Opus Team

User and Developer Documentation

04/30/2021

Swopnil Shrestha | David Will | Josh Van Sant

Table of Contents

Product

Vision

Opus Team's vision is to be a free, fast, and Open Source solution to team management and communication. It should reduce the dependency on organizational data going to 3rd parties and enable a self hosted solution for team management where organizations can fork and develop customized solutions to their specific use cases.

Mission

Opus Team's mission is to provide fast, open source, scalable and customizable collaboration solutions for small teams.

Key Solutions

- First-Party Data Ownership
- Free and Open Source
- Team Collaboration
- Minimal and Customizable

User Documentation

This documentation is deployed at https://docs.opusteam.us/

We designed Opus as a lightweight team management platform to provide small teams with the facilities to collaborate and connect with their fellow teammates. While there's no limit to how many people can be added to an Opus team, we feel that our services are best fitted to teams of less than 75 people.

Structure

When a user joins Opus, they will not be part of any teams. To access the tools on the site, they will need to either receive an invite from a user in a team that already exists, create their own team, or join an existing one. Once they are a member of a team, they will enter the team at the lowest permission level and can interact with the team features as usual. Teams in Opus have two tiers: a main team, and subgroups of that team. Subgroups can be used to divide up the team into specialized sections, perhaps by department, instrument, or position, based on the purpose of the team.

Current Features

Below is a list of tools and features that we've implemented as of May 2021. Announcements, Calendar, and Contacts pages all display their corresponding data in a table view, using the NPM package 'react-bootstrap-table-next'. This allows each of them to be sorted by any column the user chooses.

Announcements

Users within a team can create an announcement that all other users in that team can view. Each announcement is displayed in a table view with its priority (High, Medium, Low), the user that created it, the team it was created for, the expiration date of the announcement, its associated event (if it has one), and the announcement message. Announcements can be filtered by team and priority using the corresponding dropdown menu or by accessing the announcements route paired with the desired team: '/announcements/{teamName}'. Announcements have an expiration date, and will be deleted automatically after that date has passed.

Calendar

Users can create events that are associated with their team, which are then visible to all members of that team. Alternatively, users can create events that are between them and any of their contacts, without being linked to a specific team. In this case, the event is only visible to people who were invited. The events table can be filtered by team and can also display all events without a corresponding team.

Contacts

Contacts simply displays all of the users that the current user shares at least one team with. Like all of the tables, it is sortable, and the username of each row in the contact table links to the profile page of that specific person.

Teams and Subgroups

Users can create new teams by navigating to their teams page and clicking 'Create Team'. This will prompt the user for a name, and will create the new team and add it to the user's teams once they submit. On each team's specific page, users can invite new members to the team or remove members who are already in the team, and the 'Members' widget reflects these changes. Additionally, users can create and delete subgroups within the team, which are displayed in the 'Groups' widget. Currently, no functionality exists for these subgroups, but we have plans to create unique pages and permissions for each of them in a manner analogous to the teams.

Widgets

Widgets are available for each of the tools that Opus provides. Each widget provides a snapshot of the tool for the user by displaying three recent items in list form.

- On the user's dashboard: Widgets are displayed for Announcements, Calendar, Contacts, and Teams. Each one of these widgets links to its corresponding feature; in addition, the Teams widget also has individual links to the pages for each of the teams that it displays.
- On a team's page: Like the user dashboard, widgets are displayed for each of the current features. However, when viewing widgets on an individual team's page, the widget will only show data for that specific team. Additionally, the widgets will link to the corresponding feature, but the data that is shown will automatically be filtered by the team that the user navigated from. For instance, if a user clicks on the Announcements widget from the 'Basketball' team page, they will be redirected to the route '/announcements/Basketball', which will initially be filtered to show all announcements for the 'Basketball' team.

Future Features

Time constraints, testing, and documentation have limited the number of features we could implement before the end of the year. The following section includes proposed enhancements to Opus that could be taken on by future developers.

Chat Service

Many messaging platforms already exist, but it would be a bonus to include a chat system within the Opus app where users could stay in touch with contacts from the various teams that they are a part of. We would also like to implement a group chat feature so that users could have a running conversation with their entire team or subgroup, depending on the team's needs.

