Abstract

Josh, Matt and I knew we wanted to develop a mobile app for our senior project because that is what we found ourselves to be most passionate about during our time here at Cal Poly. We started to think of problems we wanted to solve using an application and we came up with a couple ideas but chose to expand on one. Students come to Cal Poly every year new to the area and the food options San Luis Obispo provides. Many of the restaurants in SLO offer a variety of deals to the community and most of them to the college students living here. We wanted to make it easier for freshman all the way up to seniors and beyond to have easy access to all the restaurant deals around SLO. Also, we wanted to provide these restaurants with a space to advertise their deals in one space where they can maximize the people seeing them. We based our prototype on the Cal Poly community but wanted to make the ability to expand our application to other college communities.

We knew there were other application out there for users to be able to see restaurants around them, rate and review them, and potentially see deals. However, we wanted to focus our application around college students by creating a UI that fits the college culture and have sayings that appeal to their community. In this app you can; create a personal account, discover new and old restaurants, rate and favorite restaurants, view new and old deals, use deals, search for nearby restaurants, and manage your current favorite restaurants. We also provided a way for a restaurant owner to create their own account, manage their restaurant, add deals, remove deals, and add images of their restaurant and deals to entice this specific base of users. One of our first steps when starting to develop KeyLime was deciding the roles each of us would have on the project. Programming assignments can be tricky to divide up if not done correctly because of merge conflicts. Below describes how we decided to spread out the work because of individual's skill sets and thinking about the components of an application (frontend, backend, and a database).

Eli:
Users/Account. Creating users and storing them in a database (Firestore). Dealing with how users interact and what is displayed in their accounts.

Josh:
Restaurants and admin accounts (deals, reviews). Dealing with how restaurant owner's accounts are displayed and what privileges they have and how to post new deals on their restaurant's page.

Matt:
UI, navigation, display and filtering. How the story of the application should flow for the user and deciding what the best UI elements to use are. Figuring out how to display and filter restaurant search results from a database.

Introduction
KeyLime was a challenge our team hadn’t really encountered before. We spent a good amount of time researching and trying to figure out the design decisions we would need to make in order to have a working prototype at the end of two quarters. Prior to working on the application, we thought of design decisions that we anticipated we would have to make in order to produce something great to use. We organized the decisions into three categories that encompasses our application.

**Frontend (UI):**

This is arguably the most important part of an application because this is what the users will be interacting with. We needed to make sure the frontend would be comfortable for the user, intuitive to use, and have everything they want to see. To make the UI comfortable we need a consistency throughout the app, for example a color palette or icons. Making our application intuitive was more challenging and required input from multiple sources. An app is intuitive if the user doesn’t need a lot or any help to navigate the application and use it for what it’s made for. To do this our team needed to look at competitor apps to see what they did right to implement those ideas and see what they could have done better on to avoid those issues. Lastly, we needed to make decisions on what the user would want to see and have access to in an app like KeyLime. These decisions include things like the ability to change your password, have a separate view for a user’s favorite and rated restaurants, allow filtering by distance, and allowing an existing account to be able to become an admin account.

**Backend:**

Although the frontend is the most important aspect of an application to users, the backend is one of two of the most important parts of KeyLime to me the developer. Backend decisions we anticipated to make include; breaking up the project between team members while reducing the number of merge conflicts, creating models for storing the data that we would grab from the database, and how/when we wanted to update the local copies of the user or restaurant. It is always a challenge working on a software project in a team setting because all the members want access to the whole project at once. We also knew we needed a way to recognize if a user has used a deal or if the restaurants want the users to use the deal more than once.

**Database:**

This is the second of the two most important things of the application to me a developer. This was one of the first major design decisions we anticipated tackling first because it decides a lot of other decisions as well. Although the database is a smaller portion of our project it is one of the most important because of how much our application uses the stored data and relies on its atomicity.
Requirements

In the case of our project we were our own pseudo-external customer. We operated as a restaurant owner that wanted an easy way to access the college student market and expand. By looking through the lens of a restaurant owner we were able to come up with requirements we needed to hit.

