dateComponents([.hour,.minute,.second], from:
currentDateTime)
        let hour = component.hour
        let minute = component.minute
        let second = component.second
        return (hour!,minute!,second!)
    }


    func closed(endTime:String)
    {
        HuffStatus.text = "Will be CLOSED until"+endTime
        HuffStatus.backgroundColor = UIColor.red
        HuffStatus.textColor = UIColor.white

    }

    func opened(endTime:String)
    {
        HuffStatus.text = "Open until  "+endTime
        HuffStatus.backgroundColor = UIColor.green
        HuffStatus.textColor = UIColor.white
    }
```

```
    func openContinental(endTime:String)
    {
        HuffStatus.text = "Hot food starts at  "+endTime
        HuffStatus.backgroundColor = UIColor.green
        HuffStatus.textColor = UIColor.white
    }

    func openSnack(endTime:String)
    {
        HuffStatus.text = "Snack Break ends at  "+endTime
        HuffStatus.backgroundColor = UIColor.green
        HuffStatus.textColor = UIColor.white
    }

    //template for hours func uptoNight()
    // {
    //
    //     let time = getTime().hour

    //     switch time
    //     {
    //     case 09...20: opened(endTime: "21") //set time for 09:00 to
20:59
    //     default:closed(endTime: "4:30")
    //     }

    // }
    func weekDay()
    {

        let time = getTime().hour

        switch time
        {
        case 1...6: closed(endTime: " 7:45")
        case 7:
            let minute = getTime().minute
            if minute < 45{
                closed(endTime: " 7:45")
            }
            else{
                opened(endTime:" 10:45")
            }
        case 8...10:
            let minute = getTime().minute
            if time == 10 && minute < 45 {
                closed(endTime: " 11")
            }
            else{
                opened(endTime:" 10:45")
```

```
        }
    case 11...12:
        let minute = getTime().minute
        if time == 11 && minute < 30{
            closed(endTime:" 11:30")
        }
        else{
            opened(endTime: " 1")
        }
    case 13...16:
        let minute = getTime().minute
        if time == 15 && minute >= 45{
            closed(endTime: " 5")
        }
        else{
            openSnack(endTime:" 4:45")
        }
    case 17...18:
        opened(endTime:" 8")

    default:closed(endTime: " tomorrow")
    }

}


func weekend()
{

    let time = getTime().hour

    switch time
    {
    case 1...8: closed(endTime: " 9")
    case 9: openContinental(endTime: " 10")

    case 10...12:
        opened(endTime:" 2:30")

    case 15: closed(endTime: " 4:30")

    case 16...18:
        let minute = getTime().minute
        if time == 16 && minute < 30{
            closed(endTime: " 4:30")
        }
        else{
            opened(endTime:" 7")
        }
```

```
            break

        default:closed(endTime: " tomorrow")
        }

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}

extension UIViewController
{
    @objc func HuffswipeAction(swipe:UISwipeGestureRecognizer)
    {
        switch swipe.direction.rawValue {
        case 2:
            performSegue(withIdentifier: "Huff2Slay", sender: self) //
segue name and location
        default:
            break
        }
    }
}
    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to do a
little preparation before navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?)
{
        // Get the new view controller using
segue.destinationViewController.
        // Pass the selected object to the new view controller.
    }
    */



//
//  CurtisClass.swift
//  Dineson
//
//  Created by Federico Read on 11/10/17.
//  Copyright © 2017 testing. All rights reserved.
//

import UIKit
```

```
import WebKit

extension Date {    //used for quick view dining times
    func dayOfWeek() -> Int? {
        let calender: Calendar = Calendar.current
        let component: DateComponents = (calender as
NSCalendar).components(.weekday, from: self)
        return component.weekday
    }
}

class CurtisClass: UIViewController {

    @IBOutlet weak var curtisWebView: WKWebView!

