FlashFund

Crowdfunding for the University Community

Mahesh Gaire

Ahmed Alsaleh

Skylar Beaty

Ian Kusner

Jimmy Mertival

Project:

Our project is to create a crowdfunding application specifically for students and other university community members who are in need of financial assistance.

Purpose:

* Student organizations require funding.
* Some students require financial help affording school supplies like laptops, software, etc.
* Traditional methods of funding like grants, scholarships, and loans may be too timely or logistically burdensome

Customers and Users:

* Kent State University
  + Administrator for our application
  + Approves/denies requests for funding
  + Requires easy to use UI
  + Given a filtered list of applications to approve/deny
* Students and Student Organizations
  + Makes applications for funding
  + Customizes their funding pages
  + Requires a student email address to be able to access funding
* Donators
  + Anyone is able to donate
  + Uses an digital wallet service to donate

Other Considerations:

* Our website should be built so that it can be easily navigated by screen reading software
* The list of applications should be filtered before given to the admin. A full list of applications will quickly make the site unusable.
* Our website must be able to handle at the very least 1000 users, with ~250 of them being concurrent.

Technical:

* Web application
* Built using HTML and PHP, with an SQL database
* Front end design implemented using the React library, version 17.0.1
* Integration of one digital wallet service.
* Features two types of account, student and donor, allotted automatically.
* Further updates with extra features will be rolled out once the first working build is completed.
* Will be worked on using the Agile development process.

Assumptions and Risks:

* Assumption: Universities will be able to adopt this application
  + We plan to make the app very lightweight and require no maintenance on the university’s end.
  + University here only serves as the administrator if they want to. They do not need to filter through the list of applications, or do maintenance on the servers.
  + The website must be easy to navigate and use, and should not require any training to do so.
* Risk: Legal issues might be involved with a crowdfunding application working closely with a public university.
* Risk: Crowdfunding funds not used for the poster’s intended purpose.
* Risk: Network security

Stakeholder Interviews

Steven M.

Former Event Organizer, Paper Aviation

*How hard was it to get Paper Aviation funded?*

“Not hard, really. We mostly just needed a room for us to use. I thought it would be nice to eventually get t-shirts made or something but the university never came through with the funding. We actually lost the room eventually too.”

*What features would you like to see in a crowdfunding application for university clubs?*

“I want it to be able to customize my page. I want to use it to get funding and attract more club members at the same time”

*Have you personally used crowdfunding apps before?*

“I used Kickstarter before.”

*How can we make it easy for people to donate using our app? What are some roadblocks that will personally prevent you from donating?*

“I can’t think of any. Maybe a problem with payment? I know I use PayPal a lot because it’s easy to log in and use, and some sites don’t take it.”

*Tell me, if you can, a horror story you’ve heard about a crowdfunding service. Can you think of a time where a crowdfunding endeavor went horribly wrong?*

“Yeah, I remember that thing with Star Citizen where people wanted to get their money back because the game was delayed over and over. The guy making it wouldn’t do it and people were trying to sue him.”

Emily B.

Kent Boxing Club

*How hard was it to get the boxing club funded?*

“It was difficult. We have to resort to club dues to take care of the small stuff, but the big problem was getting a space for us to practice. We always tried to get the combative studio in the rec, but they told us “large groups” were not allowed to sign up for studio time unless we pay for it. I tried to get the university to pay for it, since it’s the university’s rec center, you know, but they never did anything.”

*What features would you like to see in a crowdfunding application for university clubs?*

“I’m not sure. I guess I want it so it's easy to use and donate with and it will be seen by a lot of people. That way the clubs might get funded. Yeah, I want an app with a lot of traffic.”

*Have you personally used crowdfunding apps before?*

“Not really. I’ve used like those donation things before on Facebook where you just set it up and post it, but I’ve never used the big crowdfunding apps.”

*How can we make it easy for people to donate using our app? What are some roadblocks that will personally prevent you from donating?*

“Well, I want people to know about the app, so I guess my biggest concern is people actually using it. It’ll be easier to use if the word is out. I probably won’t bother setting up a funding page for the club if no one’s on there.”

*Tell me, if you can, a horror story you’ve heard about a crowdfunding service. Can you think of a time where a crowdfunding endeavor went horribly wrong?*

“I can’t think of a specific time. My roommate once shown me this video where people Kickstarter spent a bunch of money on this board game and the company making it just never made the board game. It was like, a lot of people were angry and Kickstarter couldn’t just get the money back to them.”

We also have one unrecorded interview with Katharine G. from Kent University Center for Student Improvement.

Takeaways from the interviews,

* Minimal affiliation with the university
  + This is for legal reasons. We must take care of the money side of things ourselves
  + The system administrator roles can still be built with the university in mind. This is just so university staff may administrate should they wish to.
* Ease of use
  + Application must be easy to use for all users involved
  + Donors are less likely to donate should the application not accept their desired form of payment
  + Students are more likely to request funding if the interface is intuitive and easy to navigate.

System Overview

Acronyms/Abbreviations

|  |  |
| --- | --- |
| Acronym | Meaning |
| HTTP | Hypertext Transfer Protocol |
| REST | Representational State Transfer |
| FF | FlashFund |
| API | Application Programming Interface |
| Stripe | Payment processing API |
| DWS | Digital Wallet Service |
| FF\_DB | Flash Fund Database |
| FF\_Central | Server for FlashFund |
| DPS | Digital Payment Service |
| AJAX | Asynchronous JavaScript and XML |
| SPA | Single Page Application |

FlashFund users are divided into three account types,

Doner accounts can be created with any valid email address.

Student accounts are made with a valid student email address.

Admin accounts are added directly into the system database.

*The student account creation process right now only takes into account a .edu extension within the email address. This means that former university students with a .edu account can still make a Student account and make fundraising requests. This is something we will look into remedying, as it causes clog, but since it is not particularly a security issue, it is low on the priority list.*

