Wentworth Institute of technology

COMP4960 – Software Engineering

Instructor : Atef Suleiman

Software Design Document
For
Checkit

Version 1

Prepared by
Team: SocialTask
Manyeuris Soriano, sorianom@wit.edu
Nicholas LoPilato, lopilaton@wit.edu
Bryan Murphy, murphyb18@wit.edu

Table Of Contents

1. Introduction

1.1. Document Purpose

This document is intended to provide an overview of the design of the mobile application Nexus. Throughout this document you will understand the architecture, UML class diagram, Use Case diagram, to fully explain how our group will approach creating Nexus.

1.2. Product Description

This product works as a time management software that will allow users to keep track of tasks they need to do throughout the week, while also displaying these tasks to the public and their friends in order to have a sense of accountability when people see you have not completed a task.

1.3. Problem Statements

There has been a disappointment from the task management applications of today. One study showed that only 1 in 8 people use a dedicated time management system. That compared to the average person spending two and half hours on social media a day. To help improve time management and have people complete their task on time, why not combine these ideas and have task management become a part of social media.

1.4. Definitions

- Public
  - Only viewed by user and any other user
- Friends
  - Only viewed by any other user tagged as a friend
- Groups
  - Only viewed by user and any other user tagged within a group
- Private
  - Only viewed by only the user
- UI
  - User Interface
- UML
  - Unified Modeling Language

2. System Architecture

2.1. Overall Architecture

Diagram 1.1 shows the layered architecture. Indicates how the client will interact with the software. Also how the software will interact with the database.

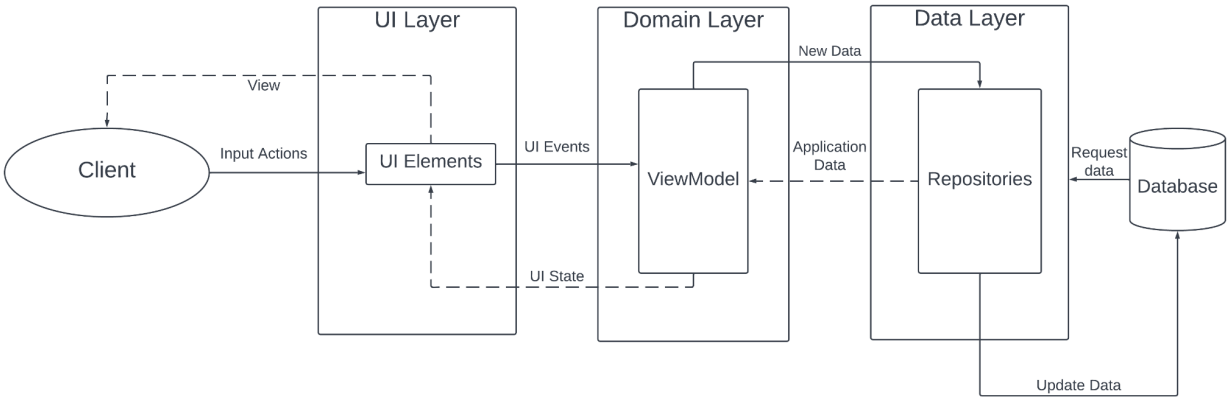


Diagram 1.1 Overall Architecture

2.2. Components mapping

2.2.1. Functional Requirements

2.2.1.1. Task

2.2.1.1.1. Title

2.2.1.1.2. Location

2.2.1.1.3. Due date and time

2.2.1.1.4. Description

2.2.1.1.5. Whether the task is repeatable

2.2.1.1.6. Whether the task is public

2.2.1.1.7. Whether other users can join task

2.2.1.2. User Profile

2.2.1.2.1. Sign up

2.2.1.2.2. Remove profile

2.2.1.2.3. Manage profile information

2.2.1.3. Friends

2.2.1.3.1. Add and remove other users as friends

2.2.1.4. Database

2.2.1.4.1. Store user data

2.2.1.4.1.1. Store username

2.2.1.4.1.2. Store password

2.2.1.4.1.3. Store email address

2.2.1.4.1.4. Store display name

2.2.1.4.1.5. Store biography

2.2.1.4.1.6. Store profile picture

2.2.1.4.1.7. Store the a list of other users tagged as a friend of the user

2.2.1.4.2. Store Task data
2.2.1.4.2.1. Store task title
2.2.1.4.2.2. Store the task description
2.2.1.4.2.3. Store the task picture
2.2.1.4.2.4. Store the task title
2.2.1.4.2.5. Store the task date
2.2.1.4.2.6. Store the location
2.2.1.5. Control bar buttons
2.2.1.5.1. Home button
2.2.1.5.2. Search button
2.2.1.5.3. New task button
2.2.1.5.4. Interactions button
2.2.1.5.5. My tasks button
2.2.1.6. Home feed screen
2.2.1.6.1. Mode button
2.2.1.6.2. Settings button
2.2.1.7. My tasks screen buttons
2.2.1.7.1. Subject button
2.2.1.8. Settings screen
2.2.1.8.1. Username button
2.2.1.8.2. Email button
2.2.1.8.3. Profile picture button
2.2.1.8.4. Linked accounts dropdown
2.2.1.8.5. Private account button
2.2.1.8.6. Theme mode button
2.2.1.8.7. Size of tasks slider
2.2.1.8.8. Save button
2.2.1.9. Selected task screen
2.2.1.9.1. Task button
2.2.1.10. Interactions screen
2.2.1.10.1. Friend request label
2.2.1.10.2. Recent activity label
2.2.1.10.3. Past activity label
2.2.1.11. New task screen
2.2.1.11.1. Title text field
2.2.1.11.2. Location text field
2.2.1.11.3. Time text field
2.2.1.11.4. Task description text field
2.2.1.11.5. Task privacy level dropdown
2.2.1.12. Upload new profile picture screen
2.2.1.12.1. Back button
2.2.1.12.2. Save button
2.2.1.12.3. Current selected task picture image
2.2.1.12.4. Device photos container

2.2.2. GUI Mapping

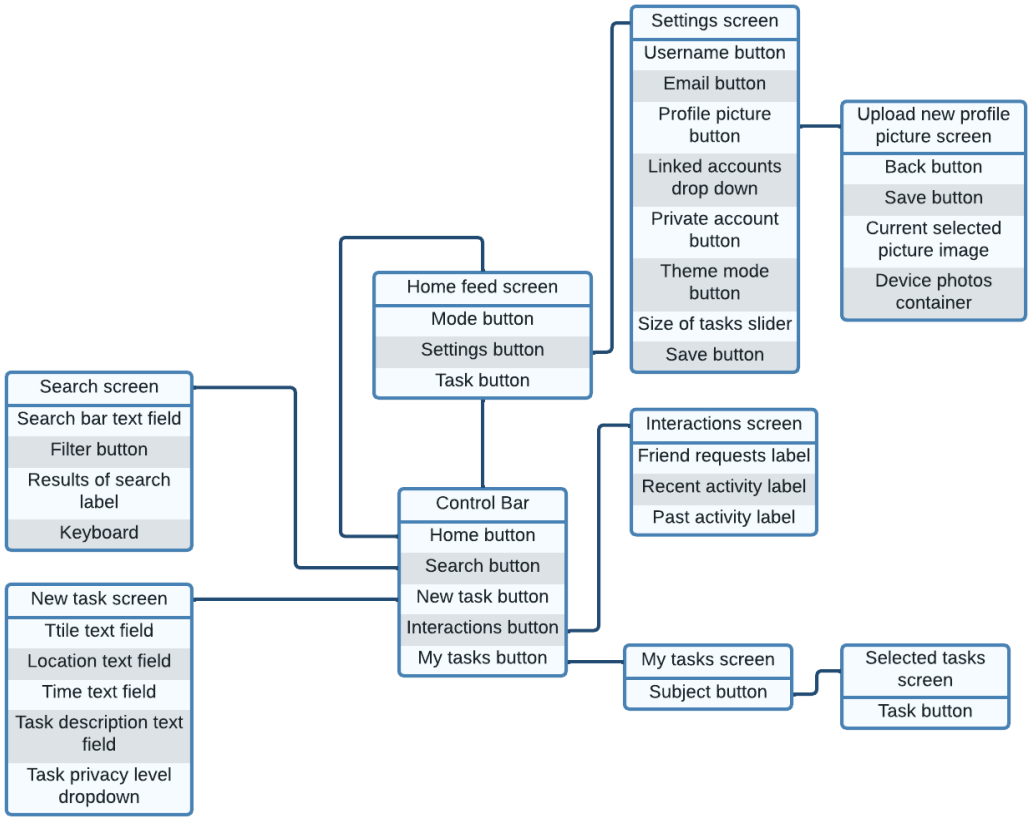


Diagram 1.2 shows the component mapping of the GUI

2.2.3. Non-Functional Requirements

2.2.3.1. Account creation

2.2.3.1.1. Provide an email to link account with

2.2.3.1.2. Username shall be unique

2.2.3.1.3. Password shall be created with at least:

2.2.3.1.3.1. One capitalize letter

2.2.3.1.3.2. One lowercase letter

2.2.3.1.3.3. One digit

2.2.3.1.3.4. One special character

2.2.3.1.3.5. Minimum character length of 8

2.2.3.1.4. User Id shall be randomly generated upon account creation

2.2.3.2. Login

2.2.3.2.1. User shall login with a username and password

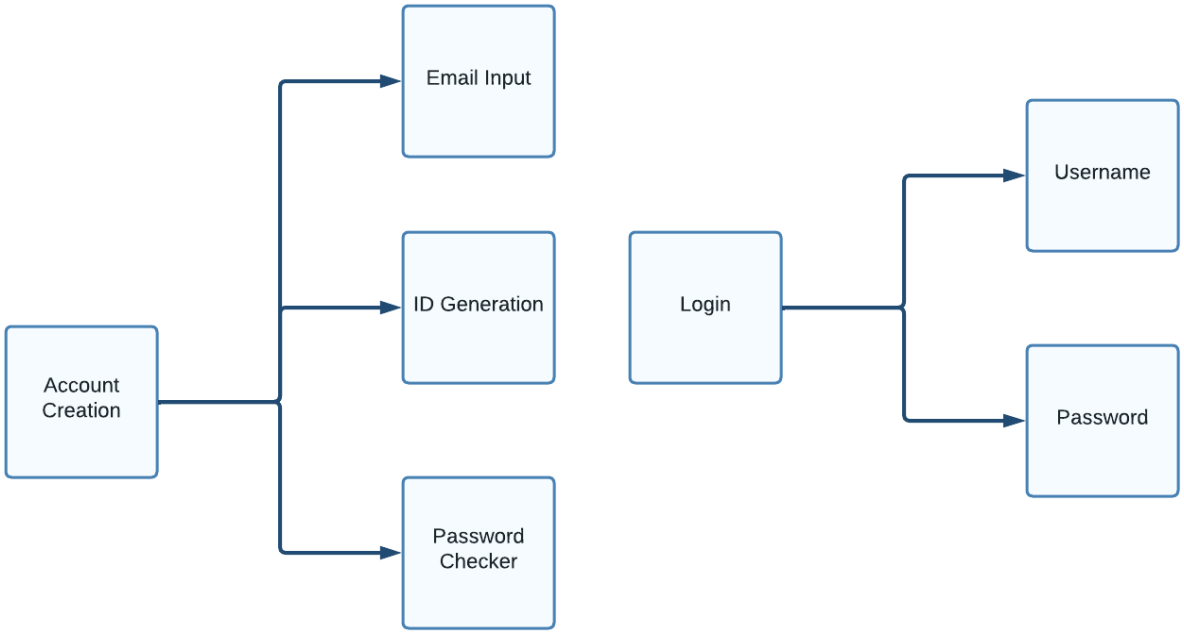


Diagram 1.3 shows the Non-functional requirements

2.3. Traceability Matrix

Design Element	Requirements
Task	5.1.1
User Profile	5.1.2
Friends	5.1.3
Database	5.1.4
Nav bar buttons	5.1.5
Home feed buttons	5.1.6
My tasks buttons	5.1.7
Settings buttons	5.1.8
Home feed screen	5.2.1.1
Nav bar	5.2.1.2
My tasks screen	5.2.1.3

