GrowHome Django Webapp Documentation (for Frontend Engineer)

Basic Idea

- GrowHome is a social-media based web-app that aims to connect Middle Eastern entrepreneurs with each other as well as with diaspora outside the Middle East
- Entrepreneurs can also fundraise for their projects on GrowHome
- GrowHome combines social media with fundraising for the Middle East
- We plan to launch in Jordan/Palestine, and expand into the Arab world (Egypt, UAE, Tunisia, etc.) and eventually to other places in the developing world (such as Kenya, Nigeria, etc.)

How the webapp works:

- There are two types of users: contributors/normal users and entrepreneurs
- Entrepreneurs can **start projects** and **fundraise** for those projects
- Entrepreneurs can **post their projects** without fundraising
- On the project page, entrepreneurs can specify what kind of collaboration they're looking for (ex: software engineering help, legal help, accounting help, looking for a business partner, etc.)
- The website has messaging features that allow people to collaborate together without leaving the website, or as a way for them to exchange contact information
- Projects that are looking for funding hold **funding rounds**, each funding round is marked complete when the funding goal is reached
- Funding rounds can still receive more than their target
- If a funding round does not reach its funding goal, entrepreneurs still receive the money
- Users can donate to funding rounds through the website's payment system (contained in the "contribute" app in Django)
- The website has certain social media features:
    - Post:
        - Only entrepreneurs can post
        - Posts appear on the project page of the relevant project
        - Posts are shown on the feeds of the followers of the project (think of your Instagram feed)
    - Like:
        - Anyone can like, there are certain things that can be liked:
            - Check the "Like" section under "Models" and "Feed"
        - Basically, when you like a project or a post, it shows up as an activity on your personal profile page (think facebook profile page)
        - The personal profile page has nothing to do with the **project page**
        - Every user has a personal profile page

    - Follow:
        - Only projects can be followed
        - Anyone can follow a project
        - Everytime you follow someone, they are notified, and the activity is shown on your personal profile page
        - Personal profile pages cannot be followed, they exist only to keep track of the activities of the specific user
    - Comment:
        - Anyone can comment
        - Comments only appear under posts
        - Comments appear as an activity on the commenter's personal profile page
        - The commenting system is a bit primitive so make sure you understand it before writing any code for it
        - Text me if you have any questions
- Users can use the search feature to search for projects, and can filter by collaboration the project is looking for:
    - So, for example, users can search exclusively for projects that are looking for software development help, or legal help, etc.
    - The "project" model's "looking_for" field holds the information relating to this

This webapp is built on Django version 2.2, Python 3.7.7

BASIC LAYOUT

Apps: this Django webapp has 4 apps in it:
- Account (handles all account related stuff)
- Projects (handles everything related to projects and funding rounds (but not payment)
- Feed (handles all the feed related stuff like posts, likes, comments, using the getstream API Django package)
- Contribute (handles payments and records them using the PayPal API)

Packages:
- We use 5 pre-made django packages as part of the backend for this app
- Some of them are related to the front end, some are not.
- The packages with a * next to the name have some requirements for the front end:
    - Django-cron
    - stream-django*

    - Django-extensions
    - django-chatter*
    - django-paypal*
- These will be explained in detail later on in this document

MODELS

Note: All models have an “id” field, this stores the integer primary key for the model in the database, which you need for some buttons and forms
#2: if youre unfamiliar with AJAX, it makes it so that you can submit forms or make requests without reloading or redirecting to a new page
- For example, when users click “follow” or “like”, there is no change in the web page they’re viewing except the follow button changing to “following” or something, there is no refresh required to complete the operation
- The backend receives the request, initiates the “follow” relationship, and returns a success or failure message, without the end user being aware of this happening

#3: I was gonna write all the fields of all models on this document, but to not make the document too long and boring, i’ve excluded them, you can check the models.py files for each app which will show you what specific information each model holds
#4: Models marked with an (*****) are **activities**, for more info on this, check the last section of this document

