Project Proposal

Introduction to the document

Your project proposal will be in four sections.

You may choose to present it as a document or a slide deck

Section 1: Project Definition

Name of the project

Shared Expense Manager Bot for Messenger

Members of the team:

- Salvador Hernandez
  - An aspiring computer scientist with an emphasis in software engineering and interested in mobile development.
- Anna Pomelov
  - A computer science major with a concentration in software engineering and an interest in mobile development.
- Joshua Smith

    - A computer science student hacking my way through school.

Project Mission / Motivation

- A common inconvenience among friends, coworkers, and roommates is tracking shared bills and expenses to ensure all parties get paid back. Bots are increasingly being built into popular apps, including Facebook Messenger, to assist users in various tasks. We propose to build a Shared Expense Manager Bot specific for Facebook Messenger. We will apply for our bot to go live in Facebook Messenger.

Project Objective

- *We will be working with our mentor Tala Huhe, over the duration of the semester in order to deliver a Shared Expense Manager Bot that will allow friends, roommates, and coworkers to easily track, pay, split, and request payments on the Facebook Messenger platform.*

Project Approach

We will have several features included in our Messenger Bot. Users will be able to make direct payments to friends, requests payments from friends, request evenly and unevenly split payments from friends, view who owes them, view who they owe, and view their past history of payments.

We will start out by getting a basic bot running by using Facebook's APIs that is capable of relaying simple messages from user to user. In order to do this we will follow Facebook's documentation about getting a basic Messenger Bot up and running. We will be using Heroku to deploy our code and we will constantly be pushing to Github. Once we have a simple bot that is capable of relaying messages from one user to another, we will then enable persistence for our bot by using PostGreSQL. Next we will create a way for admins to view the data easily to make testing our bot more convenient.

Once our bot can send messages from one user to another, we will ten explore our options for enabling our bot to make payments. We will consider the internal Facebook API for completing payments or Paypal. Once we choose which payment service to utilize, we will implement the simple direct payment feature within our bot.

After this we will expand the commands our bot is capable of responding to. We will enable a user to request a payment from another user. We will account for the case in which a user is not yet on the platform and get them authenticated.

Project Goals

- Receive approval by Facebook for our bot to go live in the Messenger app itself.
- Have a fully functioning shared expense management bot that will enable users to manage and split shared expenses between others.

Project Deliverables

- *At the end of the semester we will produce the following deliverables:*
    - *A fully functioning messenger bot for the Facebook Messenger platform with which users can*
        - *Make direct payments to friends*
        - *Request payments from friends*
        - *Request payments to multiple people at once for one shared expense*
        - *View all the users they owe money to*
        - *View all the users that owe them money*
        - *View all of their past payments and past requests*

Dependencies / Risks

- *Some risks we foresee include:*
    - *Potentially not being able to use the Facebook payment service for our bot to process payments. We may have to integrate Paypal or some other payment service*
    - *Not getting approved to have our bot go live in the Messenger app*
    - *Having trouble integrating persistence and a payment service into our code*
    - *Communication*
    - *Testing - creating sufficient test coverage of the app*
    - *Reviewing teammate code to prevent bugs*
    - *Scalability - creating features that can support an increasing amount of users*

Project Milestones

- *Monday March 6th*
    - *Create a Messenger bot that is capable of sending messages from one user to another*
    - *Implement persistence with PostGreSQL*
    - *Create admin page to improve testing*

Section 2: Use cases

Use Case One

User that owes money to friend pays friend through messenger bot

Primary Actor

User who owes someone money

Scope

Payment Bot

Level

Summary

Story

- *User opens Shared Expense Manager Bot*
- *User types “Pay”*
- *Bot responds with an interface to allow user to choose which friend they wish to pay*
- *User selects friend*
- *Bot asks for amount of payment*
- *User types amount*
- *Bot asks for a confirmation*
- *User confirms payment*
- *Payment gets sent*
- *Recipient gets notified that they have been paid*

