Schedule Optimization Product Manual

Developers: Benjamin Prince, Maynic Ye, Maria Pakinggan, Samuel Chen, Martin Zhao
Github repo: https://github.com/ucsb-cs48-w20/5pm-schedule-optimization

Table of Contents

Chapter 1. The Schedule Optimization Project Development Manual

1.1. Introduction

Welcome to the Schedule Optimization Project Development Manual! This Manual provides information on how to use the schedule Optimization Application to set up a personal calendar and display the recommended daily route on UCSB campus for travel guidance for android users. Because much of the information in this manual is general, it contains many references to other sources where you can find more detail. For example, you can find detailed information on Git, and open sources in general in many places on the Internet.

The Schedule Optimization Application Manual does, however, provide guidance and visual examples on how to use the Application for the first time.

1.2. What This Manual Provides

The following list describes what you can get from this manual:

- Information that lets users get started using the Schedule Optimization Application.
- An understanding of common end-to-end development models and tasks.
- Information on how the Team comes up with the initial idea and delivers the project through integrated development.
- Many references to other sources of related information.

Chapter 2. Getting Started With the Schedule Optimization Application

2.1. Introducing the Schedule Optimization Application

The Schedule Optimization App is a team collaboration project focused on improving UCSB students' study life experience available on Android platform. The aim for this project is to be a more scoped down, but versatile version of Google Maps, which interacts with students' calendar. Students only have to input their schedule to get their directions for the day. This project will allow for new locations other than those on the schedule, so students can customize it better. Also, there will be shortcuts so that it will automatically display the shortest path from their current location to the next recorded class. In addition, all these "daily maps" can be stored and easily pulled up so students wouldn't need to re input it again.

Note

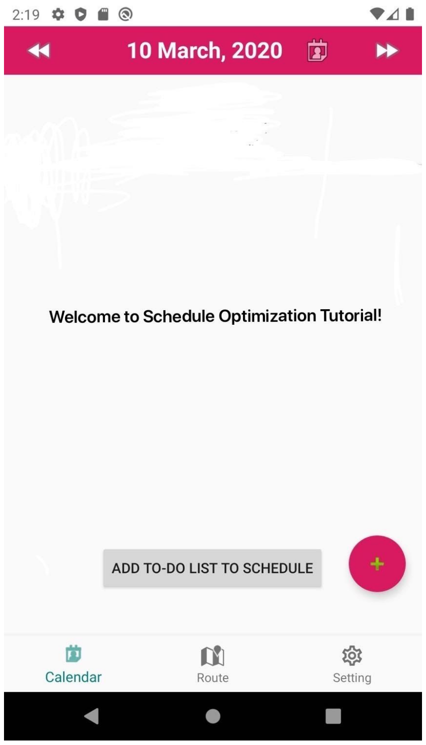
By users' consent, the Schedule Optimization app will have access to users' current location while the App is active in order to keep track of the route. Also, the daily routes are designed for the users who walk and bike on campus, so it is not recommended for users who drive to use the route as guidance.

2.2. Getting Started for new users with Visual tutorial

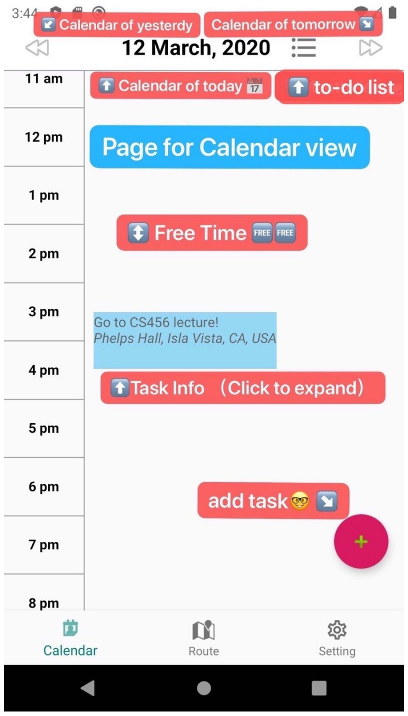
On the user's first usage of the application, the application will launch an onboarding tutorial that guides the user through the various functions and features. The tutorial is set to appear only on the first instance that the app is launched, but can be replayed through a button in the Settings tab.

The following are the images, in order, that are displayed to the viewer during the tutorial.

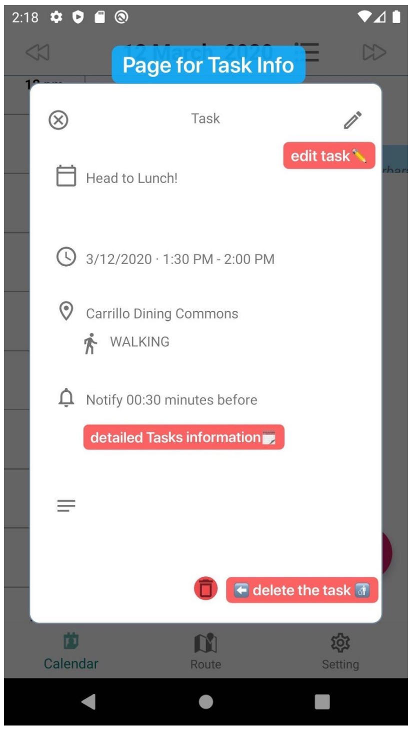
1.


2:19
10 March, 2020
Welcome to Schedule Optimization Tutorial!
ADD TO-DO LIST TO SCHEDULE
Calendar
Route
Setting

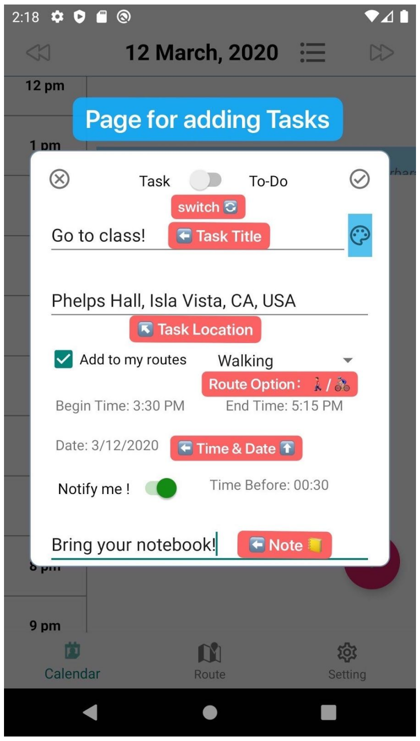
2.


Calendar of yesterdy
Calendar of tomorrow
12 March, 2020
Calendar of today
to-do list
Page for Calendar view
Free Time
Go to CS456 lecture!
Phelps Hall, Isla Vista, CA, USA
Task Info (Click to expand)
add task
11 am
12 pm
1 pm
2 pm
3 pm
4 pm
5 pm
6 pm
7 pm
8 pm
Calendar
Route
Setting

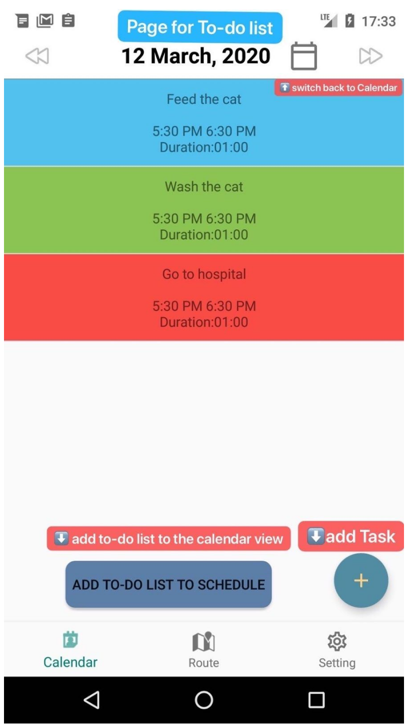
3.