Account

User:
- What:
    - Django’s default user model
    - Plenty of documentation on how you can use it
    - There are only a few fields on this model with some basic information about the user
- Why:
    - Its use is mainly for authentication purposes, since it’s easier to do with Django’s own user model
- How:
    - When a user signs up, a User model instance is created and saved in the database
    - Another model is saved in the database depending on the account type: either an instance of “Entrepreneur” or “Contributor”
    - This other model holds extra info about the user
    - you can access both the User and other model through the context at all times
- Fields:

Entrepreneur:

- What:
    - The model associated with an entrepreneur’s account
    - Stores some additional information about the user
    - On the database side, this is the instance that is referenced from the “project” model’s “creator” field
    - This is the only account type that can post, make projects, make new funding rounds
- Why:
    - To enforce permissions, enable additional features, take more appropriate data at sign-up
- How:
    - When a user signs up as an entrepreneur, an instance of the User model **and** an instance of the Entrepreneur model are created for that user
- Fields:

Contributor:

- Basically the same as the previous one but for people who don’t sign up as an Entrepreneur
- These users can **not** make projects or post
- They can only like, follow, and comment
- Fields:

Country:

- What:
    - A model that holds a string of a country’s name
- Why:
    - Used for the “country” dropdown menu at signup, so we can regulate which countries people can sign up from
    - Has no other uses, but if you need to access the name of a country make sure you use “user_obj.country.name” and not just “user_obj.country”
- Fields:
    - Name: string, the name of the country

Projects

Project*:

- What:
    - A model that holds all the relevant information about single project
    - Does not hold any information about funding rounds
- How:
    - Its created:
        - On the “new project” page, by submitting the “ProjectForm”
    - Its edited:

        - by submitting the “project form” to the “edit project” url with POST
    - Its Deleted:
        - Cannot be deleted as of now
    - Its viewed:
        - Each project has its own page with all relevant info about it
- Activity info:
    - This activity is added to the creator’s “user” feed, and is visible on their personal profile page

FundingRound*:

- What:
    - A model that holds all the information about a single funding round
    - Has a reference to the project that the funding round belongs to
    - FundingRounds have goals attached to them, Goals are separate models in the database:
        - FundingRounds have a “goals” property that returns all the goals of a funding round
- How:
    - Its created:
        - On the “new funding round” page
    - Its edited:
        - On the “edit funding round” page
        - If you would rather not have users redirect to another page to edit the project, you can allow entrepreneurs to edit the funding rounds from the project page, you just need to make sure you submit the “funding round update” form to the “edit funding round” url
    - Its deleted:
        - Cannot be deleted as of now
    - Its viewed:
        - On the project’s page
- Activity info:
    - This activity is added to the project’s feed, and is visible on the project page as well as the feeds of the project’s followers

Goal:

- What:
    - Holds the information about a specific goal. Each goal is stored separately in the database
    - Holds a reference to the funding round that it belongs to
    - Goals cannot be deleted as of now
- Why:
    - To make it easier to mark goals as complete. Reduces code complexity by a lot
    - To allow a different number of goals to be associated with each funding round (not limited to 3)

- How:
    - Its created:
        - The “AddGoalForm” needs to be POST-ed to the “add goal” url
- Fields:

Feed

Follow*:

- What:
    - A model containing the information about a specific follow action
    - Users can only follow projects
- Why:
    - To keep track of how many followers each person has
    - To manage follow/unfollow commands with the feed API using signals
- How:
    - It works:
        - When an instance of “follow” is created, a follow relationship is made between the user and the project
        - When the instance is deleted, the user “unfollows” the project
    - Its created:
        - By sending an AJAX request to the “follow” url and handling the JSON response
        - If the JSON response returns “status’ = success, either the follow instance has been created, or already existed
        - Read about JSON responses and Django, it’s very easy to parse the data and use it
    - Its deleted:
        - To unfollow, send an AJAX request to the “unfollow” url and handle the JSON response
        - Returns ‘status’ = ‘success’ if successfully unfollowed or the user never followed the project
    - To access the followers a specific project has:
        - Projects have a “followers” property that returns the number of followers the project has:
            - So if a project has 10 followers, project.followers = 20
        - If you want to get **each instance of follow**, basically info about each specific follower, there is a property called “followers_instances”
            - You can use this if you want to build functionality where users can see who the specific followers are of each project
- Activity info:
    - This activity is added to the follower’s “user” feed, and is visible on their personal profile page

