Indy-5 CompChores
(UC-266 CompChores)

CS 4850 - Section 01

Prof. Sharon Perry

Fall 2022

https://spksu2022.github.io/

https://drive.google.com/file/d/1a6sTlGvBDrxSKtPnXzPiQcl4f_Ok98BZ/view?usp=sharing

Team Lead	Myers Berry
Team Member	Ian Snyder

Abstract

Nobody likes chores, and kids surely don't like doing them. But if there was a competitive aspect and a reward, we might be able to change their minds. To do so, we will create a competition for who can get more chores completed. Parents will login and create a family, invite their kids to their family and behind posting chores. Based on their difficulty rating and estimated time, a certain number of points will be given for each chore. As the members of the family complete the chores, their score will go up and they will be able to compare how well they are doing against their family members. This could lead to parents offering incentives to the highest score throughout the week and give the kids a friendly competition while being productive. CompChores accomplishes these goals by implementing Amazon Web Services (AWS)— specifically AWS Cognito using the Amplify API, API Gateway, Lambda, and Relational Database Services (RDS) to provide a seamless experience for families to cooperatively compete.

CompChores implements Amazon Web Services as a desktop application with the Dart programming language and the Flutter framework. The application Frontend/UI is built using the Flutter built in Material library. The backend uses AWS services that allow for a scalable, secure, robust backend to support the desktop application. Users are able to create an account, join a family, create chores, assign points and priorities to the chores, mark the chores as complete obtaining points, view a scoreboard of other family members, and leave the family or change user profile information. This is all completed through api calls that run lambda functions to query the database. The result is a fast, fluid desktop application that tracks and manages chores amongst users.

Table of Contents

Background

About

Honing discipline is a lifetime learning experience. Something that one learns to develop, but not overnight. Instead, we can always fall back on motivation as a catalyst in force. Competitive Chores, hereon shortened to CompChores, aims to deliver such reasoning for errands that need to be done around a common environment. This competitive cooperative experience is driven to provide users the incentive needed to (even if falsely) become motivated in completing tasks against and with user coordinated teams, or ‘families.’

This report will detail the processes and methods followed by the CompChores development team in reaching this achievement. Initial discussion will focus on understanding the intended oriented goals of the project and how the choice of learning to improve competency led to the decision of Dart and Flutter for the development environment. The section will discuss the benefits of such a modular pair as well as certain shortcomings that certain other directions could have possibly avoided. With the process and methods, this section will also aim to understand the original process of interaction the team discussed when drafting the project.

Backend discussion will focus on the implementation of Amazon Web Services; specifically, CompChores utilizing the proprietary Cognito with Amplify API, the API Gateway, Lambda, and Relational Database and the molding of each component to format a capable application with real time services communications. The backend discussion will relate the challenges that came with trying to

provide multiple services between user and application without compromising the usability experience of users.

Frontend discussions will focus rather on how variable the presentation of the desktop application became as development further continued. The section will make sense of the balancing act between streamlining functions of the project without having to sacrifice the users' experience or mistakenly mis-implementing core functions that could effectively jeopardize the database from the users fingertips. To maintain all these moving parts seamlessly, Source Control will be provided its own section of discussion. The focus of Source Control will be to clarify the necessity of its forethought and the moments of significance throughout the project's development.

Following the discussion of its creation, this report will provide a Results and Findings discussion to elaborate the difference between what was desired and what was achieved. Here it will reiterate the reasoning behind how development of the project shifted tonally and how the various components within the project were affected in turn.

Lastly, the report will culminate with the Conclusion in which CompChores discusses varying opinions on the project. This experience as an undergraduate capstone project reflects differently between the team members, but will bridge the common and differing experiences each member had during development. Reflection will be included, as well as prospected future plans that would have undergone should the project have been continued. It is here that the report draws to discuss the educational importance of the experience behind CompChores and whether the project is considered a success.

Programming Language & Framework

When deciding between a programming language and framework to use for the application, we kept many factors into consideration. Factors such as, flexibility amongst all platforms, ease of use/development time, performance of framework, and customizability. All of these factors were fulfilled and more when looking at dart and flutter! Dart is a new powerful object oriented programming language. It has a growing developer community behind it, fast and reliable compilers, and type-safe syntax. As for flutter, the framework not only offers the flexibility of using the same code-base amongst all platforms, but it also allows for rapid development with its built-in widgets that allow for fast programming. Using dart and flutter proved to be extremely helpful in designing the desktop application as it offered many widgets to use as well as libraries to integrate with Amazon Web Services.