2:18
Page for Task Info
Task
edit task
Head to Lunch!
3/12/2020 · 1:30 PM - 2:00 PM
Carrillo Dining Commons
WALKING
Notify 00:30 minutes before
detailed Tasks information
delete the task
Calendar
Route
Setting

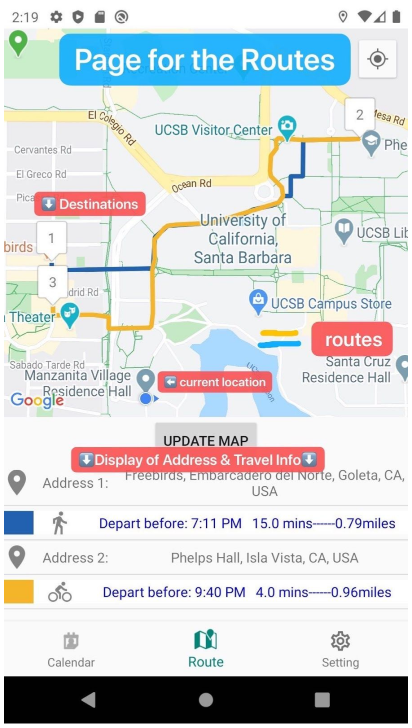
4.


2:18
12 March, 2020
12 pm
Page for adding Tasks
Task
To-Do
switch
Go to class!
Task Title
Phelps Hall, Isla Vista, CA, USA
Task Location
Add to my routes
Walking
Route Option:
Begin Time: 3:30 PM
End Time: 5:15 PM
Date: 3/12/2020
Time & Date
Notify me !
Time Before: 00:30
Bring your notebook!
Note
9 pm
Calendar
Route
Setting

5.

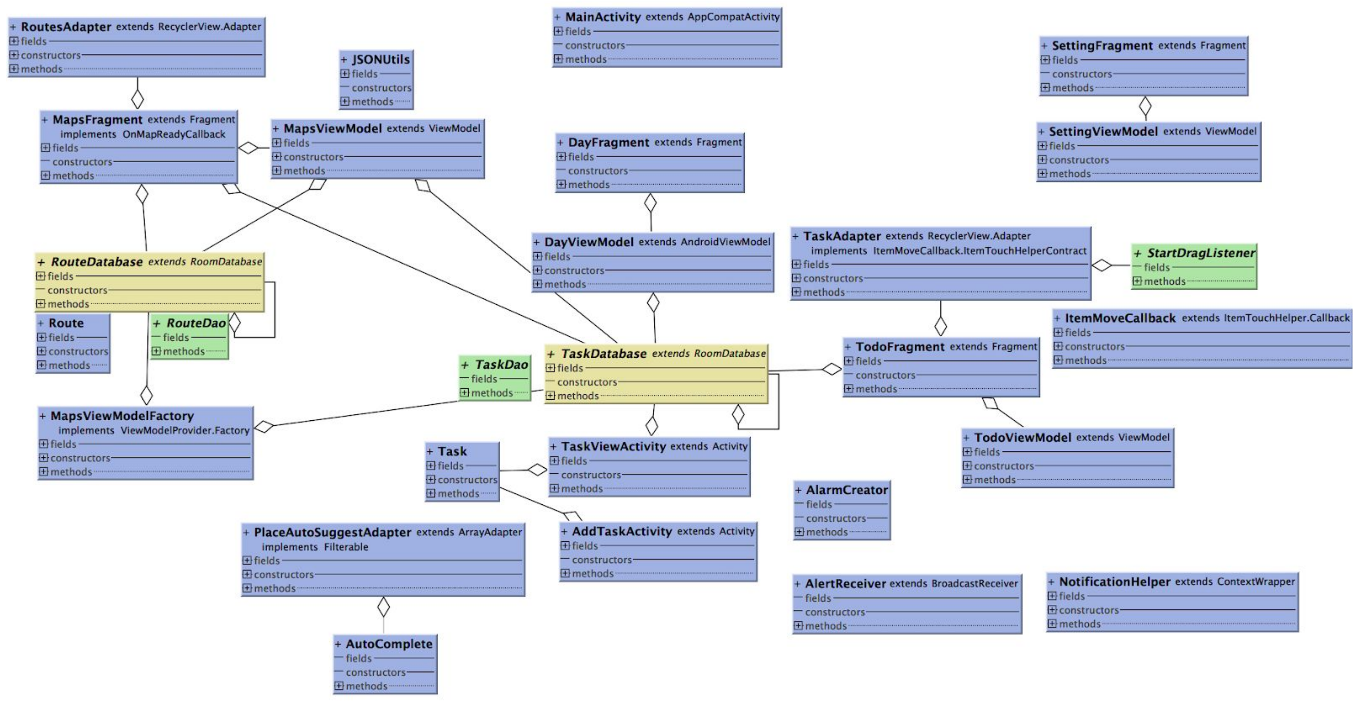

Page for To-do list
17:33
12 March, 2020
switch back to Calendar
Feed the cat
5:30 PM 6:30 PM
Duration:01:00
Wash the cat
5:30 PM 6:30 PM
Duration:01:00
Go to hospital
5:30 PM 6:30 PM
Duration:01:00
add to-do list to the calendar view
add Task
ADD TO-DO LIST TO SCHEDULE
Calendar
Route
Setting

6.


2:19
Page for the Routes
UCSB Visitor Center
Destinations
University of California, Santa Barbara
UCSB Campus Store
routes
current location
Manzanita Village Residence Hall
Santa Cruz Residence Hall
UPDATE MAP
Display of Address & Travel Info
Address 1: Freebirds, Embarcadero del Norte, Goleta, CA, USA
Depart before: 7:11 PM 15.0 mins------0.79miles
Address 2: Phelps Hall, Isla Vista, CA, USA
Depart before: 9:40 PM 4.0 mins------0.96miles
Calendar
Route
Setting

2.3. Understanding the Code base through UML

The figure below is the high level UML class diagram of our app. Adhering to the MVVM pattern, the Models are represented by the Databases, DAOs, and actual objects; the Views are represented by Fragments and helper classes, and the ViewModels are represented by the ViewModels attached to the Fragments. To see more detailed UML diagrams for each part of the project, please go to Ch.5.



