Project 2: Documentation

Group Members

Seth Damany
Whitney Devine
Alexander Dunn
Jeffrey Kim
Haocheng Xue

Table of Contents

Technologies Employed

It is a hard requirement in this project to use Codename One (CN1) in Java 11. See CN1's website for more details: https://www.codenameone.com.

Project Goal

We aim to create an mobile app to keep track of time spent on multiple tasks, both related and unrelated, with an emphasis on productivity and task switching.

User Stories

User stories provide insight into how the user would like to interact with the app, and the reasons that they would do so. While not all user stories translate to fundamental requirements, they set up a list of design features with designated priorities. We explore these priorities further below, in the section *Design (page 3)*.

- As a user, I want to measure my time spent on an assortment of tasks so as to gain accurate metrics of my productivity.
- As a user, I want to create a new task so I can begin a new assignment or project and time track it as well.
- As a user, I want to quickly switch between tasks without wasting time so I can focus my time on the actual task.
- As a user, I want to group related tasks so I can track my time spent across various related tasks as a single group.
- As a user, I want the ability to modify a task's name or various other details in case a former detail no longer applies.
- As a user, I want to assign sizes to tasks so I can more easily budget out time for each task.
- As a user, I want the time elapsed to be accurate to the second so I can precisely quantify my time spent on each task.
- As a user I want the ability to fix errors that I make in accidentally starting or stopping a task early so that I can still get accurate information even if I make a mistake.
- As a user, I want to be able to non-destructively remove old tasks, so that they are not in my way but also leaving open the possibility that they will become relevant again and I will need to check their history or revive them.
- As a user, I want to see the time I spent on a task per day so I can get a better sense of my schedule and how my productivity has changed.
- As a user, I want to see how my time spent on a task compares to other tasks so that I could potentially reconsider its size or rebudget my time accordingly.

Requirements

Here is a set of fundamental requirements based on the Project's description and the class-wide Canvas discussion forum. Requirements 1-8 (bolded) provide the fundamental functionality desired for this project. The requirements listed after had lower priority but they represent the full scope of the mobile application.

REQ1: Start & stop task quickly
REQ2: Create new task
REQ3: Archive existing task
REQ4: Group related tasks
REQ5: Assign size (S/M/L/XL) to task
REQ6: See time spent on task(s)
REQ7: Persistent data storage
REQ8: Simple and clean UI
REQ9: Chart to visual size
REQ10: Edit and delete time spent on a task

The use-cases 1-13 are based on the fundamental project requirements REQ1-8. The subsequent use-cases represent the full scope of possible interactions one can have with our app.

UC1: Start task
UC2: Stop task
UC3: Add task
UC4: View task details
UC5: Edit task
UC6: Archive task
UC7: Unarchive task
UC8: Delete task
UC9: Add tag to task
UC10: Remove tag from task
UC11: View summary
UC12: Access task history
UC13: View task history
UC14: Edit task history
UC15: Visualize task history
UC16: Visualize time spent on multiple tasks

Design

Functional Priorities

- Start and stop tasks quickly (Critical)
- Create a new task (Critical)
- Archiving and unarchiving a task (Critical)
- Easily rename a task (Critical)
- Give tasks descriptions (Critical)
- Give task a size (Critical)
- Group tasks (via tagging) (Critical)
    - Persistent data store (Critical)

- Filter tasks in summary (High)
    - Selected duration between two dates
    - Size
    - Tags
- Access history of a task (High)
- Edit history of a task (with precision up to one minute) (Medium)

Non-functional Priorities

- Simple, clean, and non-garish design (Critical)
- Visual summary of tasks (Low)

Decisions

1. Tags for grouping tasks

Tags are preferable to a hierarchy of tasks. Hierarchy is more complicated to implement but also more importantly it is more complicated for the user to use. Tagging can do all the same level or grouping as a tree hierarchy of tasks (turn each split in the hierarchy into a tag that all subtrees/nodes share), and it can also do more grouping. For example in the hierarchical method, "Class A grading", "Class A lectures" both share Class A and would go under a Class A hierarchy. However, the task "Class B grading" is tricky as you want to group it with A's grading, but not also group it with B's grading, which is impossible unless we allow multiple inheritances which are getting too complex. This scenario is fixed with a tagging system with the tags "Class A" and "grading" which allow the grouping we desire.

2. One active task at a time

Multitasking seems like needless complexity. In most cases, a person when tracking time is only doing one task at a time, so automatically stopping the old task when starting a new task would improve the speed at which a user can switch tasks and remove the annoyance of manually stopping it. A person can only really do one task at a time. Multitasking would not accurately reflect the amount of time spent on each task. Any time the user wants to run two tasks, they are probably actually trying to get a single task to count for two different things, which is solved by tags. Also, a single running task is more visually appealing to distinguish than an alternative that requires finding a way to show multiple tasks are running.

3. Calendar weeks starting on Monday

We considered how a typical user would prefer the beginning of the week. Sunday is the beginning of the calendar week, but not the work week. This app is designed with an emphasis on productivity as it relates to the concept of time tracking and scheduling. If the user is using this app with work in mind, we believe that a week should begin on Monday.

Project Timeline

Week 1: **Designing the UI/UX**
We began with independently developing our designs. This way, we could each contribute our ideas carried over from Project 1 in a way that could be integrated in Project 2. After sharing our designs with each other, we started piecing together a tentative design that incorporated our favorite elements from each of our designs. Here we included features such as starting and stopping a single task at a time and using tags to group tasks together. We developed this final design on Figma, which can be found here: https://www.figma.com/file/EIHCWTO9YZqYc3f0k8L2hD/ECS-160-P2?node-id=0%3A1

- Designed the UI separately
- Collaborated on a UI design combining the best characteristics of our individual designs

Week 2: **Implementing the UI & Delegating Tasks**
After finalizing the design on Figma, our team quickly went to implementing the UI using CN1. We initially developed certain pages/forms separately - Alexander Dunn worked on the *Home Screen*, Jeffrey Kim worked on the *Archive Screen*, and Seth Damany worked on the *Task Details Screen*. Understanding that this workflow may lead to an inconsistent aesthetic across different pages, we delegated the tasks as follows: Jeffrey Kim and Seth Damany were assigned to UI. Alexander Dunn, Whitney Devine, and Hao Cheng were assigned to the backend/business logic.

