# 6.170 Design Analysis

**Key Design Challenges**

1. Deciding what types of accounts are necessary and how to represent them
2. How to process payments from donors to the organization
3. How to represent donations to a person participating in a specific event rather than to the event as a whole or the person as a whole.

**Data Representation**

### volunteers

- name: Volunteer's name (first name)
- last_name: Volunteer's last name
- email: Volunteer's email (unique across both volunteers and organizations)
- password_digest: Hashed and salted password
- avatar_url: URL for profile image location for volunteer

### organizations

- name: Organization's name
- email: Organization's email (unique across both volunteers and organizations)
- password_digest: Hashed and salted password
- stripe_token: Stripe Connect token for the organization
- stripe_pub_key: Stripe Key for organization
- bio: Free-form bio of organization
- avatar_url: URL for profile image location for organization
- follows_count: count of how many followers an organization has

### events

- organization_id: ID of organization that's running the event
- description: Free-form description of the event
- name: Name of event
- date: Date for when the event is occuring
- time: Start time for event
- location: Location for the event
- image_url: URL for location of event image
- thank_you_email: canned thank you email provided by the organization for the event
- solicit_email: canned solicitation of funds email provided by the organization for the event

### participations

- volunteer_id: The ID of the volunteer who's participating
- event_id: The ID of the event that the volunteer is participating in
  *the tuple (volunteer_id, event_id) is enforced to be unique across participations*

- note: Free-form note provided by the volunteer to display on the participation page
- goal: The amount of money the volunteer hopes to raise for the event

### donations

- participation_id: The participation that was donated to
- amount: The amount of money donated
  *note that we do not store billing information: the one-use stripe token generated by stripe.js and sent to the server is used once to bill the donor and then discarded.*
- message: A message from the donor to the volunteer
- name: Name of donor
- email: Email of the donor
- is_message_private: boolean which stores privacy setting for the donor's message
- is_email_private: boolean which stores privacy setting for the donor's email
- is_name_private: boolean which stores privacy setting for the donor's name
- thank_you_sent: boolean which stores whether a thank you has been sent to the donor

### offline_donations

- participation_id: The participation that was donated to
- amount: The amount of money donated
- name: Name of donor
- email: Email of the donor
- donation_type: Type of donation, either cash or check
- is_email_private: boolean which stores privacy setting for the donor's email
- is_name_private: boolean which stores privacy setting for the donor's name
- thank_you_sent: boolean which stores whether a thank you has been sent to the donor

### follows

- volunteer_id: The volunteer that is following
- organization_id: The organization the volunteer is following

**Key Design Decisions**

1. Type of accounts:

*Ways to implement*:

Our system has three different types of users — organization, volunteer, and donor — who each have separate events. We could represent each type of user with a separate class, represent them all as one class with a field denoting special features of each class, or something in between.

*What we did*:

We chose to create separate classes for organizations and volunteers, both of which did not inherit from a parent class. We also chose not to create donor as a separate object for

simplicity. Also, we didn't want to require donors to have to create an account just to donate. We decided to separate organizations from volunteers since their permissions and events on the site are almost completely disjoint.

2. Payments:
*Ways to implement*:
In the simplest case we could have made payments just numbers in a database that don't actually pertain to real money (as many people did in the shopping cart project). We wanted to have a functional payment system that could actually deal with credit card information so that left PayPal and Stripe as being viable options.

*What we did*:
We decided to implement payment/donation processing through Stripe Connect so we wouldn't have to store a version of our users' payment information in our own databases. Security is key, and using Stripe means that the only information we have to store is a token, rather than credit card information, PIN numbers, etc... With Stripe, we don't have to handle any money — money flow directly from a donor to the organization. We chose not to use PayPal because we would have to store more information in our databases, and we have to give users a choice to be redirected to PayPal. PayPal also doesn't allow for a middle person to handle payments (accept payments from one entity and forward it to another).

3. Where a donation goes:
*Ways to implement*:
Our system has volunteers participating in events hosted by organizations. When a donor donates money the end result is that their money should go to the organization but we needed to represent in some way that the money is for a particular volunteer's participation in the event. This is important to keep track of since each volunteer wants to know how much money they have raised since oftentimes walks and events have a fundraising minimum. This can be implemented by having a donation object that belongs to a user and to an event and the user can find out how much money they have raised for each event by looking through all of their donations associated events. This can also be done by having a separate participation object that belongs to an event and to a user that represents the user's involvement in a specific event so that donations can then be associated with just the participation. In this model the donation belongs to the user and the event through the participation.

*What we did*:
We decided to implement the second option where a user has multiple participation objects that belong to different events. We thought this extra layer of abstraction simplified the process of organizing which donations belonged to what event. In the first option in order to calculate how much money each user has raised for an event you have to go through all of their donations and group them by event each time which. This sorting and summing in our opinion is much more complicated than just having a separate participation object that contains all of the donations already grouped by event and by user.

**Design critique**

*Summary assessment from user's perspective*

The site is very intuitive and easy to navigate. The UI is aesthetically pleasing, and the forms are easy to understand. Payment processing looks professional — not like a sketchy student project. It is also easy to upload pictures with FilePicker. The search features for organizations, events, and volunteers is pretty slick. Something that is missing is some sort of personalized dashboard for signed in users or organizations — having a newsfeed or personalized suggestions for organizations to follow or events to participate in would be great. Our newsfeed currently only includes donations that have been made to your participations.

*Summary assessment from developer's perspective*

There is appropriate separation between models, views, and controllers, and controllers are kept slim and focused. Each method in the controller is commented appropriately. The routes file only included methods that were implemented in the controller. Testing with RSpec is pretty comprehensive. Each of the controller methods was tested. The routing followed RESTful protocols. We used Twitter Bootstrap as a base and customized views on top of it.

*Most and least successful decisions/Analysis of design faults in terms of design principles*

Using Stripe was a great decision — this drastically lowered the number of security issues we had to worry about. The downside of using Stripe is that Stripe takes a percentage of each transaction and also organizations need to have an account on Stripe to sign up for an account on the site (increased barrier of entry). However, the downsides were outweighed by its simplicity and security.

Our least successful decision was not dealing with organization vetting — there is currently no process in place to verify organizations are actually who they say they are. As seen in our demo, it is extremely easy to add a new organization and start taking in donations, which is a large security loophole. One way we could have lowered the chances of fake organizations being created is verifying email addresses (such as send the organization an email when a new account is created) or having organizations create an account only after getting a unique link that we give directly to their fundraising coordinators. We could also have prospective organizations go through an application process that is reviewed by an admin (or admins) since email validation doesn't prevent people from creating an account with a legitimate name and icon but illegitimate email which volunteers might not recognize.

*Priorities for improvement*

1. Verification process for organizations (especially since money is going directly from donors to the Stripe account that an organization signs up with)
2. Personalized dashboards with a news feed about organizations you follow + suggestions of organizations/events to follow
3. Groups — allow groups of people to have a participation and allow donors to donate to the group's participation, not just an individual's participation
4. More social integration