+ RoutesAdapter extends RecyclerView.Adapter
+ JSONUtils
+ MainActivity extends AppCompatActivity
+ SettingFragment extends Fragment
+ MapsFragment extends Fragment implements OnMapReadyCallback
+ MapsViewModel extends ViewModel
+ DayFragment extends Fragment
+ SettingViewModel extends ViewModel
+ DayViewModel extends AndroidViewModel
+ TaskAdapter extends RecyclerView.Adapter implements ItemMoveCallback.ItemTouchHelperContract
+ StartDragListener
+ RouteDatabase extends RoomDatabase
+ ItemMoveCallback extends ItemTouchHelper.Callback
+ Route
+ RouteDao
+ TaskDao
+ TaskDatabase extends RoomDatabase
+ TodoFragment extends Fragment
+ MapsViewModelFactory implements ViewModelProvider.Factory
+ Task
+ TaskViewActivity extends Activity
+ TodoViewModel extends ViewModel
+ AlarmCreator
+ PlaceAutoSuggestAdapter extends ArrayAdapter implements Filterable
+ AddTaskActivity extends Activity
+ AlertReceiver extends BroadcastReceiver
+ NotificationHelper extends ContextWrapper
+ AutoComplete
fields
constructors
methods

Chapter 3. The Schedule Optimization Project Development Environment

This chapter helps you understand the Schedule Optimization Project as a team development project. In general, the team uses Agile software development as the guidance for the 10 weeks' project. Additionally, the Schedule Optimization project uses specific tools and constructs as part of its development environment. This chapter specifically addresses the team's philosophy, licensing, workflows and Git.

3.1. Team's Philosophy

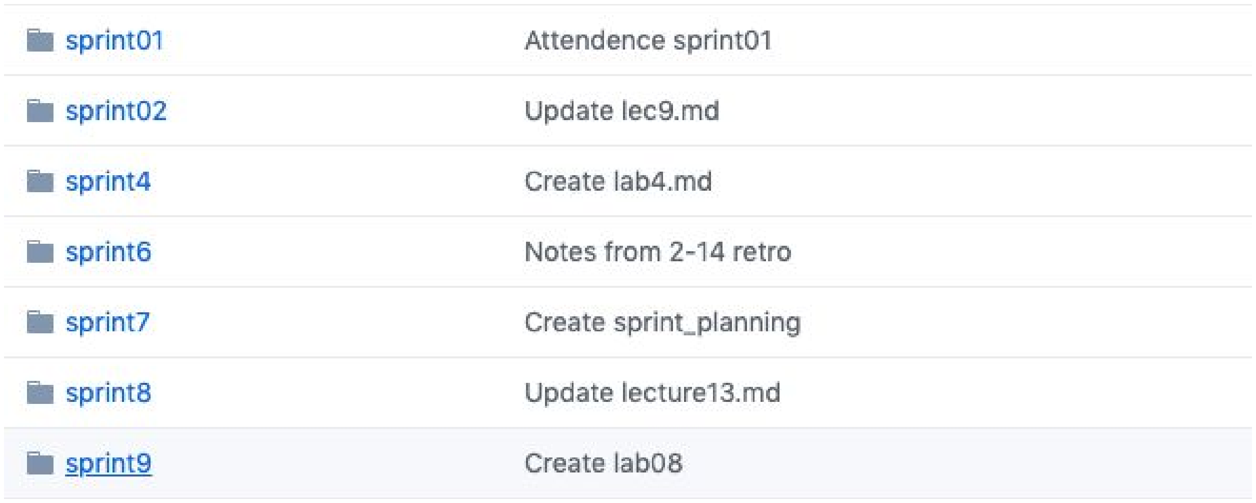
One takeaway from the Agile software development is that our team focuses on the people doing the work and how they work together. Solutions evolve through collaboration between self-organizing cross-functional teams utilizing the appropriate practices for their context. As a result, we really put emphasis on team communication. During the 10-weeks project, every class we hosted the "daily stand-up meetings" and we followed the sprint cycle for the development process. We hosted weekly sprint meetings and at the end of each sprint cycle, we had retrospective meetings. The goal is to improve and deliver the application through integrated development.
Our team believes that it's important to have mutually agreed Norms (listed below) throughout the development, which could ease and facilitate the process and incubate team chemistry.

1. Be on time to meetings.
2. Make sure to let people know if you need help with a task.
3. Never do someone else's task unless you notify them beforehand and they give consent.
4. If a team member(s) sends you a message/question on Slack, make sure to respond within one day.

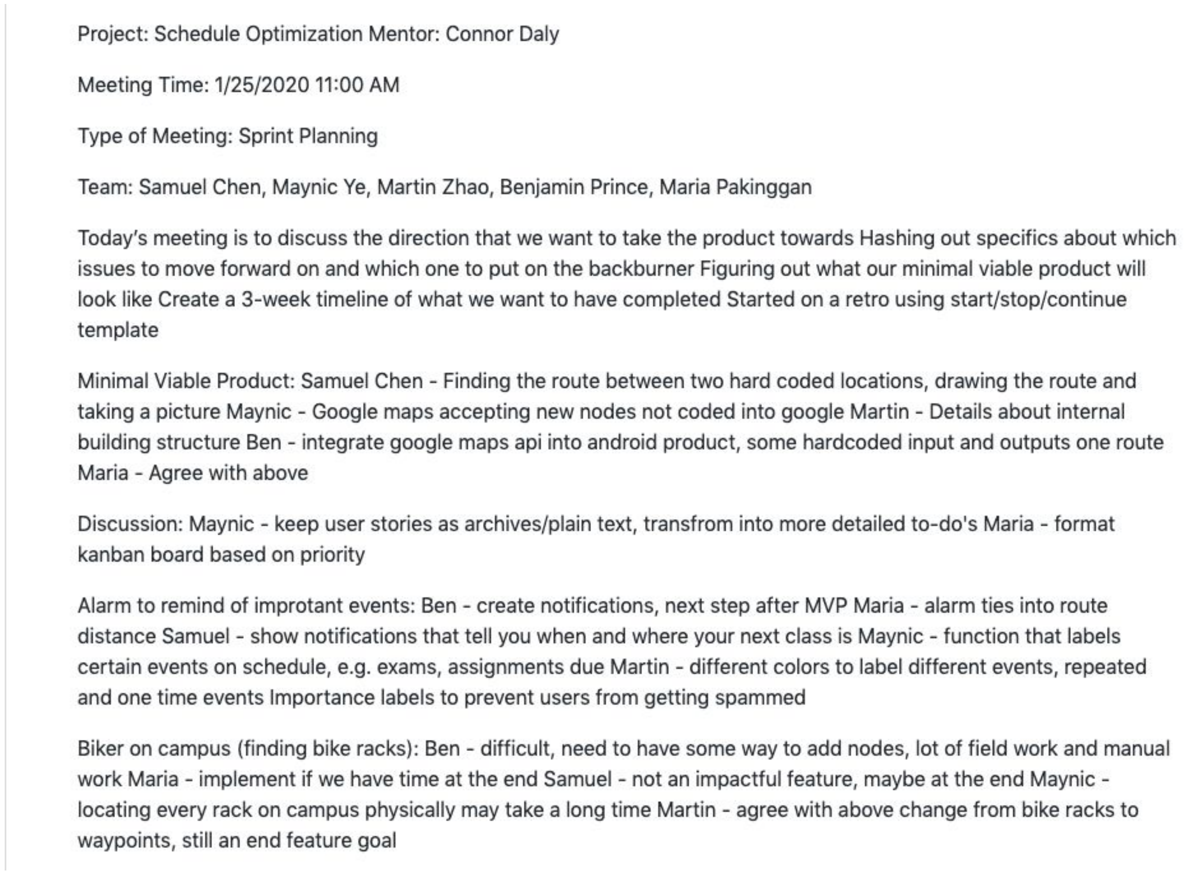
AgileAlliance.org has a good detailed historical description of the Agile Software Development Philosophy here: https://www.agilealliance.org/agile101/.