Week 3: Finalizing the Prototype & Demo Video
At this point, the framework for the backend and frontend was mostly complete. During this week the team came together to connect the front end user interface with the backend business logic. The task of connecting the front end to the backend was led by Alexander Dunn and Seth Damany and was assisted by Jeffrey Kim. Jeffrey Kim worked on finishing the TaskSpan class as well as assisting Alexander and Seth on connecting the front end to the backend. Hao Cheng began work on creating a persistent data storage solution. Whitney Devine started work on creating a graph solution for the Summary Screen. Seth worked on creating the video for the final submission, with assistance from Alexander and Jeffrey in writing the script and providing feedback and suggestions.

Week 4: Integrating Charts and Persistent Data Storage
During this week, Seth Damany, Alexander Dunn, and Whitney Devine collaborated on integrating the graph solution for the Summary Screen. Hao Cheng mostly completed the persistent data storage solution however it was still buggy and needed some work. Jeffrey Kim started finishing up the Task History screen and polishing the page.

Week 5: Refactoring and styling improvements
During this week the team came together to find and squash as many bugs as possible. There were many bugs caused by edge cases that were discovered and squashed during this process. Alexander Dunn made significant improvements to the codebase, implementing major refactors that improved readability and reduced bugs. Seth Damany continued work on

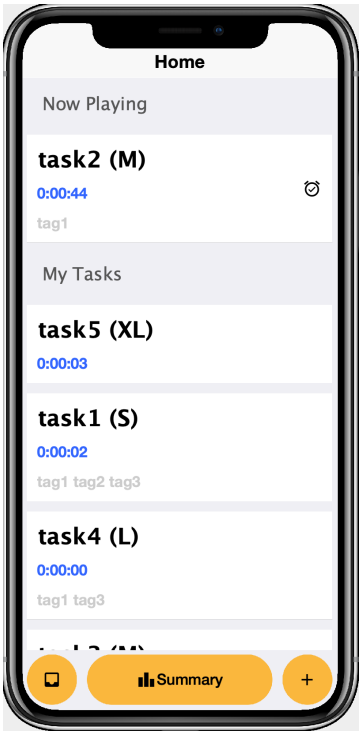
polishing the user interface and squashing bugs related to the user interface. Jeffrey Kim tested many possible edge cases and squashed several bugs that appeared on the archive screen. Whitney Devine and Seth Damany worked together to implement the line graph for the task details screen. Hao Cheng integrated his data storage solution into the project

---

User Guide

Home

The Home Screen displays all tasks important to the user. If a user wants to remove a task from the home screen but still have access to it for later use, they can simply archive the task.



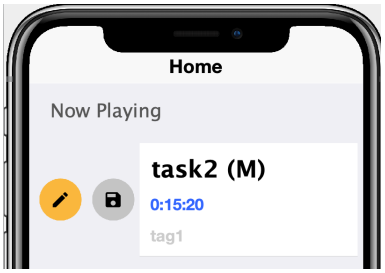
Each task UI component is a multi button that shows the task's name, size, total time elapsed, and list of tags if any. The currently active task is placed at the top with a timer icon on it to communicate that it is running. It is placed at the top so that it is quick and easy for the user to find the task and stop it or check their current time with the task. Each of these pieces of information -- name, size, and tags -- allow the user to quickly scan the screen to find their task of choice.

The buttons placed in the footer provide navigation to the archive, summary, and new task forms. We made the summary button the largest to emphasize its importance as it provides essential context to the relationships between the tasks. More on the Summary Screen later.

Functionality

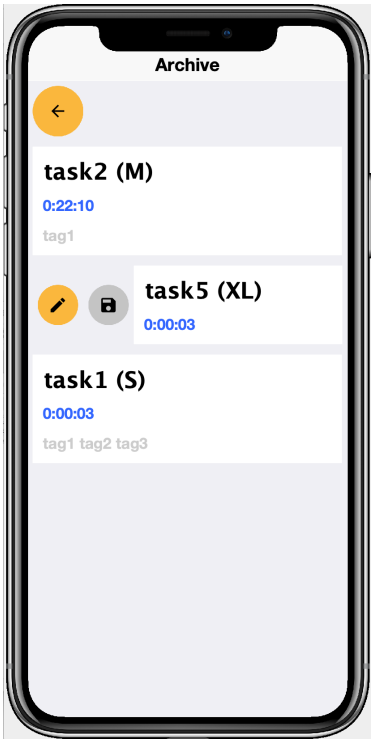
Tap on the active task to stop its timer. Inversely, **tap** on an inactive task to start its timer and stop the currently running task, if any.
This promotes quick and easy task switching and allows the user to focus on moving to the new task rather than fumbling with their time tracking app. (We recognize that we are a tool for the user to effectively use and not the main show)

Long press a task UI component to go to the Task Details Screen.



Swipe right on a task UI component for quick access buttons: edit task and archive task. Navigating to these forms without the quick access buttons usually takes two steps, so this added functionality improves the user's ease of access and lessens the time that it takes them to access the.

Archive



The Archive Screen displays all of the archived tasks. Old tasks and their histories are preserved and accessible here -- and, if needed, they can be unarchived and restored back to the Home Screen.