Backend Research

The backend was chosen because of the flexibility of Amazon Web Services. The ability to plug each service into the program and use them for their intended purpose allowed an intuitive and quick solution to back end needs. We used Amazon Cognito, Amazon Amplify, Amazon RDS, Amazon Lambda, Amazon API Gateway, and Amazon Cost Manager. These tools allowed us to build a robust backend that not only supported our program, but allowed for proper scaling! We used Amazon RDS as a relational database and built our programming data through it. We then used Amazon Amplify to allow our users to authenticate through it in order to access our database. Then once a user was allowed access, we used API gateway in order to query the lambda functions to alter the database. This allowed a division of workloads in order to allow for greater scalability amongst the users of the database! Then finally,

our lambdas not only allowed us to control the flow of traffic, but they allowed us to see the frequency and the time for each lambda. This allows us to ensure that each lambda is being used to its fullest.

Functional Requirements

Front End Requirements:

- Display Login Page
- Valid Login
- Invalid Login
- Valid Create Account
- Invalid Create Account
- Transition to Homepage
- Display Homepage
- Display Modules
- Functioning Chore List
- Functioning Settings
- Functioning Leaderboard
- Functioning Create Chore Module

Backend Requirements:

- Amplify Integration
- Cognito Integration
- RDS Operational
- Lambda Integration
- API Gateway Integration
- Create Family in DB

- Edit Family in DB
- Delete Family in DB
- Retrieve Family Data from DB
- Lambda Functions for Data Retrieval

Developing a desktop application wasn't beyond the realm for the development team. The idea of a service-based application formed quickly in a discussion to understand how to bridge lessons garnered from prior projects. The main scope: understanding how to create a functional, user-oriented application which could provide a real time database and other operand functions. Thus, the discussion shifted towards providing a service that would permit the team to familiarize such proficiency.

The unfortunate size of CompChores development team meant that a desktop application was the best of limited choices. To circumvent the shortcomings, CompChores came to the conclusion that the development environment had to permit for scalability, an ability to transform accordingly without requiring a large-scale engineering operations team. An environment that could trim as easily as it could include with little fuss or troubling fallouts. An environment that Dart and Flutter could provide, which will be further discussed in Process and Methods Used.

The development team concluded that with the tight development environment, CompChores could feasibly communicate from user to database, permit changes if necessary, all without being too taxing for the scope of its purpose. To clarify, CompChores was focused on a simple service-oriented mindset for its development. Lay the groundwork of a working chore management software before allocating any resources or time to user experience enhancements. *Grounded*, could be considered the word of the project.

For a traditional commercial project, one could consider this inefficient, dangerously conservative in thinking that could jeopardize longevity of the project. For the development team, this

was a necessary acceptance as how to realistically achieve a tangible project for C-Day without taking on futile bulk. The primary goal of the project was to better the team's aptitude towards making a realistic product, even with a team size sometimes less than half of that to other projects.

This meant constructing the practical basis and working up. This app would have the capabilities of speaking with a Relational Database and, when permitted, would allow changes to be done and communicated back to the application all in real time. As facile as the scope of the project was to be retained, it did not serve as an excuse to provide an equally scarce product. CompChores would be able to achieve this learning goal while being packaged as an entirely plausible, commercially transferable application.

Test Plan

TEST	PASS/FAIL
INITIATE DESKTOP APP	
TRANSITION BETWEEN LOGIN AND CREATE ACCOUNT	
LOGIN VALID ACCOUNT	
LOGIN INVALID ACCOUNT	
CREATE VALID ACCOUNT	
CREATE ACCOUNT; INVALID INFORMATION	
CREATE ACCOUNT; MISSING INFORMATION	
CREATE ACCOUNT; ACCOUNT ALREADY EXISTS	
LOGIN USING CREATED ACCOUNT	
RECEIVED EMAIL CONFIRMATION	
VALIDATED EMAIL STATUS	
DISPLAY HOME PAGE MODULES	
SUCCESSFULLY ADD CHORE	
ADD CHORE; INVALID, MISSING INFORMATION	
COMPLETE CHORE	
COMPLETE CHORE; POINTS AWARDED	
CHORE LIST UPDATES	
CHORE LIST SORTS	
LEADERBOARD; DISPLAYS FAMILY MEMBERS	