3.2. Developing the Schedule Optimization Project in a Team Environment

The team needed a way to focus on features to develop, streamline integration, and hold efficient meetings. As such, this led to utilizing User Stories, Github, and Scrum/Retrospective meetings, respectively. The team uses sprint meetings as a tool to integrate the development process, and here below are some screenshots from the sprint meetings.



sprint01	Attendence sprint01
sprint02	Update lec9.md
sprint4	Create lab4.md
sprint6	Notes from 2-14 retro
sprint7	Create sprint_planning
sprint8	Update lecture13.md
sprint9	Create lab08



Project: Schedule Optimization Mentor: Connor Daly

Meeting Time: 1/25/2020 11:00 AM

Type of Meeting: Sprint Planning

Team: Samuel Chen, Maynic Ye, Martin Zhao, Benjamin Prince, Maria Pakinggan

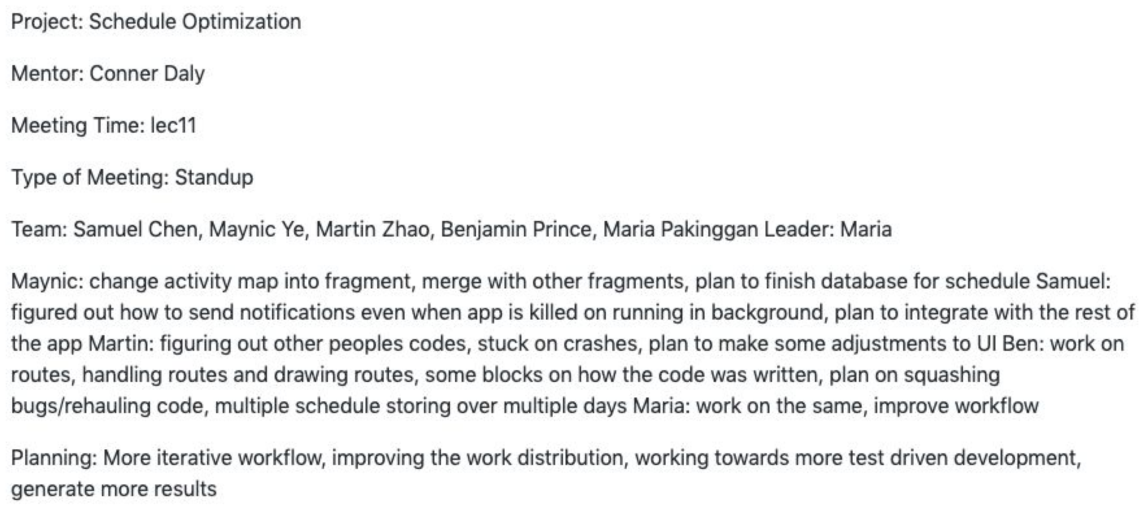
Today's meeting is to discuss the direction that we want to take the product towards Hashing out specifics about which issues to move forward on and which one to put on the backburner Figuring out what our minimal viable product will look like Create a 3-week timeline of what we want to have completed Started on a retro using start/stop/continue template

Minimal Viable Product: Samuel Chen - Finding the route between two hard coded locations, drawing the route and taking a picture Maynic - Google maps accepting new nodes not coded into google Martin - Details about internal building structure Ben - integrate google maps api into android product, some hardcoded input and outputs one route Maria - Agree with above

Discussion: Maynic - keep user stories as archives/plain text, transfrom into more detailed to-do's Maria - format kanban board based on priority

Alarm to remind of improtant events: Ben - create notifications, next step after MVP Maria - alarm ties into route distance Samuel - show notifications that tell you when and where your next class is Maynic - function that labels certain events on schedule, e.g. exams, assignments due Martin - different colors to label different events, repeated and one time events Importance labels to prevent users from getting spammed

Biker on campus (finding bike racks): Ben - difficult, need to have some way to add nodes, lot of field work and manual work Maria - implement if we have time at the end Samuel - not an impactful feature, maybe at the end Maynic - locating every rack on campus physically may take a long time Martin - agree with above change from bike racks to waypoints, still an end feature goal



Project: Schedule Optimization

Mentor: Conner Daly

Meeting Time: lec11

Type of Meeting: Standup

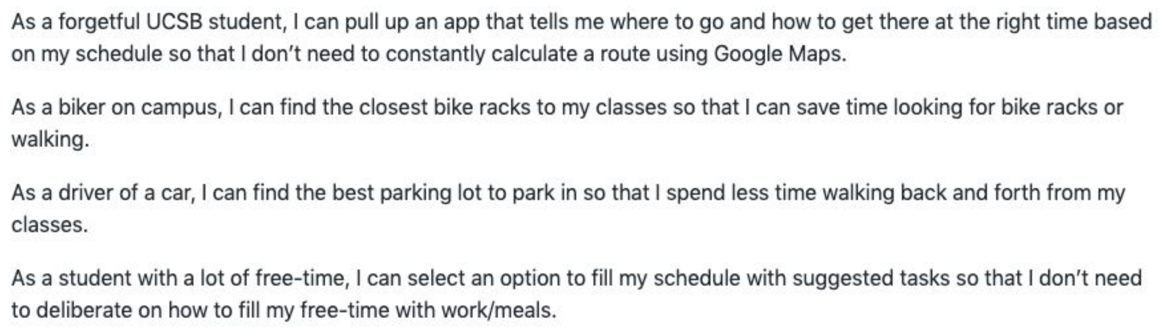
Team: Samuel Chen, Maynic Ye, Martin Zhao, Benjamin Prince, Maria Pakinggan Leader: Maria

Maynic: change activity map into fragment, merge with other fragments, plan to finish database for schedule Samuel: figured out how to send notifications even when app is killed on running in background, plan to integrate with the rest of the app Martin: figuring out other peoples codes, stuck on crashes, plan to make some adjustments to UI Ben: work on routes, handling routes and drawing routes, some blocks on how the code was written, plan on squashing bugs/rehauling code, multiple schedule storing over multiple days Maria: work on the same, improve workflow

Planning: More iterative workflow, improving the work distribution, working towards more test driven development, generate more results

3.2.a. User Story & Kanban Board

User stories are really crucial in the beginning stage of development, the format of “As a…, I wish to have the feature of… so that I could …” is really helpful for us to come up with different thoughts and features regarding the product. By taking perspective on the user end, it makes the team have a better understanding of the users’ needs. And below in the screenshots are some of our proposed user stories.