The task UI components are based on the same custom multibutton as those of the home screen; this allows for visual consistency and a nice clear message that these tasks are still around, just out of the way.

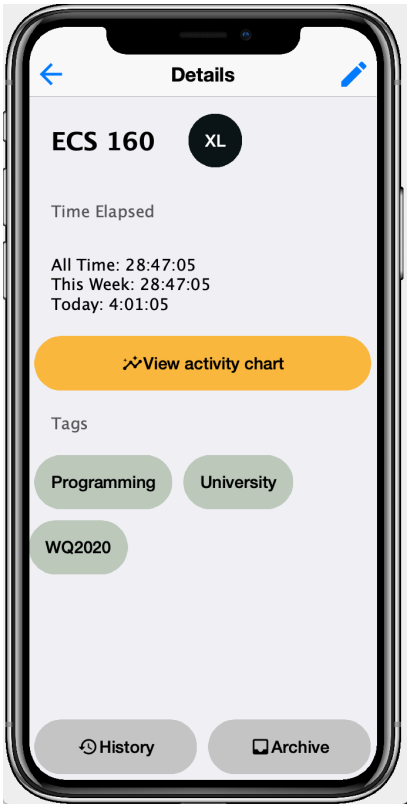
Functionality

Tap on an archived task to view its details. As this screen is intended for the preservation of history, the quicker action of short pressing should do what is most useful: viewing a task's details.

Long press on an archived task to unarchive it. We do not think it is likely the user will be frequently archiving and unarchiving tasks, therefore having that function in the slightly less convenient long press is acceptable and can prevent accidental unarchiving.

Swipe right on a task UI component for quick access buttons: edit task and *unarchive* task.

Task Details



The Task Details Screen shows all of a given task's details. This includes name, size, time elapsed for preset time spans (Day, Week, and All Time), tags, and description. If the task is active, it will also contain the Active icon to the right of the size label. Additionally, there is an option to archive the current task. This screen is the jumping off point to Edit Task Screen and the Task History Screen.

The View activity chart button opens a dialog with a line graph that displays the time spent on the task each day, by default from the start of the current work week (Monday) to the current day. Below the graph. the user can select any time window permitting that the start date is equal to or less than the end date.

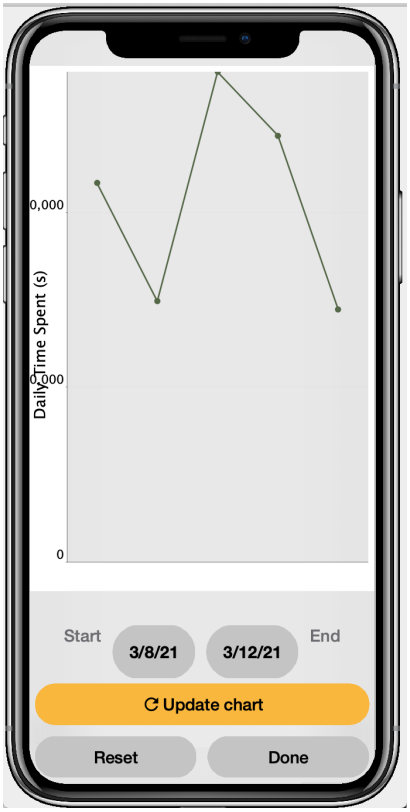
Functionality

Tap on the *Edit* button to navigate to the Edit Task Screen. This will allow the user to edit the Task's name, size, tags, and description.

Tap on the *History* button to navigate to the Task History Screen. This will allow the user to edit each task's start/stop Timespan. This is particularly useful if the user makes a mistake in starting or stopping a task.

Tap on the *Archive* button to archive the current task. This option exists on the Task Details Screen because we consider it apt if the user is reading the task, they may decide that they do not need this task readily available any more.

Tap on *View activity chart* to open the Activity Dialog.



Activity Dialog

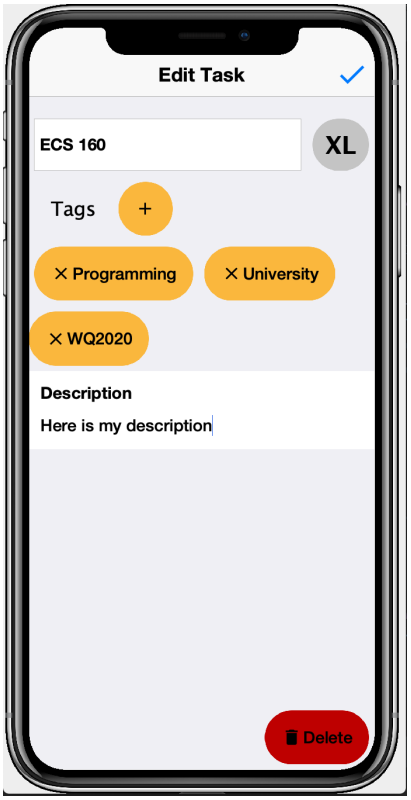
The Activity Dialog contains the chart as well as Start and End Date pickers.

Tap on Reset to make the Start Date the start of the current week and the End Date the current day.

Tap on *Update chart* to refresh it in accordance with the start and end date. If the dates provided are invalid, a warning will appear letting the user know.

Tap on *Done* to close the dialog.

Edit Task/New Task

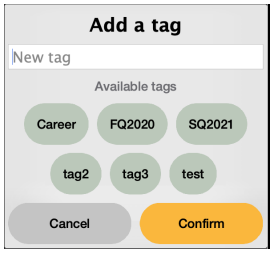


The Edit Screen is where the user will both create a new task or edit an existing task. The user has the option to edit the Task's name, size, tags, and description.

Functionality

Tap on the Title and/or Description text box to edit each one respectively.

Tap on the Size button to select a size S/M/L/XL. Default size is S.