- **Discovery**: The ability to see restaurants and information about them quickly. A way to expand and see more information about a specific restaurant. The ability to filter restaurants visible to see ones close by or by rating.
- **Deals**: The ability to see deals and information about them quickly like which restaurant they belong to and see if the user has already used a deal. A way to expand and see more information about a specific deal.
- **Search**: The ability to locate a restaurant or a type of restaurant on a map and be able to access the specific restaurant searched to interact with it.
- **Settings**: The ability to look at a user’s account information and see the restaurants and deals they’ve used. A sign out option and other account management functionality.
- **Admin**: The ability to be an owner of a restaurant and manage the information displayed be the restaurant. The displayed information includes; pictures, type of food, creating deals, and removing deals.

These requirements require an application with the three pillars described above, frontend, backend, and a database.

**Background/related work**

Other applications are sure to follow our requirements of a frontend, backend, and database. Our team has had some experience with application development and implementing some of the tools that we ended up using. I have worked on an Android augmented reality project called AR Tic-Tac-Toe. This Android project required a
frontend, backend, and database, like most do, but was heavily focused on the backend and making augmented reality work. Most, if not all, mobile application development requires an object-oriented language like Java or Kotlin with event-based programming and when we wanted to build our project for iOS, we were able to pick up Swift quicker than we would have with no mobile app development history. I ended up using Google’s Firebase which is similar to the database we chose to use for KeyLime.

I have also worked on a windows application that had similar requirements, but the execution was different on a Windows device and the goal was much different. The Windows application, called CalcBook, required specific knowledge in structural engineering since the application handled calculations of codes and stress on structures. KeyLime doesn’t need specific knowledge to be developed but I believe to be taken to the next level marketing and restaurant business knowledge would be required. Our application can almost be seen as a service application for restaurant owners. This means a restaurant business knowledge would be very valuable in designing an app like KeyLime.

Key Design Decisions

In this section I will be talking about the major design decisions I made for the team and one’s we made together.

The first minor design decision we made was figuring out what we wanted to name our application. For real development teams this is part of the project that might take the longest because it is arguably one of the most important decisions for marketing purposes. When I first was trying to come up with a name, I looked at the successful companies today in a similar industry to see what they did correct when deciding a name. The first one that came to mind was Uber. They have a popular food delivery service called Uber Eats and I thought we could kind of play off of that and name our application Uni Eats where the Uni stands for university since our application is based on colleges and the towns they reside. Next, I thought of Apple and how simple it is but no w is everywhere. I admire the simplicity, so I thought of Lime, but that is almost a direct copy, so I decided to add something to it but still keep it simple. KeyLime came to mind and I thought it was perfect, so we asked some people what they thought and at first they were unsure but everyone always came back to us remembering the name.

The first major design decision we made was after we had decided to build a mobile app for our senior project. Matt and I had both had experience with mobile app development but in Android’s Kotlin while Josh was about to head into his mobile app development class in Swift. Personally, I wanted to learn Swift because I know how much it is used in industry and mobile app development is a path I want to pursue. Kotlin is a great language and has a lot to offer, especially because Matt and I have good experience with Android, but we felt Swift was a great learning opportunity and a fun way to end our time at Cal Poly. After choosing the language the development environment came easier. There are two major IDEs when writing swift, one being
Xcode, and two being JetBrains’ AppCode. AppCode is made by a company known for great IDEs. I have experience with one of theirs called IntelliJ. Once again however, Apple’s Xcode is built by the creators of Swift, Apple. Meaning Xcode is the premium builder of iOS apps and is also free for all Mac owners so it seemed fitting to learn Xcode at the same time we were learning Swift.