System Environment

Database: MySQL 8.0.19

Server: localhost

Frontend framework: ReactJS 17.0.2

Version Control: Github: <https://github.com/Capstone-Project-team/FlashFund>

Development Server: json-server 0.16.3

Apple Pay, version 10.14.6

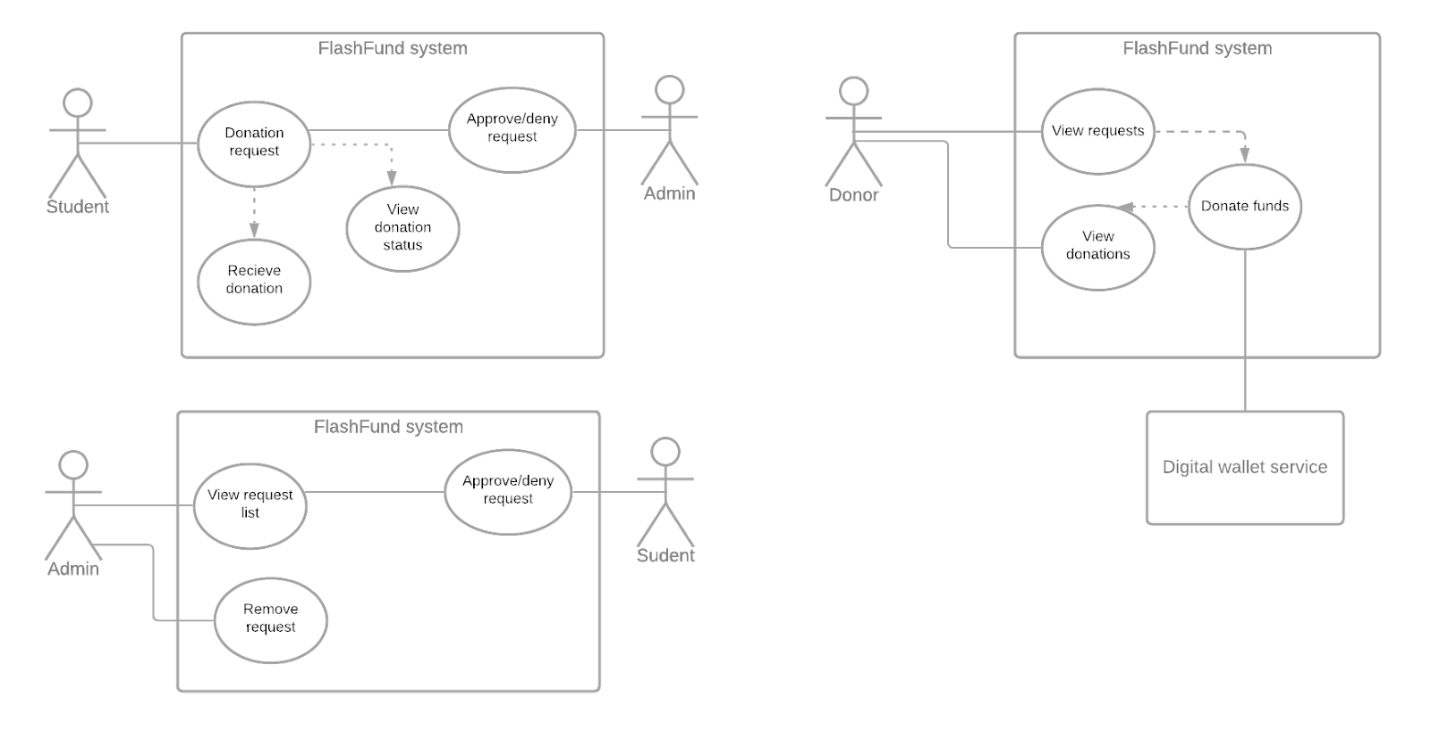
Stripe API, version 2020-08-27

Diagram

Description automatically generated

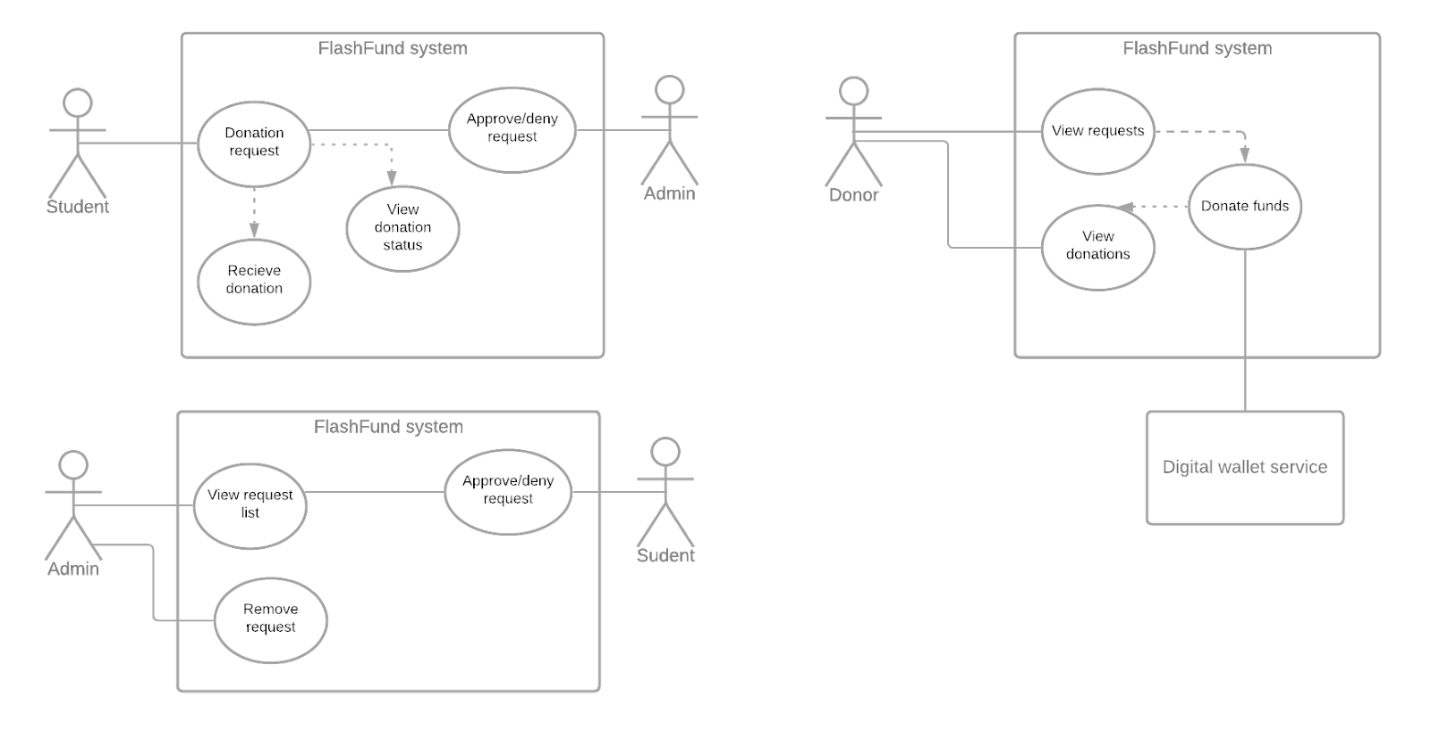
*Figure 1*

In the system overview, figure 1, Student users are the ones that create campaigns. Admins approve or deny those campaigns. Donors use the digital payment services to donate to the campaigns.



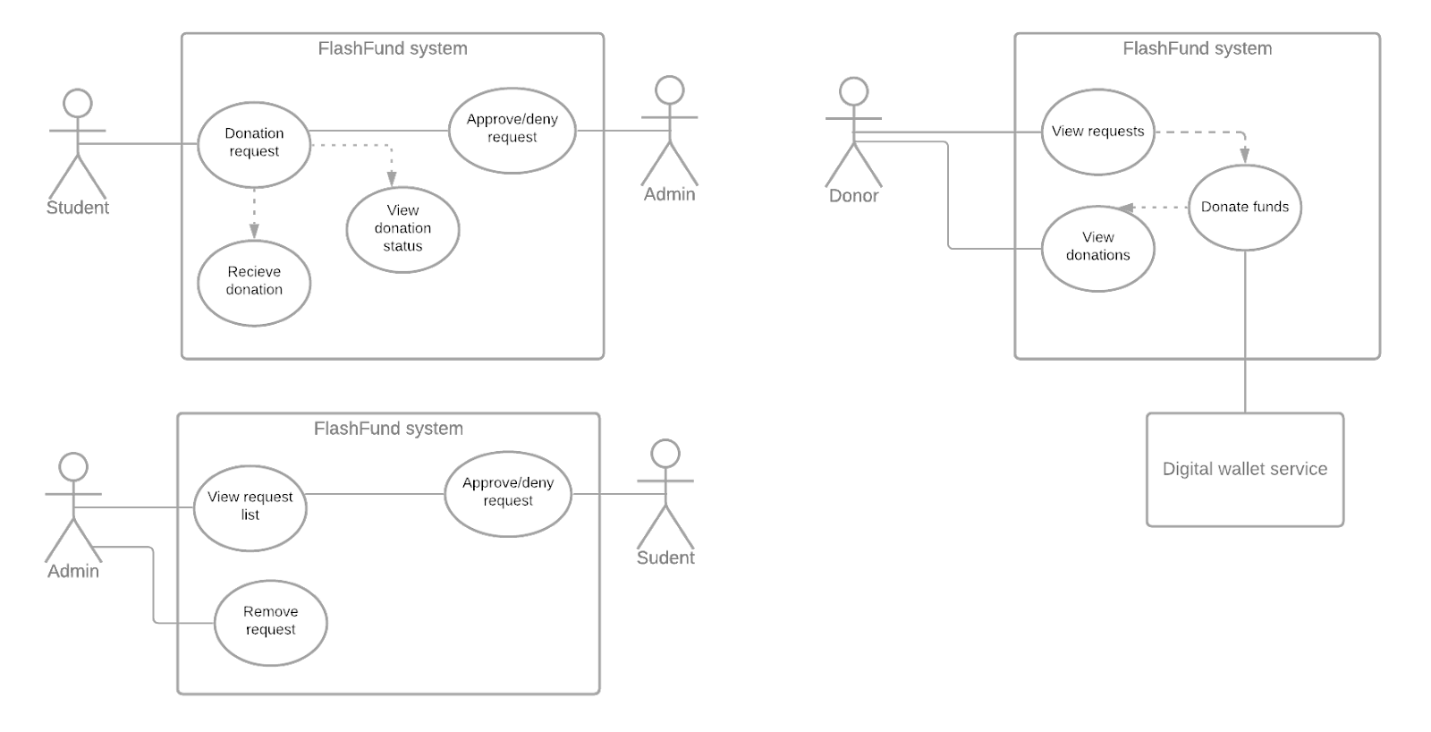
*Figure 2.1*

Administrator interactions, Figure 2.1, consists of being able to view a filtered list of requests and approve or deny requests from that list. Administrators are also able to remove fundraisers that are ongoing or finished from the FlashFund database. Administrators do not have the ability to donate or make requests for funding.



