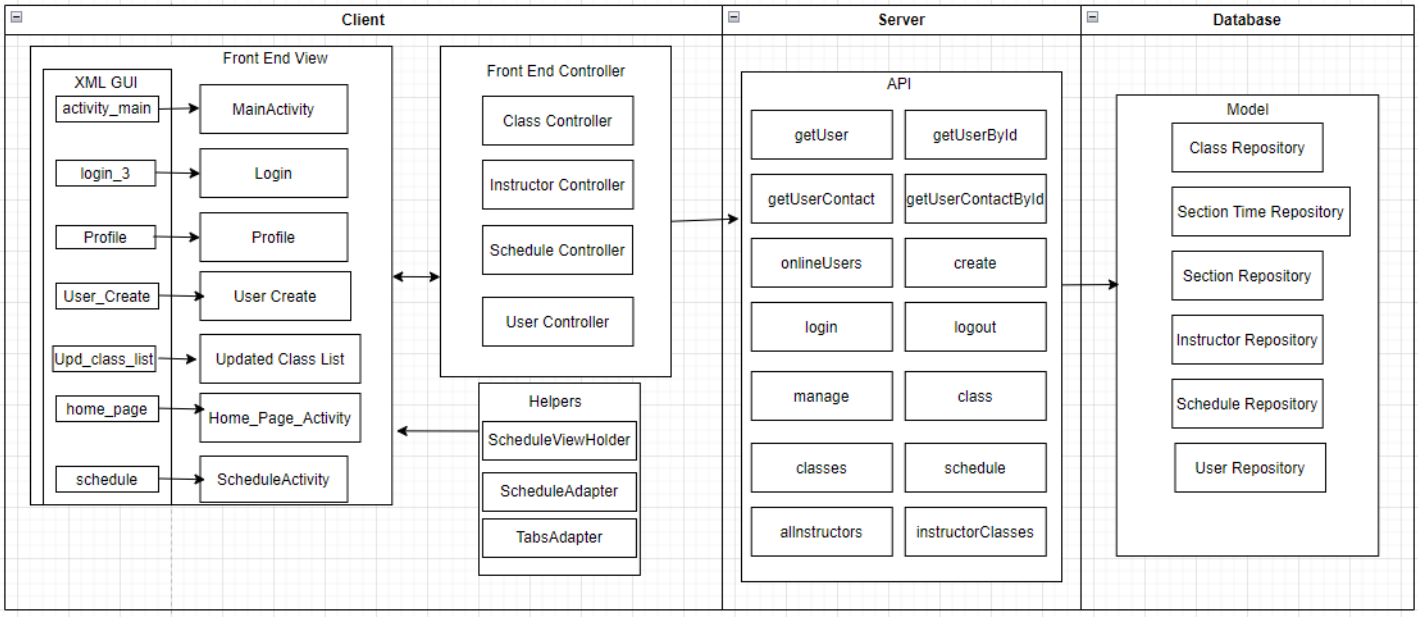
Design Document for CySchedule

Group 2_HB_4

Maxton Bromet: 33% contribution

Evan Nicely: 33% contribution

Samuel Rettig: 33% contribution


Client
Front End View
XML GUI
activity_main
MainActivity
login_3
Login
Profile
Profile
User_Create
User Create
Upd_class_list
Updated Class List
home_page
Home_Page_Activity
schedule
ScheduleActivity
Front End Controller
Class Controller
Instructor Controller
Schedule Controller
User Controller
Helpers
ScheduleViewHolder
ScheduleAdapter
TabsAdapter
Server
API
getUser
getUserById
getUserContact
getUserContactById
onlineUsers
create
login
logout
manage
class
classes
schedule
allInstructors
instructorClasses
Database
Model
Class Repository
Section Time Repository
Section Repository
Instructor Repository
Schedule Repository
User Repository

Client:

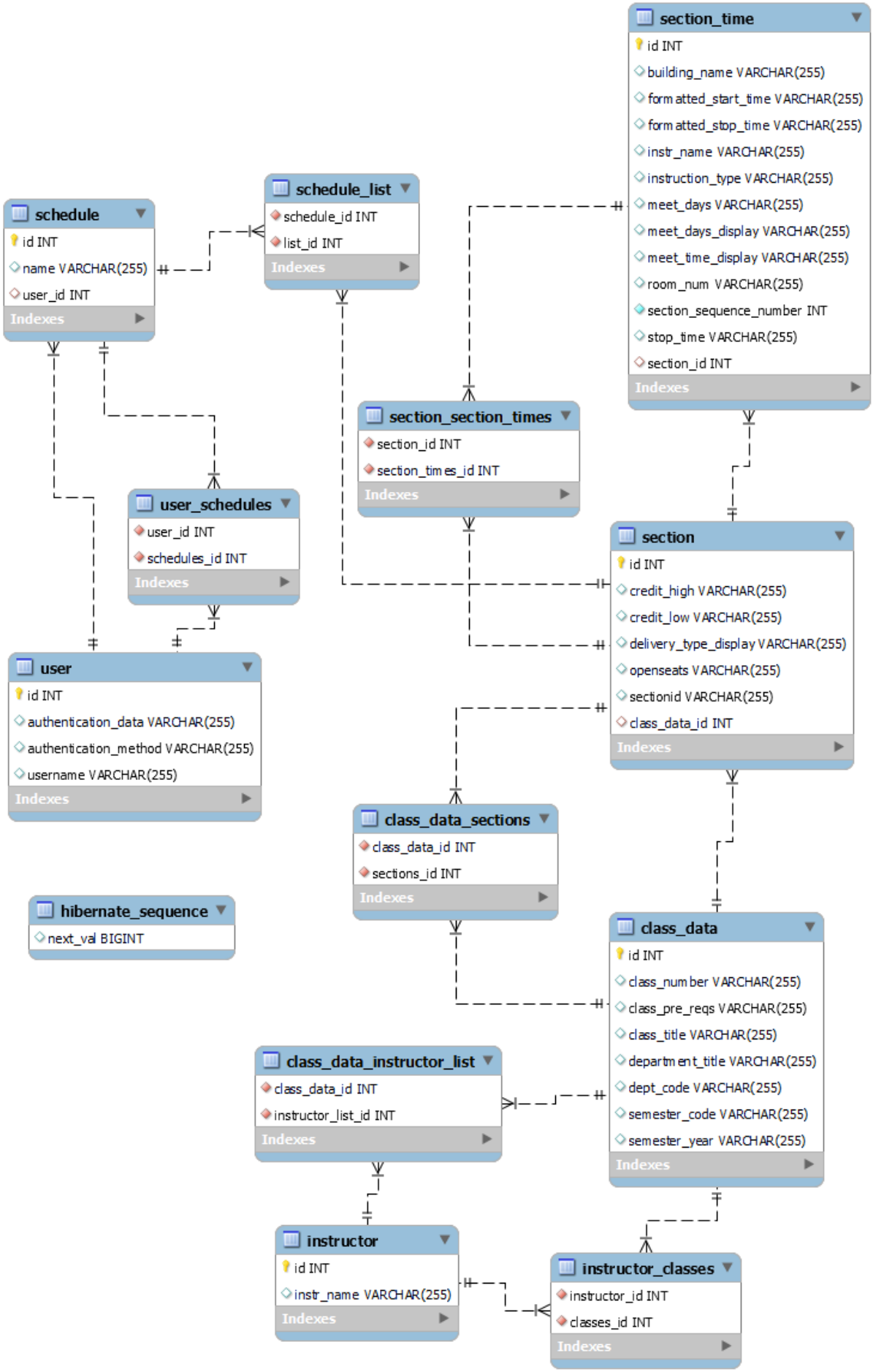
On our client side, everything starts with the Front end view. This is the aspect that the User sees upon using our application. Specifically, we use the XML GUIs in order to format the view according to our liking. The data shown to the user is controlled by the Front end controller, which acts as a way for us to send data to the server and database. Furthermore, we bolster the front end view by having helpers, such as the ScheduleViewHolder, due to the extreme complexity inherent in the screen itself. This combination effectively allows for seamless data transfer between the database, server, and view that the user sees in real time.

Server:

On our back-end is an extensive list of calls to the database. For example, getUserByID allows us to get a user by their unique ID (stored in the database). Similar examples can be held for login and create, which respectively allow for us to verify if a user exists, as well as add to the database. This is done by a combination of get and post methods. This covers nearly every aspect of the calls to the database, giving us a clean and easy way to update the on-screen information. We use Rest Api calls in our database.

Database:

The database is the back-bone of our project. It holds all of the data that we depend upon in order to show classes, create users, give users the ability to log in, and more. For example, the Section repository breaks down classes into specific sections that the user would be able to choose. This is further broken down into times per section, represented by the Section Time Repository. Without this aspect, we could not effectively show accurate data to users.


section_time
id INT
building_name VARCHAR(255)
formatted_start_time VARCHAR(255)
formatted_stop_time VARCHAR(255)
instr_name VARCHAR(255)
instruction_type VARCHAR(255)
meet_days VARCHAR(255)
meet_days_display VARCHAR(255)
meet_time_display VARCHAR(255)
room_num VARCHAR(255)
section_sequence_number INT
stop_time VARCHAR(255)
section_id INT
Indexes
schedule_list
schedule_id INT
list_id INT
Indexes
schedule
id INT
name VARCHAR(255)
user_id INT
Indexes
section_section_times
section_id INT
section_times_id INT
Indexes
user_schedules
user_id INT
schedules_id INT
Indexes
section
id INT
credit_high VARCHAR(255)
credit_low VARCHAR(255)
delivery_type_display VARCHAR(255)
openseats VARCHAR(255)
sectionid VARCHAR(255)
class_data_id INT
Indexes
user
id INT
authentication_data VARCHAR(255)
authentication_method VARCHAR(255)
username VARCHAR(255)
Indexes
class_data_sections
class_data_id INT
sections_id INT
Indexes
hibernate_sequence
next_val BIGINT
class_data
id INT
class_number VARCHAR(255)
class_pre_reqs VARCHAR(255)
class_title VARCHAR(255)
department_title VARCHAR(255)
dept_code VARCHAR(255)
semester_code VARCHAR(255)
semester_year VARCHAR(255)
Indexes
class_data_instructor_list
class_data_id INT
instructor_list_id INT
Indexes
instructor
id INT
instr_name VARCHAR(255)
Indexes
instructor_classes
instructor_id INT
classes_id INT
Indexes