Permissions

While we had planned to include this in the MVP, time constraints and backend configurations required us to move this feature to after MVP. We plan to implement this on the backend with three lists for each team, each containing the ids of the corresponding users: 'owners', 'managers', and 'users'. Owners have full access to the site's features including editing a team name, promoting members up to owner, and deleting the team that they are the owner of. Managers can create team-wide events and announcements, remove and add users to the team, create new subgroups, and promote members up to manager. Users will have the ability to leave teams, create events between themselves and other users, and view announcements and events of their corresponding team. Each permission level inherits the permissions of the levels below it. These lists can then be fetched from the API to determine what level of access the current user is allowed on the site.

To Do List

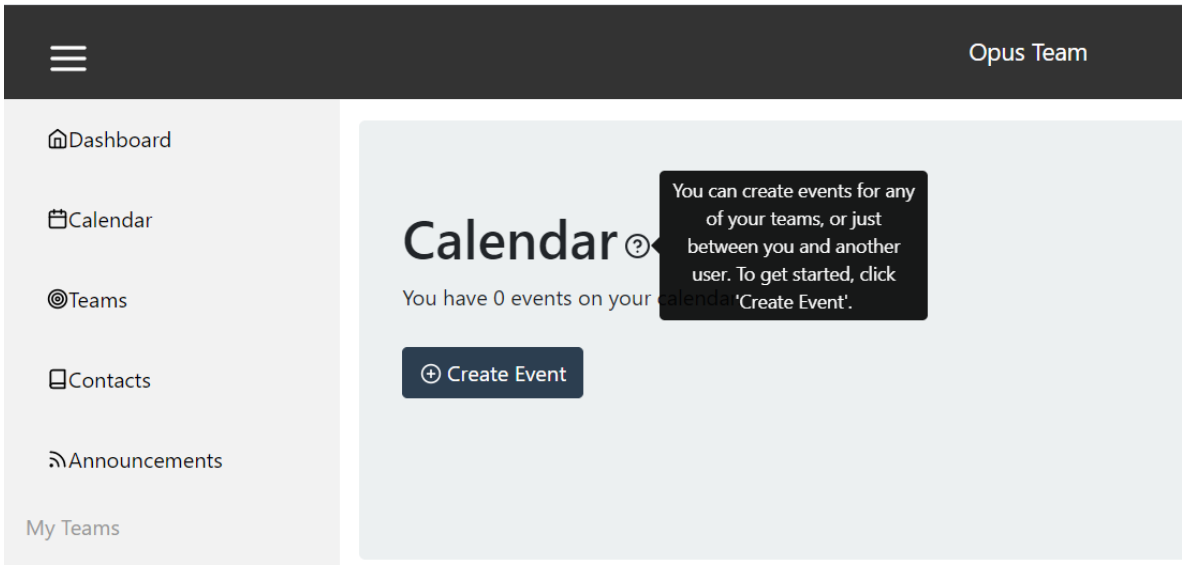
Users would be able to add or remove 'To-Do' items that can be assigned to other members of their teams, the entire team, or themselves. Users could then view their assigned to-dos or team

to-dos in a manner similar to Announcements, and then could mark items as ‘To-Do’, ‘In Progress’, or ‘Completed’, and so on.

Tooltips

Throughout the application, tooltips appear with a question mark icon. When a user hovers over these, they display some basic information about the page that the user is on, and provide some suggestions for what actions the user might take next. Below is an index of tooltip locations and their messages:

- Announcements: “Announcements are messages that are visible to entire teams. To get started, click ‘Create Announcement’.”
- Calendar: “You can create events for any of your teams, or just between you and another user. To get started, click ‘Create Event’.”
- Contacts: “Anyone that you share a team with is in your contacts. You may visit their profile page by clicking on their username.”
- Teams: “Teams enable collaboration between members. Upon joining or creating a team, you will be able to see that team's announcements, events, and members.”
- TeamView: “This is the homepage for your team. You can add or remove members, view announcements and events, leave, or delete the team from here.”
- TeamView (Groups Widget): “This feature is under development and is not currently functional. In the future, you'll be able to create subgroups within a team!”