    @IBOutlet weak var Curtis_status: UILabel!



    override func viewDidLoad() {
        super.viewDidLoad()

        self.dateCheck()

        // Do any additional setup after loading the view.
        let url = URL(string:"http://denison.cafebonappetit.com/cafe/
curtis-cafe/") //webview link
        let request = URLRequest(url:url!)

        curtisWebView.load(request)

        let RightSwipe = UISwipeGestureRecognizer(target: self,
action: #selector(CurtswipeAction(swipe:)))
        RightSwipe.direction =
UISwipeGestureRecognizerDirection.left   //segues from Curtis to
huffman pages
        self.view.addGestureRecognizer(RightSwipe)


    }

    func dateCheck()
    {
        let today = Date().dayOfWeek()

        if today == 1 || today ==  7
        {
            //print("Sunday")
            self.weekend()
        }
```

```
        else
        {
            //print("Saturday")
            self.weekDay()
        }
    }

    func getTime() -> (hour:Int, minute:Int, second:Int) {
        let currentDateTime = Date()
        let calendar = NSCalendar.current
        let component =
calendar.dateComponents([.hour,.minute,.second], from:
currentDateTime)
        let hour = component.hour
        let minute = component.minute
        let second = component.second
        return (hour!,minute!,second!)
    }


    func closed(endTime:String)
    {
        Curtis_status.text = "Will be CLOSED until"+endTime
        Curtis_status.backgroundColor = UIColor.red
        Curtis_status.textColor = UIColor.white

    }

    func opened(endTime:String)
    {
        Curtis_status.text = "Open until "+endTime
        Curtis_status.backgroundColor = UIColor.green
        Curtis_status.textColor = UIColor.white
    }

    func openContinental(endTime:String)
    {
        Curtis_status.text = "Hot food starts at  "+endTime
        Curtis_status.backgroundColor = UIColor.green
        Curtis_status.textColor = UIColor.white
    }

    func openSnack(endTime:String)
    {
        Curtis_status.text = "Snack Break ends at  "+endTime
        Curtis_status.backgroundColor = UIColor.green
        Curtis_status.textColor = UIColor.white
    }

    //template for hours func uptoNight()
```

```
    // {
    //
    //     let time = getTime().hour

    //     switch time
    //     {
    //     case 09...20: opened(endTime: "21") //set time for 09:00 to
20:59
    //     default:closed(endTime: "4:30")
    //     }

    // }
    func weekDay()
    {

        let time = getTime().hour

        switch time
        {
        case 1...6: closed(endTime: " at 7:45")
        case 7:
            let minute = getTime().minute
            if minute < 45{
                closed(endTime: " at 7:45")
            }
            else{
                opened(endTime:" 10:45")
            }
        case 8...10:
            let minute = getTime().minute
            if time == 10 && minute < 45 {
                closed(endTime: " at 11")
            }
            else{
                opened(endTime:" 10:45")
            }
        case 11...13: opened(endTime: " 2:45")

        case 14:
            let minute = getTime().minute
            if minute >= 45{
                openSnack(endTime: " 4:15")
            }
            else{
                opened(endTime:" 2:45")
            }
        case 15:
            openSnack(endTime: " 4:15")

        case 16:
```

```
        let minute = getTime().minute
        if minute < 15{
            openSnack(endTime: " 4:15")
        }
        else if minute < 30{
            closed(endTime: " at 4:30")
        }
        else{
            opened(endTime:" 7")
        }
    case 17...18:
        opened(endTime:" 7")

    default:closed(endTime: " tomorrow")
    }
}

func weekend()
{
    let time = getTime().hour

    switch time
    {
    case 1...8: closed(endTime: " at 9")
    case 9: openContinental(endTime: " 10")

    case 10...14:
        let minute = getTime().minute
        if time == 14 && minute >= 30{
            closed(endTime: " at 4:30")
        }
        else{
            opened(endTime:" 2:30")
        }

    case 15: closed(endTime: " at 4:30")

    case 16...18:
        let minute = getTime().minute
        if time == 16 && minute < 30{
            closed(endTime: " at 4:30")
        }
        else{
            opened(endTime:" 7")
        }

        break

    default:closed(endTime: " tomorrow")
    }
```

```
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to do a
little preparation before navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?)
{
        // Get the new view controller using
segue.destinationViewController.
        // Pass the selected object to the new view controller.
    }
    */