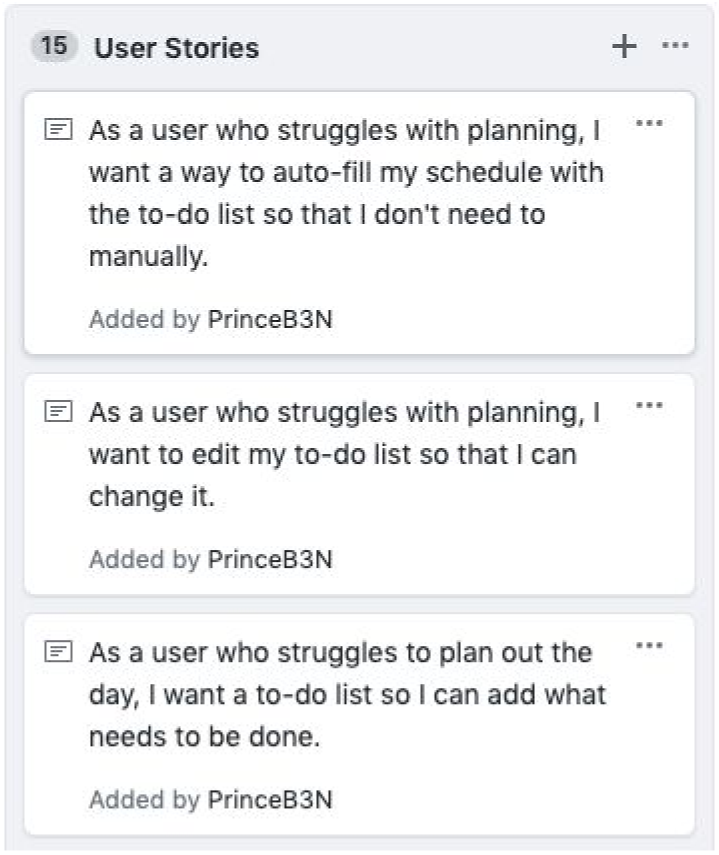


As a forgetful UCSB student, I can pull up an app that tells me where to go and how to get there at the right time based on my schedule so that I don’t need to constantly calculate a route using Google Maps.

As a biker on campus, I can find the closest bike racks to my classes so that I can save time looking for bike racks or walking.

As a driver of a car, I can find the best parking lot to park in so that I spend less time walking back and forth from my classes.

As a student with a lot of free-time, I can select an option to fill my schedule with suggested tasks so that I don’t need to deliberate on how to fill my free-time with work/meals.


15 User Stories

As a user who struggles with planning, I want a way to auto-fill my schedule with the to-do list so that I don't need to manually.

Added by PrinceB3N

As a user who struggles with planning, I want to edit my to-do list so that I can change it.

Added by PrinceB3N

As a user who struggles to plan out the day, I want a to-do list so I can add what needs to be done.

Added by PrinceB3N

Also, the team uses the Kanban board feature from Github to keep track of the development process, and assign different developers with various tasks.

3.2.b. From MVP towards Final Product

The initial design for our MVP was very simple. It encompasses the basic problem that we had decided to overcome. Our MVP was a simple application that would return a route on a map given an input of locations.

Moving forward from the simple MVP, we were able to have more discussions about the features we wanted to add to our product. Many of the features that we came up with did not eventually make it into the final product, but from those features we were able to come up with new ones that did make it in.

From the original route optimization app, we moved in the direction of a more comprehensive scheduling app. We realized many of the limitations that were caused due to the limited UI in the MVP, so we created a Calendar view to compensate. With the Calendar set up, it naturally led to some of the other features, such as assigning and removing tasks, creating todo items and setting alarms for tasks. There were of course many other ideas that were not implemented due to time constraints or suitability issues.

3.3. Licensing

The Schedule Optimization project is licensed under the terms of the MIT License. As such, licensed works, modifications, and larger works may be distributed under different terms and without source code.

3.4. Workflows

This section provides some overview on workflows of how the team uses Git. In particular, the information covers basic practices that describe roles and actions in a collaborative development environment.

The Schedule Optimization project files are maintained using Git in a “master” branch whose Git history tracks every change and whose structure provides branches for all diverging functionality. For the project, every team member is responsible for the “master” branch of the Git repository and every team member has a personal branch for the particular features they are working on. The “master” branch is the “upstream” repository where the final builds of the project occur.

There are some practices or methods that help our team development run smoothly. The following list describes some of our practices. For more information about Git Workflows, see workflow topics in the Git Community Book [https://book.git-scm.com].

- Make Small Changes: it is best to keep the changes you commit small as compared to bundling many disparate changes into a single commit. This practice not only keeps things manageable but also allows other members to more easily include or refuse changes.
- Merge Changes: The *git merge* command allows us to take the changes from one branch and fold them into another branch. This process is especially helpful for the team that members might be working on different parts of the same feature.
- Manage Branches: Branches are easy and helpful to use, we use a system where branches indicate varying levels of code readiness. For example, we have a “Name” branch to develop in, a “test” branch where the code or change is tested, a “main” branch where changes are committed into.

- Use Push and Pull: The push-pull workflow is based on the concept of developers “pushing” local commits to a remote repository (on Github). This workflow is also based on developers “pulling” known states of the project down into their local development repositories, ensuring that developers can have access to the most up-to-date version of the project.

3.5. Testing

Majority of the application is UI based, so it was difficult to automate the testing for a lot of the code. The only unit tests written were those testing the methods of the JSONUtils class, which reformatted the data from the JSON files we got from the Directions API. These were simple to test because we were only checking if the methods returned the expected outputs from parsing the JSON files. In terms of larger integration testing, the functionality of the application was manually tested. Since so much of the code depended on each other, it became difficult to automate testing for the entire application. In terms of instrument tests, some tests were written for the behavior of the application’s navigation bar, to see if the application would correctly switch between the different fragments.

Chapter 4. Common Development Models and Tasks

This section provides insights on different choices the team takes to come up with the final product. In general, the team makes decisions upon the concerns of time, economy and the usability and stability for the product. Besides, the team decides to incorporate outside resources (such as Dao Database and Google map API) to improve the application and there will be detailed description in sections below.

4.1. System Development Choice

Since the aim of the application is to be convenient and easy for users (UCSB students) to check their schedules during the day, the team decides to make the application available on a mobile platform. The team decided to choose Android over the IOS system due to many members using the Windows OS and familiarity with Android devices. Thus, team members use Android Studio as the software development tool (the minimum SDK of our application is API 14: Android 4.0, aka IceCreamSandWich). In addition, since developers are more familiar with Java over Kotlin, the team chooses the previous as the software development language.

4.2. Room Database & Data Structure

Since the application has a significant need of storing data locally, our team tried different ways, like json files and txt files. Due to the reliability and efficiency, we finally decided to use a local database. We choose the official local database for Android, Room database. Room database is an SQLite based database library provided by Google. It provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite. As for data structure, we went over three different stages. At the beginning, we use the concept of “schedule”. Schedule is for each day, and contains all the routes and tasks for that day. Later, we realized it is not a good practice to use a very large thing as the unit type. So, we divided tasks and routes into different data types and stored them separately. In the later stage of development, we had the need of storing Todos, which can be seen as a subclass of Task. After analysing the storage consuming, memory consuming, and the difficulty of implementing, we decide to not make a data type with another database, but make Task a broader one (add the attributes that Todo needs, and a “type” attribute).

4.3. Google Maps/Directions/Places API

Google Maps is a powerful tool that is ubiquitous as an app on every smartphone, but also as an integral application to any business or service that requires route or location information. It is already well-established and has a variety of feature suites that our app easily leverages. All our route and location data is parsed from RESTful calls to Maps, Directions, and Places API to drive the maps, routes, and autofill location features of our app. Although these APIs simplified the coding, the team had some difficulty with how to parse and store the data from the RESTful calls and how to manage drawing routes and markers. The solution was to parse the output using the Simple JSON library into an object, store the object using a Room Database, and then utilizing the encoded polylines, which are an encoding of the routes, to draw the routes and markers on the map.