Calendar tooltip as it appears when a user hovers over the icon

Developer Documentation

Development Environment

- **VSCode** is our preferred IDE for the project.
- **Django**
    - We use pipenv as a packaging tool and virtual environment. We also keep a separate requirements.txt in the repo in the case that we need to rebuild our environments individually.
    - It should be noted that on Apple devices psycopg2 will raise errors if psycopg2-binary and postgresql are not configured properly.
    - Django has a database configuration section in 'settings.py'. When deploying in heroku the url of interest must be input here.
- **React**
    - We use Node Package Manager(NPM) as the Node.js oceanfront manager. Yarn, in addition to other node package managers can also be used as an alternative.
    - When deploying, Heroku automatically builds an optimized react app based on the build pack defined.

Below are lists of external libraries that are used by the applications. While other versions of the dependencies can be used to replicate the intended functionality, it is recommended that the versions listed are used before development. To install the dependencies, on a terminal shell (preferred Bash or Z-Shell) a developer may run

```
$ pip install -r requirements.txt
```

to install all the required dependencies on the backend, given Python 3.5+ is installed on the host system.

For the frontend a developer must run

```
$ npm i && npm start
```

from the root directory to install all the frontend dependencies, given Node v15+ is installed in the host system.

After running these commands, the developer must run

```
$ python3 manage.py runserver 0.0.0.0:8000
```

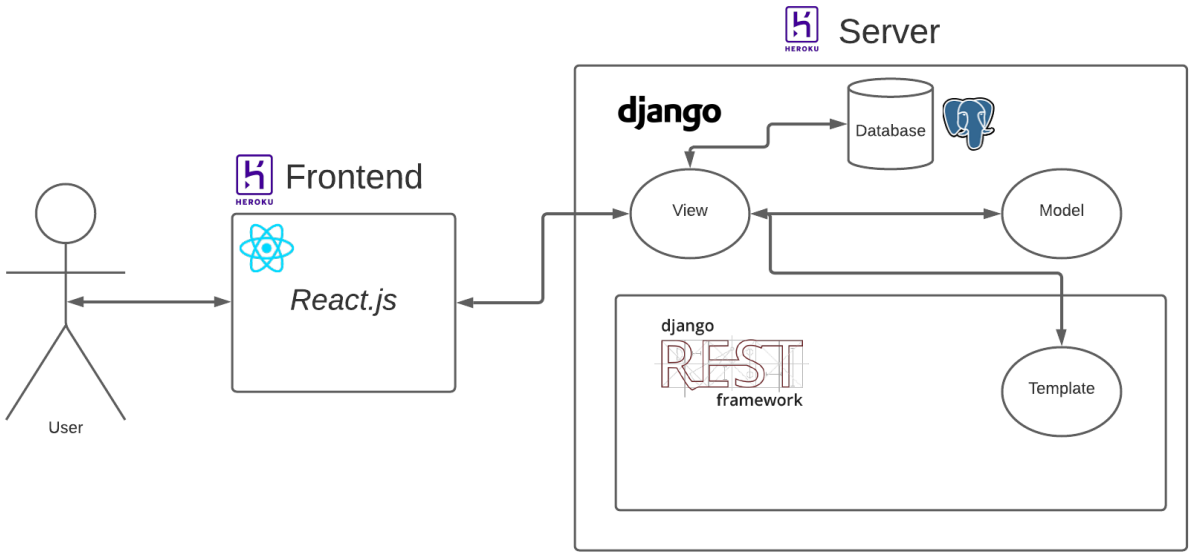
to start a local instance of the backend server, after which a local instance of both the backend and frontend will run on the host system.

Backend Dependencies	**Frontend Dependencies**
appdirs==1.4.3 asgiref==3.2.10 CacheControl==0.12.6 certifi==2019.11.28 cffi==1.14.3 chardet==3.0.4 colorama==0.4.3 contextlib2==0.6.0 cryptography==3.2.1 distlib==0.3.0 distro==1.4.0 dj-database-url==0.5.0 Django==3.1.2 django-cors-headers==3.5.0 django-extensions==3.1.0 django-heroku==0.3.1 django-rest-knox==4.1.0 djangorestframework==3.12.1 djangorestframework-jwt==1.11.0 gunicorn==20.0.4 html5lib==1.0.1 idna==2.8 ipaddr==2.2.0 lockfile==0.12.2 msgpack==0.6.2 packaging==20.3 pep517==0.8.2 progress==1.5 psycopg2==2.8.6 psycopg2-binary==2.8.6 pycparser==2.20 PyJWT==1.7.1 pyparsing==2.4.6 pytoml==0.1.21 pytz==2020.1 requests==2.22.0	"@testing-library/jest-dom": "^5.11.5", "@testing-library/react": "^11.1.0", "@testing-library/user-event": "^12.1.10", "axios": "^0.21.0", "bootstrap": "^4.5.3", "express": "^4.17.1", "jquery": "^3.5.1", "react": "^17.0.1", "react-bootstrap": "^1.4.0", "react-bootstrap-table-next": "^4.0.3", "react-dom": "^17.0.1", "react-icons": "^4.1.0", "react-input-color": "^3.0.1", "react-router-dom": "^5.2.0", "react-scripts": "4.0.0", "three": "^0.124.0", "vanta": "^0.5.21", "web-vitals": "^0.2.4"