Selected subject screen	5.2.1.4
Settings screen	5.2.1.5
Upload profile picture screen	5.2.1.6
Interactions screen	5.2.1.8
Search screen	5.2.1.9
New task screen	5.2.1.10
All chats screen	5.2.1.11
Chat screen	5.2.1.12
Create group chat screen	5.2.1.13

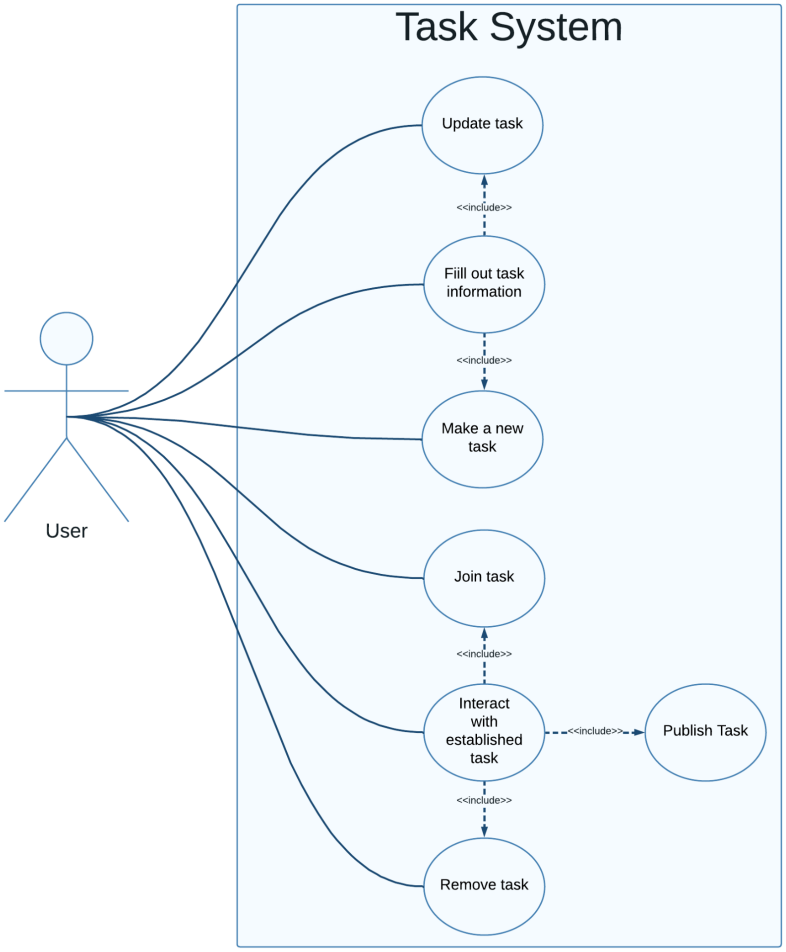
2.4. Technology Stack Selection

- **The system shall be run on IOS 15 or above or Android 12 or above**
    - It is common to support OS versions that came before the latest one. Usually it is up to the developer to decide on how many versions back it will support. For this project, we decided that supporting the latest versions and the previous version is all we needed. Users that have not updated to either of the two latest versions will be minimal and most likely will not be highly active on the program.
- **The system shall require Dart 2.18 or above**
    - Dart is the primary language used in Flutter. Flutter is the IDE we are using to implement the entire app. The Dart language is used to control the UI and the backend of the program. The widgets apart of Flutter provide quick and easy implementation and is the key feature of the IDE which only uses the Dart language.
- **The system shall communicate with Firebase**

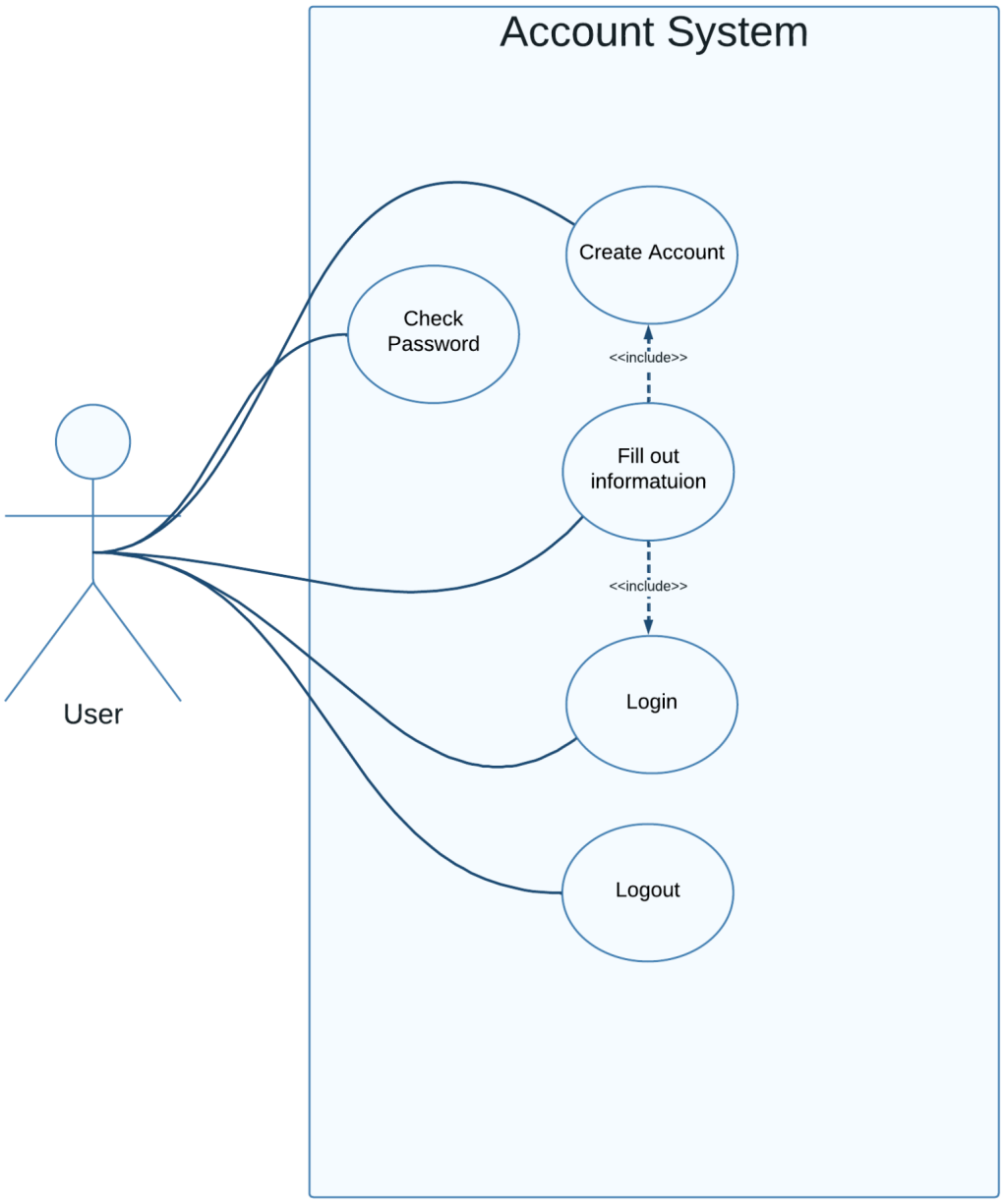
- A database is required to produce any information on the system. Since it is also a social media app, it is also required that the database is accessible through a server. Firebase offers both of these in one. It is also easily implementable in Flutter sinec they are both developed by Google. It provides useful tools for monitoring different aspects of the entre database.

3. Use Case/Activity Diagram

3.1. Diagram 1.4 shows the use case for the task system where the user can make, join, remove, and edit tasks.



3.2. Diagram 1.5 displays the use case of account creation where a user can login with a pre-existing account or create a new one