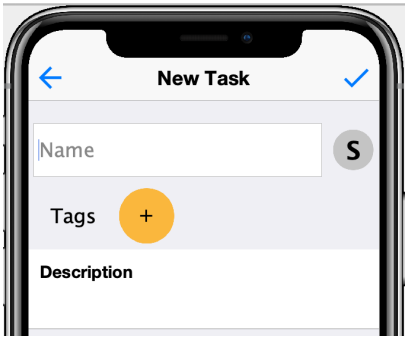


Tap on (+) button to create a new tag. This will open a dialog that allows one to either type in a new tag name or choose from the set of existing tags. *Cancel* to close the dialog. *Confirm* to add the new tag name.

Tap on the "X" on a tag to delete it. This will open a dialog asking for confirmation. *Confirm* to remove the tag from this task. Since creating/deleting tags is very straightforward, we decided to exclude the option to edit a tag name.

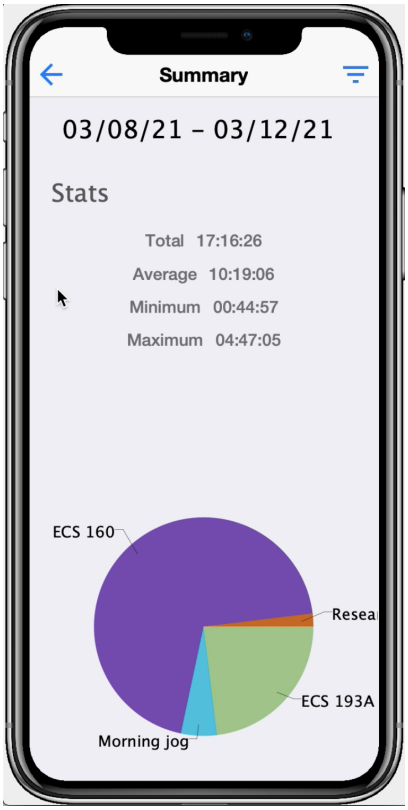
Tap on the checkmark to save your changes and return to the previous Screen.

Tap on Delete to permanently delete a task. This will prompt a dialog that confirms the deletion. **Tap** on Confirm to officially delete the task. This has far greater consequences than archiving a task which is why we buried the option deeper than any archive option, added the warning confirmation box, and made it a threatening shade of red.



To the right is the New Task screen, which bears the same basic functionality as the Edit Task screen. However, it does not include a delete button, and a back button is available if

Summary



The Summary screen shows important statistics for a set of inactive, unarchived Tasks that had activity during the selected time window.

The particular set of Tasks can be filtered by the user. It defaults to all tasks started in the past week, beginning on the most recent Monday.

The Stats row includes Total, Average, Minimum, and Maximum time spent on the specified set of Tasks.

The Graph row consists of a Pie Chart that randomizes colors for each Task within the specified set of Tasks. This visualizes and contextualizes the overall time spent on the current set of Tasks.

Below the graph (you will need to scroll due to issues with Codename One limiting ability to receive the graph) there is a table of all the tasks included in the current summary. The Tasks row consists of Task buttons that display a Task's name, size, and time elapsed.

Functionality

Tap the Filter button to open up the Filter Dialog.

Tap the Back button to go to the previous Screen.

Tap the Task button for the task's Detail page opens. While looking at a summary a user may notice some task in the list that sticks out and they may want to further investigate its details, task specific line graph, and history, so a direct link from the summary screen to details can be quite the convenience.

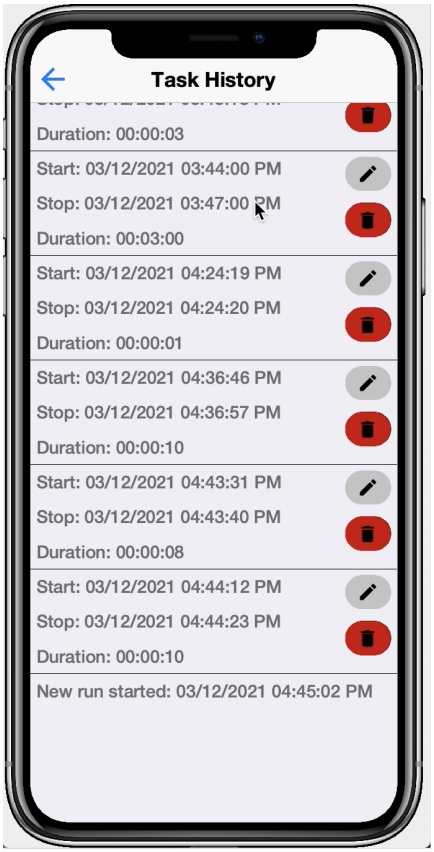
Filter Dialog

Tap on a filter button to select it. **Tap** again to deselect.

The Filter Dialog consists of Start and Stop Date Pickers, Size filters, and tag filters. The Date pickers default to the current work week. The Size and Tag filters default to unselected.

Only one Size filter can be selected a time, and if none are selected then all sizes will be included in the set. Multiple tags can be selected such that only tasks that satisfy all the selected tags will be displayed. Reset will return all the filter criteria to default.

Task History Screen



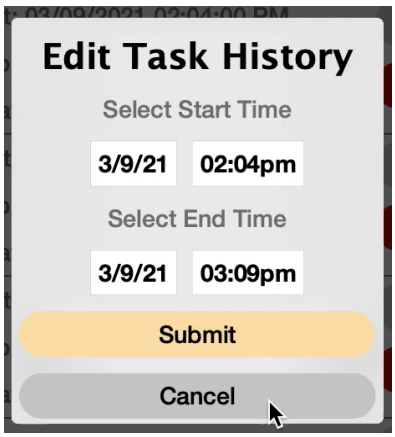
The Task History Screen displays each start/stop pair (TimeSpan) of a given task as well as the time elapsed within said TimeSpan. Each TimeSpan row has the option of editing or deleting. This is useful for when the user makes a mistake in starting or stopping a task and would like to adjust the Task's history. The user can quickly and easily fix that error in this Screen.