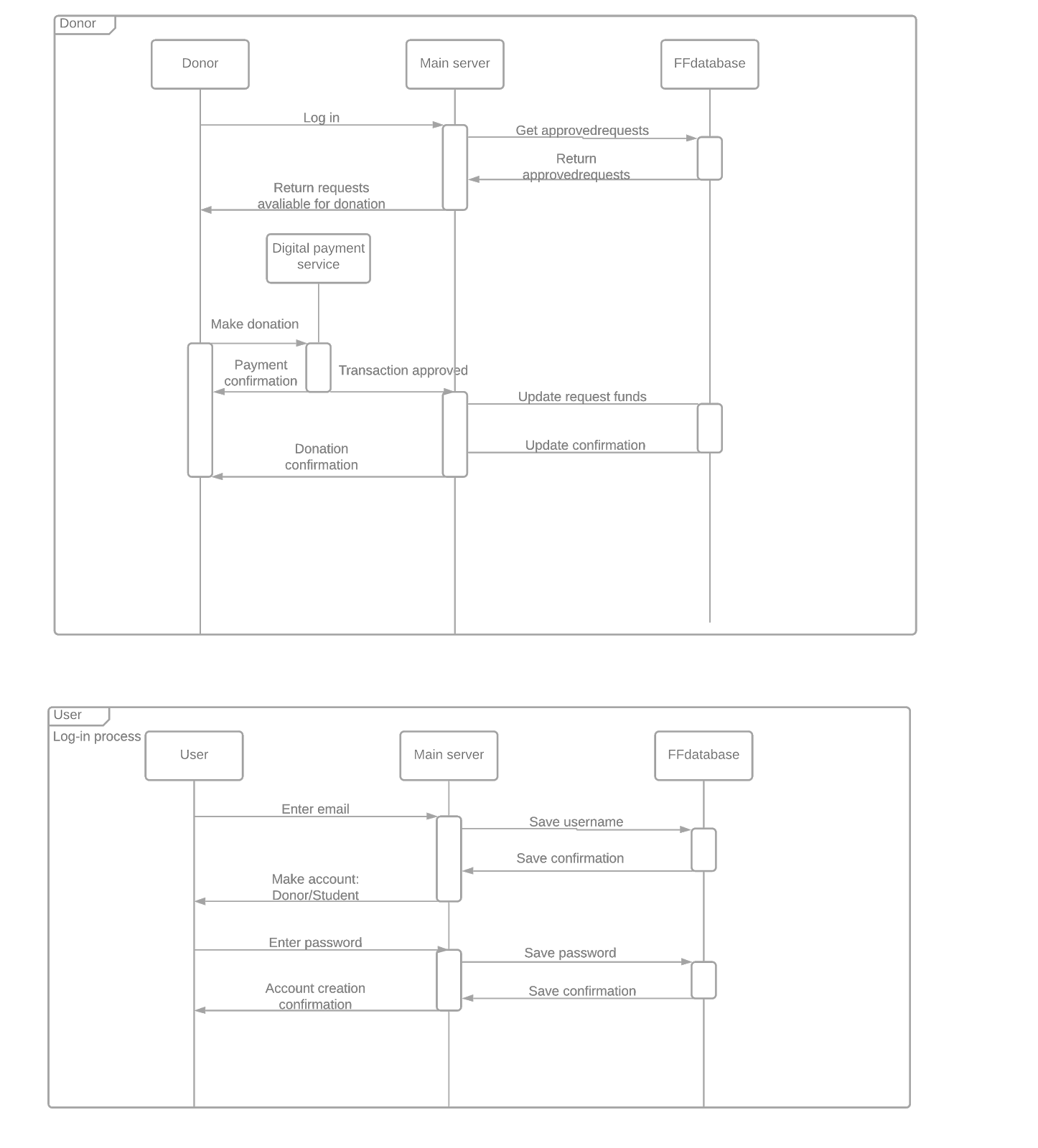
*Figure 2.2*

Donor interaction, Figure 2.2, consists of being able to view the approved ongoing requests, donating to those requests, and viewing past donations made. Donations are made with a digital wallet service. PayPal and GooglePay are currently being considered as well as other “suite” payment options such as Stripe.



*Figure 2.3*

Student interaction, Figure 2.3, consists of being able to make requests for funding and seeing the status of ongoing funding requests. If the request is denied, approved, in the donation phase, or finished, the user with the student account will be able to view the status. Once the fundraiser is finished, the user is able to receive the raised funds. Student accounts also have all the privileges of Donor accounts. They are able to donate to fundraisers just like donors.

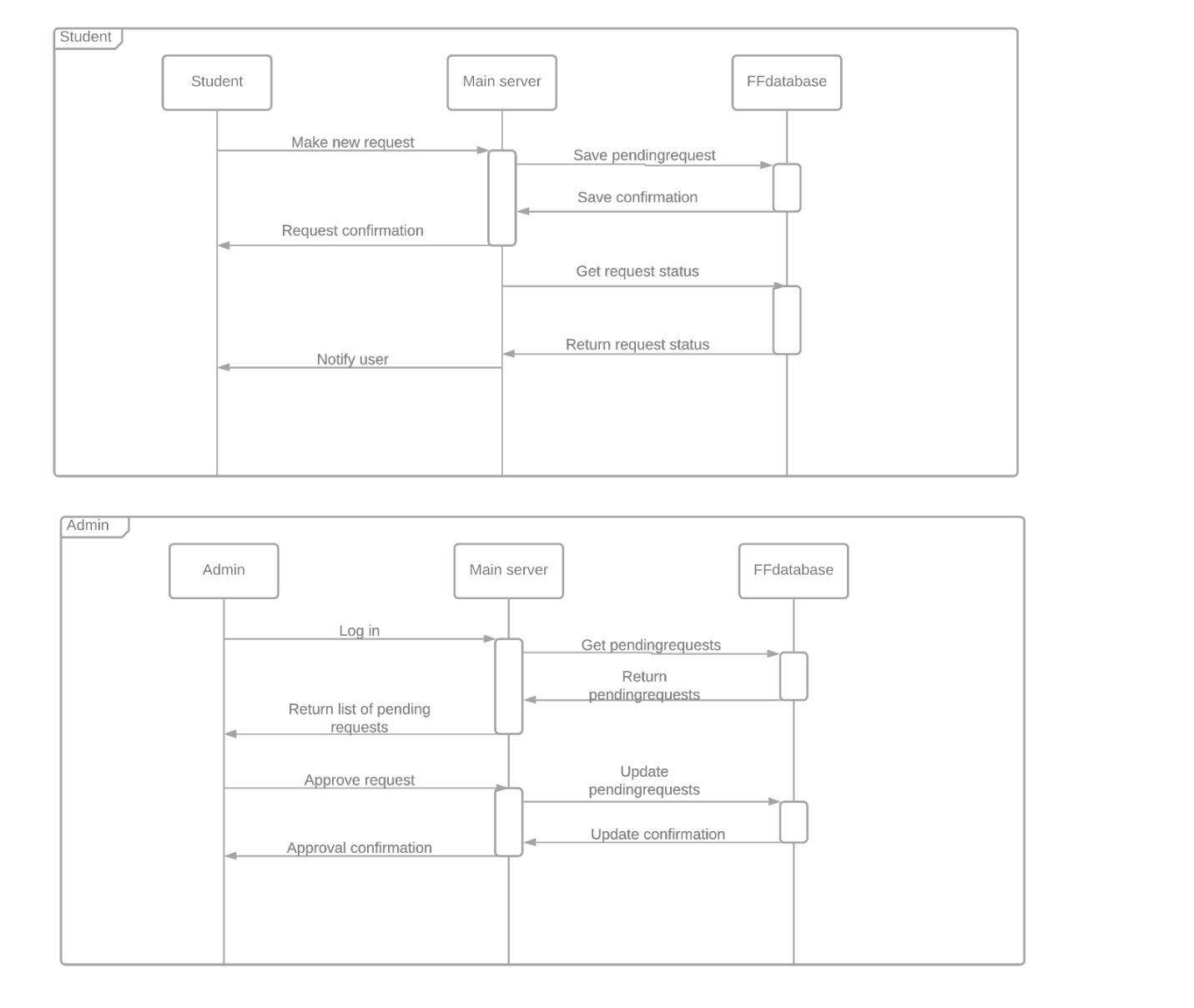


*Figure 3.1*

Login sequence, figure 3.1,

AnyUser will be asked to make an account for the FlashFund system by first entering an email address. The server will allot the User a Donor or Student account dependent on the submitted email address. It will then ask for a set password. The User will get confirmation of enrollment from the database via the server.

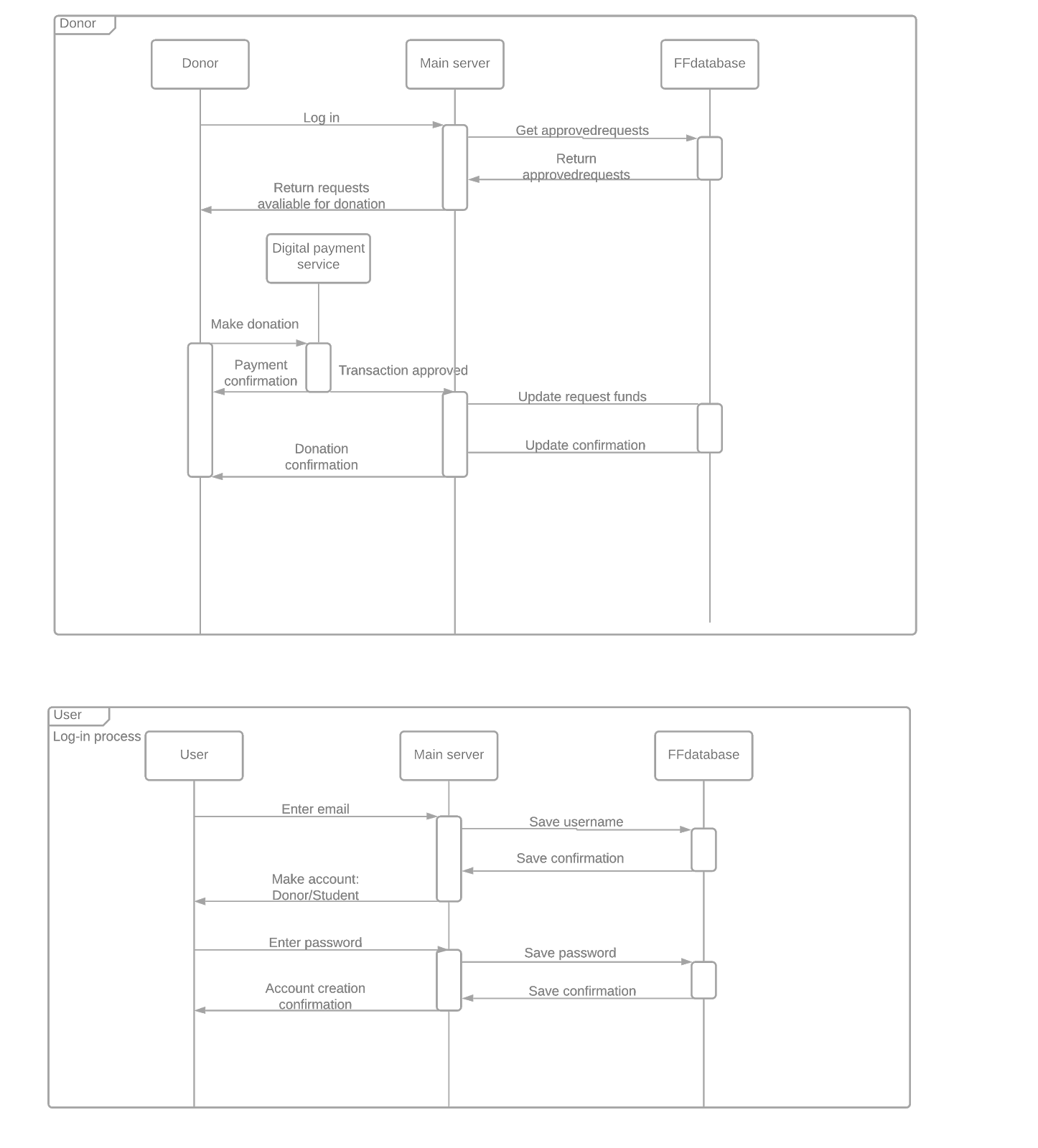
*An email verification system will be coming in a later version, which will change this sequence.*