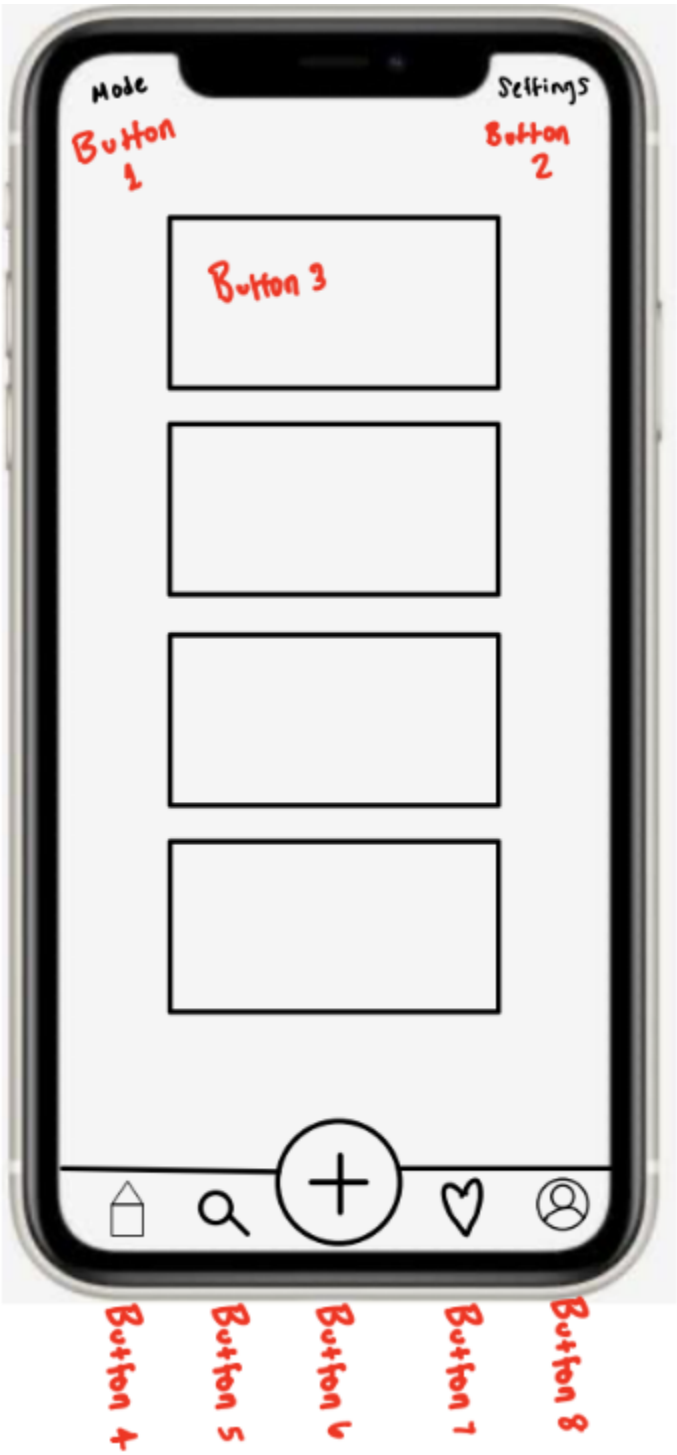
4. System Design

4.1. UI

4.1.1. Diagram 2.1 Login screen

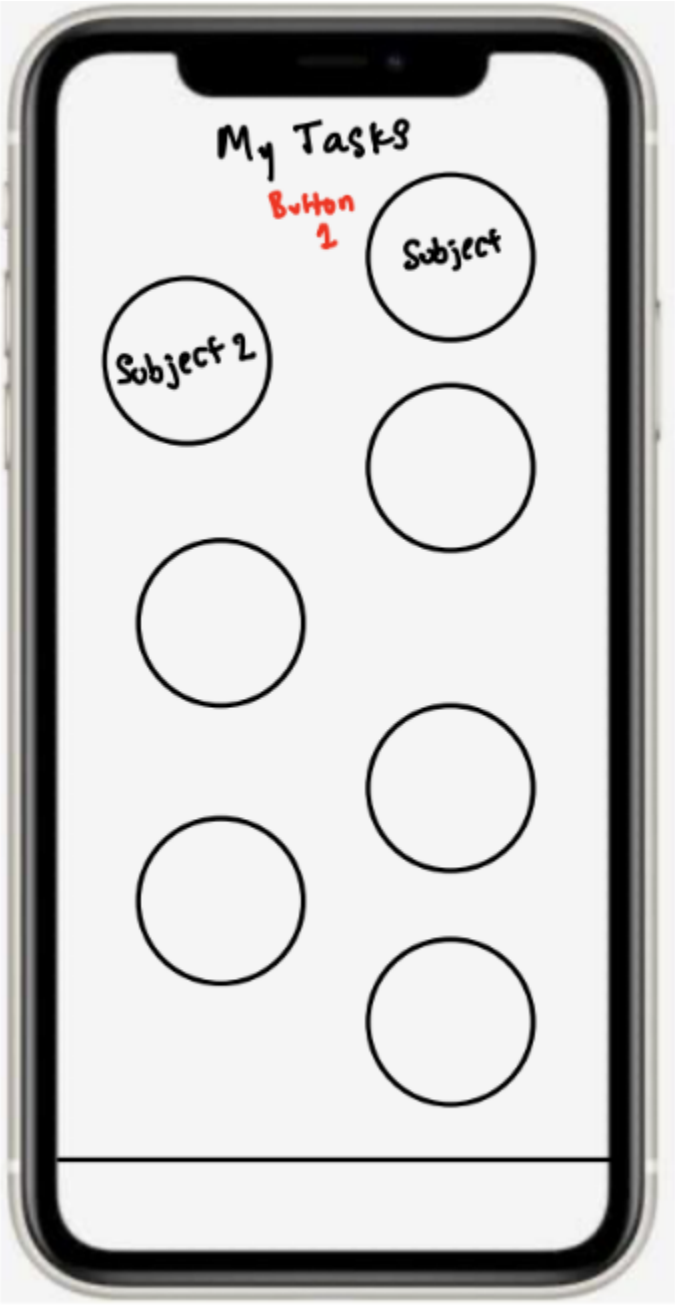
4.1.2. Diagram 2.2 Create account screen