LEADERBOARD; TOTAL POINTS UPDATED	
LEADERBOARD; INDIVIDUAL POINTS UPDATED	
LEADERBOARD; RANKINGS UPDATE	
LEADERBOARD; DISPLAY POINTS HISTORY	
SETTINGS; LEAVE FAMILY	
SETTINGS; EDIT ACCOUNT INFORMATION	
SETTINGS; DISPLAY FAMILY INFO	
CLOSE APP; REOPEN; PERSISTENT LOGIN OPERATIONAL	
LOGOFF OPERATIONAL	
ACTIVE GRAPHIC BACKGROUND DISPLAYING	

Development

Source Control

As for source control, the CompChores development team resigned to GitHub to manage progress. Due to the small size of the development team, there was less concern for accidental concurrency on the same packages that could impact progress. If either member were to work on the repository, team members were able to notify the other by means of text, discord, or any other IM mediums. Regardless, CompChores still retained the previously established GANTT charts to the best of the team's ability and any modifications to the original schedule was discussed in person during the in-person meetings. Development remained relatively stable, in that future scheduling was reactive to the agreed progress being made up to each meeting. Constant pro-active discussions between intent and results made sure that each updated version was understood between the team members, keeping source control just as stable with little to no retractions to progress.

To ensure that translations of newer or updated packages, it was vital that the changes were conducted in isolated environments that simulated in practice and in implementation. That way,

changes to be carried out were vetted to ensure risk aversion and as smooth translation to the next phase. Regardless of understood and agreed following changes, it was vital to ensure that some level of documentation was present at each stage. This came beneficial during Amplify which, being a recent development of AWS, was turbulent to changes and allowed team members to understand how each alteration was necessary.