*Figure 3.2*

Admin user sequence, figure 3.2

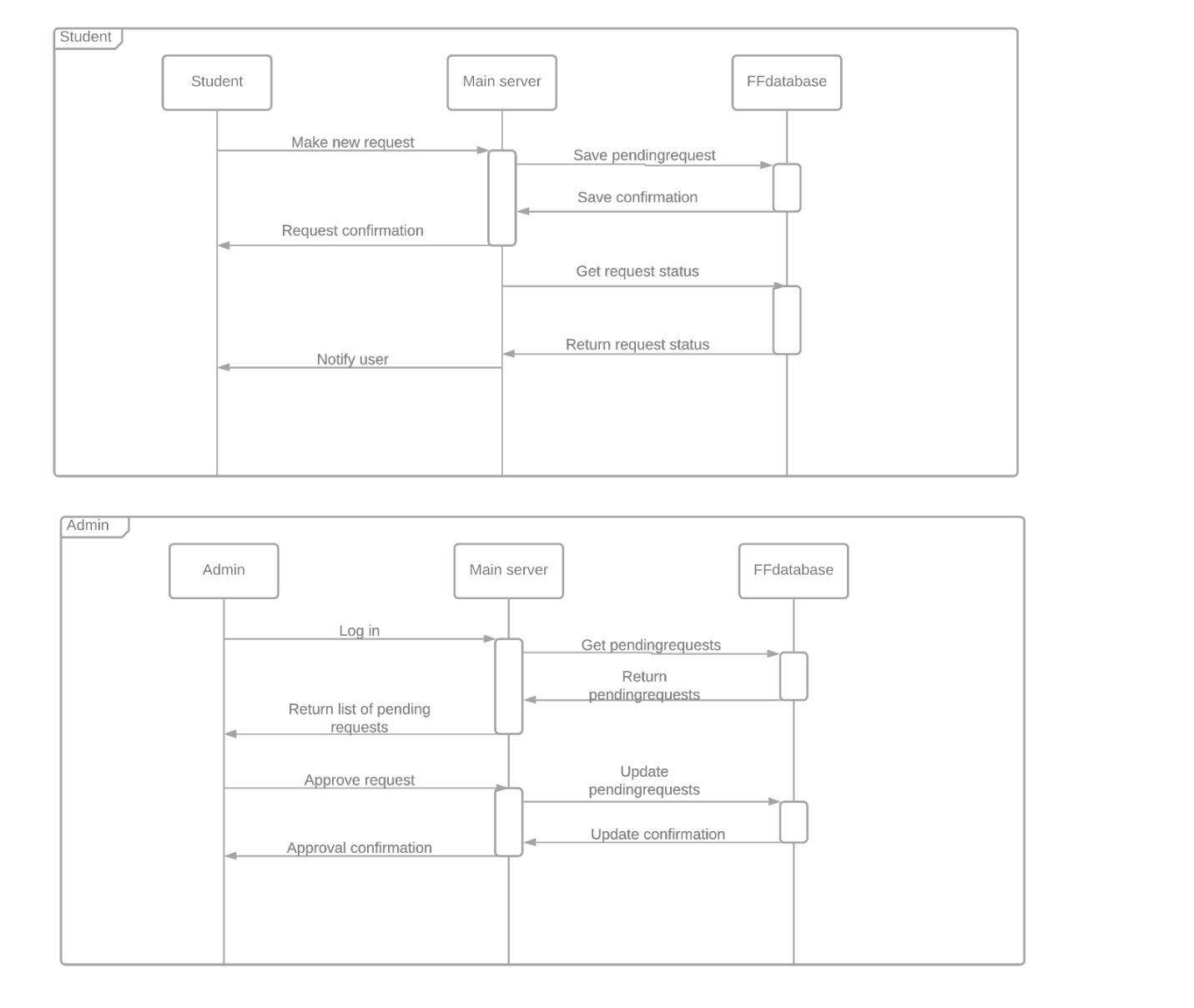
AnAdmin will log in to the server and receive a list of pending requests from the server, fetched from the database. They can process and update pending requests for fundraising to/from students via the server.



*Figure 3.3*

Donor donations sequence, figure 3.3

ADonor will log in to the server and receive a list of funding requests available for donation. They may donate to some of the fundraising requests by PayPal (or another digital wallet service). The Doner will receive a confirmation of the transaction from both PayPal and the database, where the fundraising will be updated via the server.



*Figure 3.4*

Student new request sequence, figure 3.4

AStudent will make a new request for fundraising and get confirmation of the submission from the server. The server will also automatically request the status of funding submissions from the database and notify the student.

I. Overview of the key features and capabilities supported

* Login for posters (students)
* Login for donators (anyone)
* Login for admin to moderate posters of their university
* People with valid university emails can start a donation campaign (pending admin approval)
* Anyone with an account made can contribute money to a campaign
* Admin can approve or deny anyone’s request to start a campaign

II. System Context

Diagram

Description automatically generated

*Figure 4.1*

|  |  |  |
| --- | --- | --- |
| System | Interface | Support |
| Stripe | REST | Receive payments for donation from user |
| Client | HTTP | Send requests to FF\_Central and handles response from FF\_Central |
| FF\_Central | Rest / SQL Queries | Accepts requests from Client and sends back response. Make queries to FF\_DB according to request from client. Sends request to stripe to create payment instance on their site |
| FF\_DB | SQL Queries | Create, update, delete, read from DB |

III. Hardware Components

Text

Description automatically generated  
*Figure 4.2*

The hardware components shown in Figure 4.2 consists of multiple client machines (the users), one central REST API server, and one MySQL database server that is hosting the database

IV. Software Components A picture containing text

Description automatically generated

*Figure 4.2*

We will be using class inheritance to define our three different types of account. User is an abstract class. Donor accounts are simply user accounts, but concrete. Student accounts have all the features of a donor account, but are also given access to make requests and check on their requests. Admin accounts are able to approve and deny requests. Note that admins do not have the ability to create requests.

All three account classes interact with the Request class. This class is the implementation of the funding requests. It contains all the necessary information of a fundraiser and the individual requests are also stored in the database.

Diagram

Description automatically generated

*Figure 4.3*

Our database will divide the users by Account\_type. The Request table will hold all the fundraiser information. The Donorlist table will list all the donations made for each request.

Frontend:

The web Frontend is developed using React.js, a JavaScript library, along with HTML, CSS and JavaScript. It is a SPA, so all the HTML, CSS, and JavaScript files are loaded initially. Parts of page UI is dynamically updated in response to user action, but an entire new page is not loaded. Rationale for using a SPA is creating a fast and response web app due to decreased load times. This is in line with our goals of user experience.

Server-Side Framework:

V. Users

System admin

* server maintenance / backups
* Hold money of campaigns

Application Admin

* Moderate campaigns from users of their domain

End user

* Donate or create campaigns

VI. Capability and Performance

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Value | Software Configurable | Hardware Configurable |
| Simultaneous users | 250 | Yes | Yes (expand servers) |
| Transactions per hour | 100 | No | Yes |
| Total campaign postings | 1,000 | No | Yes (expand database storage) |

User Interface

## Look and Feel guidelines:

Interface should be easy to navigate with a simplistic layout. Navigation is aided by buttons, text, and animations, directing the user where to go. Since the application is for Kent state, the color scheme is school colors.

## Menu Hierarchy and Traversal:

Traversal will be different depending on what type of user is logged in, or if the user is logged in at all.

Not logged in user:

* The user will be greeted by the landing page (Figure 4.1). On the navbar, the user can choose to click ‘register’ or ‘login’.
* If the user chooses the register page (Figure 4.7), they will fill out the registration form and then be redirected to the login form (Figure 4.2)
* On the login form user will enter login details and press submit to login
* Once logged in, a user will be redirected to the home page

Student User:

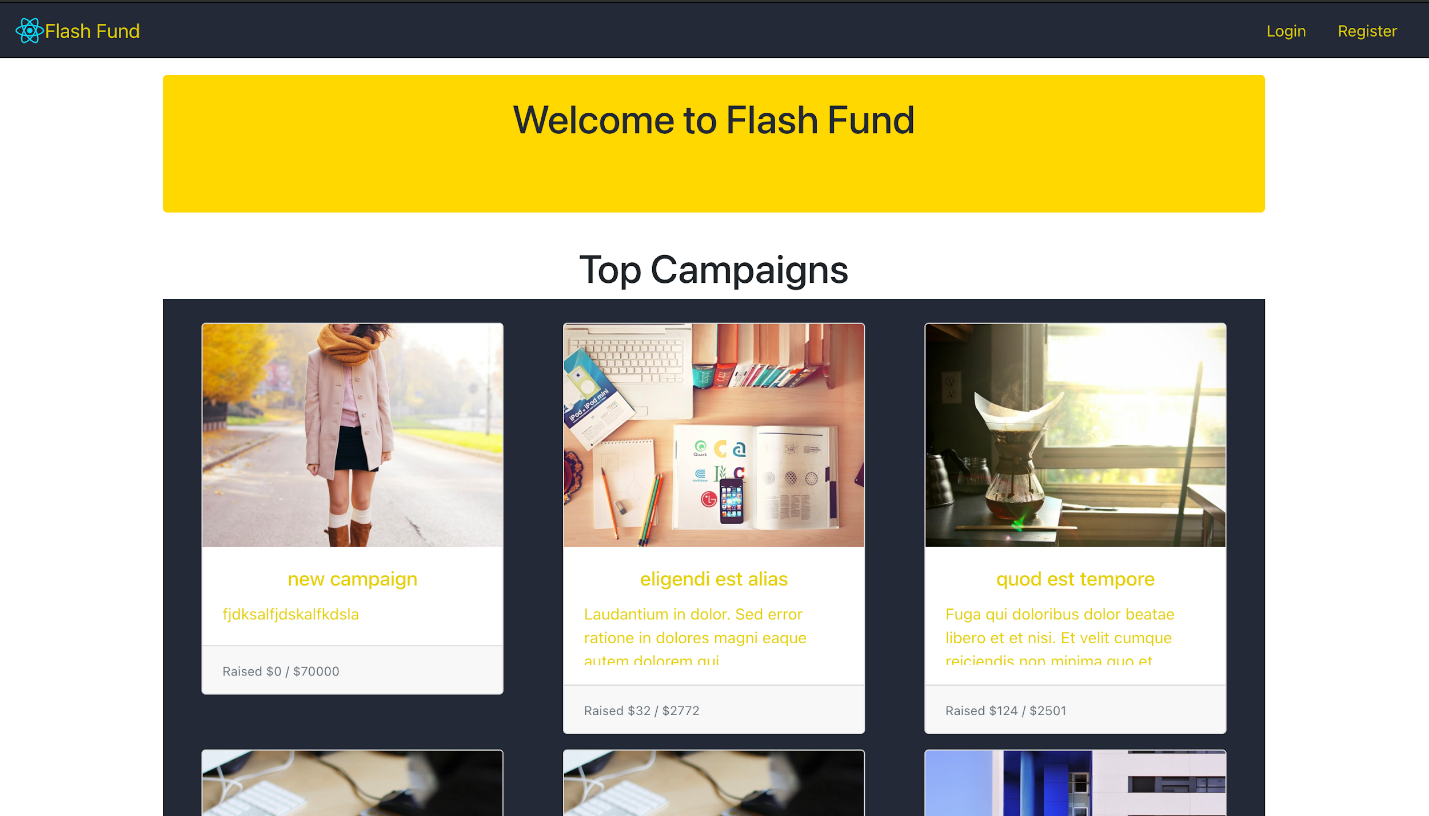
* A student user will have the home page in Figure 4.3. From that page they can traverse the campaign postings, or press the button on the home page to create a new campaign posting
* If a card is pressed the user will be brought to the page in Figure 4.4, where then they can choose to click the donate button, or press the left side of the navbar to be brought back to the home page
* A student user is able to donate, and once the donate button on the card is pressed, the user will be brought to the donation screen. After entering the amount they will be redirected to home page
* Once the student user presses the new campaign button, they will be brought to the campaign creation form page (Figure 4.9). From that page they can press the left side of the navbar to return home

Admin User:

* Instead of the new campaign button on the home screen, the admin will have a button to go to the pending requests page (Figure 4.8). Once on the pending requests page, the admin can press the left side of the navbar to return to the home page
* All other pages are the same for the admin as the student user

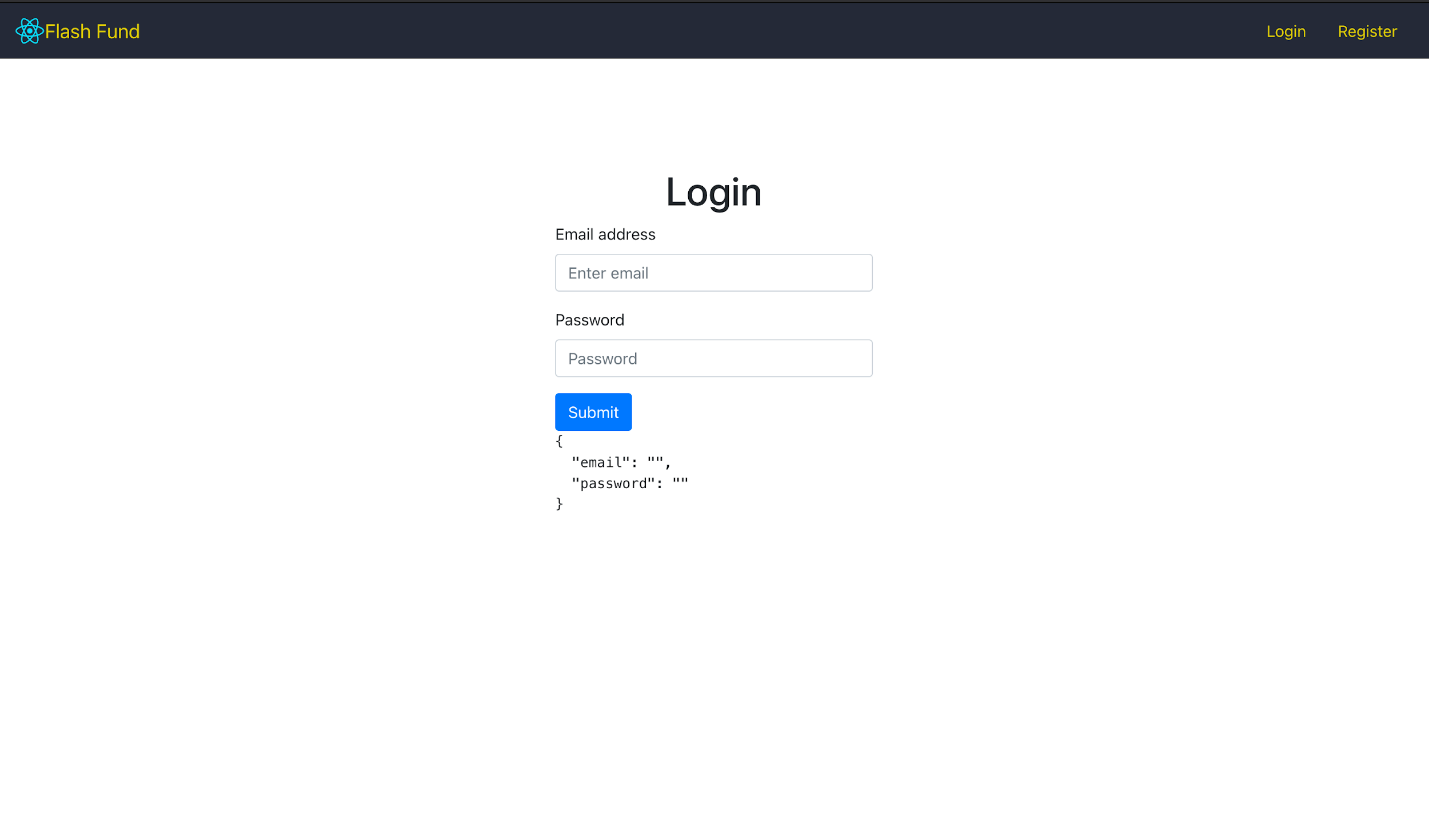
Donor User:

* Donor users can only traverse the web app through clicking on campaign postings. That traversal follows the same as the other user types