}

extension UIViewController
{
    @objc func CurtswipeAction(swipe:UISwipeGestureRecognizer)
    {
        switch swipe.direction.rawValue {
        case 2:
            performSegue(withIdentifier: "Curt2Huff", sender: self) //
segue name and location
        default:
            break
        }
    }
}
//
//  ViewController.swift
//  Dineson
//
//  Created by Federico Read on 10/31/17.
//  Copyright © 2017 testing. All rights reserved.
//

import UIKit


class ViewController: UIViewController {
```

```
   // @IBOutlet var docPopView: UIView!
    @IBOutlet var docuPopView: UIView!

   // @IBOutlet weak var visualEffectVie: UIVisualEffectView!

    @IBOutlet weak var visualEffectView: UIVisualEffectView!

    var effect:UIVisualEffect!


    override func viewDidLoad() {   //Main interface
        super.viewDidLoad()
        effect = visualEffectView.effect    //used for popup
        visualEffectView.effect = nil

        docuPopView.layer.cornerRadius = 5
    }

    func animateIn() {  //popup appear
        self.view.addSubview(docuPopView)
        docuPopView.center = self.view.center

        docuPopView.transform = CGAffineTransform.init(scaleX: 1.3, y:
1.3)
        docuPopView.alpha = 0

        UIView.animate(withDuration: 0.4){
            self.visualEffectView.effect = self.effect
            self.docuPopView.alpha = 1
            self.docuPopView.transform = CGAffineTransform.identity

        }
    }

    func animateOut(){  //popup disapear
        UIView.animate(withDuration: 0.3, animations: {
            self.docuPopView.transform =
CGAffineTransform.init(scaleX: 1.3, y: 1.3)
            self.docuPopView.alpha = 0

            self.visualEffectView.effect = nil
        }) { (success:Bool) in
            self.docuPopView.removeFromSuperview()
        }
    }

    @IBAction func popUpLoad(_ sender: AnyObject) { //on help button
press
        animateIn()
    }
```

```
    //@IBAction func popUpLoad(_ sender: AnyObject) {
    //    animateIn()
    //}
    @IBAction func popUpClose(_ sender: AnyObject) {    //on selecting
"back"
        animateOut()
    }

    //@IBAction func popUpClose(_ sender: AnyObject) {
   //     animateOut()
   // }



    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}

//
//  SweetsTableViewController.swift
//  DatabaseDine
//
//  Created by Jacob on 12/3/17.
//  Copyright © 2017 Jacob. All rights reserved.
//

import UIKit
import FirebaseDatabase
import FirebaseAuth

class SweetsTableViewController: UITableViewController {

    var dbRef:DatabaseReference!  //To be used for sending comments
    var sweets = [Sweet]()    //Holds string of characters for
comments

    override func viewDidLoad() {
        super.viewDidLoad()
        dbRef = Database.database().reference().child("Curtis")    //
initial value (top) is dining hall name

        startObservingDB()                                          //
Read current data to mobile device

    }
```

```
    func startObservingDB () {//Read in data from Firebase and display
on table

        dbRef.observe(.value, with: { (snapshot:DataSnapshot) in
            var newSweets = [Sweet]()

            for sweet in snapshot.children {
                let sweetObject = Sweet(snapshot: sweet as!
DataSnapshot)
                newSweets.append(sweetObject)
            }

            self.sweets = newSweets
            self.tableView.reloadData()