retrying==1.3.3 six==1.15.0 sqlparse==0.4.1 toml==0.10.1 urllib3==1.25.8 webencodings==0.5.1 whitenoise==5.2.0	

Architecture

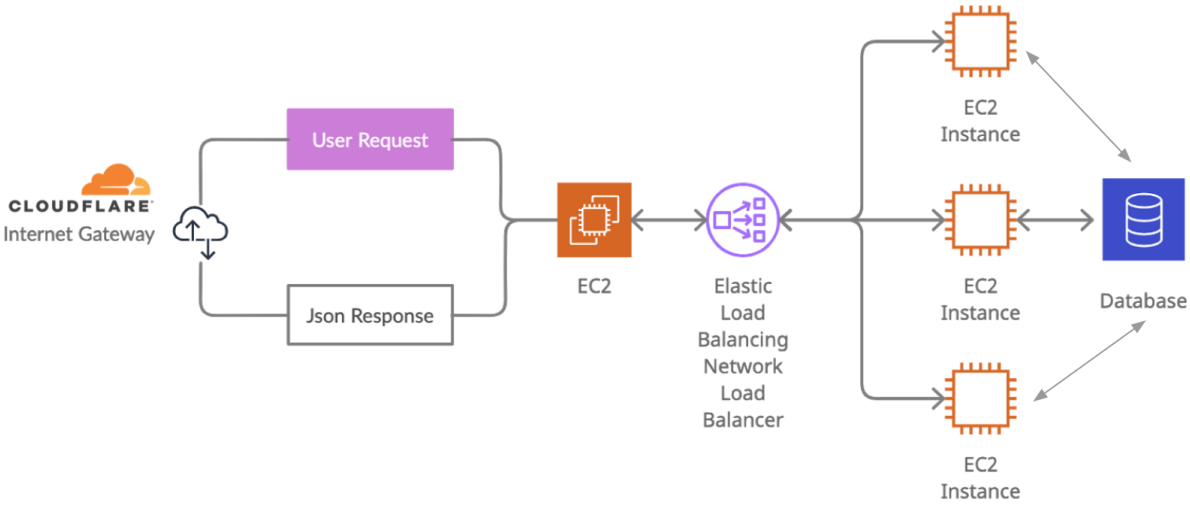
As demonstrated in the picture below, Opus uses React.js for the frontend, Django REST Framework for the backend, a PostgreSQL database, and Heroku and AWS for deployment.



Infrastructure

In terms of infrastructure, Opus is dynamically scaled and creates cloud instances based on the number of users. The Opus Team backend server runs on Amazon's Web Services's Elastic Compute Cloud 2 Linux Virtual Machine Instances. Each instance is capable of supporting close to a hundred simultaneous users, after which the Elastic Load Balancer (ELB) will clone the current instance to support a few hundred more users.

To prevent unexpected surges in cost, the maximum simultaneous instances have been limited to 100 which can support up to 10,000 simultaneous users.



Amazon Web Services Identity and Access Management

Currently access to the Amazon Web Services account for Opus Team has been limited to the internal team. AWS security groups have been set up so that Open Source contributors, if needed, may request access to limited features of the AWS Console by accessing the Opus Team Amazon Web Services Console, available at opus-team.aws.amazon.com/console.

Access to the AWS console by sending an email to contribute@opusteam.us with details of the contributor's name, email and the purpose of access as well as any associated costs with their proposed cloud solution.

Security Groups

Security groups have been set up to comply with AWS's Shared Responsibility Model, which promotes that only essential permissions be given to the AWS console for any project developers.