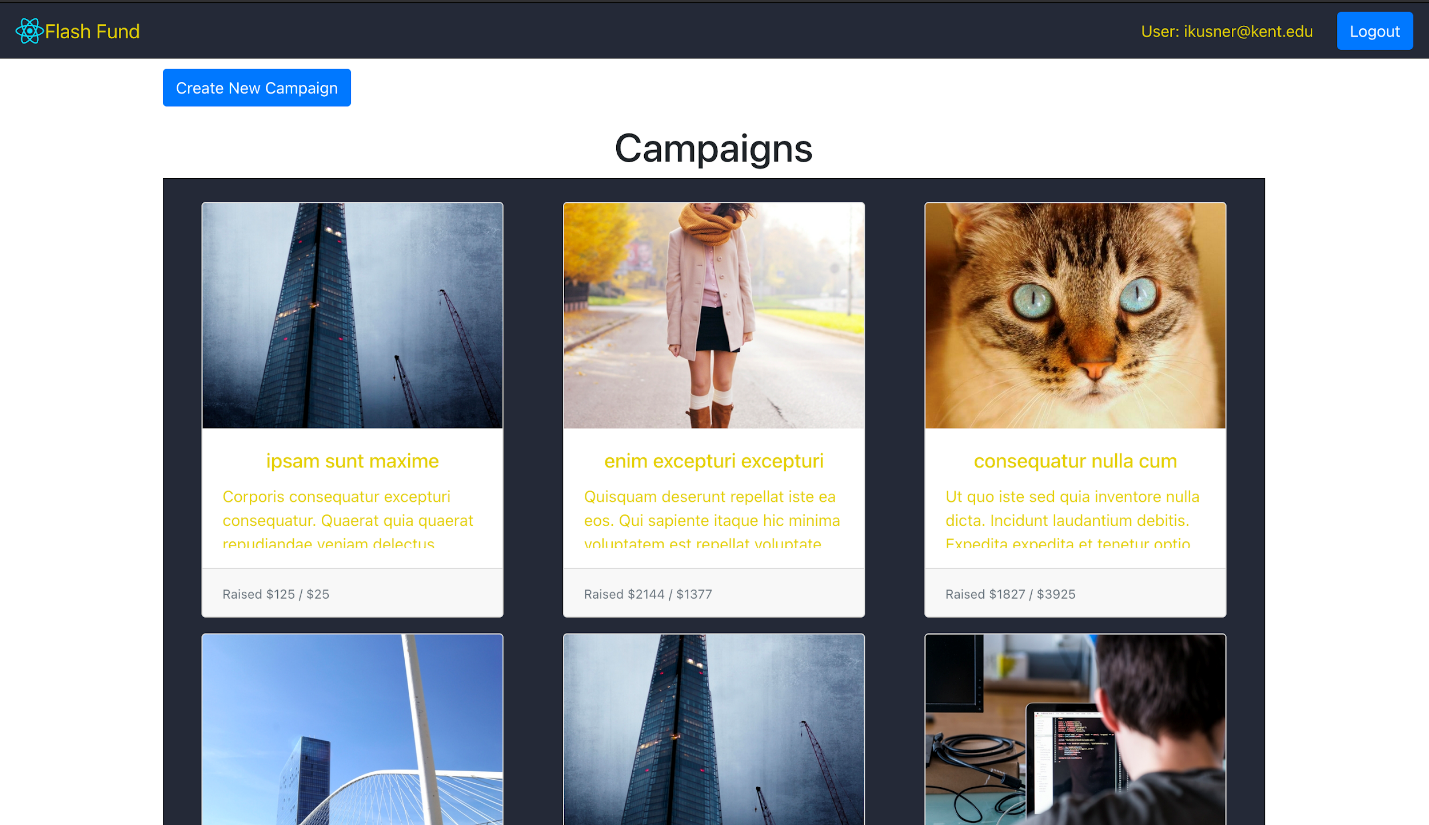
*Figure 5.1, Landing Page*

Figure 5.1 shows the landing page that a not signed in user will see. Contents on this page is a list of the top 6 currently running campaigns, a header welcoming the user, and a navbar that includes links to login and registration pages. All the content on this screen is partitioned into 3 different components, with the navbar sticking to the top, the welcome message underneath, and a container of cards in a 2 x 3 grid for the user to be able to easily view



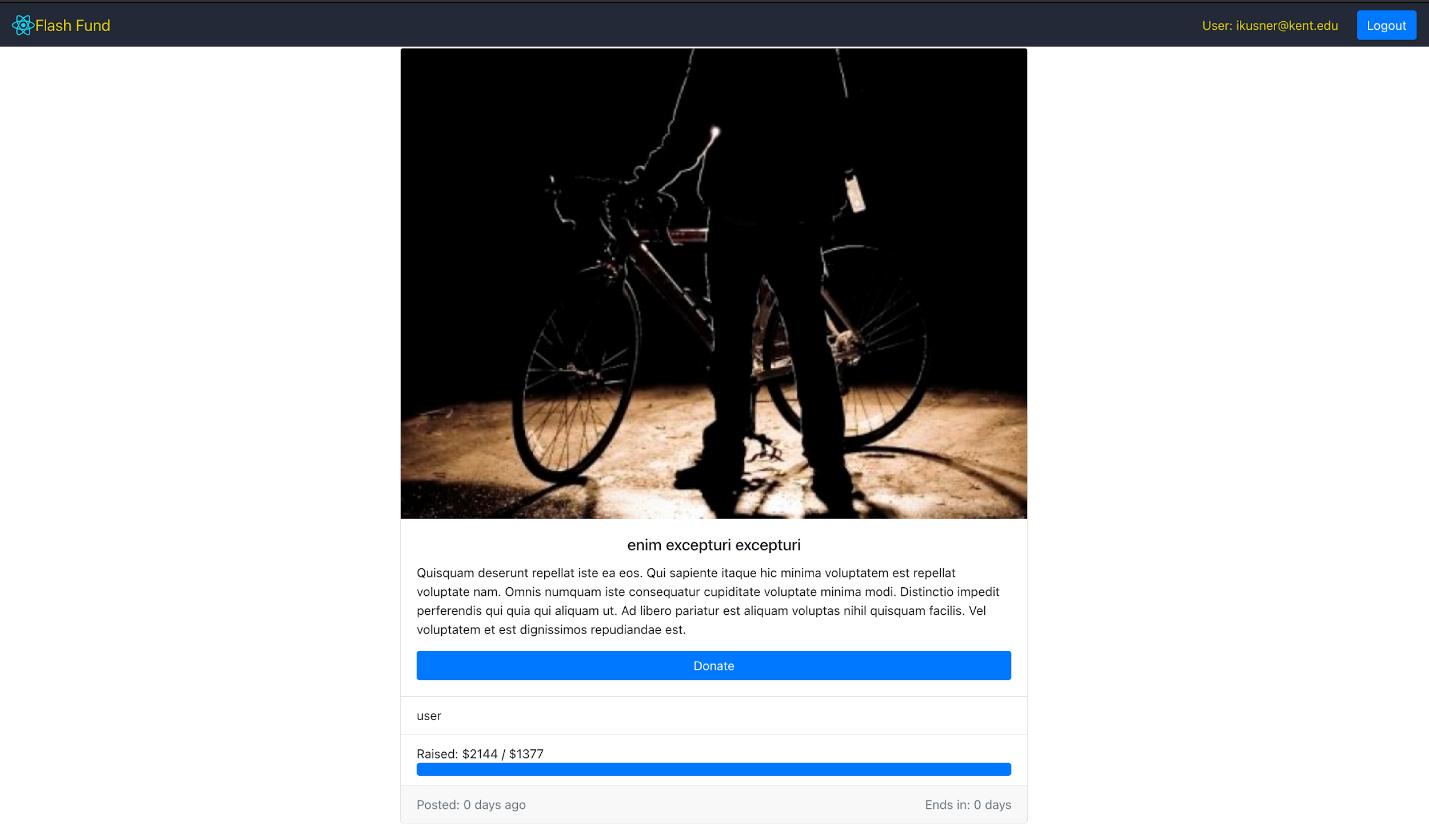
*Figure 5.2, Login page*

The login page in figure 5.2 contains 2 input forms for the user to enter email and password, along with a submit button to start the login process with the server. Login page also has the nav bar at the top, which the user can click on to navigate to the register page, or to the home page. Screen is partitioned into a form and the navbar on the top.



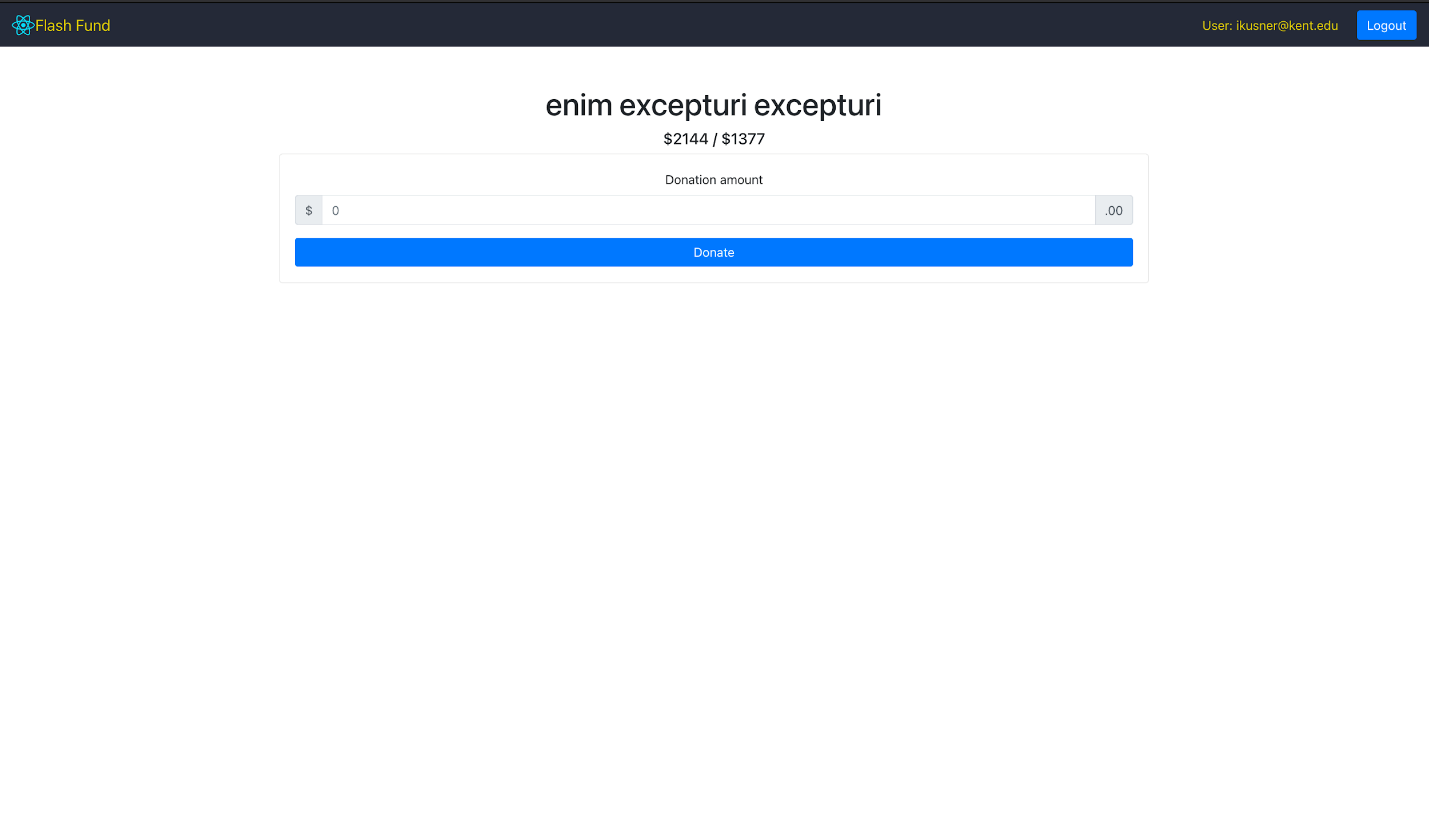
*Figure 5.3*

Figure 5.3 shows the page that a user who logged in with a @kent.edu email will see. Contents include a list of cards that are paginated (see figure 4.6), a button that will lead the user to a page to create a new campaign posting, and again the navbar at the top that the user can now use to logout. Logout will redirect back to the landing page. Screen is partitioned so the new campaign button is at the top of the screen with the list of campaigns underneath.