Functionality

Tap the edit button on a specific TimeSpan to bring up a dialog box that gives the option to select a new start and stop date time. Time can be adjusted to the minute. With this feature, you can fix errors such as forgetting to start or stop a task.

Tap the delete button for a specific TimeSpan to bring up a dialog box that will ask for your confirmation before deleting the task history. We believe that it's important for you to be sure of their actions before making the changes. Deleting allows you to remove accidental starts.

Tap on the Back button to go to the previous Screen (Task Details).

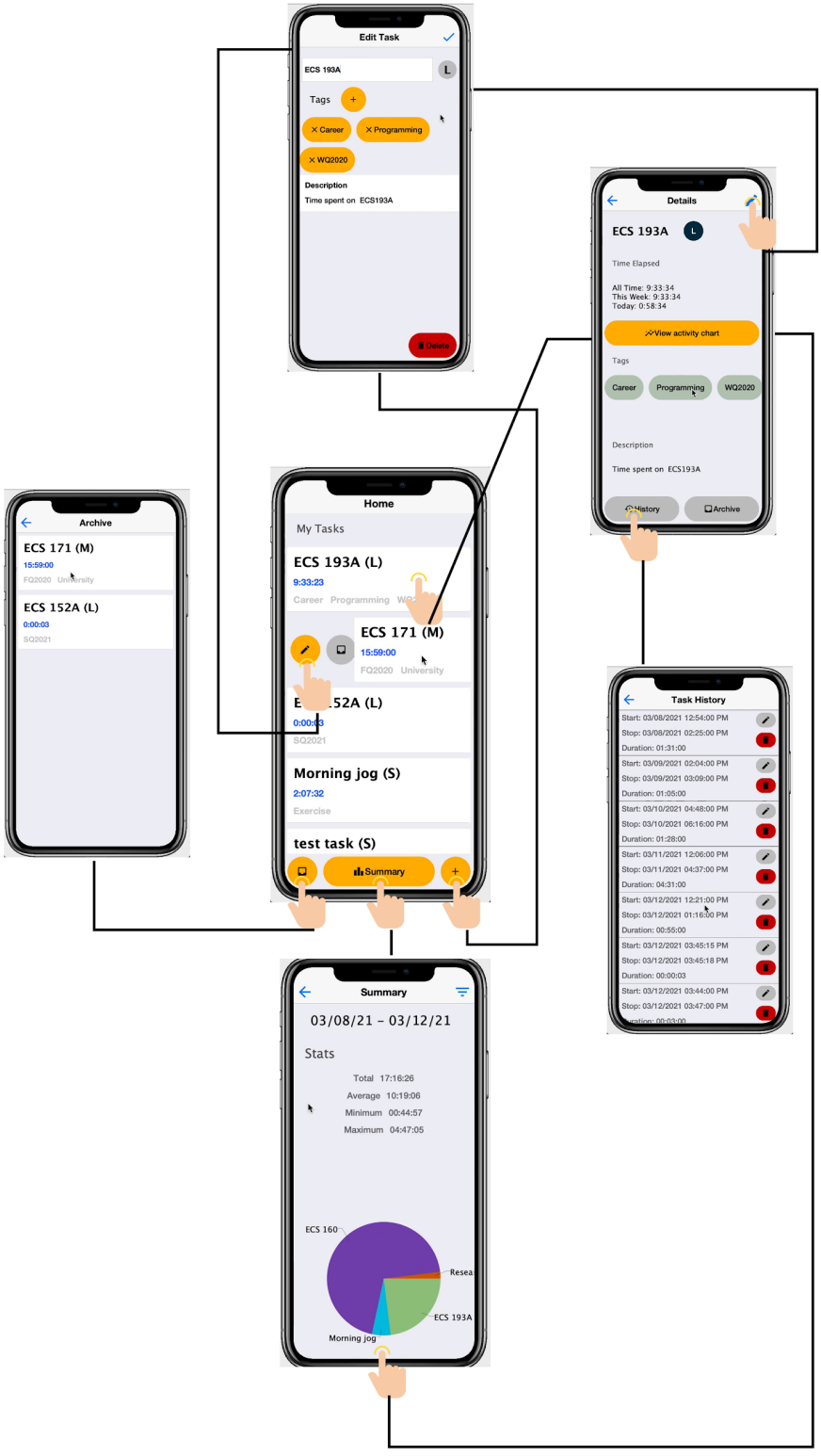


Implementation

The data in the task history list comes from custom made TimeSpan objects that hold start and stop time pairs in java's built in LocalDateTime format.

User Interface Flow Diagram (on the following page)

