Design Document for StudyBuddies

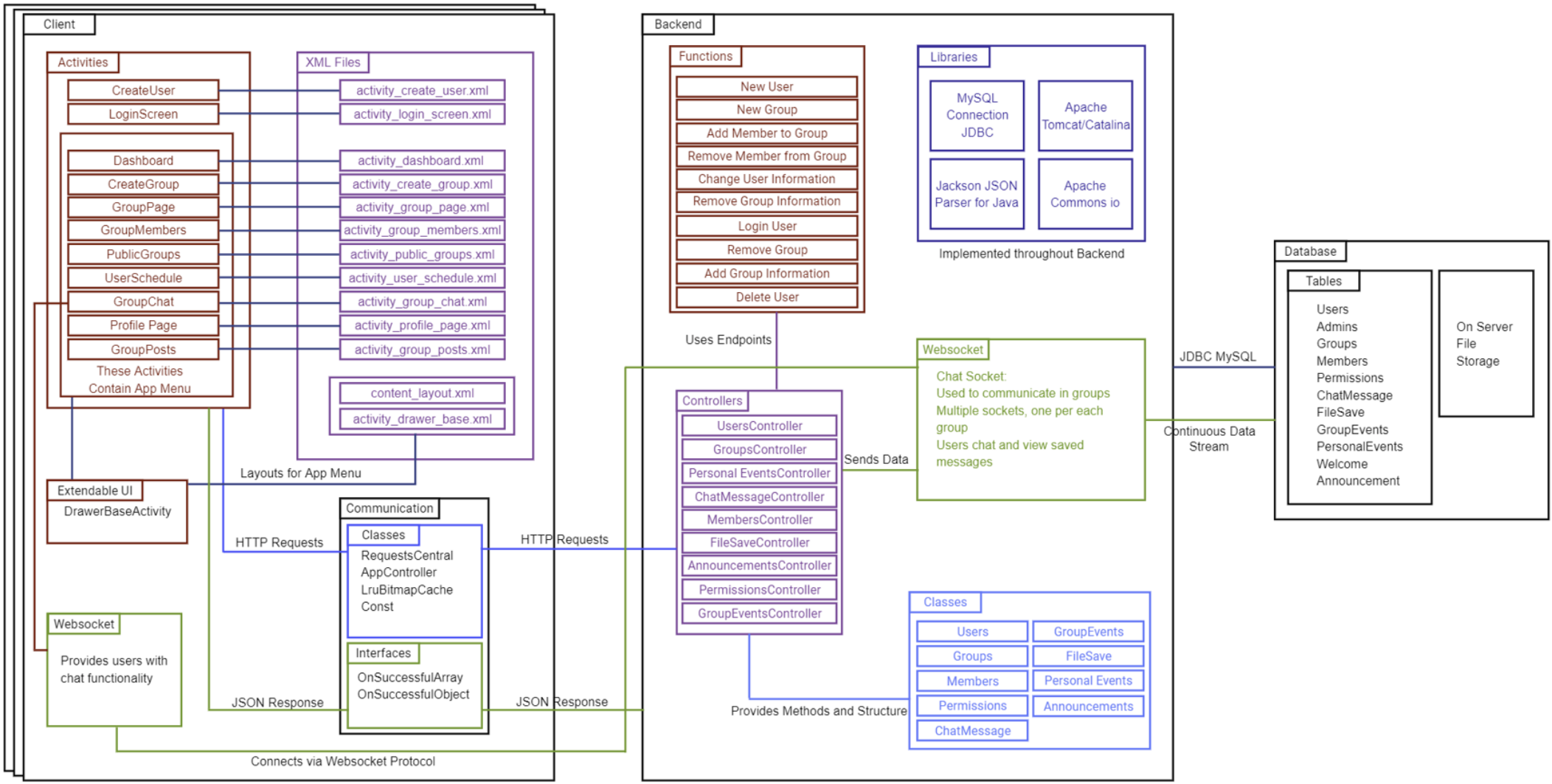
Group <2_UG_4>

Andy Bruder: % 25

Omar Muhammetkulyyev: % 25

Ryan Sand: % 25

Brady Heath: % 25


Client
Activities
CreateUser
LoginScreen
Dashboard
CreateGroup
GroupPage
GroupMembers
PublicGroups
UserSchedule
GroupChat
Profile Page
GroupPosts
These Activities
Contain App Menu
XML Files
activity_create_user.xml
activity_login_screen.xml
activity_dashboard.xml
activity_create_group.xml
activity_group_page.xml
activity_group_members.xml
activity_public_groups.xml
activity_user_schedule.xml
activity_group_chat.xml
activity_profile_page.xml
activity_group_posts.xml
content_layout.xml
activity_drawer_base.xml
Layouts for App Menu
Extendable UI
DrawerBaseActivity
Communication
Classes
RequestsCentral
AppController
LruBitmapCache
Const
Interfaces
OnSuccessfulArray
OnSuccessfulObject
HTTP Requests
HTTP Requests
JSON Response
JSON Response
Websocket
Provides users with
chat functionality
Connects via Websocket Protocol
Backend
Functions
New User
New Group
Add Member to Group
Remove Member from Group
Change User Information
Remove Group Information
Login User
Remove Group
Add Group Information
Delete User
Uses Endpoints
Controllers
UsersController
GroupsController
Personal EventsController
ChatMessageController
MembersController
FileSaveController
AnnouncementsController
PermissionsController
GroupEventsController
Sends Data