4.1.3. Diagram 2.3 Home feed screen and Control bar

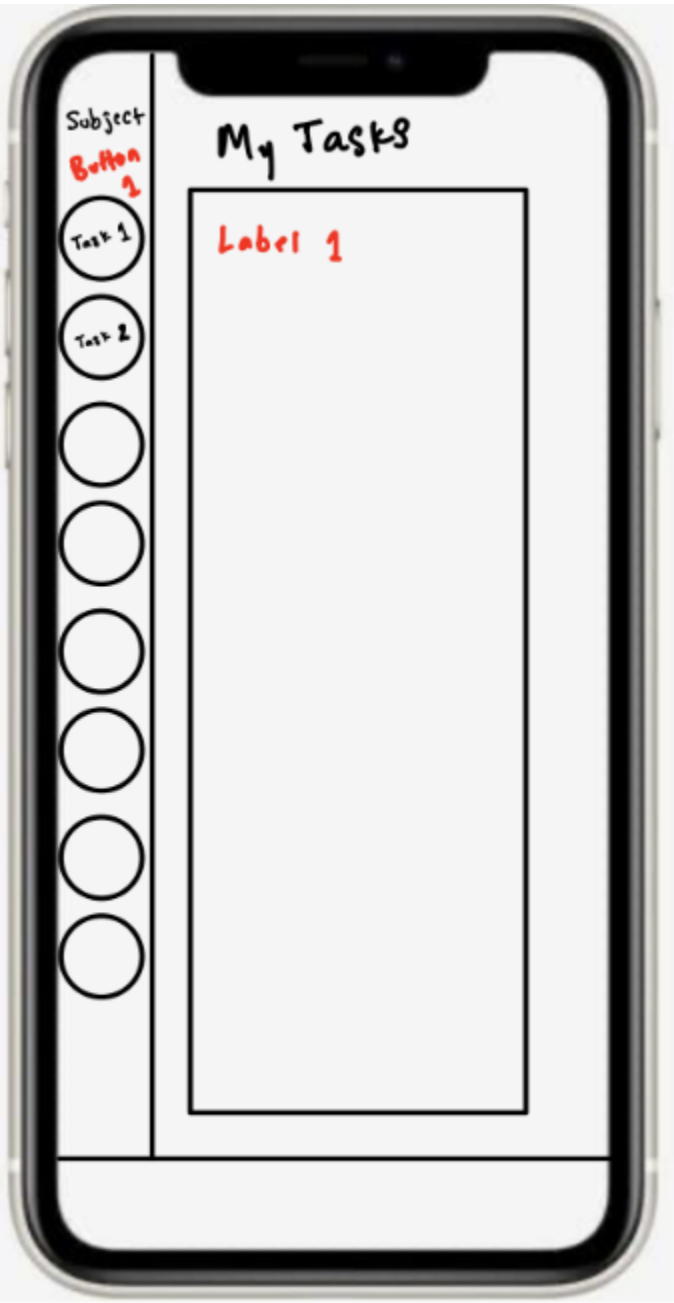


4.1.4. Diagram 2.4 New task screen

4.1.5. Diagram 2.5 My tasks screen



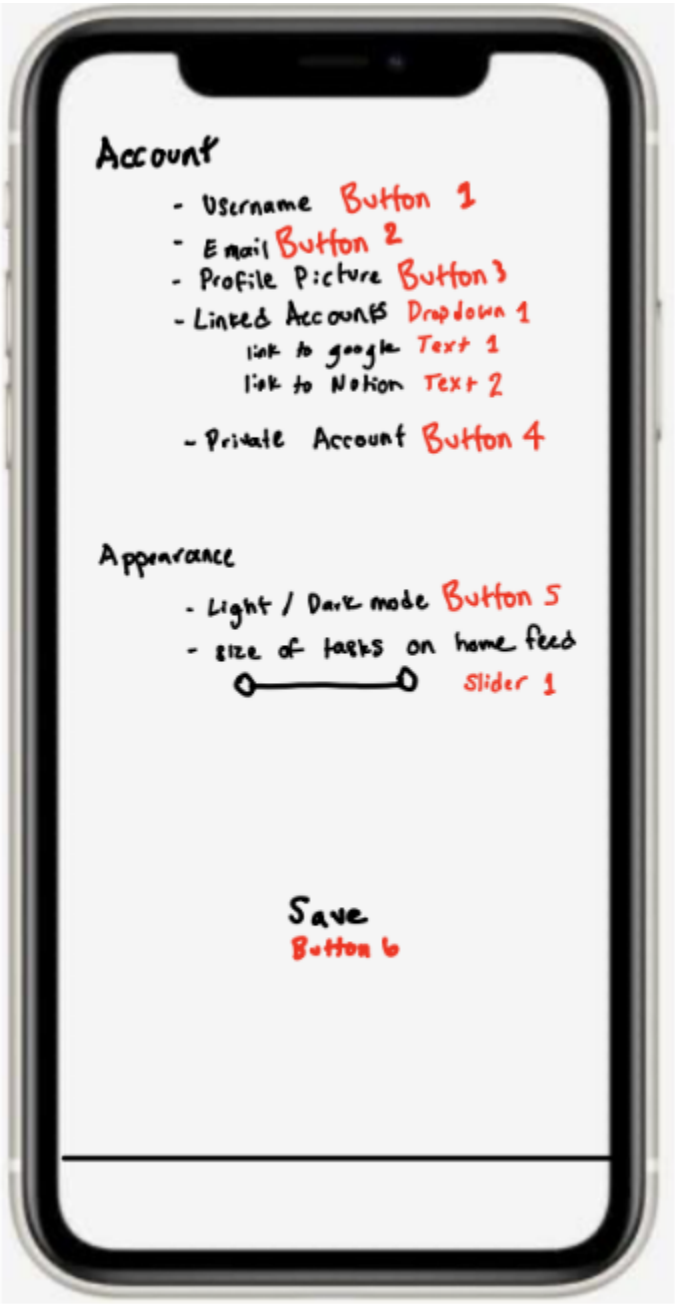
4.1.6. Diagram 2.6 Selected task screen



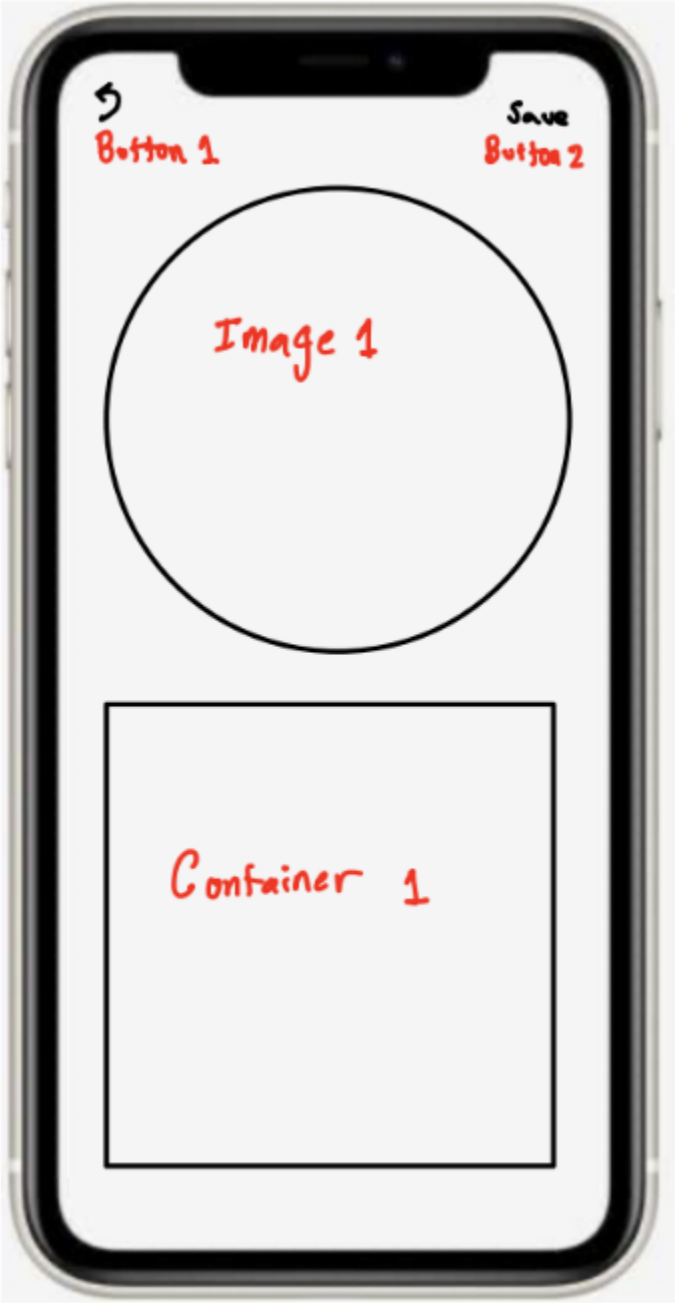
4.1.7. Diagram 2.7 Interactions screen

4.1.8. Diagram 2.8 Search screen

4.1.9. Diagram 2.9 Settings screen

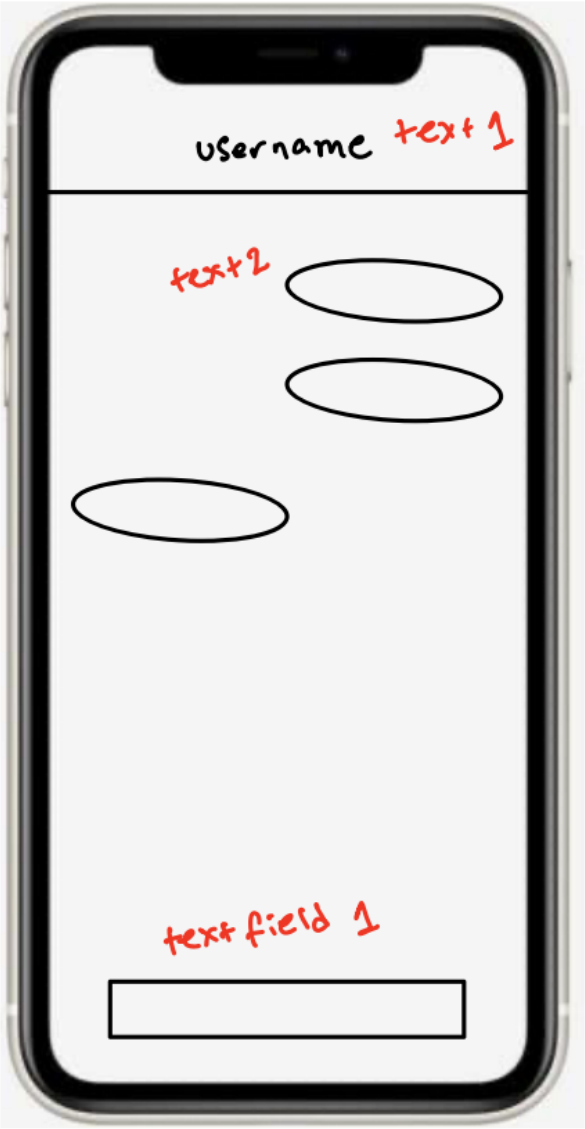


4.1.10. Diagram 2.10 Upload new profile picture screen

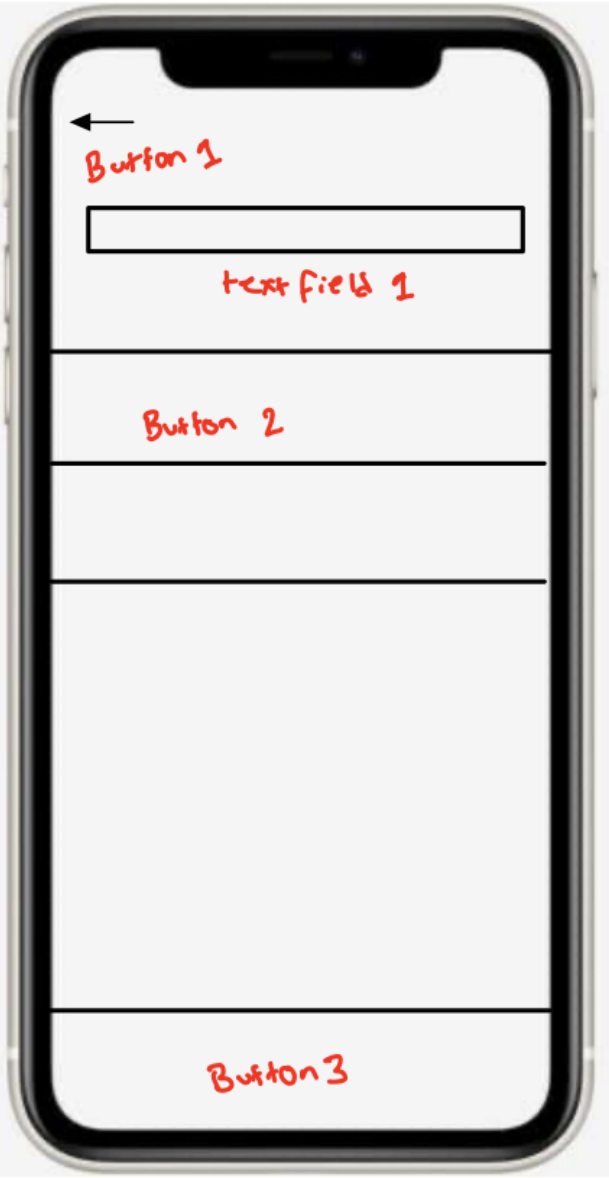


4.1.11. Diagram 2.11 All chats screen

4.1.12. Diagram 2.12 Caht screen

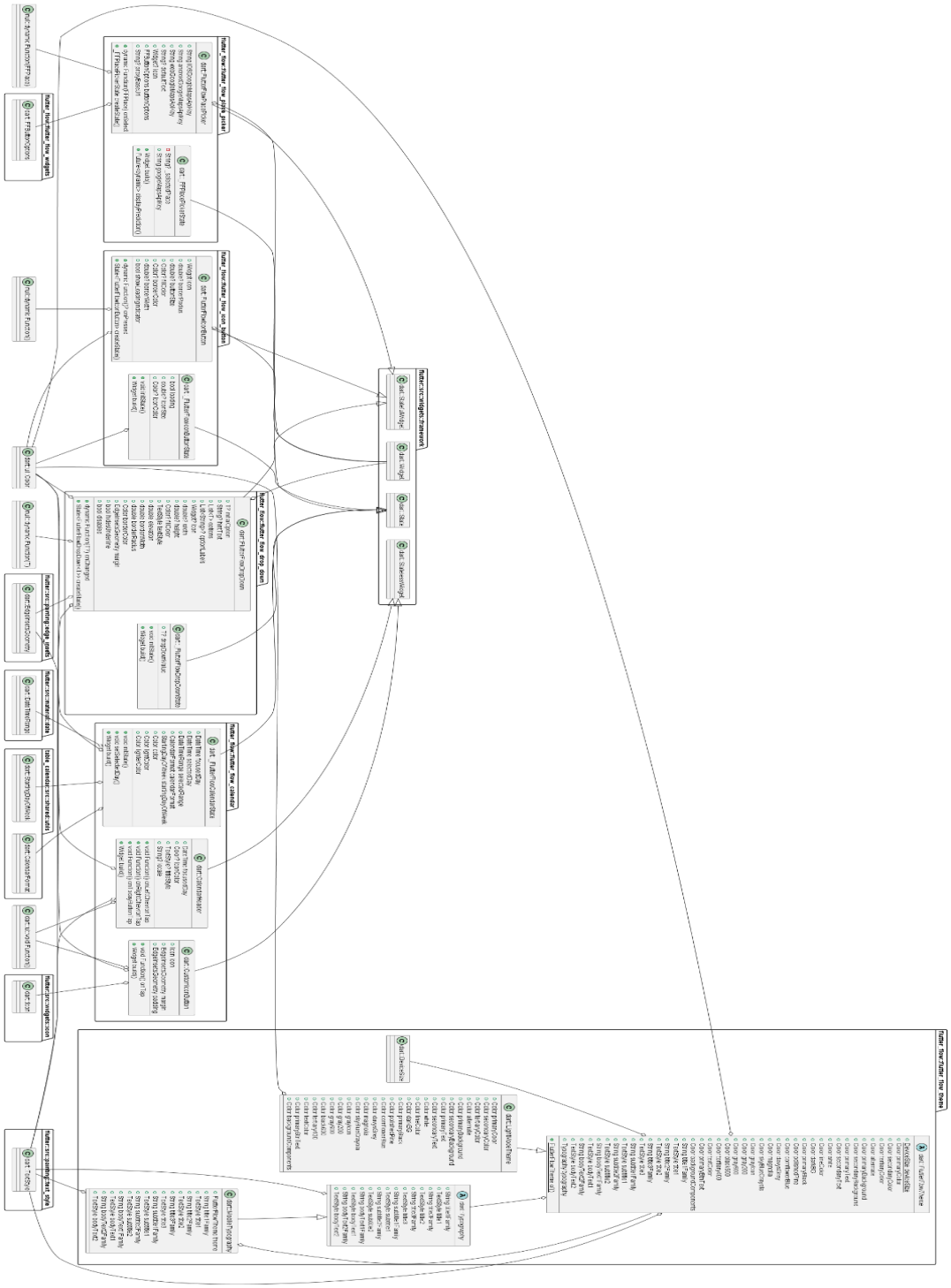


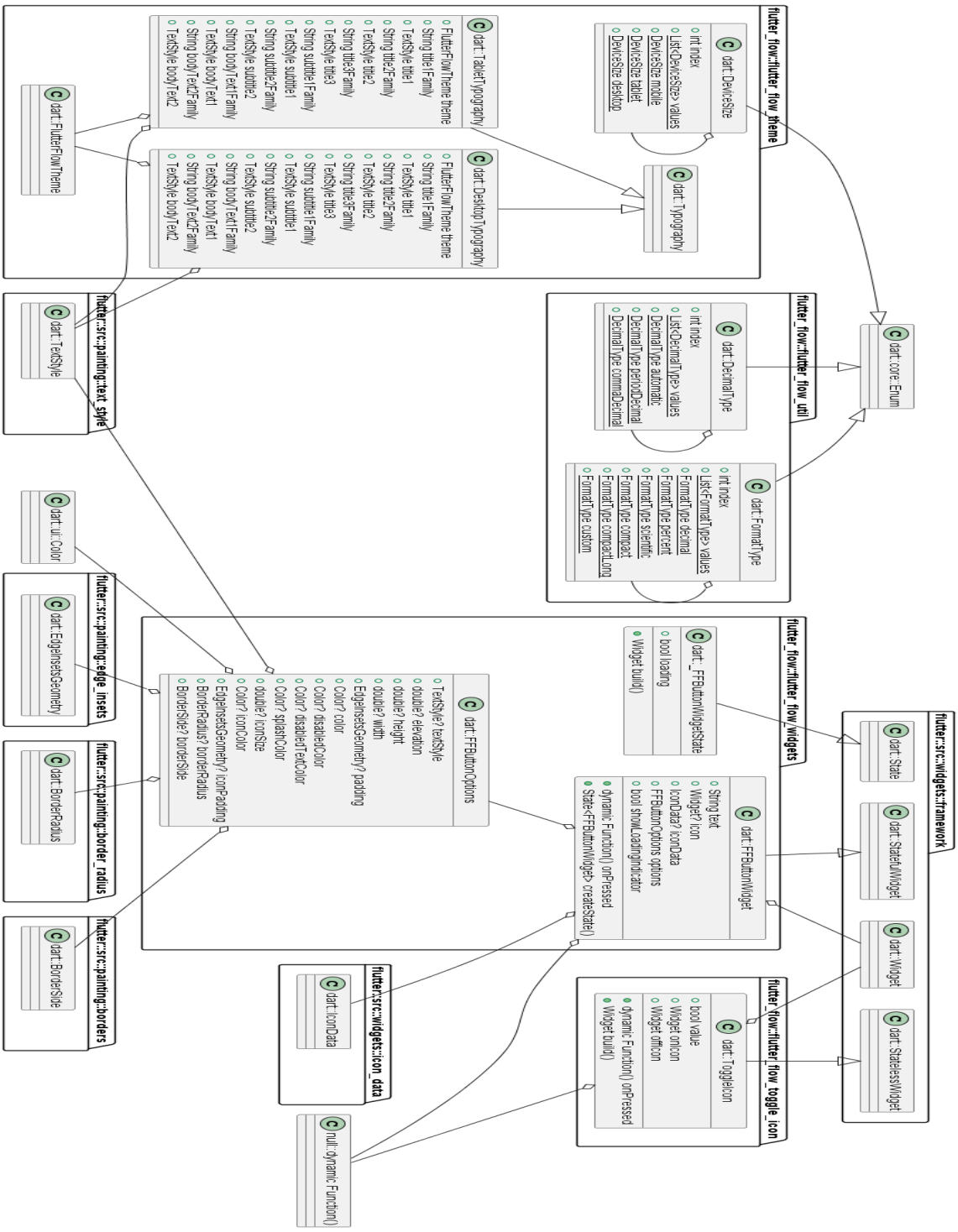
4.1.13. Diagram 2.13 Create group chat screen

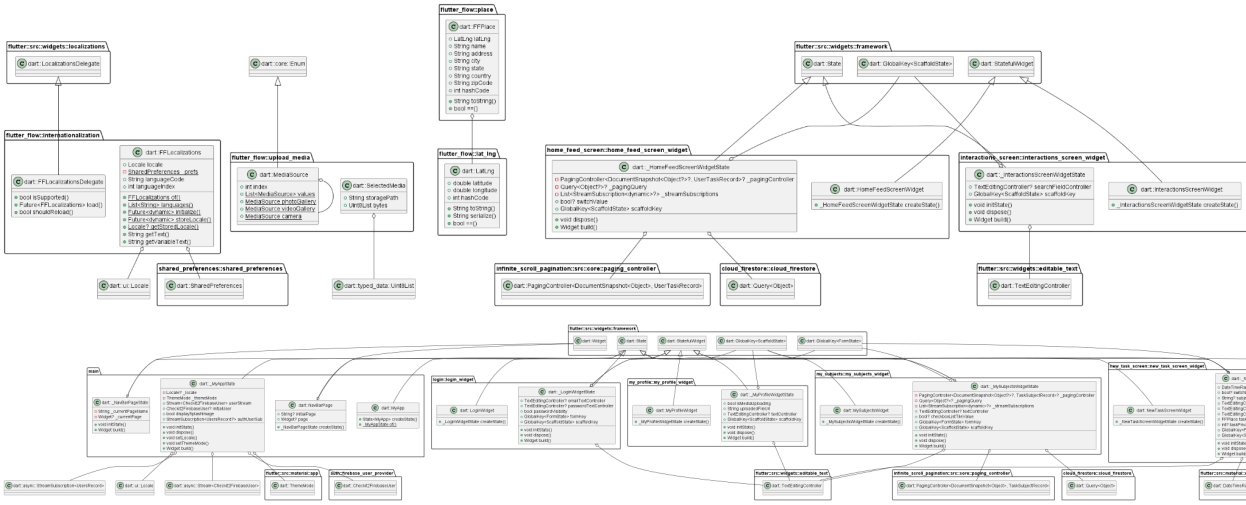


4.2. Class diagram

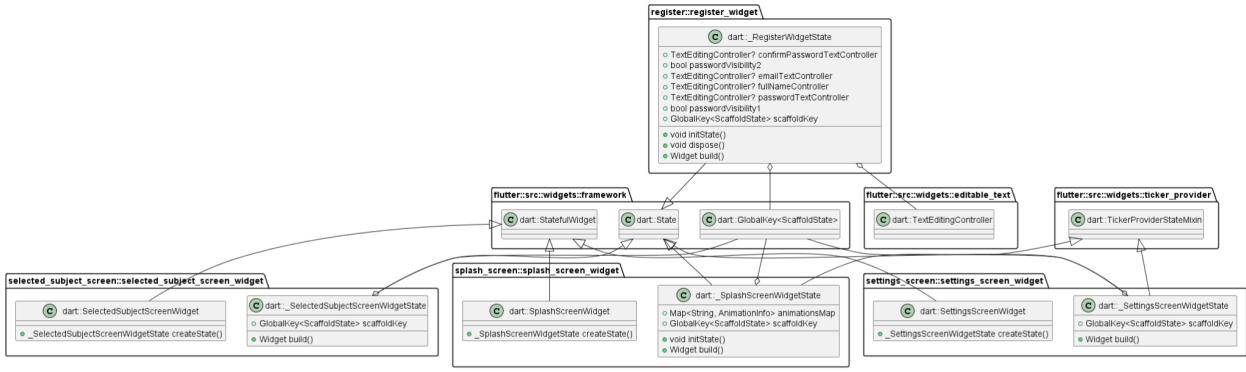
Diagram 2.11 shows the classes that are in the software and their relation to each other.

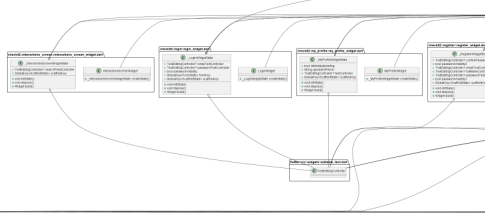


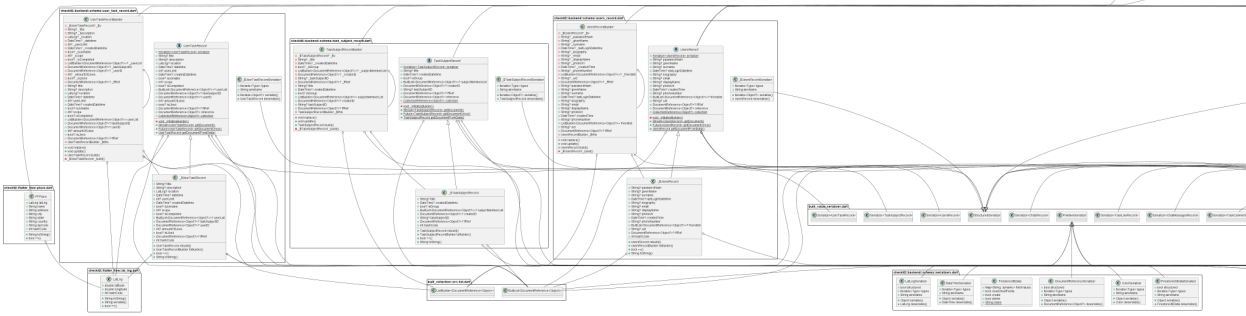

flutter_flow::flutter_flow_theme
dart::DeviceSize
int index
List<DeviceSize> values
DeviceSize mobile
DeviceSize tablet
DeviceSize desktop
dart::Typography
dart::TabletTypography
FlutterFlowTheme theme
String title1Family
TextStyle title1
String title2Family
TextStyle title2
String title3Family
TextStyle title3
String subtitle1Family
TextStyle subtitle1
String subtitle2Family
TextStyle subtitle2
String bodyText1Family
TextStyle bodyText1
String bodyText2Family
TextStyle bodyText2