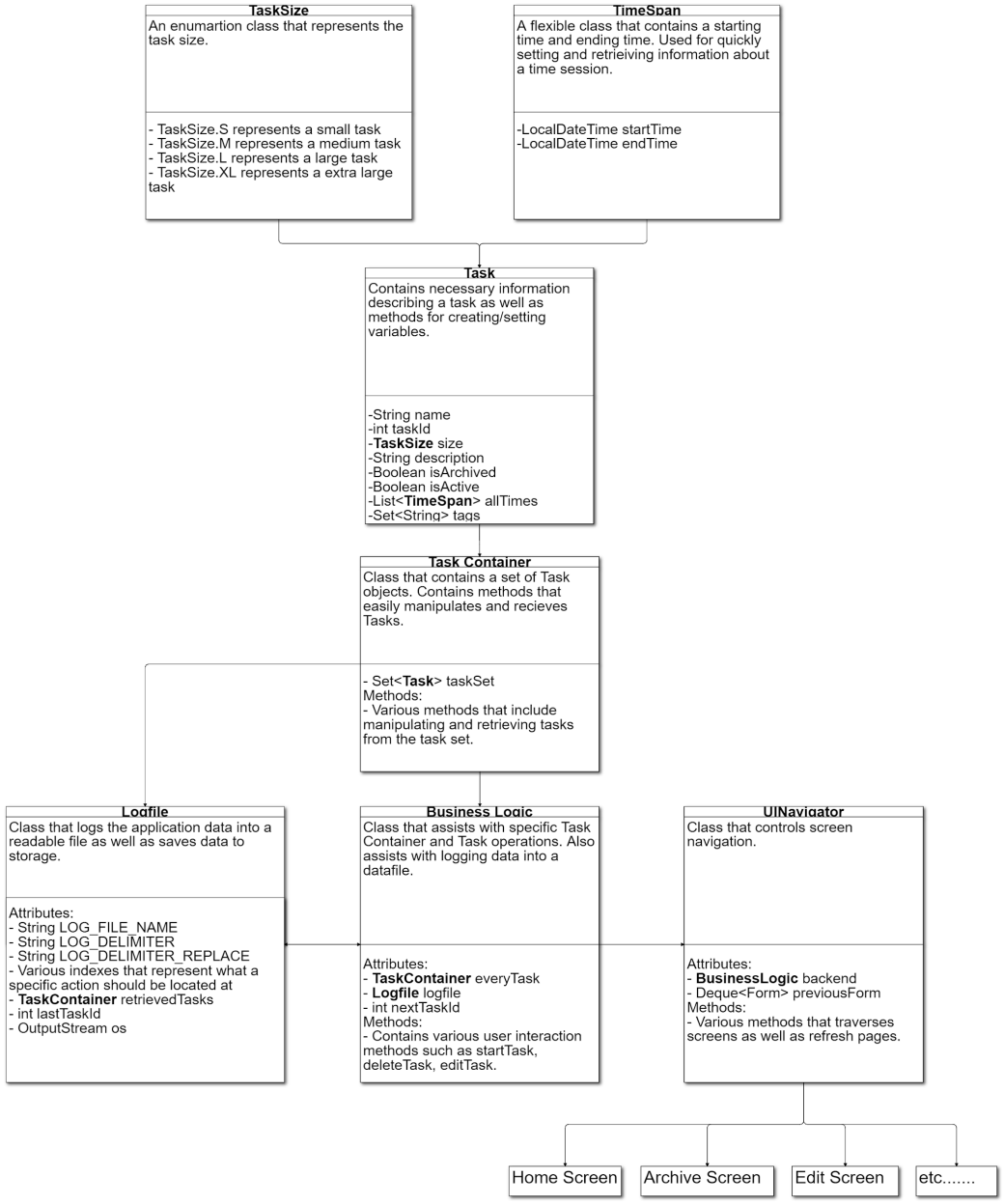
- This diagram shows the overall interactions the user can take to navigate between the different screens.
- The tap icon indicates where the user can tap. The line coming out of the tap icon indicates what screen that tap will take the user too.


Edit Task
ECS 193A
Tags
Career
Programming
WQ2020
Description
Time spent on ECS193A
Delete
Details
ECS 193A
Time Elapsed
All Time: 9:33:34
This Week: 9:33:34
Today: 0:58:34
View activity chart
Tags
Career
Programming
WQ2020
Description
Time spent on ECS193A
History
Archive
Archive
ECS 171 (M)
15:59:00
ECS 152A (L)
0:00:03
Home
My Tasks
ECS 193A (L)
9:33:23
ECS 171 (M)
15:59:00
FQ2020 University
Morning jog (S)
2:07:32
Exercise
test task (S)
Summary
Task History
Start: 03/08/2021 12:54:00 PM
Stop: 03/08/2021 02:25:00 PM
Duration: 01:31:00
Start: 03/09/2021 02:04:00 PM
Stop: 03/09/2021 03:09:00 PM
Duration: 01:05:00
Start: 03/10/2021 04:48:00 PM
Stop: 03/10/2021 06:16:00 PM
Duration: 01:28:00
Start: 03/11/2021 12:06:00 PM
Stop: 03/11/2021 04:37:00 PM
Duration: 04:31:00
Start: 03/12/2021 12:21:00 PM
Start: 03/12/2021 03:45:15 PM
Stop: 03/12/2021 03:45:18 PM
Duration: 00:00:03
Start: 03/12/2021 03:44:00 PM
Stop: 03/12/2021 03:47:00 PM
Summary
03/08/21 - 03/12/21
Stats
Total 17:16:26
Average 10:19:06
Minimum 00:44:57
Maximum 04:47:05
ECS 160
ECS 193A
Morning jog