Provides Methods and Structure
Libraries
MySQL
Connection
JDBC
Apache
Tomcat/Catalina
Jackson JSON
Parser for Java
Apache
Commons io
Implemented throughout Backend
Websocket
Chat Socket:
Used to communicate in groups
Multiple sockets, one per each
group
Users chat and view saved
messages
Classes
Users
GroupEvents
Groups
FileSave
Members
Personal Events
Permissions
Announcements
ChatMessage
JDBC MySQL
Continuous Data
Stream
Database
Tables
Users
Admins
Groups
Members
Permissions
ChatMessage
FileSave
GroupEvents
PersonalEvents
Welcome
Announcement
On Server
File
Storage

Project Description

Android Requests

All HTTP requests sent from the client will make use of the class we have implemented, “RequestsCentral”. Requests central will have methods for any type of Volley Request used. RequestsCentral methods always take a URL as a parameter, any object being sent as part of a request, and an instance of an appropriate interface. These interfaces will have a method called onSuccess() to be called within the onResponse() method of a Volley Request. Whenever a RequestsCentral method is called, the onSuccess() method from the appropriate interface will need to be implemented. This will help us pass the response from the server into the activity the request is called from, allow us to keep all requests in one class, reducing code duplication, and handle the response differently depending on the purpose of the request.