dart::DesktopTypography
dart::FlutterFlowTheme
flutter::src::painting::text_style
dart::TextStyle
flutter_flow::flutter_flow_util
dart::DecimalType
List<DecimalType> values
DecimalType automatic
DecimalType periodDecimal
DecimalType commaDecimal
dart::FormatType
List<FormatType> values
FormatType decimal
FormatType percent
FormatType scientific
FormatType compact
FormatType compactLong
FormatType custom
dart::core::Enum
flutter::src::widgets::framework
dart::State
dart::StatefulWidget
dart::Widget
dart::StatelessWidget
flutter_flow::flutter_flow_widgets
dart::_FFButtonWidgetState
bool loading
Widget build()
dart::FFButtonWidget
String text
Widget? icon
IconData? iconData
FFButtonOptions options
bool showLoadingIndicator
dynamic Function() onPressed
State<FFButtonWidget> createState()
dart::FFButtonOptions
TextStyle? textStyle
double? elevation
double? height
double? width
EdgeInsetsGeometry? padding
Color? color
Color? disabledColor
Color? disabledTextColor
Color? splashColor
double? iconSize
Color? iconColor
EdgeInsetsGeometry? iconPadding
BorderRadius? borderRadius
BorderSide? borderSide
flutter_flow::flutter_flow_toggle_icon
dart::ToggleIcon
bool value
Widget onIcon
Widget offIcon
dynamic Function() onPressed
Widget build()
flutter::src::painting::edge_insets
dart::EdgeInsetsGeometry
flutter::src::painting::border_radius
dart::BorderRadius
flutter::src::painting::borders
dart::BorderSide
flutter::src::widgets::icon_data
dart::IconData
null::dynamic Function()
dart::ui::Color