Like*:

- What:

        - Holds the data for a specific like
        - Users can only “like” projects and posts
    - Why:
        - To add “like” functionality
        - To keep track of likes

    - How:
        - It works:
            - 1 instance of like = 1 unique like
        - Its created:
            - AJAX request to the “like url” and handle JSON response
        - Its deleted:
            - To unlike, send an AJAX request to the “unlike” url
        - To access the number of likes a post or a project has:
            - Posts and projects have a “likes” property which returns the number of likes they have
            - There is also a “like_instances” property, which returns each individual instance of “like” if you want to build a functionality like Instagram’s where you click on the number of likes and it shows you which users have liked it
                - Try to minimize the usage of this, as the system scales this can slow down the site a bit if each post has hundreds or thousands of likes
    - Activity info:
        - This activity is added to the liker’s “user” feed, and is visible on their personal profile page

Post*:
- What:
    - Holds the data for a specific post
    - Can optionally contain files
    - References the Project and the project’s latest funding round (if it exists)
- How:
    - Its created:
        - Submitting the post form to the “new post” url in an AJAX request
    - Its edited:
        - submitting the “post form” to the “edit post” url in an AJAX request
    - Its deleted:
        - Can delete by sending a request (i.e. visiting) the “delete post” url with proper parameters
- Activity info:

        - This activity is added to the project's feed, and is visible on the project page as well as the feeds of the project's followers

Comment*:

- What:
    - Holds the data for a specific comment
    - There are two types of comments, this type of comment is a comment made on a post
    - Comments made in response to other comments are saved as a "CommentReply" (next model on the list)
- Why:
    - Comment system
    - To make it simple to get the replies to a specific comment
- How:
    - Its created:
        - By submitting the "comment" form to the "comment on post" url with an AJAX request and handling the JSON response (check feed/views.py to see what values the view can return)
    - Its deleted:
        - send a request to the "delete comment" url with proper parameters
    - To access comments related to a specific post:
        - Each post has a "comments" property that returns a QuerySet that includes all comments related to the post (sorted oldest to newest), excluding replies to comments
- Activity info:
    - This activity is added to the commenter's "user" feed, and is visible on their personal profile page

CommentReply*:

- What:
    - Second type of comment
    - Can be in response only to a comment (and not a commentreply)
- Why:
    - To limit how many comments can be in response to each other
    - Reduce code complexity for comments, db queries
    - So replies to a comment can be easily loaded
- How:
    - Its created:
        - By sending the "comment" form to the "comment on comment" url with AJAX
    - Its deleted:
        - send a request to the "delete comment" url
    - To access the replies of each "comment" instance

        - Each comment instance has a “comment_replies” attribute which returns a QuerySet including all replies to the commnet, sorted by date (oldest to newest)
    -
- Activity info:
    - This activity is added to the commenter’s “user” feed, and is visible on their personal profile page

Contribute

Note: The flow of the payment system should go like this:
- Users submit the “pledge form” from the project page to “growhome.app/contribute”
- The contribute page shows them a paypal payment form (which leads them to the paypal website, or does a pop-up payment page), as well as a form to change the pledge amount
    - If users want to change the amount, they have to submit the “pledge” form to the “edit pledge” url, which will reload the “contribute” page with the desired values
- Users are redirected (by paypal) to the “done” or “canceled” page, depending on whether the payment is successful

Contribution:
- What:
    - A model that holds the information of a specific successful transaction
    - Has data about who made the payment and who its intended for
    - **Contributions can have negative value in the case of someone’s bank reversing the transaction or PayPal forcing a refund**
- Why:
    - To keep track of money changing hands
    - Its needed in order to develop a system to automatically tell us which balances are due each week (i haven’t done this yet so if you have any ideas or you know how to do it then lmk)
- How:
    - Its created:
        - The PayPal server sends a special signal to our server, which handles the request
    - Its deleted:
        - Cannot be deleted
    -
- Fields:

Pledge:
- What:
    - Before a contribution is successfully processed, it is saved as a pledge
- Why:

        - Mostly for a very specific backend purpose, you can ignore it for the most part
        - Its used to get statistics about how many people click on “contribute” vs how many contributions are actually fulfilled
        - Also so when users go to “growhome.app/contribute” it’ll show them their most recent unfulfilled “pledge”, meaning it should always have something to show
    - How:
        - Its created:
            - Submitting the “pledge form” to “growhome.app/contribute”

FORMS

Note: You need to include a CSRF token in all form submissions

Account

EntrepreneurSignup:

- **What it is:**
    - Signup form for “entrepreneurs”
- **Where it should be:**
    - On the entrepreneur sign up page
- **Where it should submit:**
    - growhome.app/account/signup_entrepreneur/
- **Submit method:** POST

ContributorSignup:

- **What it is:**
    - Signup form for “contributors”
- **Where it should be:**
    - On the contributor sign up page
- **Where it should submit:**
    - growhome.app/account/signup/
- **Submit method:** POST

SignInForm:

- **What it is:**
    - Sign in for all user
- **Where it should be:**
    - On the sign in page

- **Where it should submit:**
    - growhome.app/account/login/
- **Submit method:** POST

BioUpdateForm:
- **What it is:**
    - A form for “entrepreneurs”
    - Allows them to change their bios
- **Where it should be:**
    - On the user’s profile page (if they’re the ones viewing the page)
- **Where it should submit:**
    - growhome.app/account/update_bio/
- **Submit method:** POST

Projects

FundingRoundForm:
- **What it is:**
    - Form for creating a new funding round
- **Where it should be:**
    - On the “new funding round” page
- **Where it should submit:**
    - growhome.app/projects/<int:project_id>/new/
    - project_id should be the id of the project instance, project will be passed through context so you can get the value with “project.id”
- **Submit method:** POST

FundingRoundUpdateForm:
- **What it is:**
    - Form for editing an existing funding round
- **Where it should be:**
    - On the “edit funding round” page, or on the project page if you would rather that the users dont have to redirect to a new page to edit a funding round
- **Where it should submit:**
    - growhome.app/projects/<int:project_id>/edit/
    - project_id should be the id of the project instance, project will be passed through context so you can get the value with “project.id”
- **Submit method:** POST

AddGoalForm:
- **What it is:**
    - Form for adding a goal to a funding round
- **Where it should be:**

    - On the “project” page, the form is passed if the project owner is the one viewing the project, and they have an active funding round
- **Where it should submit:**
    - growhome.app/projects/<int:project_id>/addgoal/
    - project_id should be the id of the project instance, project will be passed through context so you can get the value with “project.id”
- **Submit method:** POST

Feed

PostForm:
- **What it is:**
    - A form for creating a new post
- **Where it should be:**
    - On the “project” page, if the project creator is the one viewing the page
    - On the “post” page
    - On the homepage? Idk its kind of up to you
- **Where it should submit:**
    - growhome.app/feed/new/<int:pid>/
    - Pid is the project id
- **Submit method:** POST

CommentForm:
- **What it is:**
    - Form for creating a new comment
- **Where it should be:**
    - In the feed, as part of the template of “post”
    - On the page that shows a single post
- **Where it should submit:**
    - growhome.app/<int:project_id>/addgoal
    - project_id should be the id of the project instance, project will be passed through context so you can get the value with “project.id”
- **Submit method:** POST

SearchForm:
- **What it is:**
    - Form for search
    - Should take the form of a search bar and filters on the side of the page
- **Where it should be:**
    - On the “search page”
- **Where it should submit:**
    - growhome.app/search
- **Submit method:** GET

Contribute

PledgeForm:

- **What it is:**
    - A one-line form for submitting the desired amount of contribution to a project
    - Has one input, an integer
- **Where it should be:**
    - On the project page, should look like a small text box with a button below it that says “contribute” or something like that
    - on the “contribute” page as a way for users to edit their desired amount of contribution
        - In this case the button under should say “edit”
- **Where it should submit:**
    - growhome.app/contribute/
- **Submit method:** POST