I was tasked with choosing the database we were going to use for storing our data we needed for the restaurants, deals and users. During the first week I had done a lot of research on the best database to use for an iOS app. I had found a couple of options but two stuck out, Firebase and SQLite. SQLite is local database (data stored/processed on a device) with SQL interface and is free! However, I didn’t have any experience with SQLite, and I was already trying iOS for the first time in this project, but I did have experience with Firebase. We decided to go with a version of Firebase called Firestore that offered us a way to authenticate a user multiple different ways through third party applications like Facebook, Google, Twitter, etc. Firebase also provides an API called FirebaseUI for easy to implement UI controls for signing in users. Although I didn’t end up implementing FirebaseUI (I did it the old way instead) the documentation still helped me implement the user sign in UI elements. Firestore is basically free to an extent (limit on the number of reads and writes) but we never saw the need in the near future to purchase more access. Ultimately Firestore was a more comfortable, well documented, newer database that I thought fit perfectly with the requirements of the project.

I was also tasked with figuring out how we wanted to do users and admin accounts for KeyLime. This task included designing the database and the models used to access the database. After choosing the database to be Firestore I was able to get some help from their authentication API to do a lot of the heavy lifting when signing in users and storing that login information. I did however need to design a model of a user to be used to store other information of the user. I needed a unique identifier to be able to store each user without the worry of one be overwritten. One idea I had was to just have an incrementing number be the Id of each user. As I was researching Firebase authentication, I found that each user created by the authentication was given a unique user Id. I then used that unique user Id as the reference for a corresponding model in Firestore. Having a unique identifier for a restaurant was a whole different problem. Any easy solution was to use the name and address of the restaurant combined as a key instead of an incremented integer. This ended up working out well because it allowed the look up for a restaurant by name to require less search time. However, this wasn’t as safe as having a randomly generated code as the key as we have for the users. It was a tradeoff I wanted to make since our app is targeted toward smaller communities in specific areas.

The final collection I designed into the database was the deals. Instead of deals being a field in each restaurant I wanted to cover the case where a deal was more or less universal and there would only have to be one copy. Finding a unique identifier for deals using, for example, a name and address can’t happen because deals can be a part of the same restaurant and have any name the admin wants. Again, Firestore helps
me out by providing a unique randomly generated key consisting of 20-character string with numbers and letters. Along with figuring out how we were going to store the deals we had to come together to discuss how a user was actually going use a deal. We wanted a simple and quick way for a user to use a deal and for this way to be safe for the restaurant where they wouldn’t get people using the deal over and over if they didn’t want them to. I envisioned this happening two ways, either the user uses a code online to enter in the deal, or the user uses a QR code attached to each deal and we keep track of which user used which deal. This approach requires more a lot more logic and extra work that we didn’t have time for when making our prototype KeyLime. We decided to keep it simple, which required a little more human interaction, but still reliable. Currently the user presents the deal to the restaurant employee and that employee marks on the users KeyLime application that the deal was used. The user cannot undo this action. This is definitively not the best solution, but it works for what we wanted to accomplish in our minimal viable product. I talk more about this in Future Work.

Since I was leading the charge for user interaction with KeyLime, I worked on the design of the favorite and rating systems. There aren’t really any great alternatives to the standard favoriting system that major software companies use like Instagram and Facebook where the user presses a button and the button changes states to notify the user that they have favorited the item. The rating system on the other hand does have the ability to be done a different way and keep the intuitiveness and flow. Before I talk about my design decision, I want to acknowledge the fact that we took the standard rating scheme of using numbers and implemented phrases that suits the college community. Knowing this, phrases take up more space than numbers do when trying to convey similar information. Two ideas came to mind to allow the user to rate a restaurant and display it. First being a pickerView which displays information in a vertical table that you can scroll through, not seeing all the options at once (therefore minimizing the room needed to convey all the information about rating). One problem that arises with this method is knowing which selection you made and the other is displaying the current user rating when the view is loaded. I found that the other solution was clearer and offered some design and UI appeal that the pickerView did not. The other solution being a five-star rater that displays the phrase corresponding to the star above it. I found this method (and so did most testers) more intuitive. The UI appeal that was added was changing the stars into limes that helped tie in the overall theme.