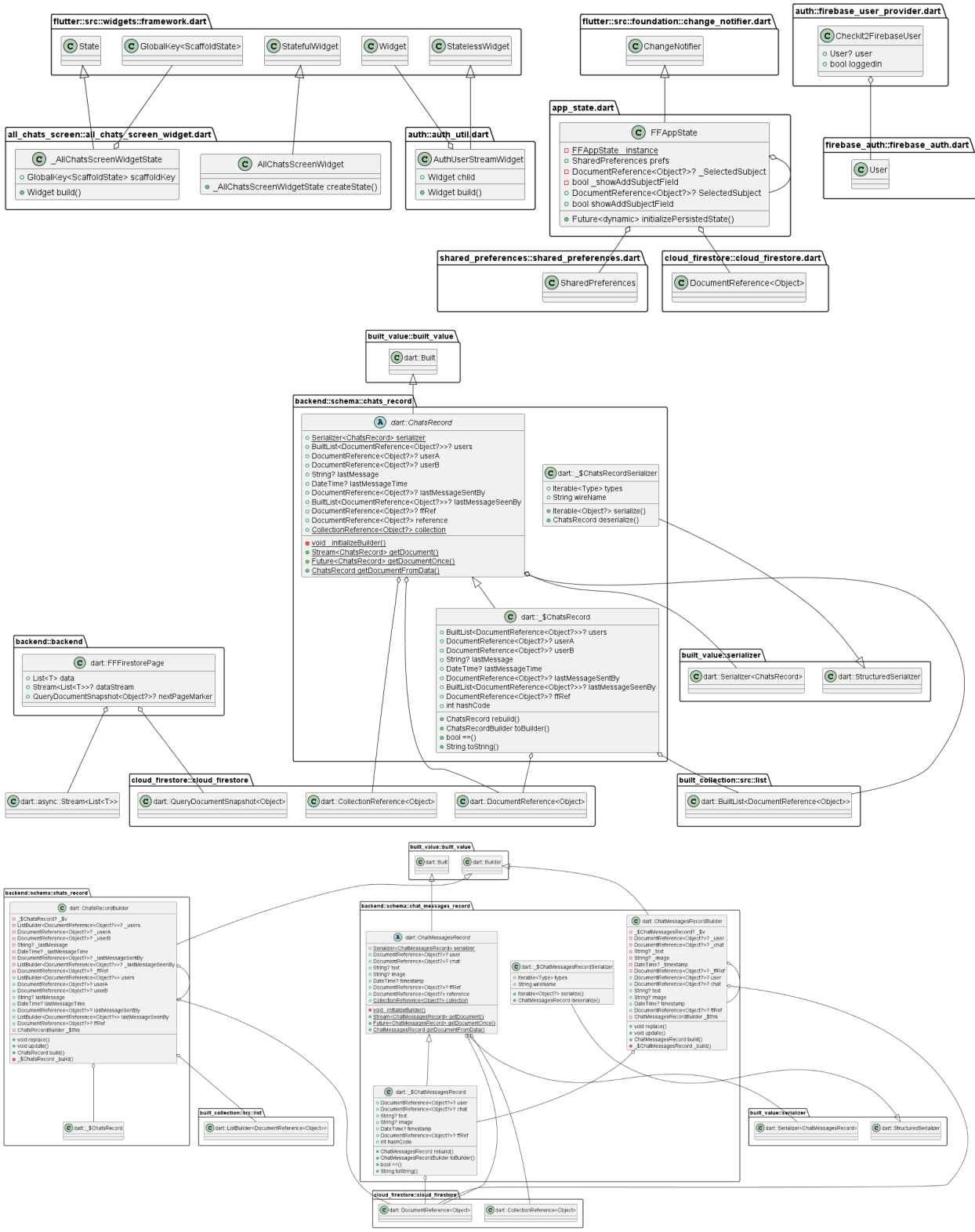

flutter::src::widgets::localizations
dart::LocalizationsDelegate
dart::core::Enum
flutter_flow::place
dart::FFPlace
flutter::src::widgets::framework
dart::State
dart::GlobalKey<ScaffoldState>
dart::StatefulWidget
flutter_flow::internationalization
dart::FFLocalizationsDelegate
dart::FFLocalizations
flutter_flow::upload_media
dart::MediaSource
dart::SelectedMedia
flutter_flow::lat_lng
dart::LatLng
home_feed_screen::home_feed_screen_widget
dart::_HomeFeedScreenWidgetState
dart::HomeFeedScreenWidget
interactions_screen::interactions_screen_widget
dart::_InteractionsScreenWidgetState
dart::InteractionsScreenWidget
dart::ui::Locale
shared_preferences::shared_preferences
dart::SharedPreferences
infinite_scroll_pagination::src::core::paging_controller
dart::PagingController<DocumentSnapshot<Object?>, UserTaskRecord>
cloud_firestore::cloud_firestore
dart::Query<Object?>
flutter::src::widgets::editable_text
dart::TextEditingController
dart::typed_data::UnitList

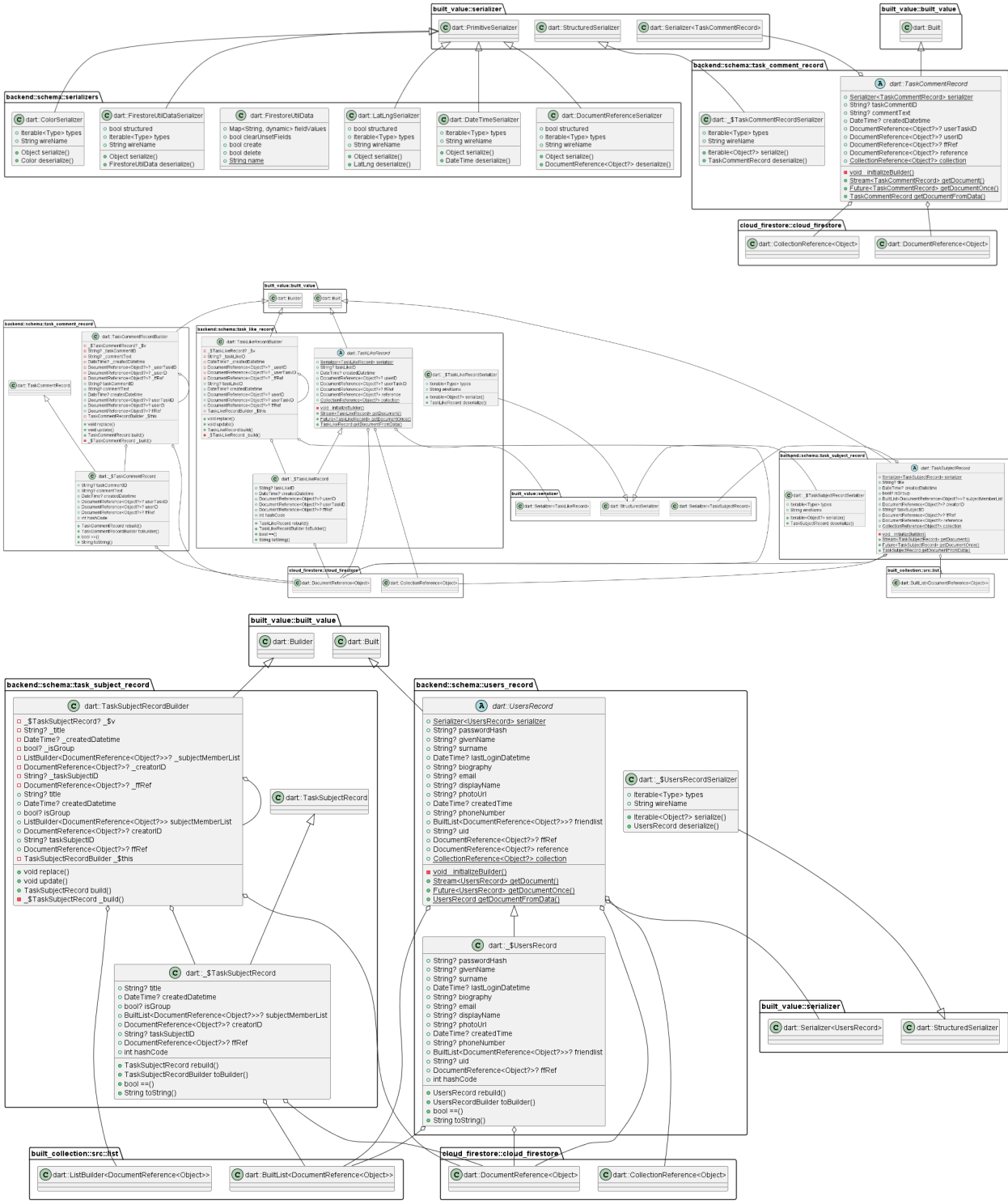
main
login::login_widget
my_profile::my_profile_widget
my_subjects::my_subjects_widget
new_task_screen::new_task_screen_widget
dart::_MyAppState
dart::LoginWidget
dart::_LoginWidgetState
dart::MyProfileWidget
dart::_MyProfileWidgetState
dart::MySubjectsWidget
dart::_MySubjectsWidgetState
dart::NewTaskScreenWidget
dart::_NewTaskScreenWidgetState
flutter::src::material::app
dart::ThemeMode
dart::Locale
dart::async::StreamSubscription<UserRecord>
dart::async::Stream<ChildFirebaseUser>
dart::MyApp
dart::NavBarPage
dart::_NavBarPageState
dart::ChildFirebaseUser
flutter::src::widgets::editable_text
infinite_scroll_pagination::src::core::paging_controller
cloud_firestore::cloud_firestore


register::register_widget
dart::_RegisterWidgetState
TextEditingController? confirmPasswordTextController
bool passwordVisibility2
TextEditingController? emailTextController
TextEditingController? fullNameController
TextEditingController? passwordTextController
bool passwordVisibility1
GlobalKey<ScaffoldState> scaffoldKey
void initState()
void dispose()
Widget build()
flutter::src::widgets::framework
dart::StatefulWidget
dart::State
dart::GlobalKey<ScaffoldState>
flutter::src::widgets::editable_text
dart::TextEditingController
flutter::src::widgets::ticker_provider
dart::TickerProviderStateMixin
selected_subject_screen::selected_subject_screen_widget
dart::SelectedSubjectScreenWidget
dart::_SelectedSubjectScreenWidgetState
_SelectedSubjectScreenWidgetState createState()
GlobalKey<ScaffoldState> scaffoldKey
Widget build()
splash_screen::splash_screen_widget
dart::SplashScreenWidget
dart::_SplashScreenWidgetState
_SplashScreenWidgetState createState()
Map<String, AnimationInfo> animationsMap
GlobalKey<ScaffoldState> scaffoldKey
void initState()
Widget build()
settings_screen::settings_screen_widget
dart::SettingsScreenWidget
dart::_SettingsScreenWidgetState
_SettingsScreenWidgetState createState()
GlobalKey<ScaffoldState> scaffoldKey
Widget build()






