**The Roommate App**

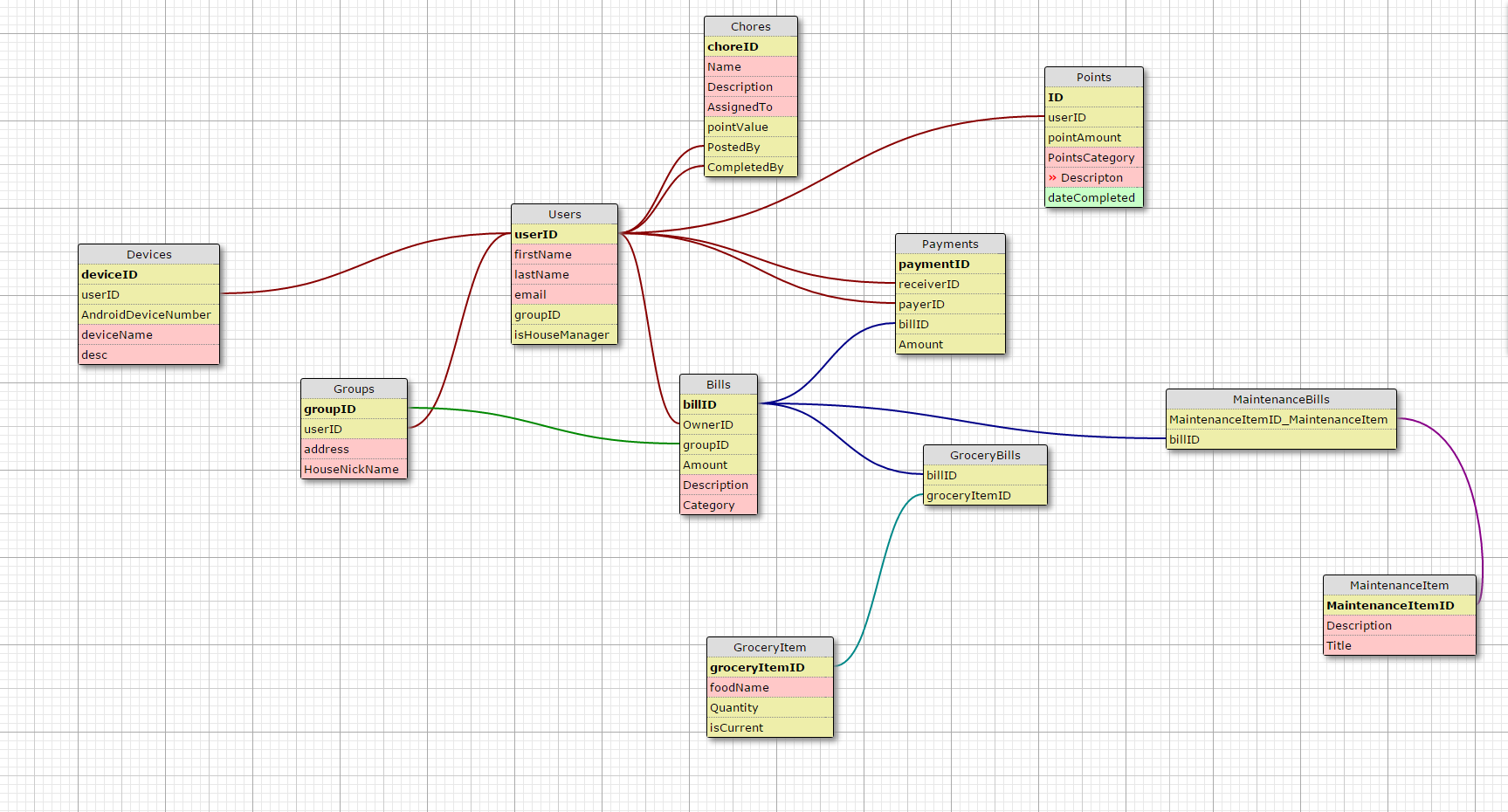
**an Android App by Albert Rynkiewicz, Alex Greco, and Matt Cieslak**

**Design Summary**

This Android app will be coded in Java using the ide, Android Studio. The project structure will be the standard Android Studio structure and we will be using fragments to layout the UI. Users are going to have to access the same data from a centralized source. To do this, we will have a SQL server hosted on Matt’s digital ocean account. We will have a php interface to access the sql server. We chose a php data access layer because we already have experience doing this and it will be easiest for us to make progress across the application. To process user to user payments, we will be using the Venmo API. For source control, we will be using GitHub, and we will be setting up the GitHub plugin within Android Studio. The application is modular by nature, so we plan on developing it in a modular fashion.