Use Case Two

User requests a payment from a friend through messenger bot

Primary Actor

User who is owed money by one friend

Secondary Actor

User who owes a friend money

Scope

Payment Bot

Level

Summary

Story

- *User opens Shared Expense Manager Bot*
- *User types “Request payment” or “request”*
- *Bot responds with an interface to allow user to choose which friend they wish to request a payment from*
- *User selects friend*
- *Bot asks for amount of payment request*
- *User types amount*
- *Bot asks for a confirmation*
- *User confirms request*
- *Request gets sent to other user*
- *Recipient gets notified that they have been requested to pay the given amount*

- *Requested user can*
    - *Pay right away*
        - *Goes through payment use case*
    - *Edit payment*
        - *User chooses edit payment*
        - *Bot asks for the edited amount they wish to pay*
        - *User confirms*
        - *Payment gets sent*
        - *Other user is notified*
    - *Deny payment*
        - *User selects deny payment*
        - *Bot asks for confirmation*
        - *User confirms*
        - *Other user is notified that the request was denied*

Use Case Three

User that is owed money requests payments from multiple people for one shared expense (ex. check splitting for shared expense)

Primary Actor

User who is owed money by multiple people for one expense

Scope

Payment Bot

Level

Summary

Story

- *User opens Shared Expense Manager Bot*
- *User types "Split" or "Request Split payment"*
- *Bot asks how many friends the user wishes to split the bill with*
- *User types N*
- *Bot asks if user wishes to split evenly or unevenly*
- *User chooses "evenly"*
    - *Bot responds with an interface to allow user to choose which friends they wish to pay*
        - *With each friend the user enters, the user will input the amount each friend owes*
    - *User confirms friends and amounts*
    - *Requests get sent to friends*
- *User chooses "unevenly"*

    - *Bot responds with an interface to allow user to choose which friends they wish to pay*
    - *User chooses n number of friends*
    - *Bot asks to confirm*
    - *User confirms*
    - *Bot sends requests*

Use Case One

User wishes to see the people they owe, people who owe them, and their payment history

Primary Actor
User who wishes to see who they owe and who owes them
Scope
Payment Bot
Level
Summary
Story

- *User opens Shared Expense Manager Bot*
- *User types "people who owe me"*
    - *Bot responds with a list of people who owe the user currently*
- *User types "people who I owe"*
    - *Bot responds with a list of people who the user owes currently*
- *User types "history"*
    - *Bot responds with a list of payments the user has requested and payments the user has made to others*

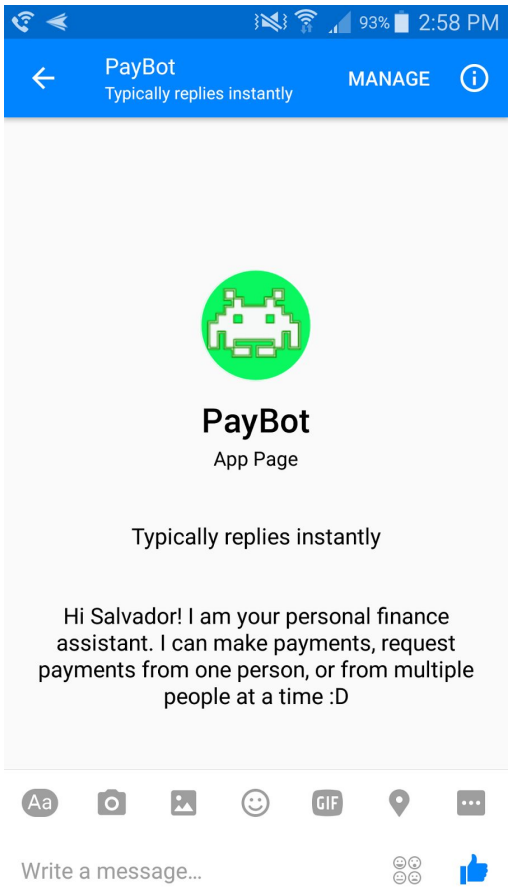
Section 3: Acceptance criteria

User should be able to interact with the bot in a variety of ways.
Users should be able to make successful payments with the bot
Users should be able to request payments through the bot
Users should be able to request multiple payments from several people
Users should be able to view their history, people they owe, and people who owe them

---

Section 4: Mocks (sketch)

The mocks show the basic setup for the three main features the MessengerBot will have



Screenshot of what the bot looks like when first opening it.