Methods

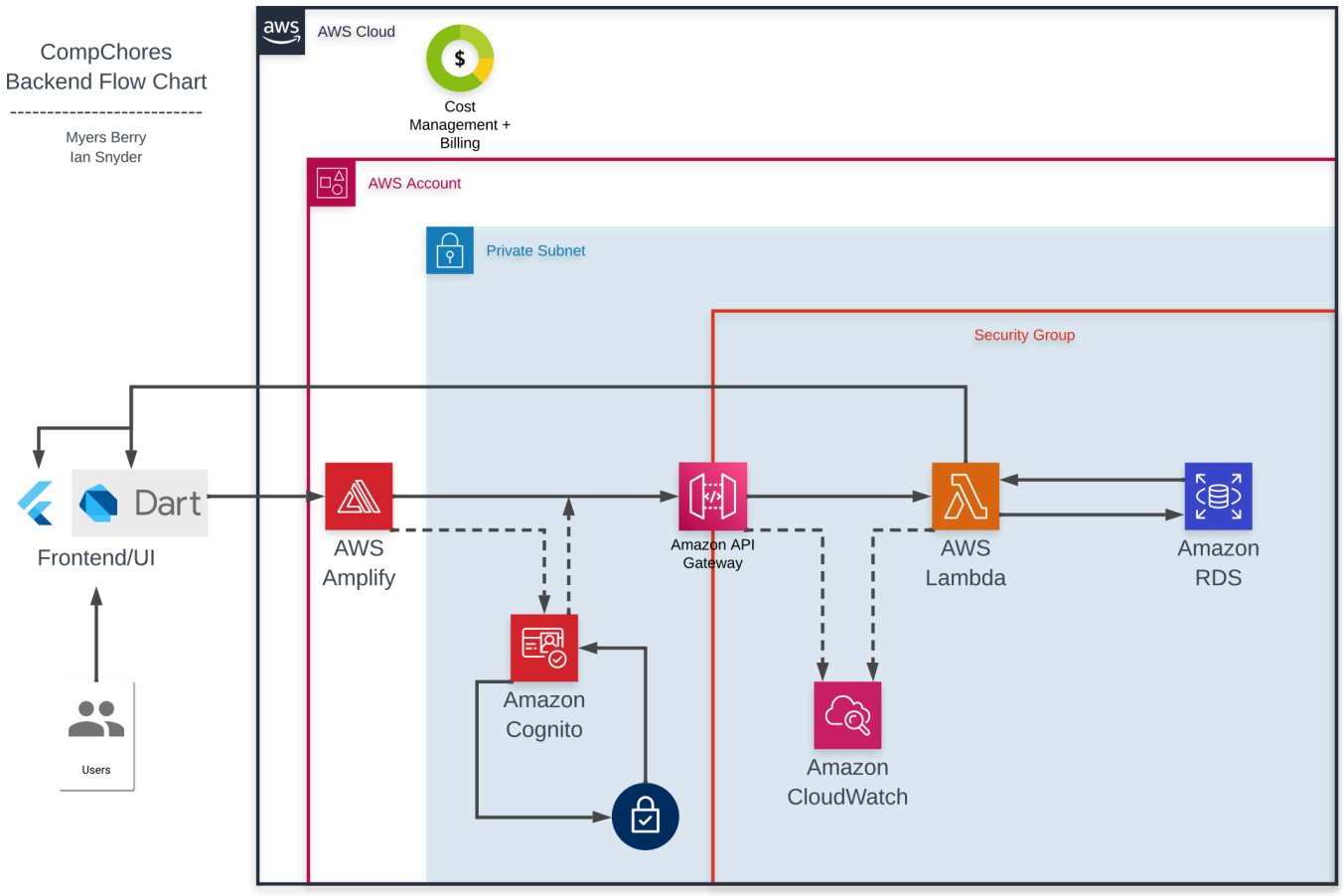
CompChores determined that the Desktop Application would act as a medium for essentially communicating with the adaptive database. Dart and Flutter would be utilized in constructing the interface portion of the application and by means of the Dart Materials library would construct containers for which the user would delegate prompts sent to the Database. CompChores would limit the users to only what prompts were provided as opposed to unrestrained freedom as the inherent risk to operations was too great when permitting communications to the Amazon Web Services.

For the chosen services, the vast array of potential services provided meant that the development team had to be reserved in selecting practical modules. Services were chosen on the merits that they could improve both the efficacy of the app's entirety as well as the users' experience with the application. With the semester long time restraint, this meant that the services chosen had to have prior, helpful documentation which could be adapted and understood within an appropriate time as to not halt development time.

Backend/Flow of User

The CompChores backend consists of Amazon Web Services integrated together to provide account verification, account creation, backend authentication/security, database security via

controlled queries, query logs, and cost management. The current flow of the user is as follows:



Users first interact with the backend upon logging in. This is done through an Amazon Web Service known as AWS Amplify. Providing a Flutter API directly integrated with the rest of the backend, Amplify allows for integration with Amazon Cognito. Cognito pools users into various User Pools that allow for user management and User control. API Gateway is then used inline with Cognito to prevent anyone attempting to login without a Cognito account access to database queries. Going as far as to only allow users access if they have verified their account when the setting has been turned on. In addition to this, custom attributes can be attached to User's accounts for later use in various uses to which we used in querying the database. If a user has properly logged in, they will be allowed access to completing database queries which is done through AWS Lambda and a user token created upon

successful login. Lambda allows developers to create queries/functions to various services, our relational database in this case. Lambda and API Gateway integrate through setting up triggers. These are URL access points that can be called on the application to run the specified Lambda Function. API Gateway allows for deployment phases as well, this allows for the development and testing of Lambda Functions before production deployment. This integrates with AWS Cloudwatch to view logs of Lambda functions. These logs keep detailed records of function's completion times, errors, outputs, and timestamps of the call. Following a successful Lambda function leads to the integration with Amazon RDS. This is our MySQL Relational Database, running off of a MySQL-Community engine. RDS allows for the creation of various relational databases all with different speeds, costs, and functionality. We decided on the basic RDS Instance as the additional functionalities of the other database options were not necessary for the scope of the project.