flutter::src::widgets::framework.dart
State
GlobalKey<ScaffoldState>
StatefulWidget
Widget
StatelessWidget
flutter::src::foundation::change_notifier.dart
ChangeNotifier
auth::firebase_user_provider.dart
CheckIt2FirebaseUser
User? user
bool loggedIn
all_chats_screen::all_chats_screen_widget.dart
_AllChatsScreenWidgetState
GlobalKey<ScaffoldState> scaffoldKey
Widget build()
AllChatsScreenWidget
_AllChatsScreenWidgetState createState()
auth::auth_util.dart
AuthUserStreamWidget
Widget child
Widget build()
app_state.dart
FFAppState
FFAppState _instance
SharedPreferences prefs
DocumentReference<Object?>? _SelectedSubject
bool _showAddSubjectField
DocumentReference<Object?>? SelectedSubject
bool showAddSubjectField
Future<dynamic> initializePersistedState()
firebase_auth::firebase_auth.dart
User
shared_preferences::shared_preferences.dart
SharedPreferences
cloud_firestore::cloud_firestore.dart
DocumentReference<Object>
built_value::built_value
dart::Built
backend::schema::chats_record
dart::ChatsRecord
Serializer<ChatsRecord> serializer
BuiltList<DocumentReference<Object?>>? users
DocumentReference<Object?>? userA
DocumentReference<Object?>? userB
String? lastMessage
DateTime? lastMessageTime
DocumentReference<Object?>? lastMessageSentBy
BuiltList<DocumentReference<Object?>>? lastMessageSeenBy
DocumentReference<Object?>? ffRef
DocumentReference<Object?> reference
CollectionReference<Object?> collection
void _initializeBuilder()
Stream<ChatsRecord> getDocument()
Future<ChatsRecord> getDocumentOnce()
ChatsRecord getDocumentFromData()
dart::_$ChatsRecordSerializer
Iterable<Type> types
String wireName
Iterable<Object?> serialize()
ChatsRecord deserialize()
dart::_$ChatsRecord
BuiltList<DocumentReference<Object?>>? users
DocumentReference<Object?>? userA
DocumentReference<Object?>? userB
String? lastMessage
DateTime? lastMessageTime
DocumentReference<Object?>? lastMessageSentBy
BuiltList<DocumentReference<Object?>>? lastMessageSeenBy
DocumentReference<Object?>? ffRef
int hashCode
ChatsRecord rebuild()
ChatsRecordBuilder toBuilder()
bool ==()
String toString()
backend::backend
dart::FFFirestorePage
List<T> data
Stream<List<T>>? dataStream
QueryDocumentSnapshot<Object?>? nextPageMarker
built_value::serializer
dart::Serializer<ChatsRecord>
dart::StructuredSerializer
dart::async::Stream<List<T>>
cloud_firestore::cloud_firestore
dart::QueryDocumentSnapshot<Object>
dart::CollectionReference<Object>
dart::DocumentReference<Object>
built_collection::src::list
dart::BuiltList<DocumentReference<Object>>
built_value::built_value
dart::Built
dart::Builder
backend::schema::chats_record
dart::ChatsRecordBuilder
_$ChatsRecord? _$v
ListBuilder<DocumentReference<Object?>>? _users
DocumentReference<Object?>? _userA
DocumentReference<Object?>? _userB
String? _lastMessage
DateTime? _lastMessageTime
DocumentReference<Object?>? _lastMessageSentBy
ListBuilder<DocumentReference<Object?>>? _lastMessageSeenBy
DocumentReference<Object?>? _ffRef
ListBuilder<DocumentReference<Object?>> users
DocumentReference<Object?>? userA
DocumentReference<Object?>? userB
String? lastMessage
DateTime? lastMessageTime
DocumentReference<Object?>? lastMessageSentBy
ListBuilder<DocumentReference<Object?>> lastMessageSeenBy
DocumentReference<Object?>? ffRef
ChatsRecordBuilder _$this
void replace()
void update()
ChatsRecord build()
_$ChatsRecord _build()
dart::_$ChatsRecord
built_collection::src::list
dart::ListBuilder<DocumentReference<Object>>
backend::schema::chat_messages_record
dart::ChatMessagesRecord
Serializer<ChatMessagesRecord> serializer
DocumentReference<Object?>? user
DocumentReference<Object?>? chat
String? text
String? image
DateTime? timestamp
DocumentReference<Object?>? ffRef
DocumentReference<Object?>? reference
CollectionReference<Object?> collection
void _initializeBuilder()
Stream<ChatMessagesRecord> getDocument()
Future<ChatMessagesRecord> getDocumentOnce()
ChatMessagesRecord getDocumentFromData()
dart::_$ChatMessagesRecordSerializer
Iterable<Type> types
String wireName
Iterable<Object?> serialize()
ChatMessagesRecord deserialize()
dart::ChatMessagesRecordBuilder
_$ChatMessagesRecord? _$v
DocumentReference<Object?>? _user
DocumentReference<Object?>? _chat
String? _text
String? _image
DateTime? _timestamp
DocumentReference<Object?>? _ffRef
DocumentReference<Object?>? user
DocumentReference<Object?>? chat
String? text
String? image
DateTime? timestamp
DocumentReference<Object?>? ffRef
ChatMessagesRecordBuilder _$this
void replace()
void update()
ChatMessagesRecord build()
_$ChatMessagesRecord _build()
dart::_$ChatMessagesRecord
DocumentReference<Object?>? user
DocumentReference<Object?>? chat
String? text
String? image
DateTime? timestamp
DocumentReference<Object?>? ffRef
int hashCode
ChatMessagesRecord rebuild()
ChatMessagesRecordBuilder toBuilder()
bool ==()
String toString()
built_value::serializer
dart::Serializer<ChatMessagesRecord>
dart::StructuredSerializer
cloud_firestore::cloud_firestore
dart::DocumentReference<Object>
dart::CollectionReference<Object>


built_value::serializer
dart::PrimitiveSerializer
dart::StructuredSerializer
dart::Serializer<TaskCommentRecord>
built_value::built_value
dart::Built
backend::schema::serializers
dart::ColorSerializer
dart::FirestoreUtilDataSerializer
dart::FirestoreUtilData
dart::LatLngSerializer
dart::DateTimeSerializer
dart::DocumentReferenceSerializer
backend::schema::task_comment_record
dart::_$TaskCommentRecordSerializer
dart::TaskCommentRecord
cloud_firestore::cloud_firestore
dart::CollectionReference<Object>
dart::DocumentReference<Object>
built_value::built_value
dart::Builder
dart::Built
backend::schema::task_comment_record
dart::TaskCommentRecordBuilder
dart::TaskCommentRecord
dart::_$TaskCommentRecord
backend::schema::task_like_record
dart::TaskLikeRecordBuilder
dart::TaskLikeRecord
dart::_$TaskLikeRecord
dart::_$TaskLikeRecordSerializer
built_value::serializer
dart::StructuredSerializer
backend::schema::task_subject_record
dart::TaskSubjectRecord
dart::_$TaskSubjectRecordSerializer
cloud_firestore::cloud_firestore
dart::DocumentReference<Object>
dart::CollectionReference<Object>
built_collection::src::list
dart::Built_list<DocumentReference<Object>>
built_value::built_value
dart::Builder
dart::Built
backend::schema::task_subject_record
dart::TaskSubjectRecordBuilder
dart::TaskSubjectRecord
dart::_$TaskSubjectRecord
backend::schema::users_record
dart::UsersRecord
dart::_$UsersRecord
dart::_$UsersRecordSerializer
built_value::Serializer
dart::Serializer<UsersRecord>
dart::StructuredSerializer
built_collection::src::list
dart::ListBuilder<DocumentReference<Object>>
dart::BuiltList<DocumentReference<Object>>
cloud_firestore::cloud_firestore
dart::DocumentReference<Object>
dart::CollectionReference<Object>

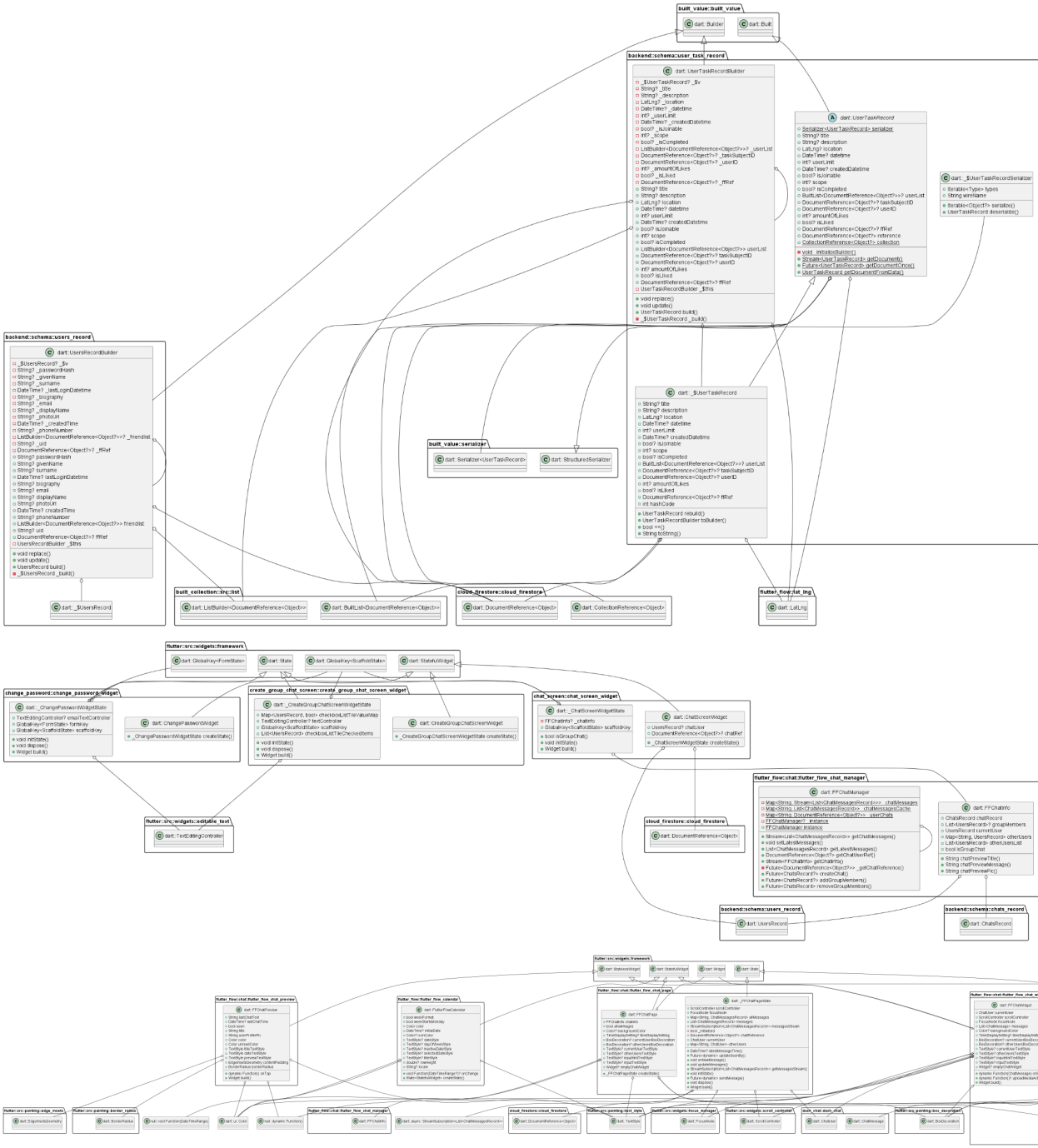


Diagram 2.11 shows the UML class diagram

4.2.1. UserLoginRepository

This is where the client's login information will be retrieved from and saved to the database. It will also verify if the username and password are valid. This class will also generate the unique user ids when saving a new user. This repository implements the **Contact** struct

4.2.2. UserRepository

This class is where the client's profile information will be retrieved from and saved to the database.