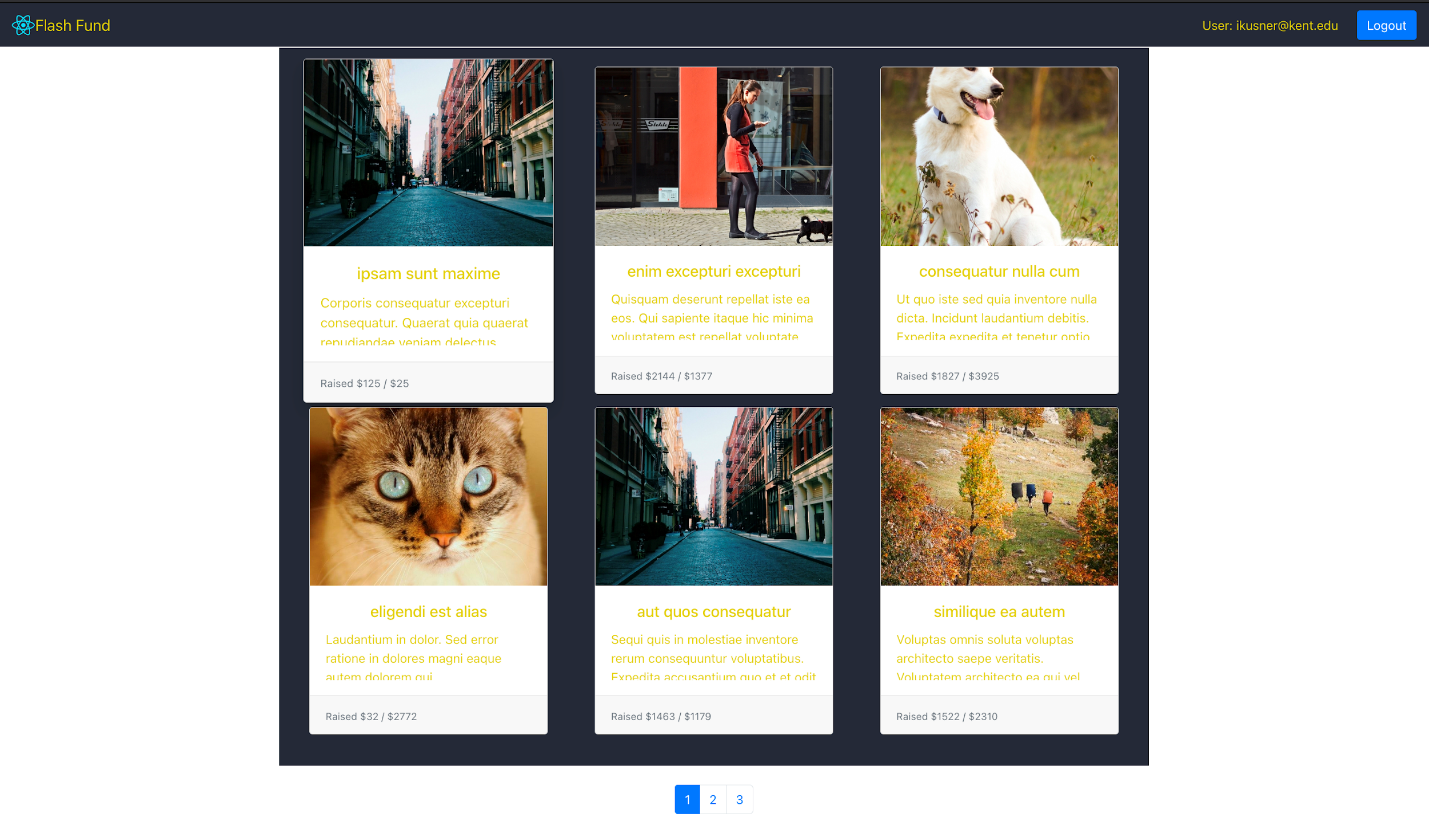
*Figure 5.4, Campaign card*

Once a user clicks on a card displayed on the landing page or home page, they will be brought to the screen in Figure 5.4. Contents on this page include the card information from the landing or home page (image, title, description, funding), along with the user of the post, and days ago posted, and days till posting ends, and the donate button. Screen is partitioned to provide an easy view of the campaign that the user can choose to donate to.



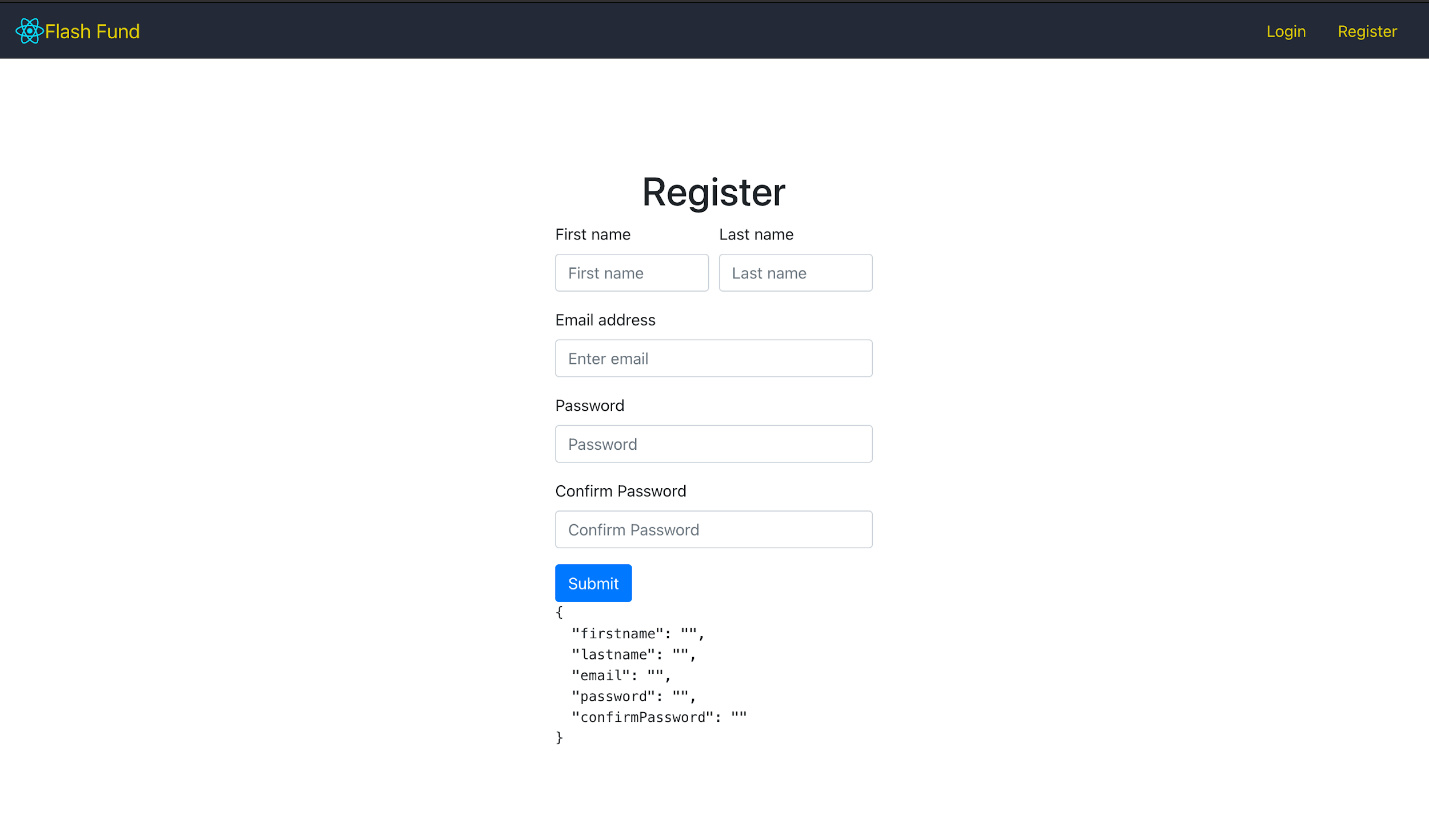
*Figure 5.5, Donation page*

Once the user clicks the donate button and the user is logged in, they will be brought to the page in Figure 5.5. Content includes a donation form that the user can enter their amount to donate, along with a button to submit the donation. Nav bar is once again at the top. Screen is partitioned to have the donation form in the middle of the screen.