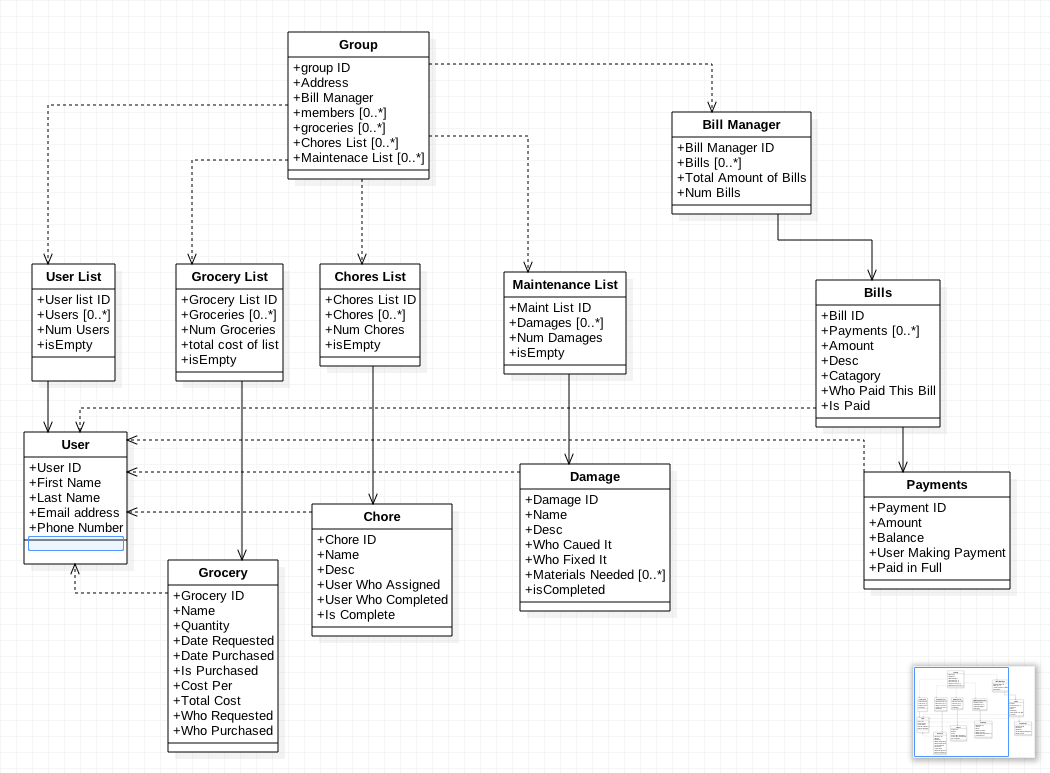
**Database Design**

The database will exhibit the following structure:

We envision a database with this structure will encapsulate all the data we need, when we need it. The main identifying table here is the User Table. The application is user driven; the other database objects all relate to a user in some way. Another key note, is the structure of the bills. Each grocery bill, for example, can have multiple Grocery Items and the same goes for Maintenance Bills and Maintenance Items. The way we designed the database will allow for the most straightforward access to all the data. This will be important for the implementation of the application because there will be heavy database usage, almost everything is going to be stored in it, so we are going for as intuitive of an approach as possible.