Roadblocks

I encountered a good number of roadblocks during the whole application development process with my assigned roles. Also, as a team we encountered many roadblocks as well.

There were two major group roadblocks we encountered. First to use Firebase and other SDKs we had to use cocoapods which was something none of us had heard before. We did some research on using Pods and at first it seemed to go smoothly but Git “hated” us and our project. Not all of the pod files that were added to our project
didn’t seemed to get pushed so everyone but the original person who created the Pod files wouldn’t be able to run the application. This could have been due to the Xcode built in source control since we weren’t using the command line. Eventually we figured it out and manually pushed over 300 files and got the project to be pulled and built. The second large roadblock we encountered was the fact that Xcode uses storyboards to organize the flow of an app. This was a smaller issue than Git but caused us to lose some of our work to overwrites. When we first started using Xcode we didn’t know about the almost unresolvable conflicts you get when two people work in the same storyboard. When we first encountered this issue, we didn’t know of a great solution, so we just went into the storyboard file and used a text editor to manually resolve the conflicts. Terrible idea. Eventually we learned that using multiple storyboards was the best way to work. This way people could work on storyboards at the same time just not the same exact one.

User creation and implementation went smoothly for the most part but adding Facebook login was more challenging than I first thought. I first tried to use FirebaseUI which is an all in one API for Firebase apps like KeyLime that allows third party logins and provides all the correct buttons and functions. This was too good to be true because I wasn’t able to find any helpful Swift documentation on the API and wasn’t able to implement it. Instead I went manual and had to register the application with Facebook through there dashboard.

One roadblock I fast that I didn’t expected occurred when I was implementing the rating system on restaurants. I initially stored the overall rating for the restaurant and then for each user I stored which restaurants they have rated. This initial design only partially worked because the overall rating would get updated locally based on the user rating and pushed to the database but if the user wanted to change the rating they gave there was no way of doing this. For a work around I decided to store the actual user rating in the database as new field in the user in an array. The rating for a specific restaurant would be stored at the same index as the restaurant IDs in the rated restaurant array. By storing the user rating, I was able to calculate the overall restaurant rating before they had even rated it then calculate the new rating based on the rating they currently gave.

The final roadblock I would like to talk about is time management. This roadblock isn’t project specific to KeyLime but can and probably will occur with most projects that are large in magnitude comparatively to what you’ve done before. We had two quarters to work on our Senior Project and we originally thought that is a lot of time, so we don’t need to rush. Knowing we had a difficult first quarter we decided that we would do the design process the first quarter along with setting up the Swift foundation and then do most of the programming work in the second quarter. Looking back I wish we had been able to manage our time better and start the deep programming late in the first quarter so we would have been able to come back second quarter and get some testing done early and maybe even do some redesign based on user feedback like we were able to do with the rating system.
Future Work

The best applications never die (even when they probably should). I think we were able to get our minimal viable product out there but weren’t able to accomplish much past that. Here I will give a high-level update schedule for what I would like to see for KeyLime.

Version 2.0

The first update should be to the backend of KeyLime. When I was first designing how the current user was being managed in the app, I didn’t think a lot about the cost of updating and then grabbing the user from the database and storing it locally. At first, I did those calls often but as I kept coding, I began to think about the optimization more and how often I am updating the database. I want version 2.0 to have that optimization in every view that requires the current user by passing the local copy around and only reading and writing when necessary. That optimization can actually be applied to a lot of our grabs from the database. Another optimization that I would want to be applied in version 2.0 is in the rating system. Currently I created my own radio button model to hold an array of buttons and update them accordingly based on the one pressed. Each button has a similar function but as of version 1.0 each button has its own function instead of a common one with different values passed in as parameters.