Implementation

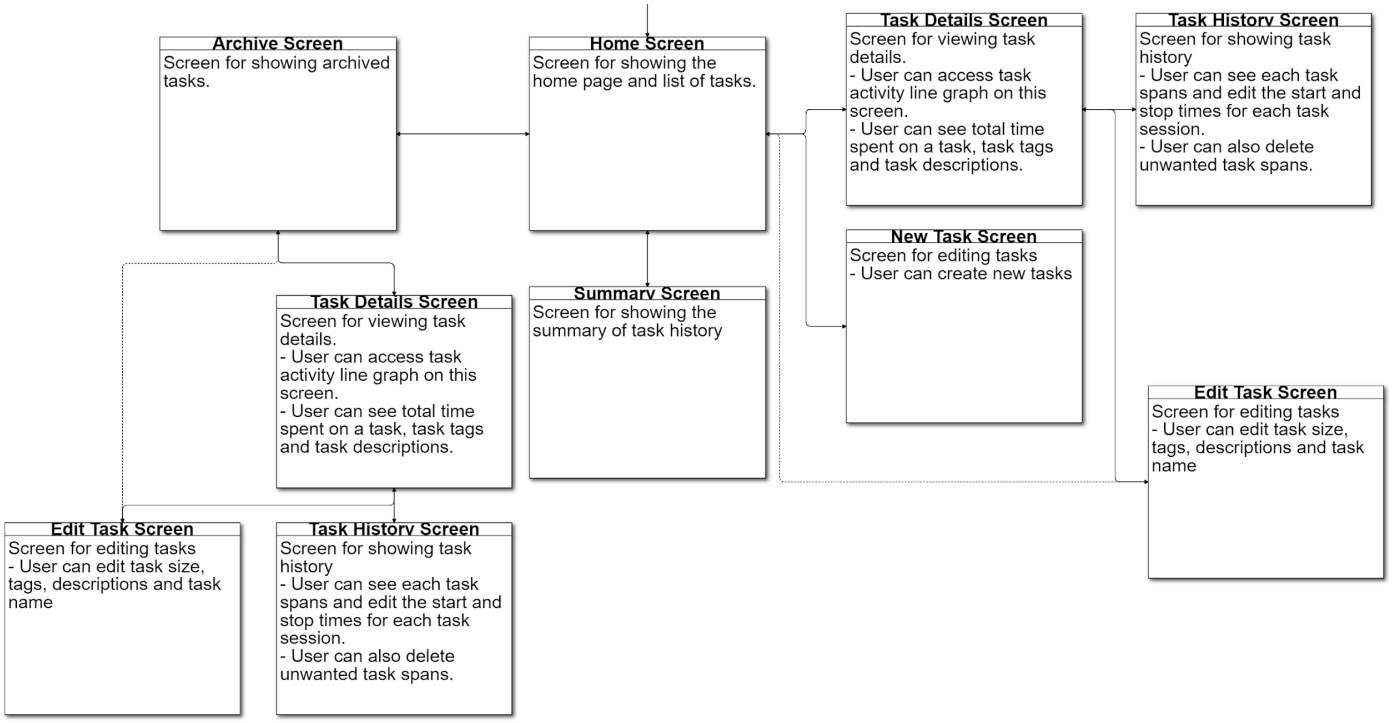
Class Diagram of Project

Here you can see the structure of the project. UINavigator manages which screen is showing and transitions between the screens. It is then linked to the backend by BuisnessLogic which serves as the connection between the frontend and the backend and is what frontend screens call and must go through if they wish to get tasks or modify data that is in the backend. The LogFile stores a record of what is being done to allow for persistent memory across app restarts. It connects to BuisnessLogic so that it can stay up to date with any changes that happen. All the data is then stores in TaskContainers, which in turn store Tasks which store some data locally but also use subclasses to store their size in a TaskSize enum and their various runs in a list of TimeSpan objects.



Navigation/Flow Diagram for UI

- This diagram shows how a user will be able to navigate to various screens. The arrow direction represents a potential location that the current screen can send the user to. An example of this is with the Home Screen. The Home Screen is the jumping off point for the rest of the application. It has arrows that are sent out to many different screens such as the Archive Screen, the Summary Screen, the Task Details Screen, the New Task Screen and the Edit Task Screen. This means that from the Home Screen you can quickly access all the screens listed above.
- Each box represents a screen. The screen name is at the top of the box. The screen description is below the screen name.
- Shortcuts are a big part of our program in order to enhance the user experience. In the diagram, unique user interactions such as short cuts are represented by dotted lines.



Front End

UINavigator

The class UINavigator manages navigation and passing relevant information to each of the 6 separate pages. maintains a stack of visited pages as well as a BusinessLogic object. It also contains methods for navigation, such as goBack(), goHome(), etc. Each of these methods are called by the appropriate buttons' Event Listeners in the separate Screen classes.

Public Variable

```
Deque<Form> previousForm // Stack of forms representing the user's navigation path
```

Note: This is used as a stack for tracking backwards navigation. Whenever you move to a new screen, the previous one is pushed to the top of this stack. The method **goBack()** (outlined below) pops from this stack to receive the previously accessed Form. It then calls **showBack()** to reveal the Form with a backwards animation to communicate the navigation.

Public Methods

- `refreshScreen()     // calls show() on current Form`
- `goBack()            // shows previous Form`
- `goHome()            // pops all Forms from stack and shows base Form`
- `goDelete(Task)      // deletes Task and returns Home`
- `goDetails(taskName) // shows details form of specified taskName`
- `goEdit(taskName)    // show Edit form for specified taskName`
- `goNew()             // show Edit form for a new Task`
- `goArchive()         // show Archive form`
- `goHistory(taskName) // show History form of specified taskName`
- `goSummary()         // show Summary form`

Next, we have six classes that extend form for each of the app's pages:

1. HomeScreen
2. ArchiveScreen
3. TaskDetailsScreen
4. TaskHistoryScreen
5. EditTaskScreen
6. SummaryScreen

Each screen has overridden show() and showBack() methods that first recreate the Screen (in effect, updating its contents) and then call Form's default show() and showBack() methods. This is done so that they always have up to date information whenever the screen is refreshed or gone back to. HomeScreen() and TaskDetailsScreen() each have overridden animate() methods to update the timer display periodically. Every screen apart from the HomeScreen contains a Toolbar for navigation. Each screen is composed of various containers in Flow, Box, Grid, and Border layouts. Since the screens share many components - such as task items, buttons, labels, dialogs, and date pickers - we abstracted them away to the class UIComponents. UIComponents has a set of custom classes for creating our styled components without having to excessively violate the DRY principle.

The Screen/form classes have several functions that are not-ideal in verbosity. We found that styling in particular came at the heavy cost of wordy and repetitive code. In an effort to remedy this, we created the following two classes to contain important components and constants.