**Object Diagram**

The following is a UML diagram that outlines the basic structure of our application:



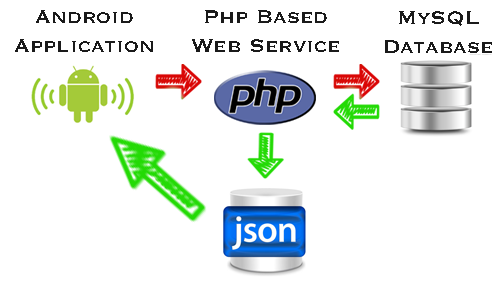
This UML diagram is very similar to the database design. There will be a class for each group. That group will be made up of user objects. The group will also be home to its own grocery list, chores list, maintenance list and bill manager. All of these list will contain information about themselves and will be filled with respective objects. Each of the objects in any of the list will be tied to 1 or more users. This is because if you have a chore, there needs to be a reference to who assigned the chore and who is supposed to complete it. This is the same for all the different object.

This is the structure of the application we have deliberated on and decided it is the best for our initial purpose. Of course, as with any software, we may come across a

a better way to achieve our goal. For the purpose of the initial development of this application, we believe this a suitable design

**Backend Design**

After thorough research of the Play framework we decided that we will indeed use php to handle database transactions on the backend. Some of the reasons include lack of support in the framework for critical functionality required by our app, for example, because the framework is relatively new its support for interacting with http requests from a mobile source and executing a high volume of SQL queries is said to be suboptimal or require verbose work-arounds. Other reasons include the fact that Play is designed to work with asynchronous I/O, making code unmaintabley messy considering java lacks features that typically keeps async code consistent like closures. A lot of existing Java libraries are synchronous/blocking, so it often means that you have to be careful with which ones you use in an async environment devoid of blocking code execution. In addition to that it would slow down time of development as we would have to adapt to the idiosyncrasies of the framework itself. Since nearly every bit of functionality will require the ability to talk to the server and perform database transactions, writing a web service in php would be advantageous to our efforts. The implementation is rather simple. the php handles POST and GET http request initiated by our app and simply returns cleanly formatted json.



The php scripts exist on the server and more or less acts like an API. We make a GET request, for example, to the script on the server from our app like so

HttpClient httpclient = new DefaultHttpClient();

HttpResponse response = httpclient.execute(new HttpGet(URL));

subsequently using output stream readers to process the json response and handle the data accordingly. conversely posting data via http is just as simple

HttpClient httpclient = new DefaultHttpClient();

HttpPost post = new HttpPost("http://server-ip-address/database-script.php");