Version 3.0

The second update would be towards the UI and making it cleaner and more attractive. Currently most of the views used are the default ones given by Apple in their SDK. By using the default UI elements from Apple, the application loses some uniqueness. Updating the application with new custom elements and views would add to the flow of the app along with the feel. We tried to do as much customization to the default elements as we could to help with consistency. For example, most of the buttons and accent colors are a lime green that matches the logo we used for KeyLime. The restaurant view has barebone labels and collection views that have the potential to look very cool, but we decided as a group to focus more on the functionality of the application than the layout and graphic design. Since that functionality will be mostly complete and where we want it to be version 3.0 could be focused on graphics and the overall aesthetic of KeyLime. I would like to add some dimensionality to the application by giving the effect of different layers on the UI. The dimensionality adds to the attractiveness but could also benefit the user experience and effectiveness by using helpful icons, colors, and positioning. Different colors attract people to different areas on the screen. For example, something that is red usually means it is a “delete” or “undo” type action, while yellow is a rewarding color and used for notifying a user of something good they have done.

Version 4.0 and beyond

I believe KeyLime has a lot of potential and with more time can become a really unique and useful app that is used throughout the country at most university cities. To get there we need to implement some of our extra goals we set in our project proposal. These goals include adding a social aspect to the application between users, fleshing
out what a user can do, changing the way a deal is used, and some other minor functionality fixes.

- **Social Media:** The most popular applications today are the ones that connect users to their friends and favorite people. By adding friends into KeyLime we expand the market that we can reach. You would be able to add and remove friends along with viewing their profile where you can see the restaurants the favorited and the ratings, they gave those restaurants. We could keep track and display the number of favorites a restaurant has to promote that social aspect of the app.
- **User ability:** This ties into the social media aspect of the app. I would like the user to be able to do more things in the application like leave comments on deals or even restaurants. This would keep the user in the application longer and give them more to do.
- **Deals:** We had a hard time thinking of the best way for us to implement deals and keep track of when a deal was used and couldn’t be used anymore. Currently the way we have it works and is easy to keep track of but can easily be worked around. I would like to reimplement the deals with maybe a QR system or some other code system but that involves a lot more overhead and extra communication with the restaurants.

**Reflection**

I have really enjoyed the projects I’ve done in my senior year. Capstone, Senior Project and Mobile App Development have been amazing because of the reflection I have gotten to do on these projects. Each time I turn one in I realize things that I could have done differently and implement those new ideas in the next project I work on. As for KeyLime there are multiple things I would have done differently. Broadly speaking, it is hard to design something perfectly because challenges arise that you don’t see coming when you first start brainstorming.

I talked about this a little in Future Work, but I wish we could have designed the deals better and found a way to manage the deals when a user wants to use one. In the current implementation I believe it is not properly supervised and relies on the human interaction too much.

During the research and design process I tried to think of everything we needed in the database in each collection but along the KeyLime journey I needed to add a couple of fields and repopulate the database accordingly. Thankfully we implemented an easy way to grab restaurants off of the map, but it would have still been a lot easier to know each field I needed to accomplish all of the necessary tasks. Coming into the project I had never used Swift before and hadn’t used Xcode. As mentioned in Roadblocks Git caused us a lot of problems during the whole of the project, but mostly during the first couple of weeks. We didn’t know the best way to work simultaneously on an iOS project with storyboards and we paid the price by losing someone’s work at times. We eventually learned to split the Main Storyboard into
multiple storyboards and that helped immensely with reducing the amount of merge conflicts.

**Conclusion**

Senior Project has been a great learning experience and an even better example of learn by doing. I wanted to choose a project that allowed me to both gain more teamwork experience and work with my friends. We jumped into Senior Project as an opportunity to try something new that we might not have been able to explore as much as we would have wanted to while at Cal Poly and show off what we have learned at the same time. All while having fun. KeyLime is one of the projects I’m most proud of because it was basically all taught and down by us. It turned out to be something really cool and has the potential to be useful for many college communities. Through the Key Design Decisions and the Roadblocks, we were able to overcome we created something from scratch that we said we were going to do. KeyLime is one of the few projects that I’ll be able to continue to work on after college and I’m looking forward to spending more time on it and seeing where we can take it.