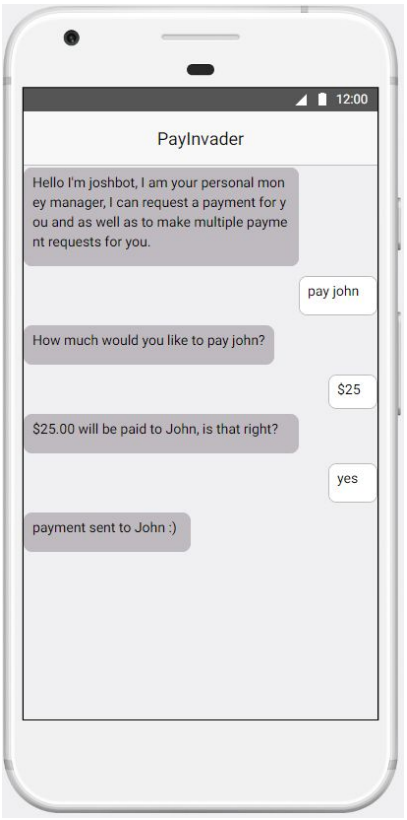


Figure 1.

Figure 1 shows a sample conversation that the user would have when the user wants to **send a payment** to someone.

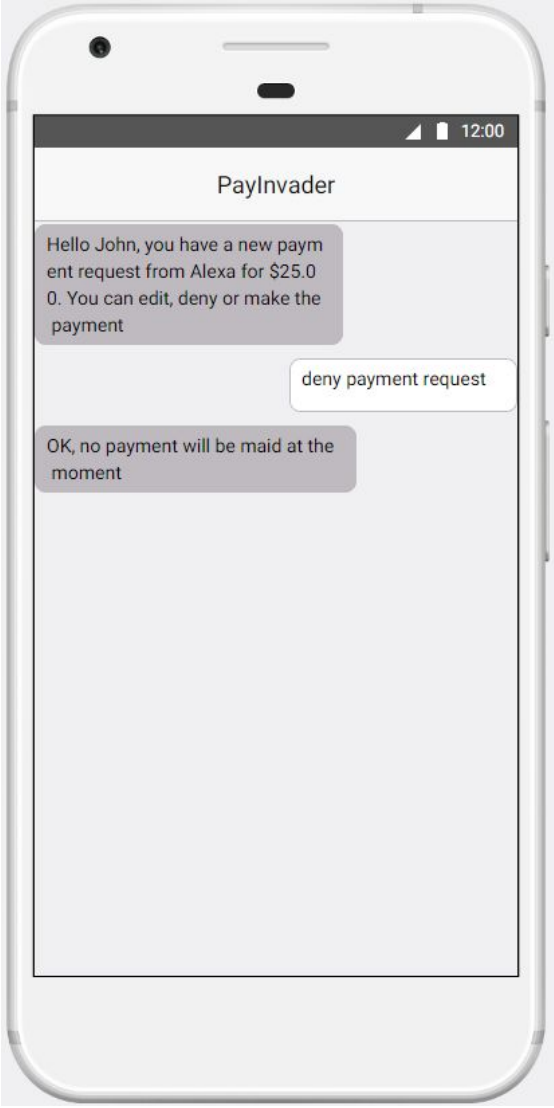


Figure 2

Figure 2 shows one of the flows that would occur when the user receives a **payment request**. In this particular case, the user denied the payment being requested.