**Frontend Design**

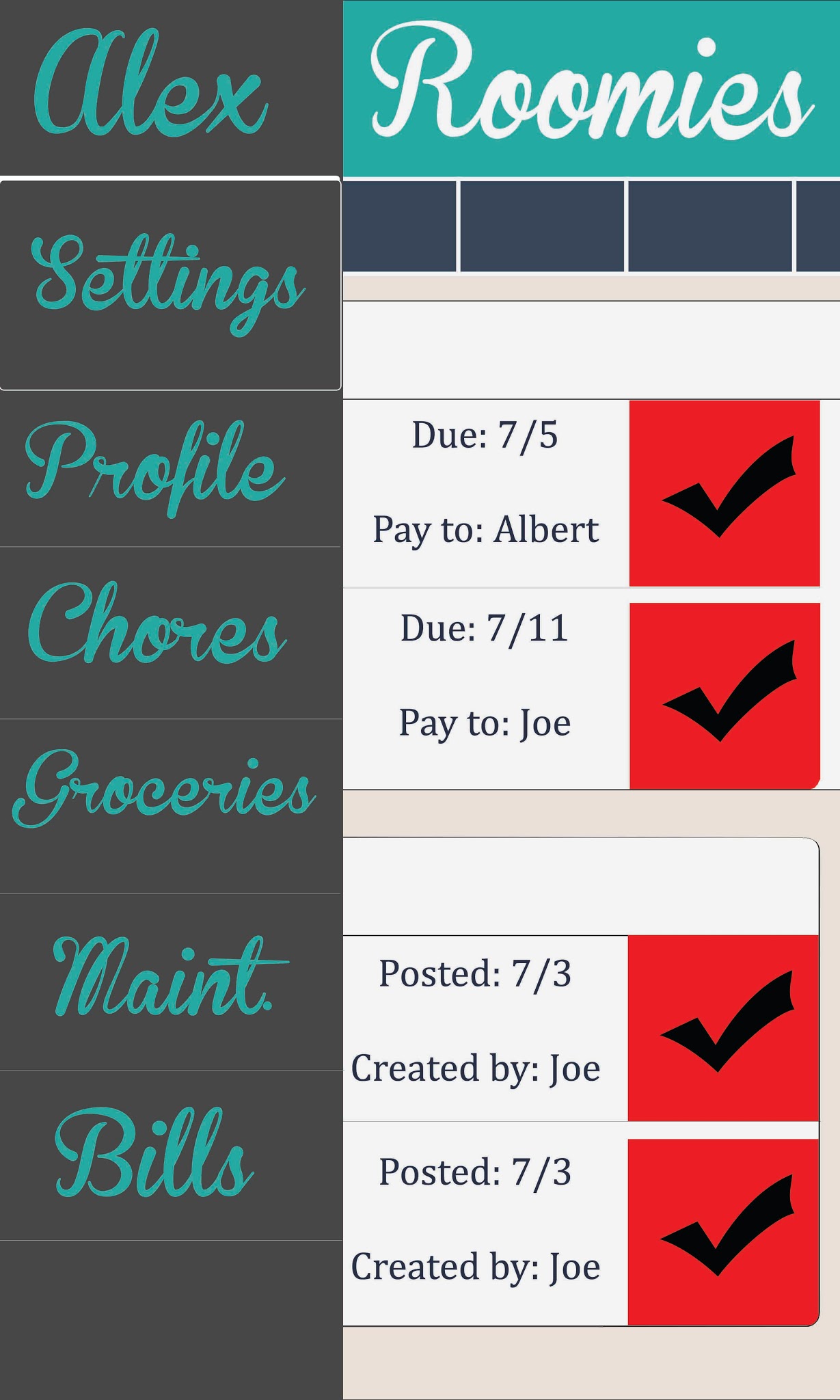
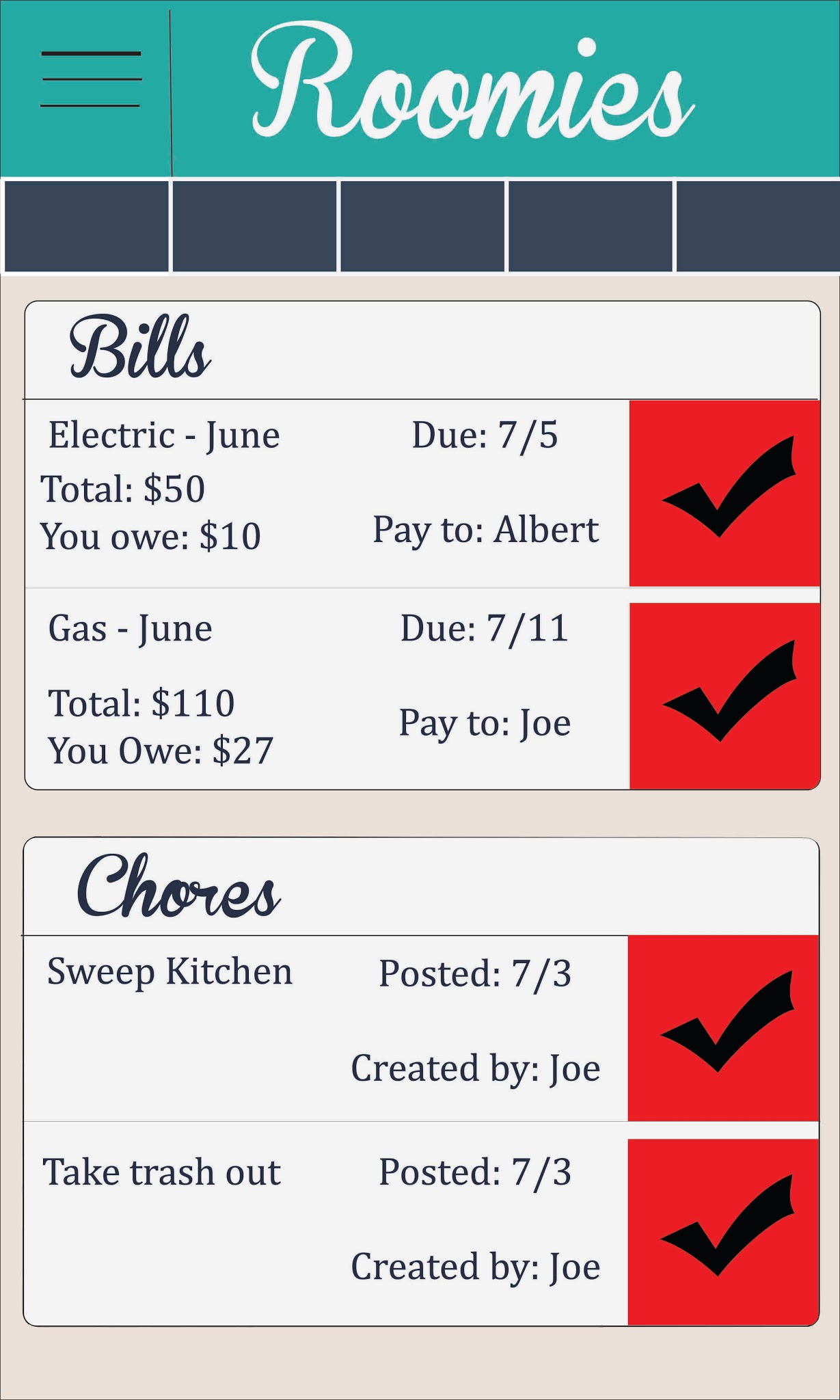
The goal and theme that we are trying to achieve with our front end design is simplicity. We want our front end to be very intuitive to any user trying to use it. We want to keep the experience of our app simple and clean, while still having robust functionality. When we were planning the design of this app, we all looked into the front end design of our favorite or most used apps. They all have something in common, their design shows the user what they need to see and saves extraneous information for different specialized views. We plan to follow the same type of design. We want our main screens to show just enough information to inform the user of what they need, but not add too much information so that the user is looking at 12 different things at the same time.