Our database had two iterations. The first iteration consisted of four tables, a Family table, User table, Chore table, and a Scoreboard table. The Family table acted as the main table that other tables would reference through foreign keys. Family had a primary key of FamilyID that was an auto-incrementing integer value. This would be referenced in the Users and Chores tables as foreign keys. Allowing for queries to grab users and chores based on the family. Then the Scoreboard table used the primary key in Users in a similar fashion to grab only scoreboard data based on the user. This implementation allowed for the division of data based on family. This iteration was effective until AWS Amplify and Cognito were implemented, thus making the need for a Users table obsolete.

The next iteration consists of three tables. A Chores table, Scoreboard table, and a Family table which acts as the main table that the other two link to through Family's primary key. The Chores and Scoreboard tables use the primary key in Family as a foreign key to allow for queries grabbing information regarding the family to only grab information from the specified family. When making the

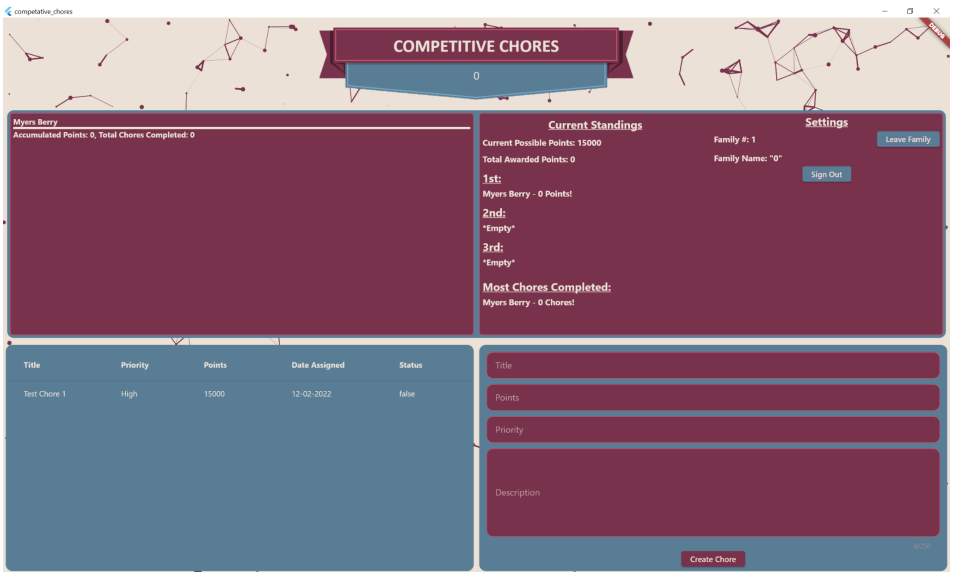
call to the lambda function, the custom attribute on the User's account containing the family ID number is used to query the proper information against the tables in the database.

Amazon Web Services' provided modules have an integral ability to work congruently with each other, granted implementation is successful. This meant ensuring that with the proper calls, users could easily undergo changes to the RDS without having to switch between contexts within the Dart app. With the AWS properly integrated, CompChores development team were able to mitigate the number of listeners, both active and passive, that were present. Fortunately, the development team was successful in supporting each service into the program after recognizing how each component could effectively be integrated. What this allowed us to achieve was an ability to essentially create an intricate highway to and from the RDS without compromising the users experience.

Login especially saw significant amelioration with the Amplify API allowing all the general abilities of conventional login modules to be inserted all from a single package. Function calls were sent through the AWS API Gateway, and the application was able to use Lambda to efficiently communicate database calls. Future prospects to the project were the use of S3 Bucket to provide further media enhancement support for the application. This sort of plug and play approach to CompChores meant that backend operations could be rapidly deployed and updated as fast as a programmer could come to understand how and where to utilize this. CompChores thus enhanced it's scalability potential, as the tier system with AWS could be adaptive to the demands of fluctuating number of users. Should the RDS demand more, CompChores could update a tier and without any time sacrifice be just as operational ready as before. These autonomous modules also had their own development cycles internally, meaning as the modules themselves were enhanced, so too could the capabilities of CompChores. Granted, this would require a keen monitoring of development schedules and efficient management of utilized modules, with the ever present risk of syntax modification and their dependencies.