UITheme

Our color palette, padding sizes, and icons are defined in UITheme. Separating these styling constants allows for more modularity, making it easier to modify and add to the User Interface. Placing our styling preferences in a designated class comes with the benefit of modularity and the freedom to adapt the application's visual interface.

- `Colors  // enhances the ability to test out and add new color palettes`
- `Padding  // defines the size of all UIComponent.ButtonObjects`
- `Icons  // defines the set of icons used in the application`

UIComponents

Contains custom buttons, labels/spanlabels, containers, and dialogs to be used by the Screen classes.

- `ButtonObject  // extends Button, assigns text, color, icon, and padding`
- `SizeButtonObject  // extends Button, similar to below but with eventListener()`
- `SizeLabelObject  // extends Label, assigns UITheme.color according to size`
- `TextObject  // extends Label, assigns text, color, size, and isBold`

- `TaskObject(Task, UINavigator)`
  - `extends Container`
  - `sets name, tags, and all event listeners of the Task`
    - `Handles short press and long press interaction`
  - `enables slider for access to quick navigation`
  - `Animates if task is active`
- `TaskHistoryObject // extends Container, sets the correct buttons for the Task History's page entries`
- `DatePickerObject // extends Picker, assigns color and padding`
- `StartEndPickers // extends Container, sets the layout for the start/end date picker entries`
- `showWarningDialog // extends Dialog, sets the text and padding to a Warning dialog that disposes() upon pressing the OK button`

Back End

Task

The Task class is an abstraction of a user's task. It contains all of the information necessary to create a class. This includes:

- A specialized task id that will never change and will be used to track the task across renames. Despite being so important, it is not actually something that is added when the object is created. This is because there are times when creating a new task where you do not know if the user will actually want to keep that task, therefore we make a sock puppet task that has no ID and only fills in the position of a task until the user decides they want to save the task, at that moment it is assigned a unique id.
- A task name for the user to know the task by. This is not constant as the user can change it whenever they please. Names do not need to be unique, but multiple tasks with the same name are confusing to the user however so outside checks prevent them from creating or renaming two tasks to have the same name.
- A task size based on Agile's use of T-Shirt sizes. There is always a task size set, with the default being S. There are four possible task sizes (S,M,L,XL) and they are contained within an enum called TaskSize.
- A simple string that will hold the description of the task. Rather than allow multiple descriptions, having it be one string that the user can come back to to change or add more on to, seems more intuitive from a user point of view.
- Booleans to easily check if the task is archived, or if it is currently active.
- A set of all the tags that the set has assigned to them, each stored as a simple string.
- And a list of all TimeSpan objects that the task uses (explain ahead)

Many of the classes functions are related to getting different info out of the object in order to provide rich data to the user. One of the methods that is used quite frequently is the `public Duration getTimeBetween(LocalDateTime start, LocalDateTime stop)` Method which iterates over all the TimeSpans and gets the total amount of time that the task was active but only within the given window of time specified by the start and stop parameters. This filtering by time allows us to give much more relevant information to the users as often a user would be more interested in their recent time spent on a task or the time spent over a certain period like the past week or something rather than just a total time for the task's entire lifetime/

TaskContainer

The TaskContainer class is an abstraction of a container that holds a collection of the user's tasks. It has several methods that manipulate and retrieve Task objects as well as get certain info like total time and time statistics over sets of tasks.

The task container is a very convenient class with its great filtering and finding capabilities. The `Task find(Predicate<Task> selector)` method allows the user to specify their own condition to find a certain task out of the group, whether that is it matching a certain name, being the active task, or having a certain ID. Other functions like `getActiveTask()` use the find method but with a friendlier and easier to identify name.

The `TaskContainer filter(Predicate<Task> selector)` method is another especially useful function as it allows us to provide a lambda that can be used to get a specific subset of the current TaskContainer, because it also returns a new TaskContainer, we can chain these filters on top of each other and allow the user to specify quite specific filters on the Summary Screen (for example only including tasks that are not archived, have a task size of M, were active between 3/10/2020 and 3/13/2021 and have all the tags: “tag1”, “tag2”, and “tag5”).

The class also implements `Iterable<Task>` so that one can easily iterate over all the tasks included in a set with a simple for-loop. This and the previously mentioned features have made the TaskContainer quite flexible so that it can be used both in the backend to hold large groups of all the tasks, and in the front end as smaller containers for specific subsets of tasks.

TimeSpan
TimeSpan is a class that represents a session for a task and is used by the Task class to keep track of time.

The TimeSpan class keeps track of the start time and end time. It has various easy to use methods for returning important information such as duration in either a string or a date format and calculating stuff such as for how long, if any, amount of time it was active within a certain time window.

Once a TimeSpan is created its start time is set and it is considered active. Once its stop function is then call it notes the time at which it was to stop. These however are not set in stone. To allow the user greater control they are given the power to edit the start and stop times from the Task History screen.

Business Logic
The business logic class is the intermediary between the backend and the frontend. It is where the main TaskContainer that contains every task of the entire project is kept, it is also where the data storage is linked in with the project and where it is told when and what to update. Different parts of the frontend will use various methods from the Task and TimeSpan classes to get different pieces of data, however when it comes to setting or changing the data those methods are all called through this class, that way we can keep the data storage up to date with any changes and so we can hide some of the logic involved in setting the data.

Data Storage

The data storage is handled by the LogFile class. This class allows for the task objects to persist between the app being started up and shut down. It accomplishes this by using Codename One’s Storage class and creating a log that line by line lists each operation that is performed to modify the Tasks and or their TimeSpans (similar to how it was done in project 1). When the app is then started this log is read in and the operations are all performed again, leading us back to the state that the app was at when it last closed. The person creating this chose this implementation as it was what they were most comfortable with (they had just finished another project where they had to do this after all).