Overview:

For the minimum viable product for our group payment web application, we will have a few users and use cases that we need to focus on. We have two types of users on our app. The first is the organizer, who organizes a transaction and shares the link to others in their party. The second type is the link receiver, who receives the link to join the group. Both of our users would interact with our app to make splitting bills easier and in real time.

Parts of the App:

We will need to build a form that is used by the organizer. Fields on this form include the total amount, the number of people to split a bill with, a checkbox that indicates whether the bill is split evenly, and the price per person, which is disabled when the split evenly checkbox is marked. Once the organizer submits the form, it generates a link that everyone should go to and select their name and pay. The organizer will have a page that tracks user agreements, and once everyone has paid, the organizer will have access to the one-time use card for payment.

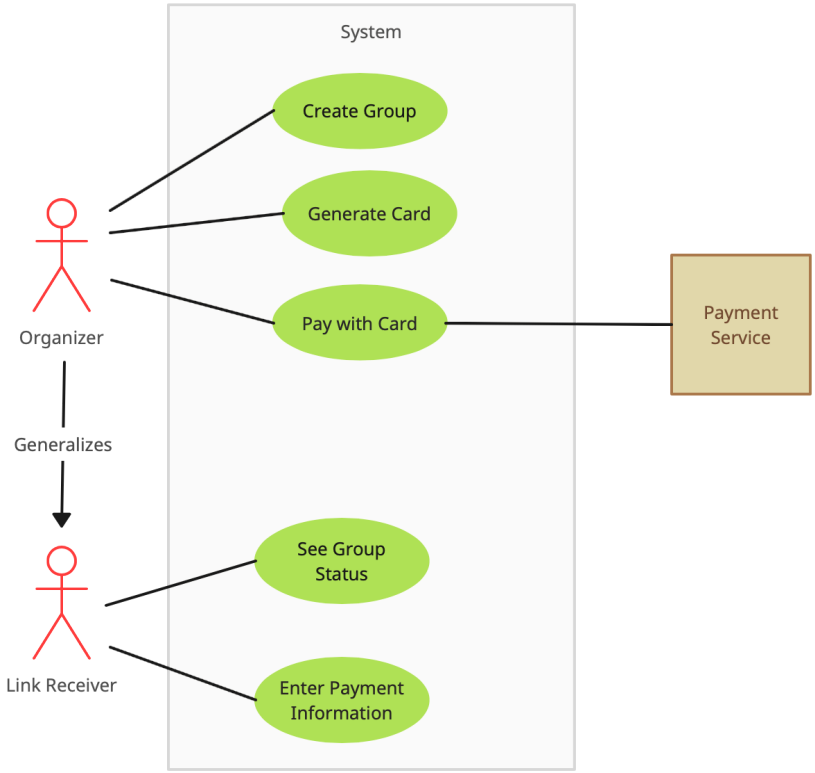

Figure 1: Use-case Diagram

Figma Mocks and Architecture:

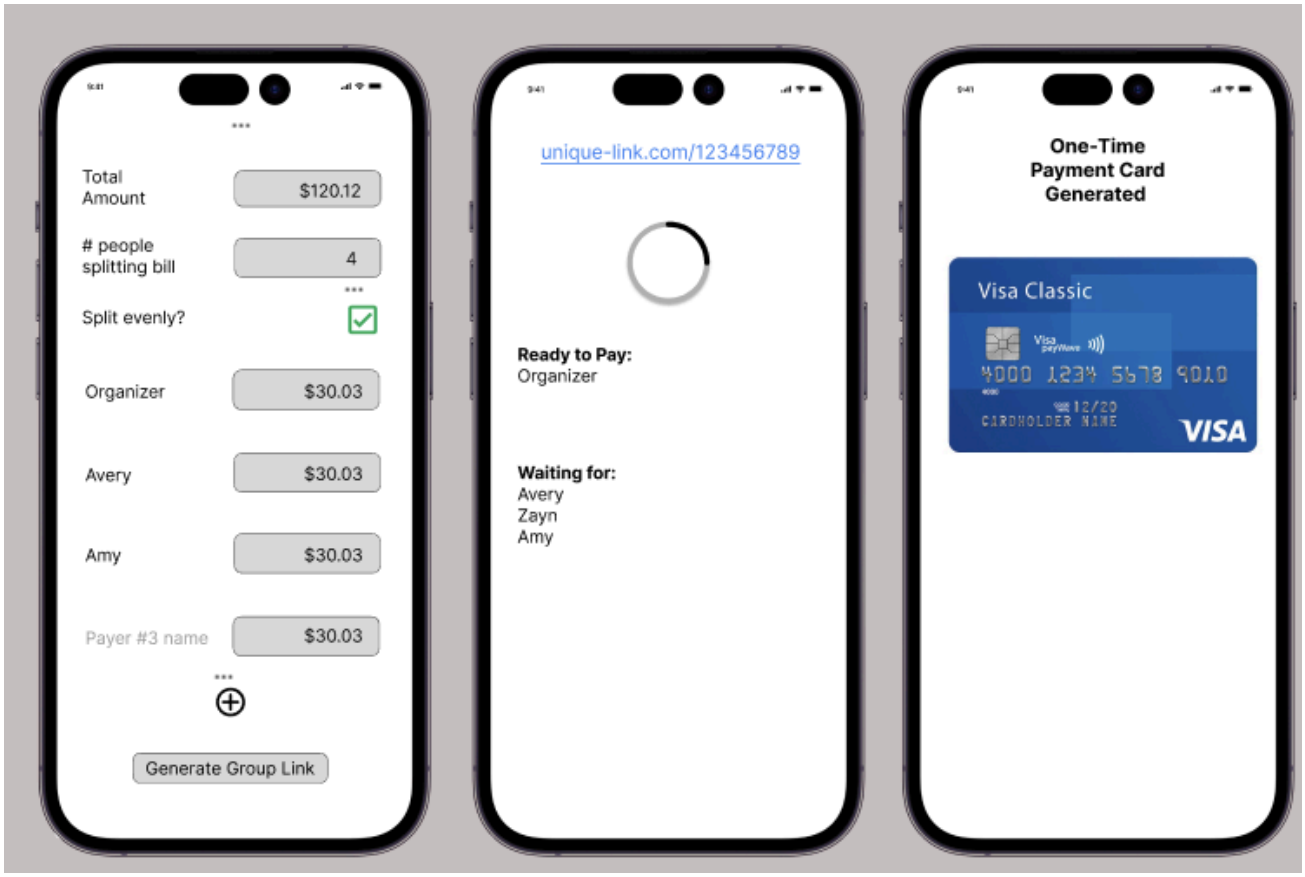

Figure 2: Figma Frames for the Organizer

The starting page for the organizer will be a form that, once filled out and submitted, will generate a unique link to be shared with all members of the group. Then the organizer will be brought to a page showing what members have agreed to the payment and have entered the card details. When all the members have agreed and entered their payment, a one-time card payment will be generated for the organizer which they can use to pay. Once done, we will remove the stored information for this transaction.