Domain Host and Naming Conventions

The current Domain Host that Opus is using is CloudFlare. CloudFlare access, like AWS is limited to the core team. Any changes to the domain, subdomain and Domain Name System Settings can be proposed by sending an email to contribute@opusteam.us.

The current DNS configurations include:

Record Type	Name	Content	TTL	Proxy status	Comment
CNAME	api	opusteam-api.herokuapp.com	Auto	Proxied	Backend Hosting
CNAME	_b3874ea3941d5d714f27dd51506405bb	_2c82d4693c62a0ad1ebb6a1535d9b474.zjfbrrwmzc.acm-validations.aws	Auto	DNS only	Google Verification
CNAME	docs	opustm.github.io	Auto	Proxied	Documentation Hosting
CNAME	_domainconnect	connect.domains.google.com	Auto	Proxied	Google Domain Connect
CNAME	opusteam.us	d2is42j4y7om3m.cloudfront.net	Auto	DNS only	Domain Security
CNAME	www	d2is42j4y7om3m.cloudfront.net	Auto	DNS only	Domain Security
MX	opusteam.us	alt4.gmr-smtp-in.l.google.com 40	Auto	DNS only	GMail Forwarding
MX	opusteam.us	alt3.gmr-smtp-in.l.google.com 30	Auto	DNS only	GMail Forwarding
MX	opusteam.us	alt2.gmr-smtp-in.l.google.com 20	Auto	DNS only	GMail Forwarding
MX	opusteam.us	alt1.gmr-smtp-in.l.google.com 10	Auto	DNS only	GMail Forwarding
MX	opusteam.us	gmr-smtp-in.l.google.com 5	Auto	DNS only	GMail Forwarding
TXT	_github-challenge-opustm	a576074b28	Auto	DNS only	GitHub Verification

Code Quality Assurance

Frontend Linting and Code Formatting

The frontend uses both ESLint and Prettier for code formatting and linting to ensure consistency in syntax and style across multiple developers. Our specific rules are defined in the configuration files at the top level of the /src/ folder and may be changed by editing that file. For more information on either of these tools, visit https://eslint.org/ and https://prettier.io/.

For the frontend repository, linting can be configured with the .eslintrc file. To generate a log of all linting errors, navigate to the source folder for the frontend and write to the console:

```
$ eslint --write .
```

Along with linting code formatting standards have also been followed to ensure all code is consistent across the repository.

Here are some general guidelines and recommendations for code formatting:

Basic guidelines for formatting code samples:

- Don't use tabs to indent code; use spaces only.
- Follow the indentation guidelines in the relevant coding-style guide. For most programming languages, that means indent by two spaces per indentation level, but some contexts, such as Python and the Android Open Source Project, use four spaces instead. This guidance applies to formatting code samples, not to formatting commands.
- Wrap lines at 80 characters.
  If you expect readers to have a relatively narrow browser window, or to print out your document, consider wrapping at a smaller number of characters for readability.
- Mark code blocks as preformatted text. In HTML, use a <pre> element; in Markdown, indent every line of the code block by four spaces.

To ensure everyone follows the same code format, we have set up a npm script that before build verified that the code formatting standards have been followed. If not followed, the script will automatically go and reformat all the code in the entire repository that has formatting errors.

The code below details eslint configuration settings. For the full document, refer to https://github.com/standard/eslint-config-standard/blob/master/eslintrc.json

```
{
  "parserOptions": {
    "ecmaVersion": 2021,
    "ecmaFeatures": {
      "jsx": true
    },
    "sourceType": "module"
  },

  "env": {
    "es2021": true,
    "node": true
  },

  "plugins": [
    "import",
    "node",
    "promise"
  ],

  "globals": {
    "document": "readonly",
    "navigator": "readonly",
    "window": "readonly"
  },

  "rules": {
    "no-var": "warn",

    "accessor-pairs": ["error", { "setWithoutGet": true,
"enforceForClassMembers": true }],
    "array-bracket-spacing": ["error", "never"],
    "array-callback-return": ["error", {
      "allowImplicit": false,
      "checkForEach": false
    }],
….
```

Backend Linting and Code Formatting

The backend uses Pylint and Black for code formatting and linting to ensure consistency in syntax and style across multiple developers.