URLS FOR BUTTONS AND HYPERLINKS

Note: You don’t have to know all of the info in this section, it’s just for if you need to link something and need to know the URL

Account

growhome.app/account/

Name:

- Personal profile page

What is here:

- This url leads to the “profile” page, which renders the “view.html” template with the user’s own data

growhome.app/account/__username__

Name:

- Profile page

What is here:

- This url leads to the “profile” page, which renders the “view.html” template with the information of the user who’s username is passed instead of __username__

growhome.app/account/signup_entrepreneur

Name:

- Entrepreneur signup page

What is here:

- Loads entrepreneur signup template

growhome.app/account/signup

Name:

- Signup page

What is here:

- Signup page for “regular” users
- Loads contributor signup template

growhome.app/account/login

Name:

- Login/signin page

What is here:

- Login page which loads the login template

growhome.app/account/logout

Name:

- Logout url

What is here:

- Logs the user is out if they are logged in, and redirects them to the homepage

growhome.app/account/update_bio

Name:

- Update bio url

What is here:

- This is the URL that the “update bio” form needs to be POST-ed to
- After processing the data, it returns them to the their profile page

growhome.app/account/change_password

Name:

- Change password url

What is here:

- Page for changing password

Projects

growhome.app/projects/__slug__

Name:

- View project page

What is here:

- This page shows the “view.html” template of the project app

- Uses the information of the project who’s “slug” attribute is equal to what is passed as “__slug__”

growhome.app/projects/new
Name:
- New project page

What is here:
- This page leads to the “new project” page, which shows the “new.html” template of the projects app
- Uses the information of the project whose “slug” attribute is equal to what is passed as “__slug__”

growhome.app/projects/delete/__int__
Name:
- Delete project url

What is here:
- Url that deletes the project whose “id” attribute is placed instead of “__int__” in the url
- If the person that visits this URL is the creator of the project whose “id” is passed, the project will be deleted
- Projects **cannot** be deleted once they have received funding
- Whether or not deletion is successful, the user is redirected to the home page

growhome.app/projects/__int__/new
Name:
- New funding round page

What is here:
- This page renders the “newfundinground.html” template of the projects app
- The id of the project that the funding round is for should be passed instead of “__int__”

growhome.app/projects/__int__/edit
Name:
- edit project page

What is here:
- This page renders the “edit.html” template of the projects app
- The id of the project should be passed instead of “__int__”

growhome.app/projects/__int__/edit_fundinground
Name:
- edit funding round page

What is here:
- This page renders the “editfundinground.html” template of the projects app
- The id of the project the funding round is related to should be passed instead of “__int__”

growhome.app/projects/__int__/addgoal
Name:
- Add goal url

What is here:
- This page processes the POST submission of the “add goal” page
- The id of the project the goal is related to should be passed instead of “__int__”

Feed

growhome.app/
Name:
- Homepage
- Feed index

What is here:
- The homepage of the whole site for logged in users
- Not logged in users are redirected to “growhome.com/about”

growhome.app/__int__/new
Name:
- New post url

What is here:
- This url is meant to take submissions of the “post” form
- This page accepts only POST requests, and returns a JSON response
- The id of the project the post is related to should be passed instead of “__int__”
- The user is redirected to the project page after completion

growhome.app/__int__/edit
Name:
- edit post page

What is here:
- Shows the “edit post” template
- The post’s “post_identifier” field should be passed instead of __int__

growhome.app/__int__/delete
Name:
- Delete post url

What is here:
- Sends a request to a view which deletes the intended post and redirects to the project page
- The post’s “post_identifier” field should be passed instead of __int__

growhome.app/__int__/follow

Name:
- Follow url

What is here:
- Sends a request to a view which “follows” the intended project from the user’s account and returns a JSON response
- The project’s “id” field value should be passed instead of __int__

growhome.app/__int__/unfollow
Name:
- unfollow url

What is here:
- Sends a request to a view which “unfollows” the intended project from the user’s account and returns a JSON response
- The project’s “id” field value should be passed instead of __int__

growhome.app/__str__/__int__/like
Name:
- like url

What is here:
- Sends a request to a view which “likes” the intended project from the user’s account and returns a JSON response
- The project or post’s “id” field value should be passed instead of __int__
- __str__ can be either “post” or “project”