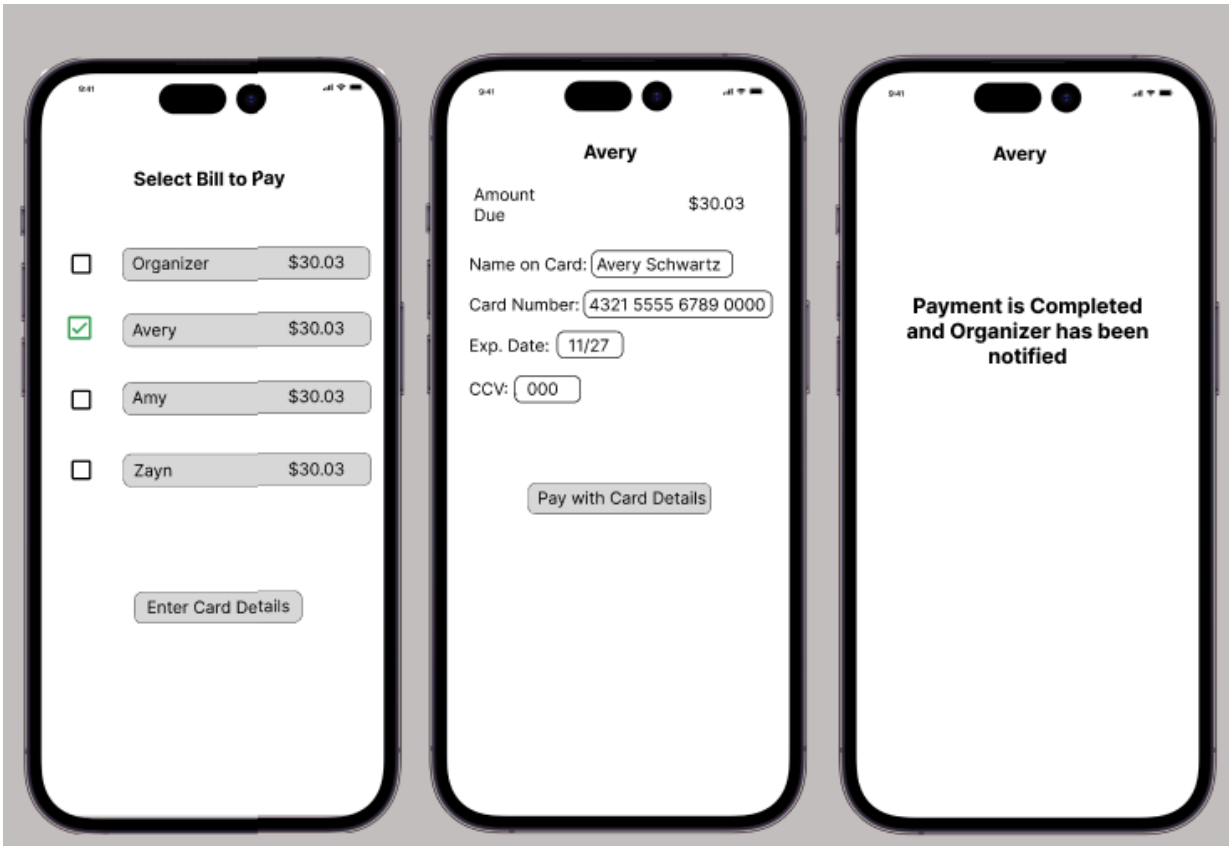

Figure 3: Figma Frames for the Link Receivers

Everyone will use the uniquely generated link to be able to check how the bill is being split. They select their name after seeing the amount they should pay which serves as user agreement. They can then proceed to put in their card details and once it is entered, it will instantly update on the organizer's screen when they are done.

Tech Stack:

We will have a frontend written in Next.js and React. Our backend will be created with Django. We will store transaction data in an embedded SQLite database and use GraphQL to fetch it. We are using Next.js because we used it for our first mini project and it allows us to use server-side rendering. We considered just using normal React since we're more comfortable using it but it doesn't really use server-side rendering, which we will require. Next.js also has a low ramp-up time since all teammates are familiar with it since we already used it in our first mini-project. We're using Django because we prefer Python over having to deal with the Java Runtime Environment. Our team is also much more familiar with Python than Java. We know SQLite is

included with Django by default so it makes sense to use it to store our data. We are choosing to use GraphQL because we want to utilize the subscription feature in order to have real-time updates in our application.

Branching strategy

We want to use a branching strategy similar to GitHub-Flow. Every time we complete a task, we create a new branch to commit to, and then make a pull request. We review it then merge it into the main and close that branch.