Frontend Repo Documentation

Opus Team

website up license MIT

About

Opus is a Team Management and Communication application for small teams that are looking to host and deploy their own personal solution to team communication. Opus allows users a fast and flexible way to create meetings, announcements, and teams with other users. This repository hosts the frontend codebase and deployment for Opus Team.

Take a look at the following links

- User Documentation: https://docs.opusteam.us
- Production Deployment: https://opusteam.us

Dependencies

This is a React project, and the major dependencies for this project are

- React.js v17.0.1
- Node Package Manager (npm)

The language that we use is JavaScript (ES7) with linting provided by eslint. Refer to `package.json` for further documentation about dependencies.

Contributing

Setup

- Clone the repository

```
git clone git@github.com:opustm/backend.git
```

- Install npm
- Install all node dependencies `npm i`

Installing Additional Packages

- Install additional dependcies with npm

```
npm i <package-name>
```

License

MIT.

Backend Repo Documentation

Opus Team API

website up license MIT Python v3.9

About

This is the Opus REST API which serves the Opus Frontend. It currently contains documentation and endpoints for users, teams, requests, events, announcements, JWT, Django Admin.

Available at https://api.opusteam.us.

Dependencies

- Python v3.9
- Pipenv 2020.11.5
- PostgreSQL 13.2
- All Python Modules under `./Pipfile.lock`

Contributing

Setup

- Install Python
- Install Pipenv `pip install pipenv`
- Install PostgreSQL for your system here
- Alternatively you can install it on MacOS using Homebrew:

```
brew install psql
```

- On Debian/Ubuntu use `apt-get`

```
sudo apt-get install psql
```

- Using Pipenv install the other dependencies

```
pipenv install
pipenv install --dev
```

- For MacOS users, install `psycopg2-binary`

```
pip install psycopg-2 binary
```

Installing Additional Packages

- Start the Python virtual environment and use `pipenv` for package installations

```
pipenv shell
pipenv install <package-name>
```

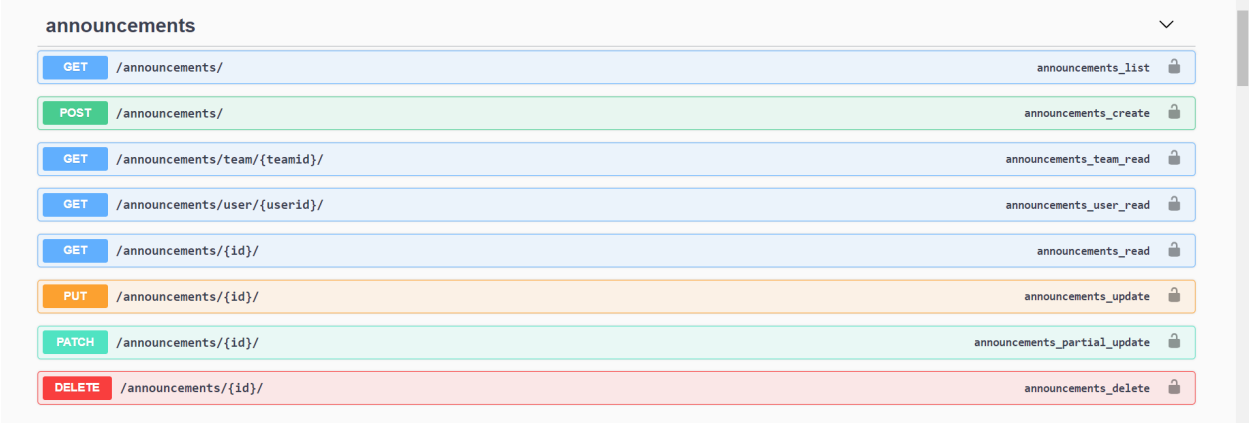
- Clone the repository `git clone git@github.com:Instaline/Instaline-Markdown.git`
- Activate the virtual environment `pipenv shell`
- Install Dependencies `pipenv install`
- Run the CLI `python3 parse.py`