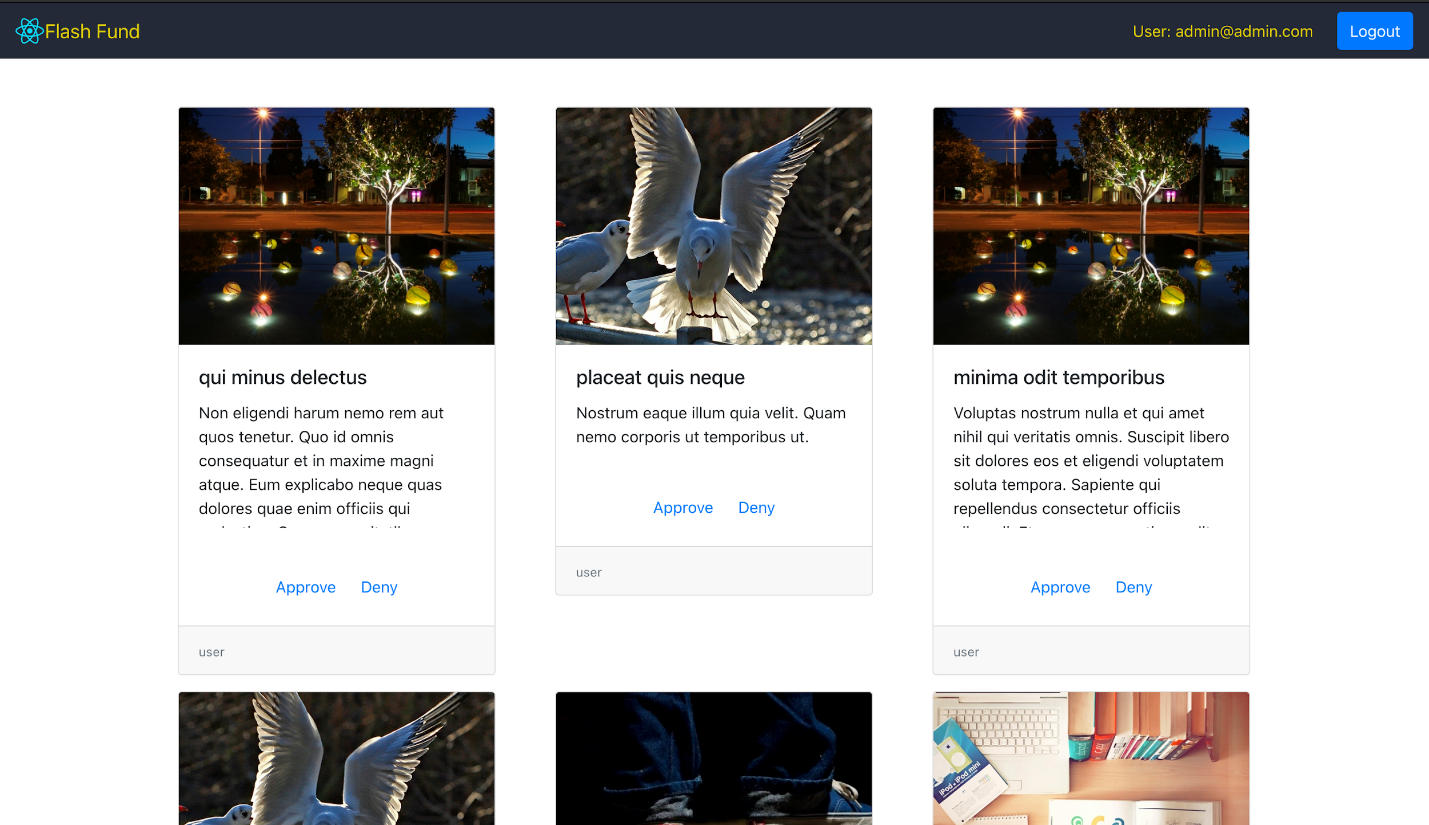
*Figure 5.6, Home page card view w/ pagination*

Figure 5.6 shows that the home page shown in Figure 4.3 has a paginated list of cards. List is partitioned this way to provide an easier view of cards for the user.



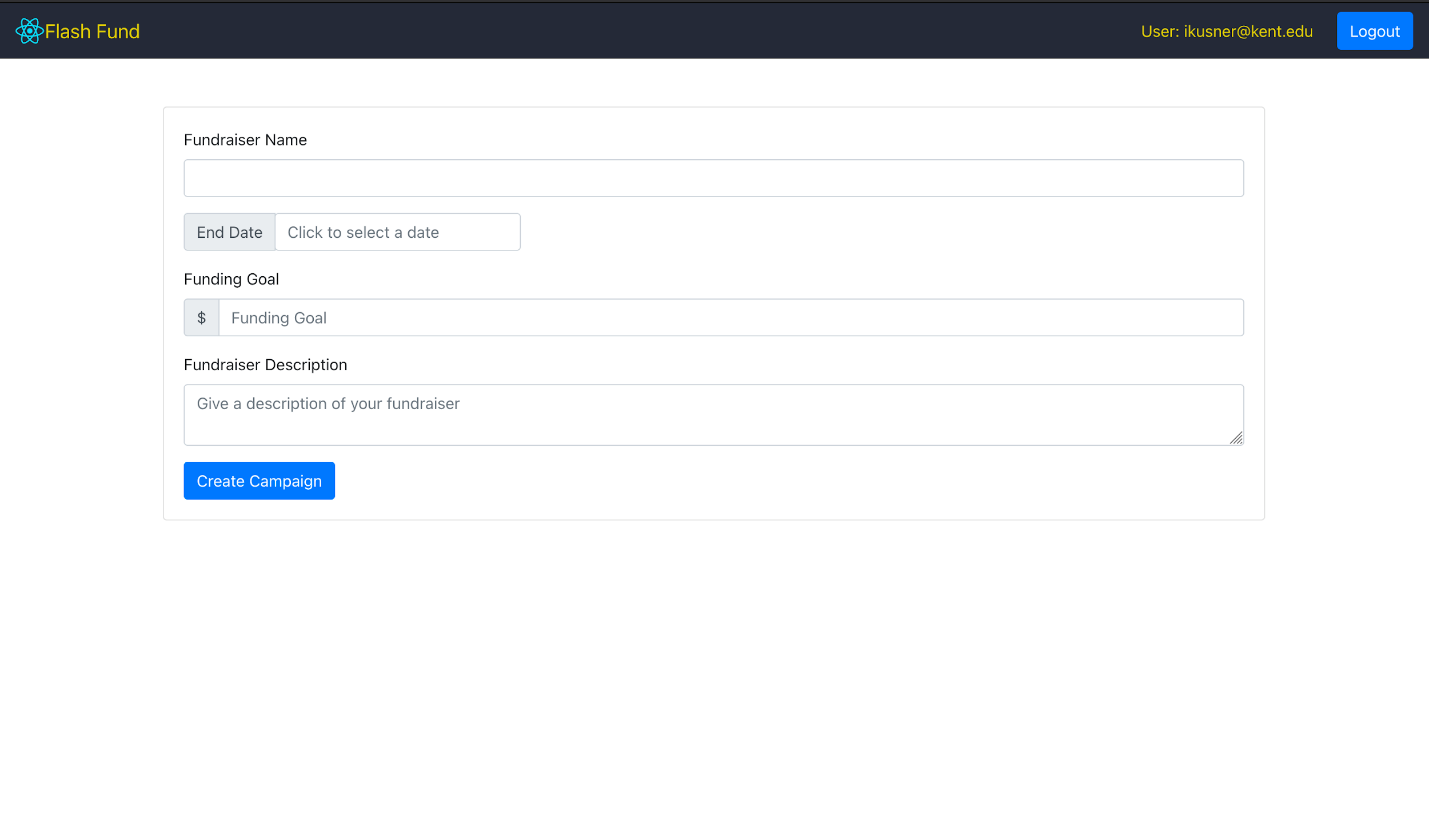
*Figure 5.7, Registration page*

Figure 5.7 is the Registration page that includes a registration form. Form is centered on the page. Submitting the form will bring the user to the login page.



*Figure 5.8, Admin Approve/Deny view*

Figure 5.8 is the page admin's access to either approve or reject a pending request. Page includes a list of cards that have a button to either approve or reject the request from appearing to others to donate to. List is partitioned in columns of 3 to make viewing of cards easier for the admin user



*Figure 5.9, New Campaign page*

Once a student user clicks the create campaign button, they will be brought to the screen in Figure 5.9. Page includes a form a user will fill out to create a new posting. Once the button is pressed, the posting will be added to the list of pending campaigns that the admin will need to approve.

Graphical user interface, application

Description automatically generated

*Figure 5.9, New Campaign page*

After a user clicks the ‘donate’ button shown in Figure 5.5 they will be brough to Stripe’s Checkout page in Figure 5.9 to make the transaction.

Error Recovery

## System could encounter the following errors:

User form validation:

* The frontend will display error messages when a form’s input value is deemed invalid.
* For example, when the user is asked to enter their email address to login, the email must be in proper email format (@domain.top-level-domain) for that entire form to be submitted to the server. User will be asked to re-enter if value is invalid

AJAX errors:

* When a request is sent from the client that results in an error response from the server. The client will then handle the error appropriately.
* An ‘Unauthorized’ error with a 401 HTTP status will be sent back to the server if the user is trying to access content they shouldn’t, or enters incorrect login credentials
* A 400 HTTP status code will be sent back to the user if a bad request is made due to bad input

## Recovery:

User form validation:

* During a form validation error, the user will be unable to submit a request to the server until the form is validated. Once the user fixes the input error that are displayed, the request will be sent

AJAX error:

* The client will catch the error it receives from the server. During this time a page could either not be rendered with data, or data could not be posted to the database.
* After error is recovered, the system will function once again, with the server properly accepting requests

Development Team Organization

