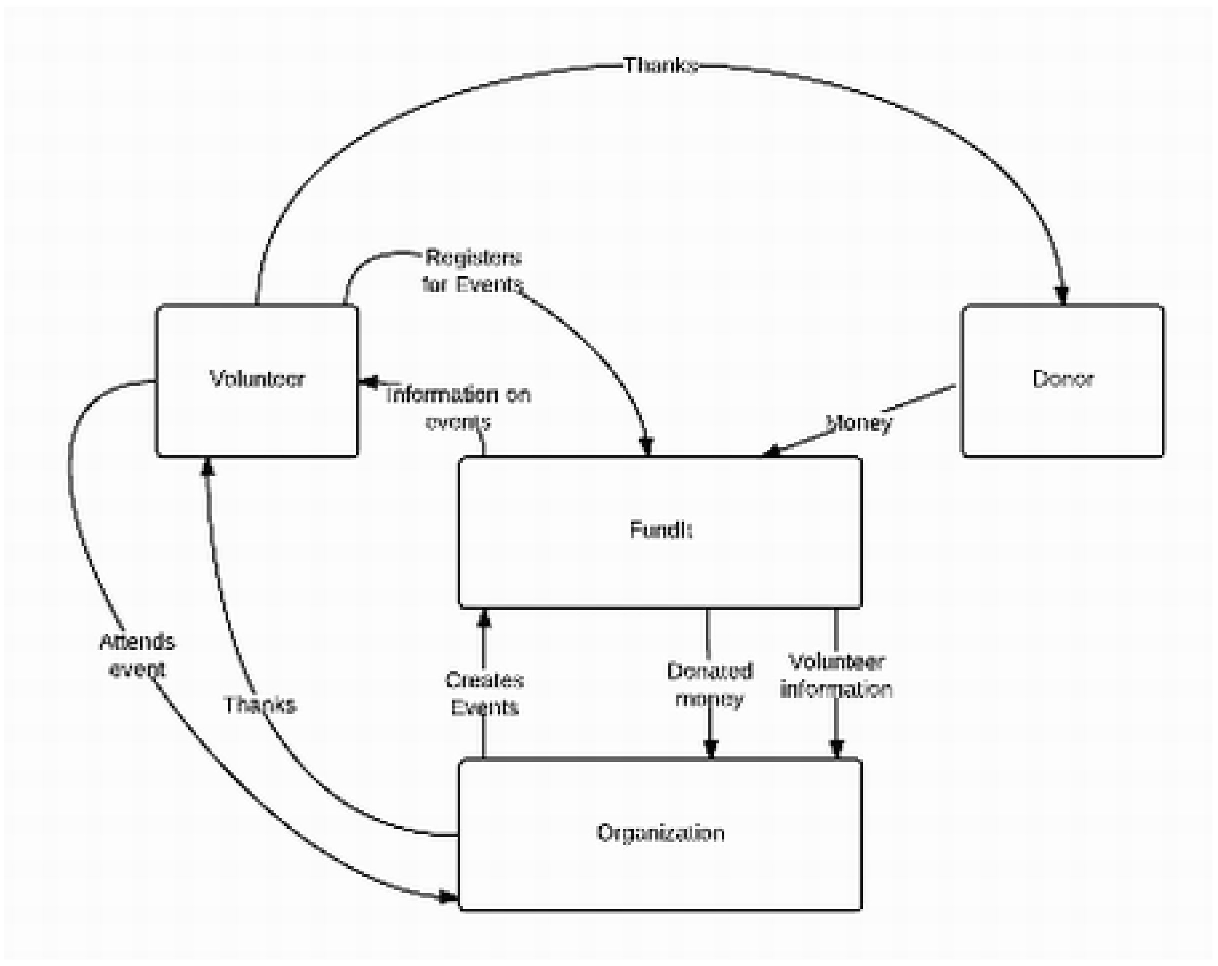
OVERVIEW

Purpose and goals

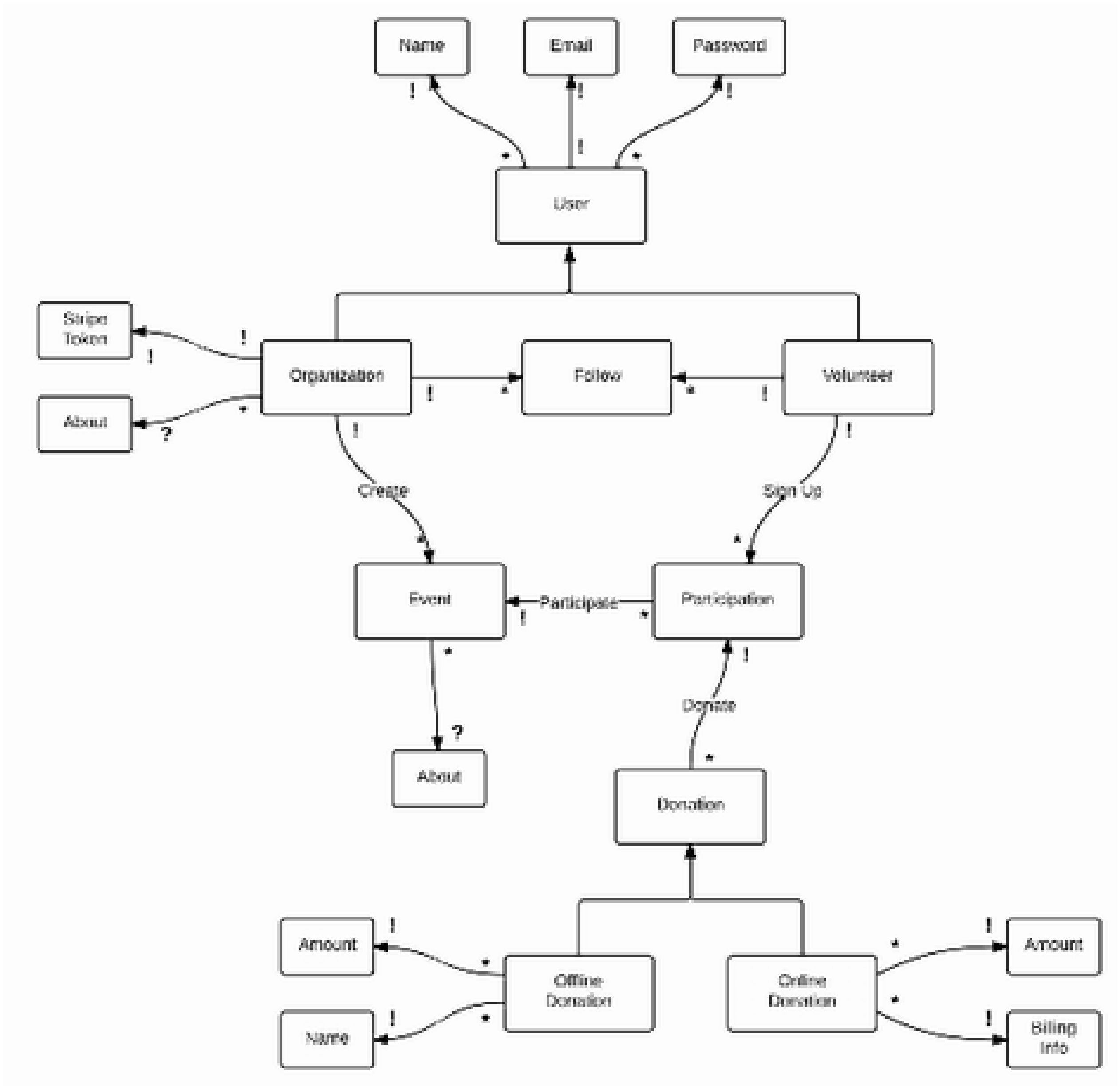
FundIt is a fundraising platform to organize fundraising events — like walks, bike rides, and dinners. Like Meetup.com, people can sign up once and easily join events on the site and donate via Stripe. The website will help volunteers quickly set up donation pages and allow users to discover fundraising events they are interested in from new new non-profits and follow events from a list of their favorite non-profits.

Context diagram



DOMAIN

Object model



In our model there are two types of users who can have accounts: organizations and volunteers. The third type of user for the site are donors but in our minimum viable product we do not have accounts for donors.

Each user who wants to participate in an event has a participation object associated with the event that donors can then donate to rather than having donations be directly associated with just an event or just a user.