License

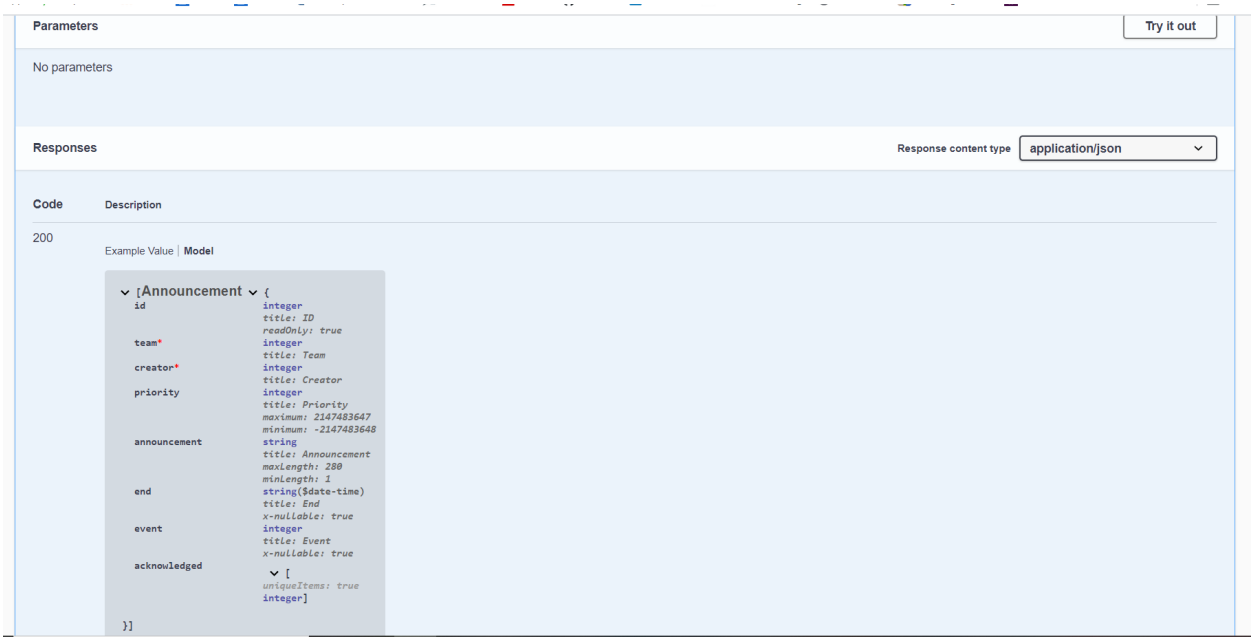
MIT.

Swagger API Documentation

Our frontend uses many backend API routes to get information about users and their teams, so we used a module called “Swagger” that automatically generates basic API documentation for Django projects. This documentation is available in its entirety at https://opustm-api.herokuapp.com/



Auto-generated Swagger Documentation for our ‘announcements’ routes



Example of the documentation that expands when clicking on a route; in this case, a GET request to /announcements

Version Control

This information is also available at https://github.com/opustm/docs in the CONTRIBUTING.md file.

Structure

- Our main branch is named "main", and will contain the code that Heroku uses for deployment
- We have one branch off of main named "staging"
- To merge new code into main, ensure that the code is functional on the staging branch both by running it locally and by checking the staging deployment.
- From staging, each developer may create branches for their assigned bugs, enhancements, and so on.
- Branch names should be concise and indicate to others what the new code is meant to do.

Introducing New Code

Once a developer has finished working on their individual branch, they should merge that branch back into staging by following the steps below.

- Stage your changed files (git add . or click “Commit” on SourceTree and stage all)
- Commit and add a message (git commit -m “{Insightful_Message}” or enter the message on SourceTree on the bottom and click commit)
- Push changes to remote (git push --set-upstream origin {branch_name} or click push on SourceTree)
- Next, create a merge request on GitHub
- Choose the source repository/branch and destination repository/branch
- Destination branch should always be “staging” for merging an individual branch
- Add a title and a brief description to indicate what reviewers should be looking for in your changes
- Assign the pull request to one or two other people
- Reviewers should follow the code review guidelines and ensure that their review is complete before approving new changes.
- Once the original developer has received approval from their reviewers and resolved any comments, they may complete the merge into staging.

Notes

- When merging staging into main, ensure that the staging branch is fully functional before doing so. Failure to do this could introduce breaking changes into the deployed version of the main branch.
- Merge conflicts can typically be resolved by back-merging the target branch into your branch before merging your changes back in.

Code Review

Motivation

Code Reviews are a valuable process to a development team like ours since it increases the overall efficiency, quality, and decreases the complexity of the code. It also serves as a way for a development team to learn and grow faster to become better engineers.