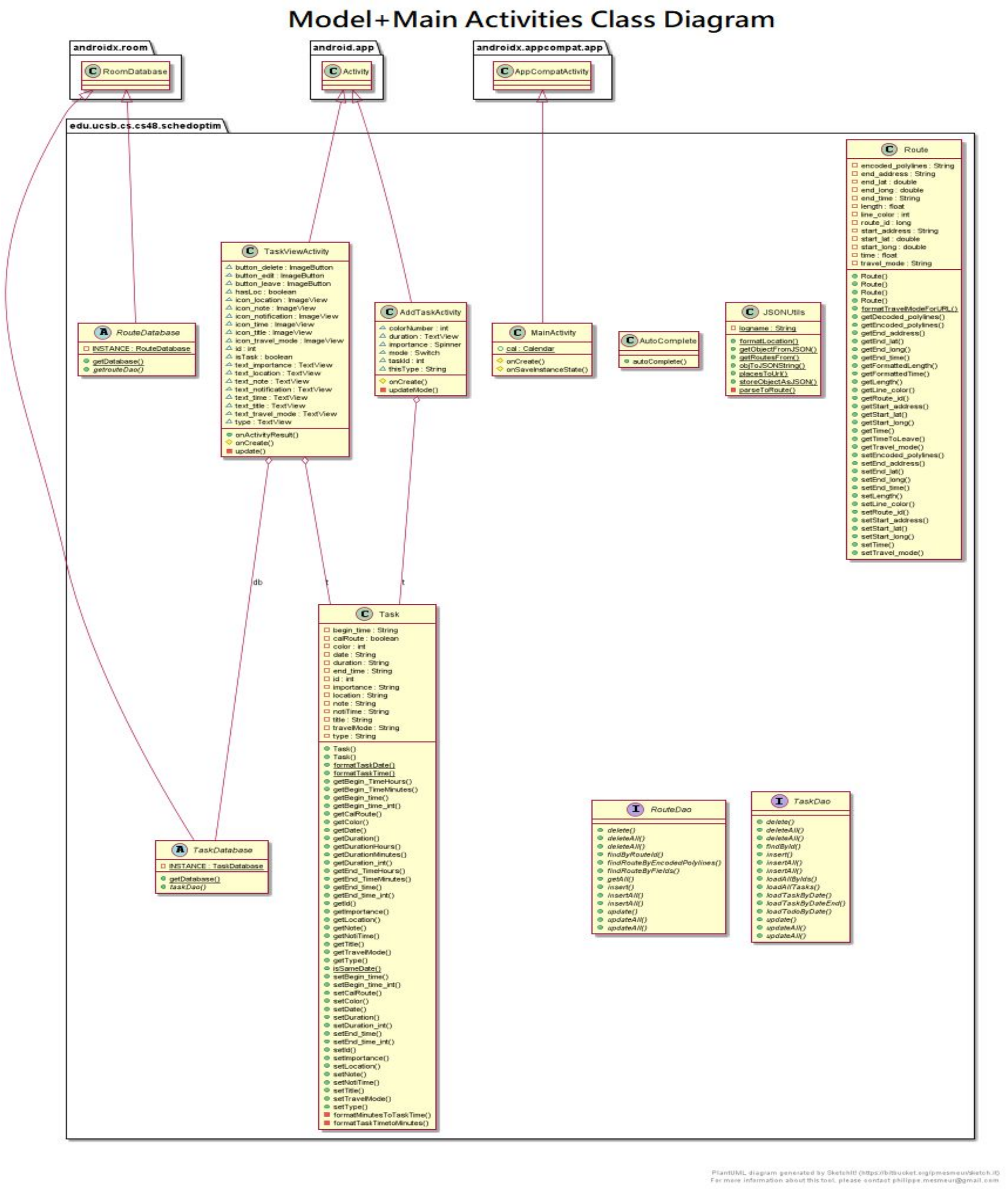
4.4. Summary of Design Decision

As such, the main reason to choose an Android mobile environment is to complement the purpose of the app: flexible, and can be adjustable on-the-go. The Google APIs also helped with setting up the maps, addresses, and routes so that the team could focus on developing novel features and a beautiful UI that utilized the power of those libraries. Also, choosing the Room Database allows for the user to use routes that were already pre-calculated so that the app can still work, which again complements the on-the-go theme for our app. This demonstrates that each design choice has a rationale behind it that supports the app’s functionality rather than hindering it.

Chapter 5. In-depth UML Diagrams

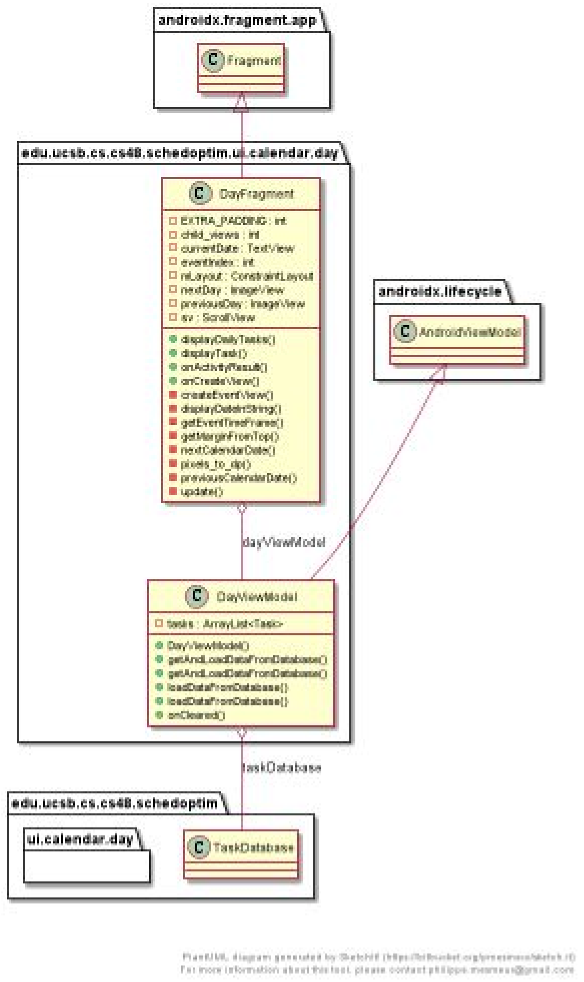
This section covers detailed diagrams of the major parts of the project. These UML diagrams are not meant to be exhaustive, but rather expose enough detail to enable a clear understanding of the project while maintaining readability for the reader.