        })
    }

    @IBAction func addSweet(_ sender: Any) {//on "+" button press
        //First user prompt (comment)

        let sweetAlert = UIAlertController(title:"New Comment",
message: "Enter your Comment", preferredStyle: .alert)
        sweetAlert.addTextField { (textField:UITextField) in
            textField.placeholder = "Your comment"
        }
        sweetAlert.addAction(UIAlertAction(title: "Send",
style: .default, handler: { (action:UIAlertAction) in
            let sweetContent = sweetAlert.textFields?.first?.text
            //Second user prompt (Name)
            let sweetAlert2 = UIAlertController(title:"Name", message:
"Enter your Name", preferredStyle: .alert)
            sweetAlert2.addTextField { (textField:UITextField) in
                textField.placeholder = "Your name"
            }
            sweetAlert2.addAction(UIAlertAction(title: "Send",
style: .default, handler: { (action:UIAlertAction) in
                //update reference in database
                if let sweetName = sweetAlert2.textFields?.first?.text
{
                    let sweet = Sweet(content: sweetContent!,
addedByUser: sweetName)

                    let sweetRef =
self.dbRef.child(sweetName.lowercased())

                    sweetRef.setValue(sweet.toAnyObject())
                }
            }))
```

```
            self.present(sweetAlert2, animated: true, completion: nil)
        }))
        self.present(sweetAlert, animated: true, completion: nil)

    }



    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
        return sweets.count
    }

    //Establish table view, vertically inputing comments (older
towards the bottom)
    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
"Cell", for: indexPath)

        let sweet = sweets[indexPath.row]

        cell.textLabel?.text = sweet.content
        cell.detailTextLabel?.text = sweet.addedByUser

        return cell
    }




}
//
//  Sweet.swift
//  DatabaseDine
//
//  Created by Jacob on 12/3/17.
//  Copyright © 2017 Jacob. All rights reserved.
//

import Foundation
import FirebaseDatabase
```

```
struct Sweet {

    let key:String! //Title name (slayter, Name)
    let content:String! //Child of key
    let addedByUser:String! //Child of key and key name
    let itemRef:DatabaseReference?  //location in database

    init (content:String, addedByUser:String, key:String = "") {
        self.key = key
        self.content = content
        self.addedByUser = addedByUser
        self.itemRef = nil
    }

    init (snapshot:DataSnapshot) {  //initialize location
        key = snapshot.key
        itemRef = snapshot.ref

        let snapshotValue = snapshot.value as? NSDictionary
        if let sweetContent = snapshotValue!["content"] as? String {
            content = sweetContent
        }else {
            content = ""
        }

        let snapshotValue2 = snapshot.value as? NSDictionary
        if let sweetUser = snapshotValue2!["addedByUser"] as? String {
            addedByUser = sweetUser
        }else {
            addedByUser = ""
        }
    }

    func toAnyObject() -> Any { //create comment with timestamp
        return ["timestamp": ServerValue.timestamp(), "content":
content, "addedByUser": addedByUser]
    }

}
//
//  HuffmanTableViewController.swift
//  DatabaseDine
//
//  Created by Jacob on 12/4/17.
//  Copyright © 2017 Jacob. All rights reserved.
//

import UIKit
import FirebaseDatabase
import FirebaseAuth
```

```
class HuffmanTableViewController: UITableViewController {

    var dbRef:DatabaseReference!  //To be used for sending comments
    var sweets = [Sweet]()     //Holds string of characters for
comments

    override func viewDidLoad() {
        super.viewDidLoad()
        dbRef = Database.database().reference().child("Huffman")    //
initial value (top) is dining hall name
        startObservingDB()                                          //
Read current data to mobile device
    }


    func startObservingDB () {//Read in data from Firebase and display
on table
        dbRef.observe(.value, with: { (snapshot:DataSnapshot) in
            var newSweets = [Sweet]()

            for sweet in snapshot.children {
                let sweetObject = Sweet(snapshot: sweet as!
DataSnapshot)
                newSweets.append(sweetObject)
            }

            self.sweets = newSweets
            self.tableView.reloadData()

        })
    }

    @IBAction func addSweet(_ sender: Any) {    //on "+" button press
        //First user prompt (comment)
        let sweetAlert = UIAlertController(title:"New Comment",
message: "Enter your Comment", preferredStyle: .alert)
        sweetAlert.addTextField { (textField:UITextField) in
            textField.placeholder = "Your comment"
        }
        sweetAlert.addAction(UIAlertAction(title: "Send",
style: .default, handler: { (action:UIAlertAction) in
            let sweetContent = sweetAlert.textFields?.first?.text
            //Second user prompt (Name)
            let sweetAlert2 = UIAlertController(title:"Name", message:
"Enter your Name", preferredStyle: .alert)
            sweetAlert2.addTextField { (textField:UITextField) in
                textField.placeholder = "Your name"
            }
            sweetAlert2.addAction(UIAlertAction(title: "Send",
```