growhome.app/__str__/__int__/unlike
Name:
- unlike url

What is here:
- Sends a request to a view which “likes” the intended project from the user’s account and returns a JSON response
- The project or post’s “id” field value should be passed instead of __int__
- __str__ can be either “post” or “project”

growhome.app/post/__int__
Name:
- Post view page

What is here:
- Page that shows the post and all its comments
- The post’s “post_identifier” field value should be passed instead of __int__

growhome.app/search
Name:
- Search page

What is here:
- Shows search template

growhome.app/explore
Name:
- explore page

What is here:
- Shows explore page template (which is the same as the search template)

growhome.app/comment/new/__int__
Name:
- Comment on post url

What is here:
- A view that handles new comments, returns JSON response
- "__int__" should be the "id" attribute of the post that this is a comment on

growhome.app/comment/reply/__int__
Name:
- Comment on comment (comment reply/reply comment) url

What is here:
- A view that handles comments as replies to other comments, returns JSON response
- The "id" of the comment that this comment is a reply to should be passed instead of "__int__"

growhome.app/comment/delete/__int__
Name:
- Delete comment url

What is here:
- A view that handles comment deletion, returns JSON response
- The id of the comment or commentreply to be deleted should be passed instead of "__int__"

Contribute

growhome.app/contribute
Name:
- Contribute/contribution page

What is here:
- The page that shows the Paypal form and contribution summary
- Accepts POST request of the "pledge form"

TEMPLATES

Note: Read about Django template extending before writing any code: you need to make a base template with elements that are loaded on each page - there are a couple of requirements for this base template:

- A navigation bar with
    - A search bar that submits its content using "GET" to the search url (check URL section),
        - the name of the input field on the search bar should be 'query'
    - Some way of showing notifications (more on this later)
    - Button that leads to the profile page
    - Button that leads to the messaging page (growhome.app/chat)
    - If the user is an entrepreneur, a button to the entrepreneur dashboard

Note: If you decide to **not** use Django's form rendering functionality in the templates, make sure that the values of all html input fields are set to what they should be (you can get these values from the form passed through the context)

Note: Some things will always be available through context **if the user is logged in.** If the user is not logged in, these values will **not** be in the context :

- Login_user = instance of the User object, corresponding to the user that is logged in, is equal to None when the user is not logged in
- User_object = an instance of Entrepreneur or Contributor, depending on the type of the user
- Type = string, can either be "contributor" or "entrepreneur"
- Notifications = notification feed
- All_likes = list of all things liked by this user
- All_follows = list of all project followed by this user
    - The use of the two above context variables is to check whether a user has already liked or followed the thing that's being shown
    - So if the user has already followed a project, when they view the project (or a post related to the project), the "follow" button will turn into a "following" button, and the same for the "like" button
    - This is pretty trivial, should take only a couple of lines of code, but makes the website much more functional, as users know what they've liked
    - Read the section on feeds at the end, basically how this will look like in the template (in psuedocode):
        - For activity in activities:
            - render activity
            - If activity.object in all_likes:
                - Change the "like" button to "liked"

            - If activity.actor in all_follows:
                - Change the “follow” button to “followed”
        - This doesn’t scale very well, it’s just a shortcut for now, it’ll probably have to be changed later on for efficiency

Account

entrepreneursignup.html:

Description:

- A signup page for entrepreneurs

Requirements:

- Show EntrepreneurSignup form and error fields
- Show information about signing up, what it means to be an entrepreneur on this platform
- Info on how to apply for an invite code

Context:

- form = EntrepeneurSignup (form)

contributorsignup.html:

Description:

- Signup page for non-entrepreneurs (referred to in the code as “contributors”)

Requirements:

- Show contributor signupform and error fields
- Show information about signing up, what it means to be a contributor on this platform

Context:

- form = ContributorSignup

signin.html:

Description:

- Sign in page

Requirements:

- Show SignInForm and error fields
- Show any relevant extra information

Context:

- form = SignInForm

view.html:

Description:

- Profile page for all users

Requirements:

- Show information about the user that is passed through the context, including activities streams (check end of the document for what this means)
- Be able to handle either contributor account type or entrepreneur account type

        - Note: This template can be split up into two different templates to handle each account type if it would be easier. If you decide to do this let me know.
    - Keep in mind all QuerySets can have a value of “None”, depending on what data is available about the user (ex: if the user has not posted anything, the ‘posts’ attribute in the context will be None)
    - There should be a button for sending a direct message to this person (check the second to last section of this document on messaging)

Context:

- Entrepeneur only:
    - posts = QuerySet of Post objects
    - projects = QuerySet of Project objects
- Both:
    - country = Country object
    - activities = QuerySet of enriched activities (likes, follows)
- Profile owner only:
    - contributions = QuerySet of Contribution objects
    - bio_update_form = BioUpdateForm

Change_password.html:

Description:

- A page for changing your password
- All you really have to do for this one is render the form on the page

Context:

- Form = Django’s PasswordChangeForm

Projects

dashboard.html:

Description:

- A dashboard for entrepreneurs, that helps them access whatever the need to, quickly
- You can make this look however you want, there are no real requirements for it
- You can show total $ raised, or number of total likes, views, etc. depending on what you feel would be most usable and useful
- If you need info passed through context lmk and I will write the backend code

Context:

- projects = all projects created by the user

new.html:

Description:

- Page for creating a new project

Requirements:

- Show form for creating new project, and error fields
- Any relevant extra info about project creation

Context:

- form = ProjectForm

edit.html:

Description:

- Page for editing an existing project

Requirements:

- Show form for editing project, and error fields
- Any relevant extra info about projects

Context:

- form = ProjectForm, prepopulated
- project = Project object instance

newfundinground.html:

Description:

- Page for creating a new funding round

Requirements:

- Show form for creating a new funding round, and error fields
- Any relevant extra info about funding rounds

Context:

- form = FundingRoundForm

editfundinground.html:

Description:

- Page for editing an existing funding round

Requirements:

- Show funding round update form, and error fields
- Any relevant extra info about funding rounds

Context:

- form = FundingRoundUpdateForm, prepopulated
- funding_round = FundingRound object for the relevant funding round

view.html:

Description:

- Project view page, shows all the info for a specific project

Requirements:

- Show project information and activity feed
- Show most recent funding round's information (with option to view past funding rounds)

- Show PledgeForm on the page for those who want to contribute
- Show social media share buttons
- Show like/follow buttons
    - Info about whether a user has already liked or already followed will be available through context
- Show project creator-specific functions if project creator is the one viewing:
    - Forms:
        - PostForm
        - If the user has active funding rounds, AddGoalForm is also passed through context
    - Buttons:
        - Edit button (redirect to edit project page)
        - Edit funding round button (redirect to edit funding round page)
        - If funding_round.goals_finished: show new funding round button (redirect to new funding round page)

Context:

- Activities = list of enriched activities (posts, funding rounds), i.e. all posts, and funding round declarations attached to a project (in the template you should render these using the render_activity template tag, check the last section of the doc)
- Project = Project object instance
- Funding_round = FundingRound object instance, the most recent funding round
- Past_rounds = QuerySet of finished funding rounds, completed funding rounds
- Is_creator = Boolean value. Is true when the project creator is viewing the project
- liked = Boolean value. True if the user has liked this project
- following = Boolean value. True if user is following the project
- Only passed when project creator is viewing project:
    - If the user has active funding rounds, add_goal_form = AddGoalForm
    - post_form = PostForm
- **Not** passed when project creator is viewing project but is passed otherwise:
    - pledge_form = PledgeForm

Feed

About.html:

- This template is independent of the rest of the site
- You do not have to import the base into this one if you want to make it look different
- The only information this page should show is info about the platform
- This will be the site people are taken to when they go to growhome.app and they're not logged in
- You have total freedom on this, there are no technical requirements

index.html:

Description:

- This is the homepage of the whole site for users that are logged in, it shows the timeline feed for users (which contains updates from projects they follow). You can make this look however you want, but make sure it shows all the relevant stuff. Try to model it loosely around the twitter homepage (in terms of functionality)
- If you want to add any extra features and need something passed through the context lmk or make the appropriate change to the “index” view in the “feed” app