5.1. Models and Main Activities

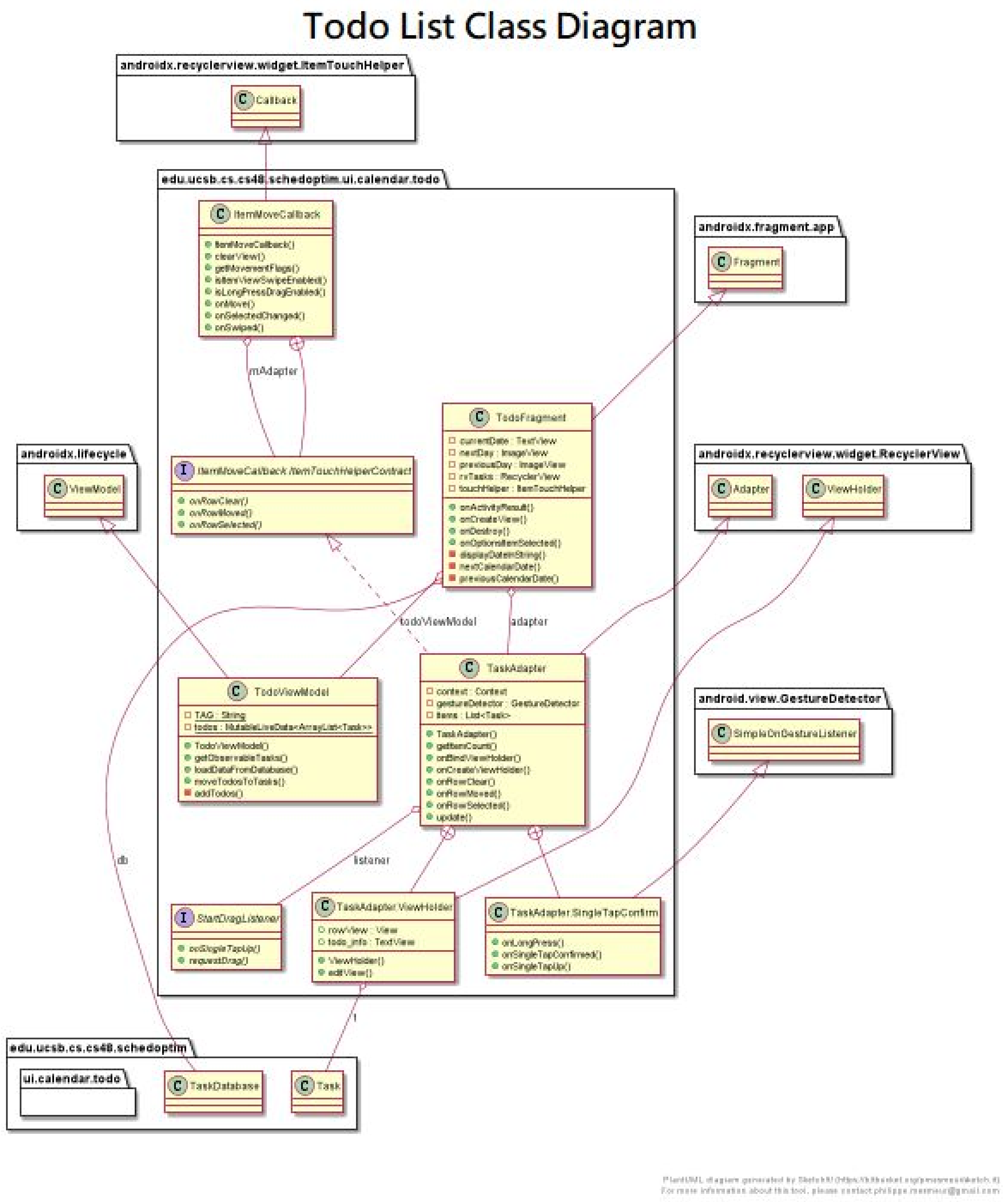


5.2. Task List

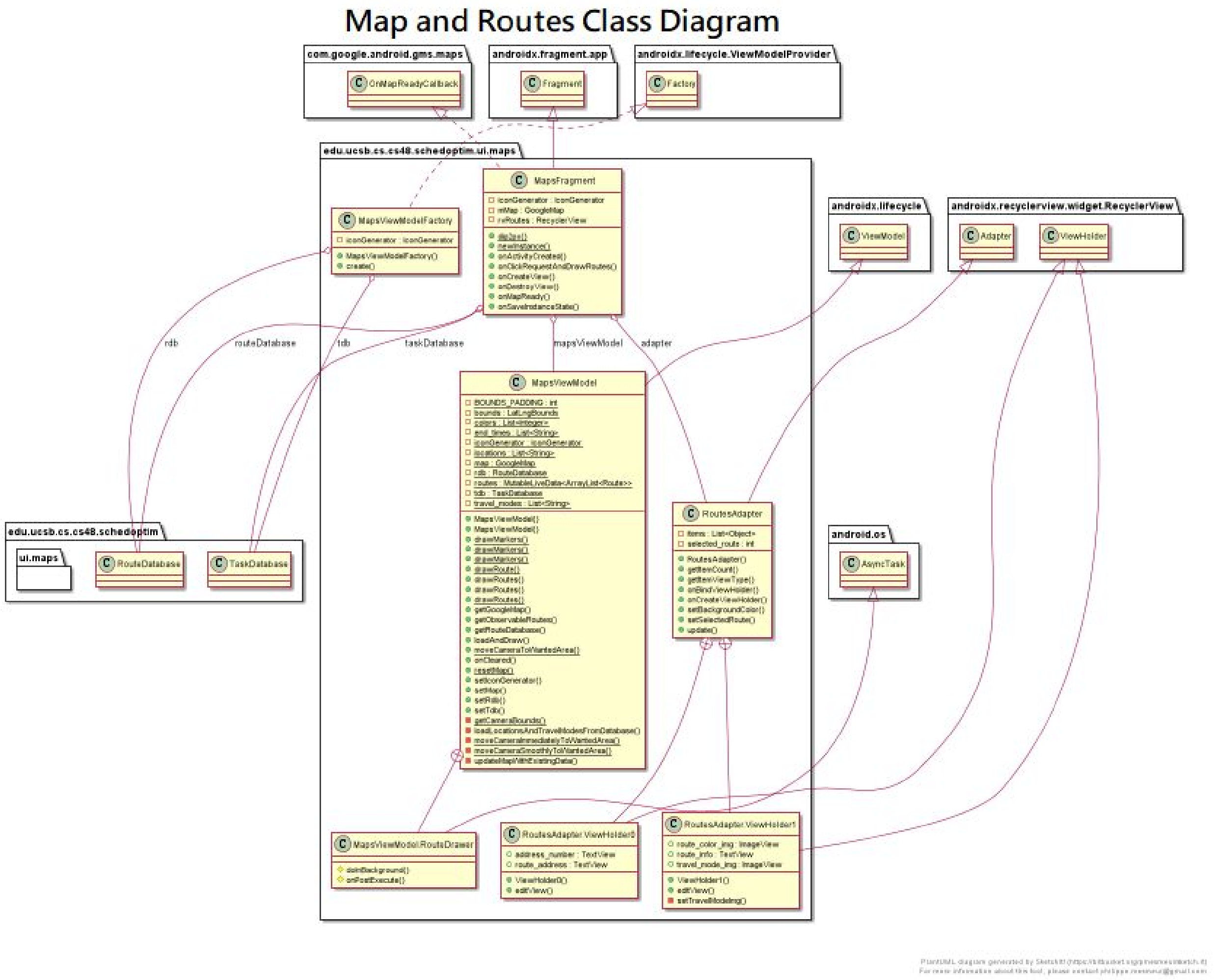
Task List Class Diagram



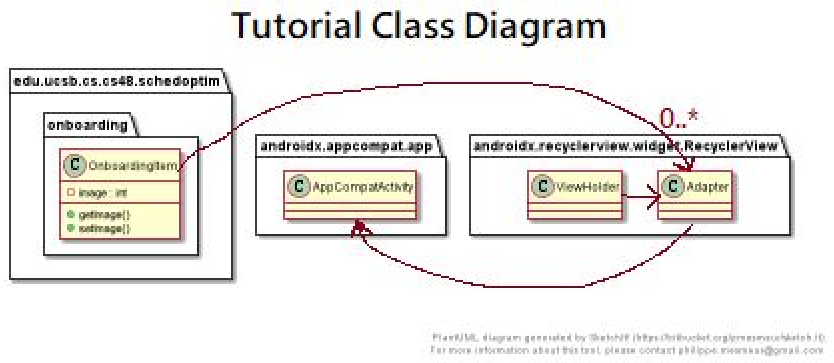
5.3. TodoList


Todo List Class Diagram
androidx.recyclerview.widget.ItemTouchHelper
Callback
edu.ucsb.cs.cs48.schedoptim.ui.calendar.todo
ItemMoveCallback
ItemMoveCallback()
clearView()
getMovementFlags()
isItemViewSwipeEnabled()
isLongPressDragEnabled()
onMove()
onSelectedChanged()
onSwiped()
mAdapter
ItemMoveCallback.ItemTouchHelperContract
onRowClear()
onRowMoved()
onRowSelected()
androidx.fragment.app
Fragment
TodoFragment
currentDate : TextView
nextDay : ImageView
previousDay : ImageView
rvTasks : RecyclerView
touchHelper : ItemTouchHelper
onActivityResult()
onCreateView()
onDestroy()
onOptionsItemSelected()
displayDateInString()
nextCalendarDate()
previousCalendarDate()
androidx.lifecycle
ViewModel
androidx.recyclerview.widget.RecyclerView
Adapter
ViewHolder
todoViewModel
adapter
TaskAdapter
context : Context
gestureDetector : GestureDetector
items : List<Task>
TaskAdapter()
getItemCount()
onBindViewHolder()
onCreateViewHolder()
onRowClear()
onRowMoved()
onRowSelected()
update()
TodoViewModel
TAG : String
todos : MutableLiveData<ArrayList<Task>>
TodoViewModel()
getObservableTasks()
loadDataFromDatabase()
moveTodosToTasks()
addTodos()
android.view.GestureDetector
SimpleOnGestureListener
db
listener
StartDragListener
onSingleTapUp()
requestDrag()
TaskAdapter.ViewHolder
rowView : View
todo_info : TextView
ViewHolder()
editView()
TaskAdapter.SingleTapConfirm
onLongPress()
onSingleTapConfirmed()
onSingleTapUp()
1
edu.ucsb.cs.cs48.schedoptim
ui.calendar.todo
TaskDatabase
Task

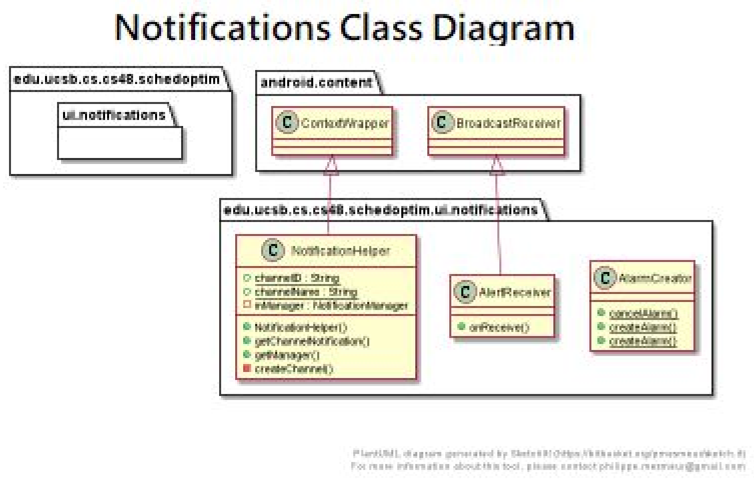
5.4. Map and Routes



5.5. Tutorial

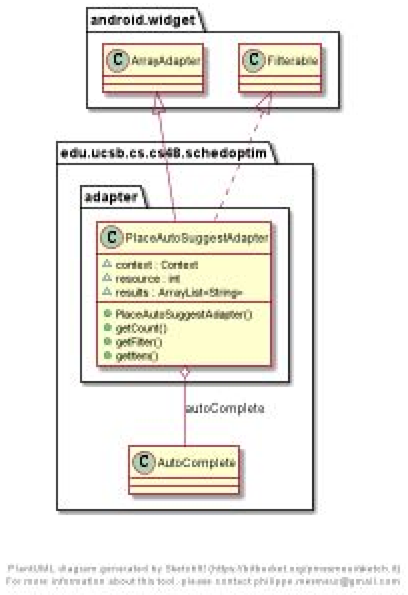

Tutorial Class Diagram
edu.ucsb.cs.cs48.schedoptim
onboarding
OnboardingItem
image : int
getImage()
setImage()
androidx.appcompat.app
AppCompatActivity
androidx.recyclerview.widget.RecyclerView
ViewHolder
Adapter
0..*
PlantUML diagram generated by SketchIt! (https://bitbucket.org/pmesmeur/sketch.it)
For more information about this tool, please contact philippe.mesmeur@gmail.com