Event model

Events:
register_organization: An organization can register for an account with an email/pwd
register_volunteer: A volunteer can register for an account with an email/pwd
sign_in_organization: An organization can sign in to its account to modify/add/delete events
sign_in_volunteer: A volunteer can sign in to sign up for events
sign_out_organization: An organization can sign out from its account
sign_out_volunteer: A volunteer can sign out from his/her account
add_event: An organization can add a new event
modify_event: An organization can modify an existing event
delete_event: An organization can delete an existing event
register_to_volunteer_for_event: A volunteer can sign up to volunteer for an event that an organization created
follow_organization: A volunteer can follow an organization to get notifications from the organization (notifications have not been added yet)
log_offline_donation: A volunteer can log donations he/she receives off the FundIt website
send_thankyou: A volunteer can send a thank you email to a donor
donate_public: A donor can donate to a participation with both his/her name, amount donated, and message to volunteer public
donate_hide_info: A donor can donate to a participation and choose to hide his/her name, amount, or message from the public.

Event = organization_events | volunteer_events | donor_events
organization_events = register_organization sign_in_organization add_event [modify_event] [delete_event] [sign_out_organization]
volunteer_events = register_volunteer sign_in_volunteer [register_to_volunteer_for_event] [log_offline_donation] [send_thankyou] [follow_organization] [sign_out_volunteer]
donor_events = donate_public | donate_hide_info

BEHAVIOR

Feature descriptions

Add event: Organizations can create events. Volunteers can then sign up for these events, and donors can donate to individuals for a certain event. Each event has a page (belonging to the organization that created it), and each volunteer for the event has his or her own donation page for that particular event.

Payment: The site will use Stripe Connect to handle payments/donations.

Notifications: Followers of an organization can get a notification when the organization creates a new event.

“Follow” an organization: Volunteers can follow organizations to get email notifications when a new event is created by an organization they follow.

Log offline donations: Volunteers can log the donations that they receive outside of the FundIt site.

Private/public donations: Donors can choose to donate to a participation anonymously (the name and/or amount and/or message to volunteer can be hidden).

Thank you emails: Volunteers can send thank you emails to people who donate to their participations.

Security concerns

Roles

Our system interacts with three types of users: nonprofits, who are able to create and manage events; volunteers, who participate in events; and donors, who contribute money to a volunteer for a particular event.

Threat Model

We assume that attackers have access to the website, and can sign as any type of user. We assume that attackers only have access to our system via HTTP: they do not have access to the server infrastructure (e.g. we don’t defend against attacks by Heroku or AWS employees).

Key Concerns

The most important security concern is keeping donors’ billing information safe. Secondarily, the amount donated should be kept as private as possible. Finally, volunteers’ contact information should be kept private unless and organization has a particular reason to contact the user.

Data Privacy

Nonprofits:

name and bio: can be edit by the nonprofit; can be viewed by anyone.

events: can be created by nonprofits; nonprofits can edit their own events; anyone can view events.

Volunteers:

name and bio: can be edited by the volunteer; can be viewed by anyone.

email: can be edited by the volunteer; can be viewed by nonprofits running events the volunteer has signed up for

amount raised: anybody can see the total amount a volunteer has raised for a particular event.

Donors:

payment information: only our system can bill donors.

donor names, emails, and donation amounts: donor emails are only shown to the volunteers they donate to; donors can choose whether to keep their names and amounts private.

Payment Processing

Payment processing will use Stripe Connect: https://stripe.com/connect

In this system, nonprofits connect their account to a Stripe account. We get a token that lets us bill donors on the organization's behalf. When we present a payment form to donors, we user stripe.js to have the donor's browser send the information directly to Stripe over HTTPS. Stripe gives us a customer token that lets us charge the user on behalf of the organization they're donating to: the donor's money goes directly to the organization, and the donor's credit card details never even hit our server.

Standard Defense Mechanisms

SSL: Rack::SSL is used to redirect users who attempt to access the site over HTTP to the equivalent HTTPS page. It also sets the Strict-Transport-Security header, which instructs browsers to treat any issues with the site's SSL certificates and fatal errors (rather than prompting the user to ignore the error), and marks all cookies as secure (preventing them from being sent over non-SSL connections).

XSS Attacks: all user-provided data will be sanitized on output using standard Rails sanitization helpers or those built into out templating system.