4.2.3. MessageRepository

This class saves and retrieves messages from the database, it implements the **Message** struct.

4.2.4. SubjectRepository

This class saves and retrieves subjects that hold a list of tasks. This repository is also incharge of saving tasks to the database.

4.2.5. TaskRepository

This class retrieves tasks without having to go through the subject repository. This class depends on the enumerable **TaskScope.**

4.2.6. SubjectElement

This class will be in charge of the visualization of subjects, while also containing a saved instance of a task_subject entity that was retrieved from the database using **SubjectRepository**.

4.2.7. TaskElement

This class will be in charge of the visualization of tasks. Display their information. While also containing a saved instance of a user_task entity and retrieving it from the database using **TaskRepository.**

4.2.8. UserProfle

This class will hold all the actions the client is allowed to perform. It has a saved instance of the client information

4.2.9. ControlBarElement

This class contains the buttons and UI control for the controlbar. This will contain the methods to switch between screens, and will be persistent throughout all screens, being the parent of the **Screen** class.

4.2.10. Screen

This is an abstract class that will define the base blueprint for all screens. This is a child of **ControlBarElement.**

4.2.11. HomeFeedScreen

This class is a child of **Screen** that will display the feed. It will have methods that update the feed and any action unique to the feed. This will implement **TaskElement**.

4.2.12. SelectSubjectScreen

This class is a child of **Screen** that will display the **SubjectElement**. This is where **SubjectElement** is implemented.

4.2.13. NewTaskScreen

This class is a child of **Screen** that will allow the client to input information pertaining to create a subject_task entity.

4.2.14. MyTaskScreen

This class is a child of **Screen** that will display **TaskElement.** This is where **TaskElement** is implemented.

4.2.15. SettingsScreen

This class is a child of **Screen** that will collect client input. This is where the client can update their **UserProfile.**

4.2.16. UploadNewProfileScreen

This class is a child of **SettingsScreen** that will display textField. This is where the client can upload a photo that will save in their **UserProfile.**

4.2.17. SearchScreen

This is a child of **Screen** that will display the result found from the client input. This will implement **TaskElement, SubjectElement,** and **UserProfile**.

4.2.18. InteractionsScreen

This is a child of **Screen** that will display the action other clients have done towards the clients account. This will also contain tasks that have been recently completed and tasks that are due soon.

4.2.19. Contact

A struct that holds client personal information.

4.2.20. Message

A struct that holds the owner of the message and the contents of the message.

4.2.21. TaskScope

An enumerable that allows for better defined visibility levels of objects.

5. Data Model

In this model there are various entities and relations in the database. Every entity primarily revolves around the User entity. This is because each user creates all the different data in the application. For example, if a new User Task is created then it must have one and only one User that relates to it. The User Task also has one and only one Task Subject. The Task Subjects are also created by a user so they must have a single User that creates it, but also has one or many Subject Members total.

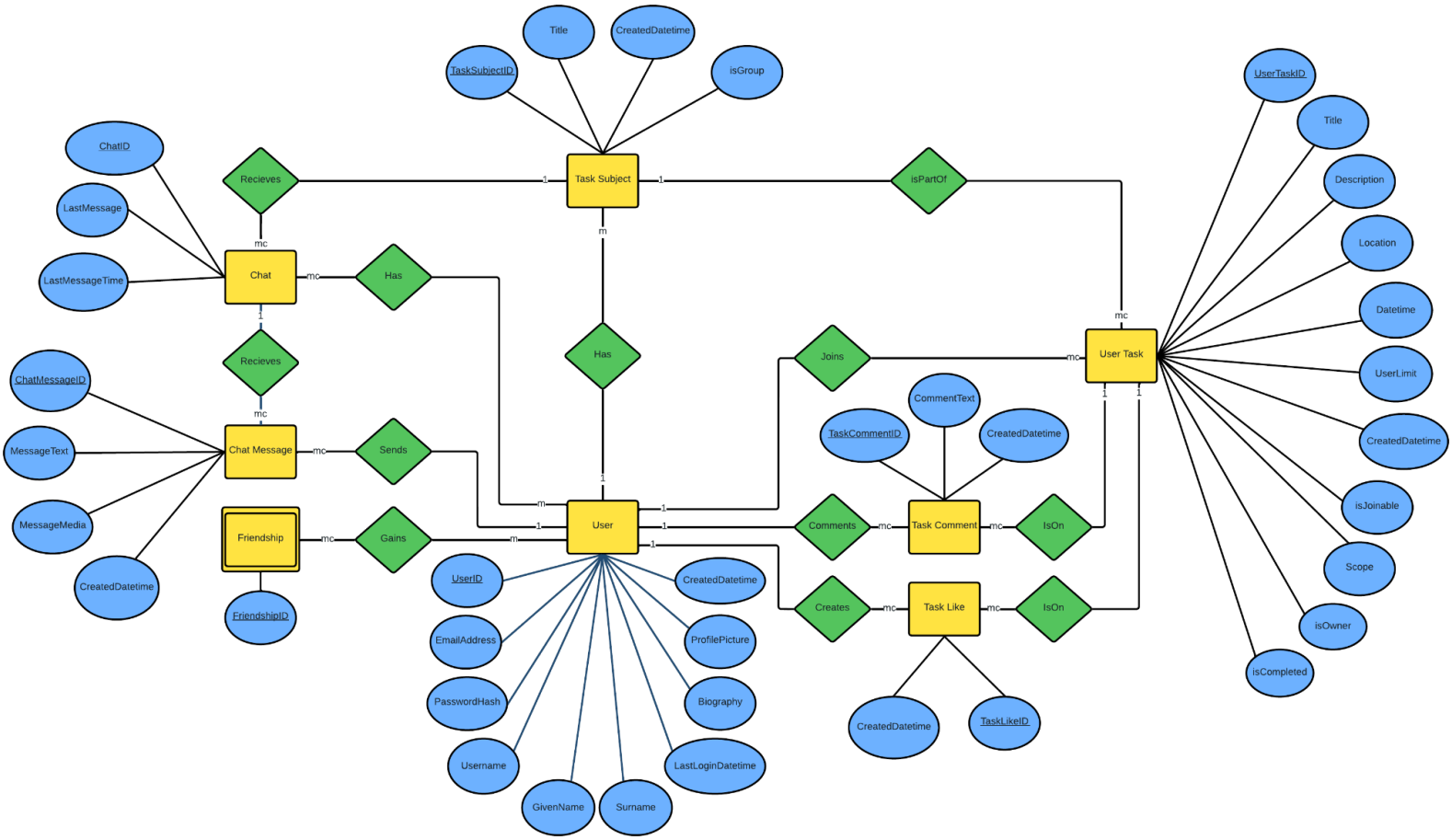


Diagram 3.1

5.1. Entities

5.1.1. **user_profile**

5.1.1.1. contains user profile's basic personal information

5.1.1.1.1. contains user profile's email

5.1.1.1.2. contains user profile's password hash

5.1.1.1.3. contains user profile's given name

5.1.1.1.4. contains user profile's surname

5.1.1.2. contains user profile's app related information

5.1.1.2.1. contains user profile's username

5.1.1.2.2. contains user profile's last login datetime

5.1.1.2.3. contains user profile's biography

5.1.1.2.4. contains user profile's profile picture

5.1.1.2.5. contains user's profile created datetime

5.1.2. **task_subject**

5.1.2.1. contains a foreign key to the user profile that owns it

5.1.2.2. contains the subject title
5.1.2.3. contains the created datetime of the subject
5.1.2.4. contains a boolean variable to signify if the subject is a group or not

5.1.3. **chat_message**
5.1.3.1. contains one and only one foreign key to the chat that is receiving the message
5.1.3.2. contains one and only one foreign key to the user profile that sent the message
5.1.3.3. contains the text of the message
5.1.3.4. contains any media that is part of the message
5.1.3.5. contains the the datetime the message was created

5.1.4. **user_task**
5.1.4.1. contains a foreign key to the profile that the task belongs to
5.1.4.2. contains a foreign key to the subject that the task belongs to
5.1.4.3. contains the title of the task
5.1.4.4. contains the description of the task
5.1.4.5. contains the location of the task
5.1.4.6. contains the datetime of the task
5.1.4.7. contains the datetime of the creation of the task
5.1.4.8. contains a boolean on if the task is open to join or not
5.1.4.9. contains the use limit of the task
5.1.4.10. contains the scope of the task which is who it is shared with
5.1.4.11. contains a boolean on if the task is completed or not
5.1.4.12. contains a boolean of if the user profile is the owner of the task or not

5.1.5. **task_like**
5.1.5.1. contains a foreign key to the task the like belongs to
5.1.5.2. contains a foreign key to the user profile who liked the task
5.1.5.3. contains the datetime the like was created

5.1.6. **task_comment**
5.1.6.1. contains a foreign key to the task the comment belongs to
5.1.6.2. contains a foreign key to the user profile that commented on the task
5.1.6.3. contains the text of the comment
5.1.6.4. contains the datetime the comment was created

5.1.7. **friendship**
5.1.7.1. contains a foreign key to the user profile that sent the friend request

5.1.7.2. contains a foreign key to the user profile that accepted the friend request

5.1.8. **chat**

5.1.8.1. contains a foreign key to the user profile that sent the message

5.1.8.2. contains a foreign key to the user profile that is receiving the message

5.1.8.3. contains the text of the message

5.1.8.4. contains any media in the message

5.1.8.5. contains the datetime the message was created

6. References

How much time do people spend on social media in 2022? Techjury. (n.d.). Retrieved November 14, 2022, from https://techjury.net/blog/time-spent-on-social-media/#gref

Log in to access the Lucid Visual Collaboration Suite. Lucid visual collaboration suite: Log in. (n.d.). Retrieved November 14, 2022, from https://lucid.app/

Time Management Statistics (new research in 2022). Timewatch. (2022, October 24). Retrieved November 12, 2022, from https://www.timewatch.com/blog/time-management-statistics-in-2022/

7. Other

Last Updated: 12/13/22