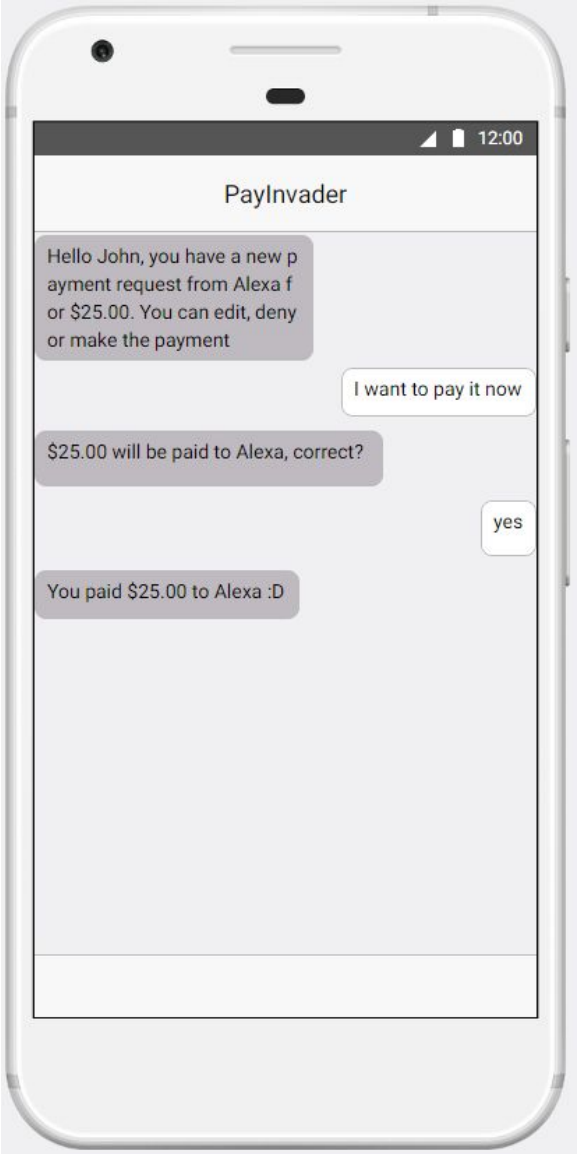


Figure 3

Figure 3 shows one of the flows that would occur when the user receives a **payment request**. In this particular case, the user **accepted to pay the requested payment**.

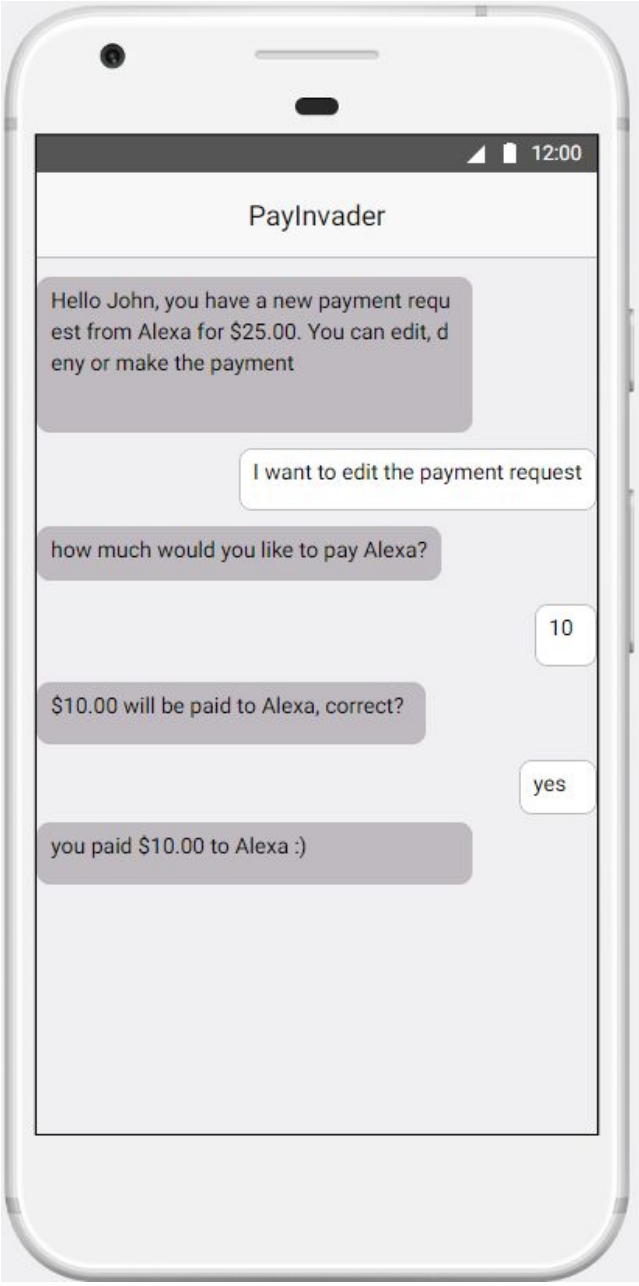


Figure 4

Figure 4 shows one of the flows that would occur when the user receives a **payment request**. In this particular case, the user decided to **edit the payment amount** to be made.