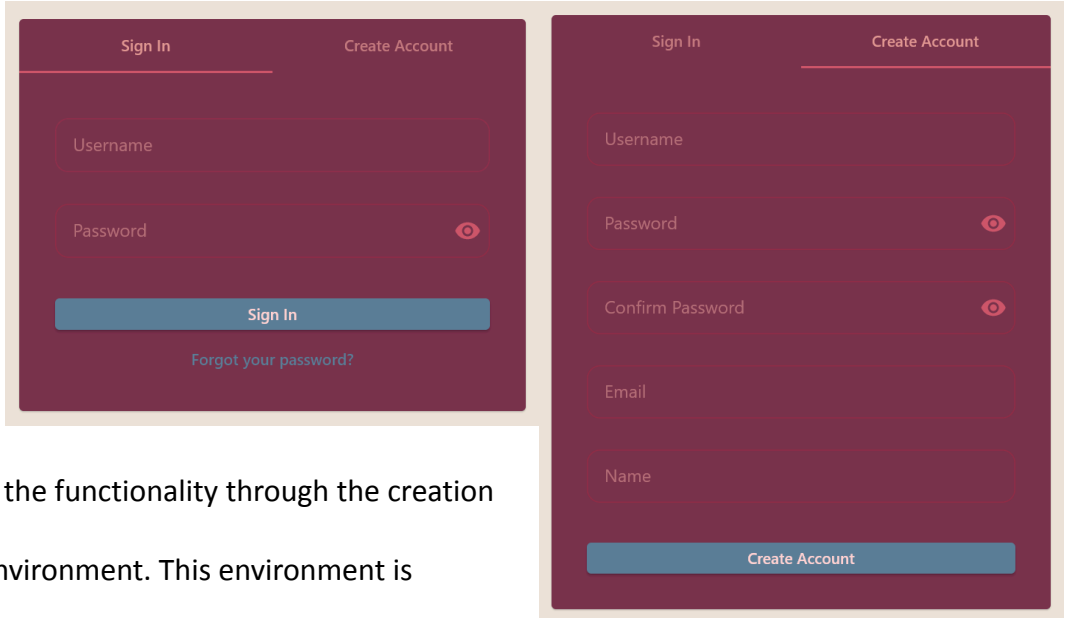
Frontend/UI

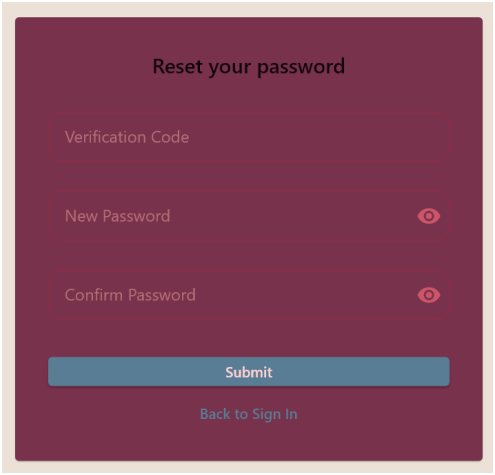
The UI of CompChores uses Flutter's default Material Library. This library integrates with the Dart programming language to create a fluid and beautiful UI experience without sacrificing functionality. Initially, CompChores was designed to have multiple pages that would break the application into three main sections, a scoreboard, a chore screen, and a main family home screen. This was done through the use of Dart's built in listeners that allowed portions of the application to be rebuilt in real-time as values of variables were changed. Allowing us to change the page based on an index value the user would change through a pop out menu. This was later changed to all be displayed on one main screen. Allowing us to make better use of the screen space and eliminate the need for users to constantly switch screens to see information. As tumultuous the integration of several Web Services to CompChores was, an advantage to this approach was the removal of the prior necessary listeners. No longer was it necessary to promote functions that distressed calls to updates and instead were able to provide live changes and updates from the single context provided on screen.



This adaptation was also conclusive to improving user experience with the application. After regressing the number of navigating pages to the single presentation, ease of functionality improved significantly.

Using AWS as a backend allowed for the integration of AWS Amplify! This service is in the developer phase in terms of integration with Flutter, but it allows us to create a login screen that directly connects to AWS Cognito. Allowing us to focus on the visuals of the screen as Amplify automatically creates the functionality through the creation of an AWS Backend Environment. This environment is created through the AWS Command Line Interface. This CLI walks the user through setup customizations and allows us to implement functionality such as email verification and account recovery! This allows the user to recover their account, reset the password, and allows the developers to manage each account with Cognito.