The first step to designing our front-end was to look out how the object diagram and database were structured. We wanted to make the front-end sleek and simple, but we had to make sure that we could still include all of our functionality. So we thought what would be the best way to incorporate the right information in the right way. We decided that having an activity for each of our main features would be the best approach. We also decided that having a main “Home” screen would be needed as well. This will be used to show the user a quick glance at updates and activity from the roommates.

The next thing we did was draw everything out on a whiteboard. This is a crucial step in the design process because it is an easy way to express your ideas with a committee of people. It also allowed us to view several different options before deciding which one would best meet our needs.

The last part was to photoshop a rough mock of what we want the app to look like. These were not intended as actual representations of what the final product will look like but rather a good guideline for what appearance we are aiming for. As you can see from figure 3 below, we have decided that the traditional list type view would be the best for our application. This image is a mock of what the home screen will look like. It will have the apps name across the top with a menu selector on the top-left corner. The blue buttons that are not filled in will be buttons that switch between the home screen and chores list, grocery list, etc. The white rectangles in the center of the screen will be a quick feed from all of the other categories in the app. This screen will be scrollable downward to reveal all of the different categories. The red check marks will be buttons that the user can click when that task is completed. Figure 4 is the home screen with a fragment view for all settings and profile management options.

Figure 3. Home Screen Figure 4. Home Screen with Settings Fragment



Each of the main elements of the app, including the chores list, bill manager, grocery list, etc., will have their own screen. Figure 5 illustrates what the bills list will look like. It will follow the typical list style view. The red box with the check mark will be a button the user can push to confirm they paid the bill or push it to open a transaction between themselves and another user. The question mark will open a modal with information about the bill. All of the other windows for the other type of activities will follow a similar format.

Figure 5. Chores List