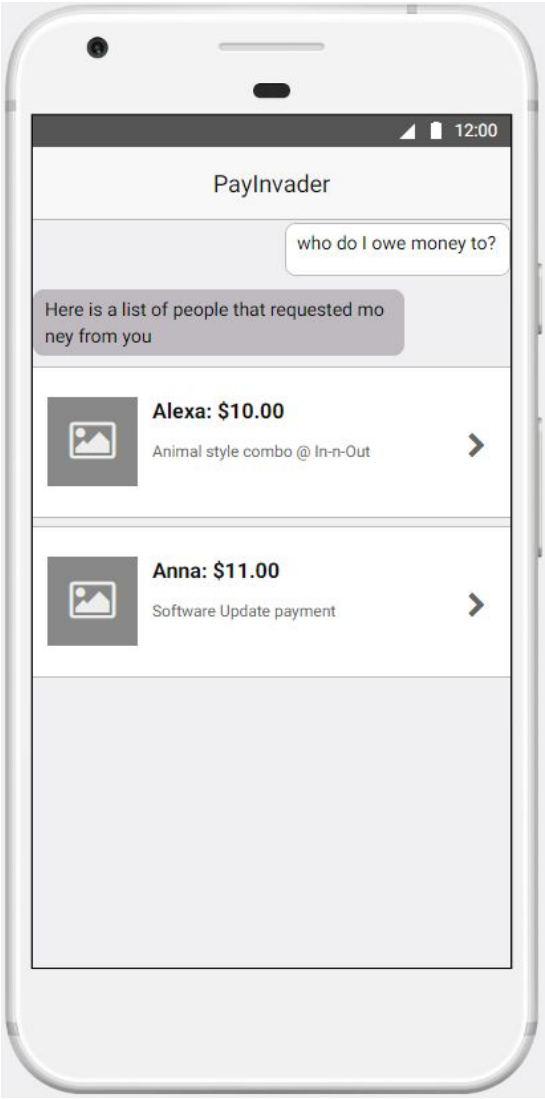


Figure 5

Figure 5 shows what happens when the user wants to **see the payment requests they have**.

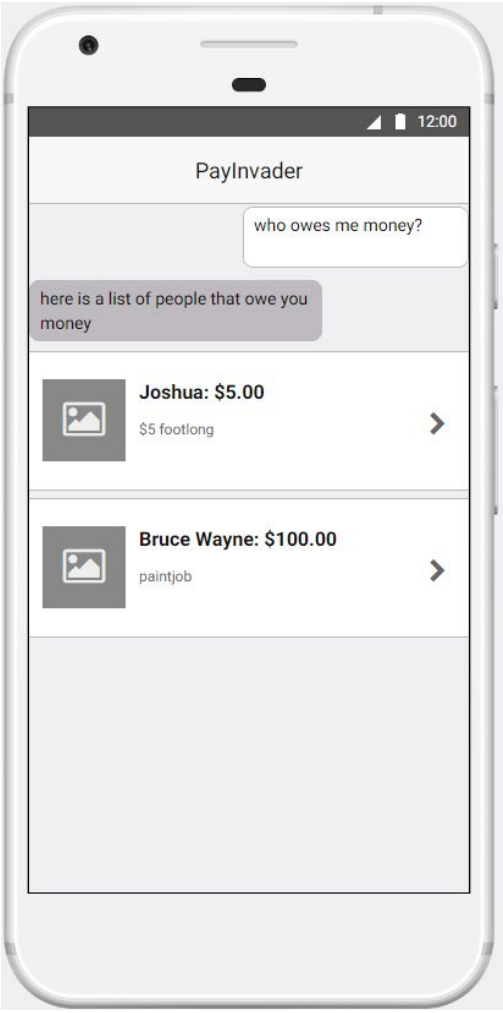


Figure 6

Figure 6 shows what happens when the user wants to **see the payment requests they have made**.