5.6. Notifications


Notifications Class Diagram
edu.ucsb.cs.cs48.schedoptim
ui.notifications
android.content
ContextWrapper
BroadcastReceiver
edu.ucsb.cs.cs48.schedoptim.ui.notifications
NotificationHelper
channelID : String
channelName : String
mManager : NotificationManager
NotificationHelper()
getChannelNotification()
getManager()
createChannel()
AlertReceiver
onReceive()
AlarmCreator
cancelAlarm()
createAlarm()
createAlarm()
PlantUML diagram generated by SketchIt! (https://bitbucket.org/pmesmeur/sketch.it)
For more information about this tool, please contact philippe.mesmeur@gmail.com

5.7. Auto-complete Address

Auto-complete Class Diagram


android.widget
ArrayAdapter
Filterable
edu.ucsb.cs.cs48.schedoptim
adapter
PlaceAutoSuggestAdapter
context : Context
resource : int
results : ArrayList<String>
PlaceAutoSuggestAdapter()
getCount()
getFilter()
getItem()
autoComplete
AutoComplete
PlantUML diagram generated by SketchIt! (https://bitbucket.org/pmesmeur/sketch.it)
For more information about this tool, please contact philippe.mesmeur@gmail.com

Chapter 6. Acknowledgement of External Resources

6.1. Tutorials

Google Official tutorials of RecyclerViews, RoomDatabase, Integrating Google Maps, Calling the Directions API, Drawing markers and routes
https://developer.android.com/guide/

Using JsonSimple Library to parse JSON output:
https://mkyong.com/java/json-simple-example-read-and-write-json/

Auto fit text size for Android API25 and earlier:
https://blog.csdn.net/zhangphil/article/details/79891765

Use and custom date/time pickers:
https://www.jianshu.com/p/d3744c2b480a

Visual display of the schedule:
https://inducesmile.com/android/android-custom-calendar-day-view-example/

Draggable RecyclerView items:
https://www.journaldev.com/23208/android-recyclerview-drag-and-drop

6.2. Open Source Projects

Color Picker (by kristiyanP):
https://github.com/kristiyanP/colorpicker

Android Debug Database (by amitshekhariitbhu):
https://github.com/amitshekhariitbhu/Android-Debug-Database

GSON Library (by Google)
https://github.com/google/gson

Json-simple (by fangyidong)
https://github.com/fangyidong/json-simple