```
style: .default, handler: { (action:UIAlertAction) in
                //update reference in database
                if let sweetName = sweetAlert2.textFields?.first?.text
{
                    let sweet = Sweet(content: sweetContent!,
addedByUser: sweetName)

                    let sweetRef =
self.dbRef.child(sweetName.lowercased())

                    sweetRef.setValue(sweet.toAnyObject())
                }
            }))
            self.present(sweetAlert2, animated: true, completion: nil)
        }))
        self.present(sweetAlert, animated: true, completion: nil)

    }



    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
        return sweets.count
    }

        //Establish table view, vertically inputing comments (older
towards the bottom)
    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
"CellH", for: indexPath)

        let sweet = sweets[indexPath.row]

        cell.textLabel?.text = sweet.content
        cell.detailTextLabel?.text = sweet.addedByUser

        return cell
    }
```

```
}
//
//  SlayterTableViewController.swift
//  DatabaseDine
//
//  Created by Jacob on 12/4/17.
//  Copyright © 2017 Jacob. All rights reserved.
//

import UIKit
import FirebaseDatabase
import FirebaseAuth

class SlayterTableViewController: UITableViewController {

    var dbRef:DatabaseReference!    //To be used for sending comments
    var sweets = [Sweet]()          //Holds string of characters for
comments

    override func viewDidLoad() {
        super.viewDidLoad()
        dbRef = Database.database().reference().child("Slayter")    //
initial value (top) is dining hall name
        startObservingDB()                                          //
Read current data to mobile device
    }


    func startObservingDB () {          //Read in data from Firebase
and display on table
        dbRef.observe(.value, with: { (snapshot:DataSnapshot) in
            var newSweets = [Sweet]()

            for sweet in snapshot.children {
                let sweetObject = Sweet(snapshot: sweet as!
DataSnapshot)
                newSweets.append(sweetObject)
            }

            self.sweets = newSweets
            self.tableView.reloadData()

        })
    }

    @IBAction func addSweet(_ sender: Any) {    //on "+" button press
        //First user prompt (comment)
        let sweetAlert = UIAlertController(title:"New Comment",
message: "Enter your Comment", preferredStyle: .alert)
```

```
        sweetAlert.addTextField { (textField:UITextField) in
            textField.placeholder = "Your comment"
        }
        sweetAlert.addAction(UIAlertAction(title: "Send",
style: .default, handler: { (action:UIAlertAction) in
            let sweetContent = sweetAlert.textFields?.first?.text
            //Second user prompt (Name)
            let sweetAlert2 = UIAlertController(title:"Name", message:
"Enter your Name", preferredStyle: .alert)
            sweetAlert2.addTextField { (textField:UITextField) in
                textField.placeholder = "Your name"
            }
            sweetAlert2.addAction(UIAlertAction(title: "Send",
style: .default, handler: { (action:UIAlertAction) in
                //update reference in database
                if let sweetName = sweetAlert2.textFields?.first?.text
{
                    let sweet = Sweet(content: sweetContent!,
addedByUser: sweetName)

                    let sweetRef =
self.dbRef.child(sweetName.lowercased())

                    sweetRef.setValue(sweet.toAnyObject())
                }
            }))
            self.present(sweetAlert2, animated: true, completion: nil)
        }))
        self.present(sweetAlert, animated: true, completion: nil)

    }



    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
        return sweets.count
    }

    //Establish table view, vertically inputing comments (older
towards the bottom)
    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
```

```
"CellS", for: indexPath)

        let sweet = sweets[indexPath.row]

        cell.textLabel?.text = sweet.content
        cell.detailTextLabel?.text = sweet.addedByUser

        return cell
    }




}
```