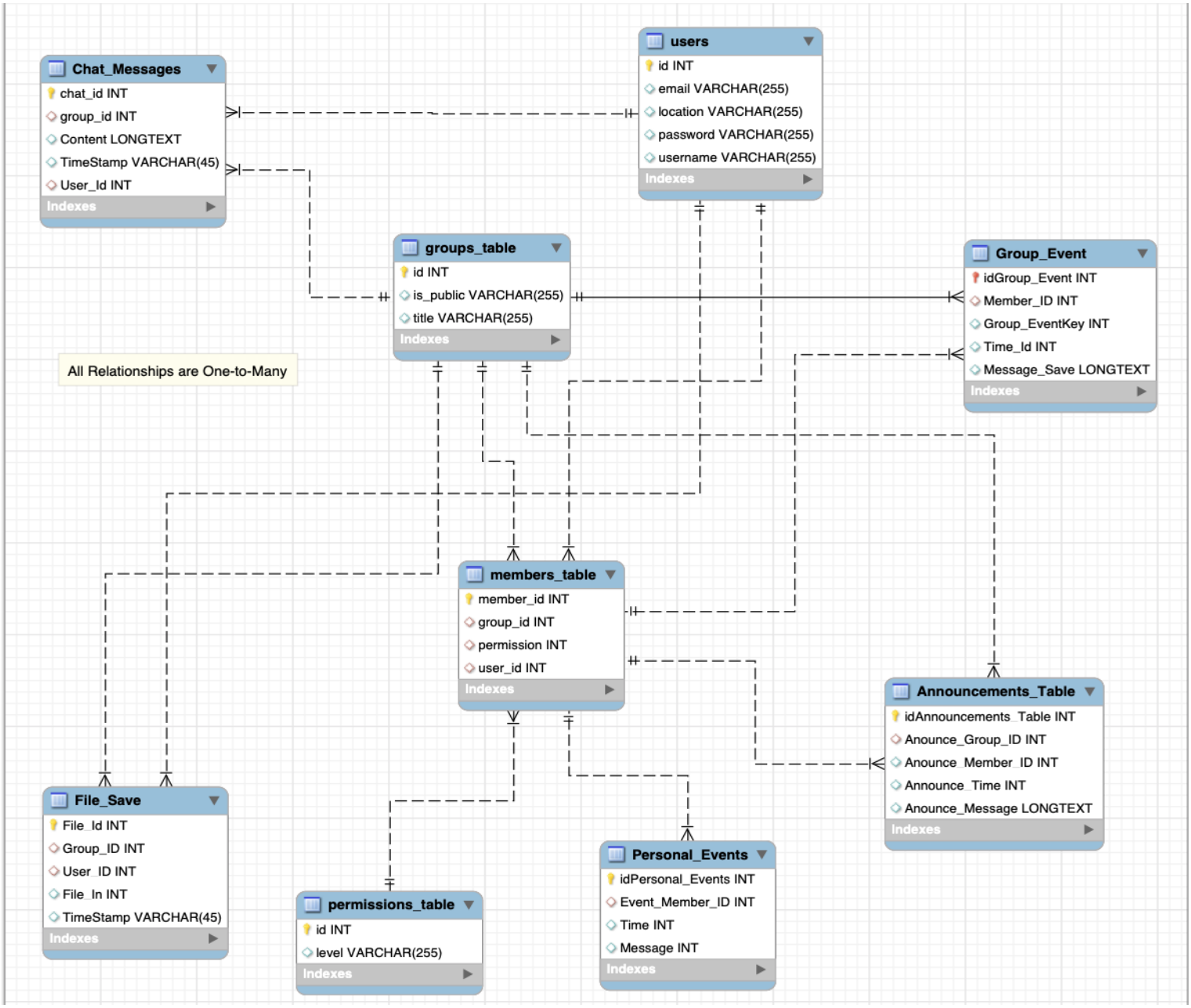
Android User GUI

The application’s front-end consists of several activity Java classes and their corresponding XML layouts. Each activity represents a screen of the application and processes most of its own logical operations. However, some of the activities that need access to a common functionality like main menu and page action bar extends the base class DrawerBaseActivity that provides such functionality. This helps keep the code clean and scalable should we decide to add more screens to our application. Having an extendable base class also makes it easy to maintain activities that do not need the functionality since we can choose which screens get the attributes. All of the activities communicate with the backend whenever they need via helper classes and interfaces as described above.

Controllers

Each of the 9 tables have the basic create, read, update, delete, and list functionality. These requests form the basis of the rest of the functionality in our project. Because all of our relationships in our database are many to one, the Controllers all have similar layouts and purposes. We have laid out the endpoints to all follow the same naming conventions to help keep the code clear and easy to understand. The format includes /class/new for the create endpoint, /class/id for the targeted read endpoint, /class/id as well to specify which instance of the class to update, /class/id again to delete instead of change, and /class/ to get a list of all of the saved instances in the database. The Android application uses these endpoints for each table to get and process the information needed to run the project. The Backend uses Repositories to save and retrieve the given classes. We are also using a Websocket system to provide a better channel for our chat feature to operate. This allows us to only open the communication line once and then send data back and forth using the WS protocol whenever we need to.

Relationships Diagram



Tables and Fields and Relationships

***All relationships listed below are ONE-to-MANY relationships**

Chat_Messages

chat_id (Primary Key)

group_id (Foreign Key linked with groups_table: id)

Content

TimeStamp

User_id (Foreign Key linked with users: id)

users

id (Primary Key)

email

location

password

username

groups_table

id (Primary Key)

is_public

title

File_Save

File_Id

Group_ID (Foreign Key linked with groups_table: id)

User_ID (Foreign Key linked with users: id)

File_In

TimeStamp

permissions_table

id (Primary Key)

level

Personal_Events

idPersonal_Events (Primary Key)

Event_Member_ID (Foreign Key linked with member_table: member_id)

Time

Message

Group_Event

idGroup_Event (Primary Key)

Member_ID (Foreign Key linked with members_table: member_ id)

Group_EventKey (Foreign Key linked with groups_table: id)

Time_ID

Message_Save

Announcements_Table

idAnnouncements_Table (Primary Key)

Anounce_Group_ID (Foreign Key linked with groups_table: id)

Anounce_Member_ID (Foreign Key linked with memberss_table: member_id)

Announce_Time

Anounce_Message

members_table

member_id (Primary Key)

group_id

permission

user_id