The overall theming and color scheme came from the development of our website. When creating the website a Jekyll theme called Merlot was used and provided a soft on the eyes theme which we managed to recreate in the application. This was done through the use of a styling class in which class variables can be called from anywhere in the program holding the specific fonts, colors, and alignments needed.

Results

Analysis

At the conclusion of the work cycle, CompChores was able to sustain a working development build that incorporated goals from both the Front End and Back End planning phases. In terms of the UI for the Desktop App, the incorporation of the AWS packages and desired functionality lead to mandatory retraction from the original interface design. This final presentation is a skimmed variant which presents all of the agreed vital information within a single display. In practice, this had benefits in the case of ergonomics on the users' end. This simplification to one screen for the DesktopApp permitted users to quickly open the app, conduct any necessary changes from the comfort of the single screen and then leave. This simplistic outcome of making chore management software as quick and simple to use with minimum fuss demonstrated an almost Occam's Razor level revelation that ergonomics holds an unprecedented degree of importance in the end. All these integrated AWS would mean little to nothing for one of the central pillars to a desktop application's success: the user base.

Utilizing specialized tools for compounding projects shows merit in the mélange of specialized tools rather than reliant on proprietary developments. It is important for one to know what they choose to integrate, since bloating a project without proper comprehension of when and where said package would most efficiently fit leads to sacrifice of optimization. However, what this project succinctly taught was that merely looking inward for optimization can lead to tripping on the front end. All too easy is it for one to find a potential fix or useful module without realizing how incompatible it would be wholly, as demonstrated by Amplify's API constraining non-modularity. This led to the

unfortunate death of the original graphic login initialization, but gave birth to a more streamlined login process once control of the API was reigned.

The Amazon Web Services themselves were beyond utilitarian. The sweat and effort that went into their incorporation proved fruitful in the end with efficient calls being made from device to database and vice versa. The AWS modules were treated as precision-based tools to an otherwise overall operation: specialized, congruent capable instruments for desired functions. One could argue that on a superficial level, CompChores serves more as a medium for these services. This does not retract the educational merit garnered from construing the appropriate channels to embed separate, near stand-alone elements in a functional presentation. It is a balancing act on the tight wire of harmony over the pitfalls of disjunction, and overloading your weight with unnecessary features only serves to become detrimental to the unity in its entirety.

Conclusion

In conclusion, this project proved to be a fantastic challenge while also a creative learning situation. Amongst our team we learned many of the skills it takes to start a project from the ground up, plan the roadmap for the future of the project, communication, source control, and working with requirements for a project. This project has challenged the skills we have learned through our journey of Computer Science at KSU while still giving us the freedom to make the project ours. We are grateful for the opportunity to work on a project of this nature and with people we may have not otherwise worked with. This project has been a great learning lesson and opportunity to show what we have learned throughout our time at KSU, we hope you feel the same!

References

Amazon API Gateway Documentation. https://docs.aws.amazon.com/apigateway/index.html.

Hollands, Mark. “Amplify.” *Amazon*, Publisher Not Identified, 2015, https://docs.aws.amazon.com/amplify/?icmpid=docs_homepage_fewebmobile.

Jenkins, Geoff. “Clouds Project Cloudwatch.” *Amazon*, Royal Meteorolog. Soc., 2000, https://docs.aws.amazon.com/cloudwatch/?icmpid=docs_homepage_mgmtgov.

“Material Library.” *Material Library - Dart API*, https://api.flutter.dev/flutter/material/material-library.html.

Musgrave, David. “Lambda.” *Amazon*, Europa Editions, 2022, https://docs.aws.amazon.com/lambda/?icmpid=docs_homepage_featuredsvcs.

“RDS.” *Amazon*, Research Defence Society, 2007, https://docs.aws.amazon.com/rds/?icmpid=docs_homepage_featuredsvcs.

Roose, Herman. “Cognito.” *Amazon*, De Auteur, 1987, https://docs.aws.amazon.com/cognito/?icmpid=docs_homepage_security.