CSRF Attacks: we will use Rails' built-in CSRF protection to reject requests that do not including a valid authenticity token.

SQL Injection: we will use ActiveRecord's query interface to perform queries, which will automatically sanitize input data.

Clickjacking Attacks: we will use the X-Frame-Options header to prevent pages on our webiste from being embedded in a frame or iFrame on a malicious website.

Password Storage: Passwords will be salted with a per-user salt, and then hashed with BCrypt. The process will be handled by an existing library, such as has_secure_password or Devise.

Operations

GET /

Effects: Returns a welcome page that allows volunteers and organization to sign in or register, and shows upcoming events to entice users to join the system.

POST /login

Requires: an email and password
Modifies: session
Effects: Authenticates an organization or volunteer with the username and password. If successful, gives the user an authentication cookie so they can make further requests.

DELETE /logout

Requires: user is logged in
Modifies: session
Effects: destroys the current user's authentication token.

GET /organizations

Effects: returns a list of registered organizations.

POST /organizations

Requires: an email, password, and Stripe Connect token
Modifies: Organizations
Effects: creates a new organization account and logs the current user in as that organization.

GET /organizations/:id

Requires: an organization with the given id exists
Effects: Returns information about the organization: name, bio, and upcoming events

PUT /organizations/:id

Requires: the current user is logged in as the organization with the given id; an updated set of attributes for the organization
Modifies: the organzation with the given id
Effects: the organization with the given id is updated with the given set of attributes

PUT /organizations/:id/follow

Requires: A user is logged in; the logged in user does not follow the organization with the given id; an organization with the given id exists.
Modifies: Follows
Effects: The currently logged in user stop following the given organization

PUT /organizations/:id/unfollow

Requires: A user is logged in; the logged in user follows the organization with the given id; an organization with the given id exists.
Modifies: Follows
Effects: The currently logged in user stops following the given organization

GET /organizations/:org_id/events

Requires: an organization with the given org_id exists
Effects: returns a list of events created by the organization with the given org_id

POST /organizations/:org_id/events

Requires: the current user is logged in as the organization with the given org_id; an event name and description
Modifies: Events
Effects: creates a new event associated with the given organization

GET /organizations/:org_id/events/:event_id

Requires: an event with the given event_id exists
Effects: returns information about the given event: a description, top fundraisers, etc.

PUT /organization/:org_id/events/:event_id

Requires: the current user is logged in as the organization with the given org_id; an event with the given event_id exists and is owned by the organization with the given org_id; an updated set of attributes for the event
Modifies: the event with the given event_id
Effects: updates the event with the given event_id to have the updated attributes

POST /volunteers

Requires: an email and password
Modifies: Volunteers
Effects: creates a new volunteer account and logs the current user in as that volunteer.

GET /volunteers/:id

Requires: a volunteer with the given id exists
Effects: Returns information about the volunteer with the given id: their bio and list of events they're participating in

PUT /volunteers/:id

Requires: the current user is logged in as the volunteer with the given id, an updated set of attributes for the volunteer
Modifies: the volunteer with the given id
Effects: the volunteer with the given id is updated with the given set of attributes

POST /volunteers/:v_id/participations

Requires: the current user is logged in as the volunteer with the given v_id; an event id
Modifies: Participations
Effects: creates a new Participation for the volunteer with the given v_id participating in the event with the given event id.

GET /volunteers/:v_id/participations/:p_id

Requires: a Participation with the given p_id exists
Effects: returns information about the Participation with the given p_id

POST /volunteers/:v_id/participations/:p_id/donate

Requires: billing information; donation amount
Modifies: Donations
Effects: charges the user for the donation amount and creates a new Donation to record the amount

POST /volunteers/:v_id/participations/:p_id/thank?donation=:d_id

Requires: The current user is logged in as the volunteer with id v_id, the donation with id d_id exists, has an email attached, and belongs to the participation with the id p_id; an email subject and body is provided
Effects: Sends a thank-you email to the donor with the given subject and body

POST /volunteers/:v_id/participations/:p_id/offline_donate

Requires: The current user is logged in as the volunteer with id v_id, the participation with id p_id exists; a donation amount, name, email, and method are provided.
Modifies: OfflineDonation
Effects: Creates an OfflineDonation for the participation with id p_id with the given attributes.

User interface

See the mockups folder.