Requirements:

  - You need to use activity rendering and handle feeds from the stream-django plugin (check the “activity streams” section at the end of the doc)
  - Should be able to handle both types of accounts, you decide if it looks the same for contributors and entrepreneurs or if entrepreneurs have extra functionality

Context:

  - activities = list of enriched activities to be rendered with the render_activity template tag

edit.html:

Description:

- Page for editing a post

Requirements:

- Show edit post form, and error field
- Link to the project the post is associated with

Context:

- post = the Post object
- form = PostForm, pre-populated

post.html:

Description:

- A page displaying a single post, and all the comments associated with it

Requirements:

- Show comments and comment replies
- Show number likes, number of comments
- Social media share buttons
- Link to the project the post is associated with
- If viewed by the post’s author:
  - Edit post button
  - Delete post button

Context:

- post = the Post object
- Comments = QuerySet of Comment objects (check the documentation of the comment model to see how the comment reply system works)

search.html:

Description:

- This template is used for both search and explore pages. It takes the search parameters passed to it and displays the search results (if you just go to growhome.app/search or growhome.app/explore, it returns the featured projects)

Requirements:

- Show all projects passed to it in an easily readable way, (these are serach results)
- You decide how the 'pages' system will work for the results, there is a Django paginator object passed through the context, but you don't have to use it
- Show the SearchForm, which includes the search bar and the additional filters
    - There are some small things that need to be done so that the search form works properly
        - 1. The search form needs to be submitted with a GET instead of POST request
        - 2. If you decide to use django pagination, there needs to be a hidden integer input field 'page_number' which is sent along with the form request. This is set by the backend, and the user doesn't interfere with it. Its used to keep track of page numbers so the view can return the correct page each time
        - 3. If you use Django pagination, the "next" button needs to be a form submit button (i.e. you need to make a form submission to get the next page)
        - 4. Like mentioned earlier, if you do not use Django form rendering, make sure the values of the form input fields are equal to the initial value for the fields on the Form object

Context:

- projects = page from Paginator object showing 20 search results
- form = SearchForm, prepopulated
- all-projects = all search results (if you want to implement a page system with JS that doesn't involve multiple requests, you can use this)

Contribute

process.html:

Description:

- This template is for the processing of payments, it shows the contribution amount, the paypal form, and an input field to change the contribution amount before the user pays.

Requirements:

- Show the Pledge amount
- Show the PledgeForm, which on submit sends the data to the edit pledge page
- Render the paypalpaymentsform with {{ form.render }}
- Show information about how money is handled

Context:

- Form = PayPalPaymentsForm

done.html:
Description:
- A page with a success message for successful payments, paypal will redirect users here when a payment is successful

canceled.html:
Description:
- A page with a failure message for failed payments, paypal will redirect users here when a payment has failed

MESSAGING/CHAT

- You dont really have to do much for this, i'm using a pre-made django package
- https://github.com/dibs-devs/chatter
- You can customize the template if you want
- The documentation shows what links to go to to initiate specific actions

ACTIVITY STREAMS

You can ignore the notifications stuff for now if needed

- For social media style feeds, I'm using a feed API.
- https://github.com/GetStream/stream-django
- Check the templating section of the API documentation ^, and make sure you understand that as well as what an enriched activity is
- Any model that's marked as an activity should have its own corresponding template:
    - Its up to you what each activity looks like
    - whenever one of the models marked "activity" is saved and added to a feed, the template corresponding to that activity will be rendered with the enriched activity passed as context
- There are three different "feeds"
    - There's the user feed, which shows the activities of a specific user (liking or commenting), think of it like a facebook profile feed
    - There's also a "project" feed, which is the only kind of feed that can be followed. It shows post and funding round activities shared by projects. It's like the user feed but for projects
    - The notification feed: shows when specific actions happen related to the user
- All notifications implemented so far:
    - Likes (notifies the post/project creator)
    - Follow (notifies the person being followed)

- Comment (notifies the post/project creator)