- Reviewed code allows development teams to change free form defects, establish coding conventions and solve a problem in the most effective and reasonable way, ensuring that the product is of a high quality.
- It helps save time in the overall development and debugging process of a program.
- Code reviews also share knowledge between team members, and help both the reviewer and submitter familiarize themselves with the team's coding process and technology.

Submitters:

The person submitting the pull request will keep the size of the pull request to less than 300 lines. Many articles and studies have discussed how as the amount of code that is changed increases, the reviewers' willingness to comb through and understand the entire change decreases. Keeping these pull requests compact and focused on a particular issue will help both the submitter and the reviewer in this way (note: since the code freeze has already passed, it's unlikely that there will be any changes that require this much coding). Additionally, developers should aim for more commits that have fewer lines rather than the opposite. This will help reviewers understand the process of how the changed code developed.

To get into the habit of writing readable and intuitive code, the person submitting the code for review should assign it to a member of the team that is not working on the "side" of the project that the code is related to. For example, Josh works primarily on the backend while my work is usually focused on the frontend. Enlisting the other to help when one of us has a pull request to review will increase our team's net understanding of the project, rather than compartmentalizing the frontend and the backend. Where necessary, developers should include comments to describe the purpose of blocks of code that are more complex.

Reviewers:

A main component that many of the articles discussed on the reviewer side was maintaining a balance between the level of detail put into the review while also turning the review around quickly. This is important because all the involved parties should have the code fresh in their head while working with it, and a delay in code reviews will cause confusion all around.

Moving forward, reviewers should return all pull requests within 24 hours of their being opened; this will be facilitated and made more manageable by the reduced size of the pull requests as

described above. Care should also be taken when reviewing to make comments that are relevant to the scope and intent of the current pull request. For example, if a change deals primarily with standardizing formatting or structure, reviewers shouldn't include comments about the functionality of certain functions that haven't been changed by the submitter.

If someone notices an issue that isn't related to the changes that have been made, they will make an issue on Github or contact their teammates via Slack. Reviewers should of course remember that they are reviewing the code and not the submitter, and therefore should follow the etiquette of providing constructive feedback for the submitter

Completing Code Reviews:

Once the 24-hour window has passed, all the reviewers should have made their comments so that the person who submitted the code can look at them and make the necessary revisions. For complex chunks of code, in-line comments should also be expected. Then, the code should be committed with a clear commit comment about what the changes are about. Because the number of lines is relatively small, the submitter should be able to make the changes within six to twelve hours of receiving the reviews, though this will be more flexible based on the scale of the changes that need to be made.

Upon completion, the submitter should run the unit tests corresponding to their changes a final time to ensure that the tests have not identified any errors. If the tests flag errors, the submitter will need to identify the issue and determine whether the code or the test is at fault, and then make the appropriate change. Once the tests have passed and the reviewers have signed off, the submitter may then merge the request into the respective branch and verify that the deployment was successful.

Bug Documentation and Discussion Forum

All developers have access to the GitHub Issues feature which allows developers to discuss any possible enhancements, bugs, or help required for a particular issue. Because the repository is labeled as open source, any GitHub user may open an issue if they encounter a bug or have a suggestion for an enhancement. Both of these will allow developers and end users to report bugs as they find them, and will also create a list of items for developers to work on. GitHub also enables comments and conversation on specific issues, which provides an easy, built-in way for users and developers to interact.

Code Structure: Frontend

On the frontend, there are five top-level folders that developers should be familiar with. Below is a description of each of those folders, as well as individual files that appear at the top level.

Folders:

- *build*: This folder is automatically generated by Node Package Manager (NPM) when the user runs the command 'npm start' or 'npm build'. Developers should not need to modify any files in this folder.
- *coverage*: This folder is automatically generated by NPM and is used for running the test suites. Developers should not need to modify any files in this folder.
- *node_modules*: This folder is automatically generated by NPM and is used for package management only. Developers should not need to modify any files in this folder.
- *public*: The site icon is located in this folder with the conventional name *favicon.ico*. If desired, developers may change this icon by deleting the current file, uploading a new one, and then renaming the new file to *favicon.ico*
- *src*: This folder contains all of the code that renders on the website. It has six internal folders where code is sorted based on its purpose:
    - *apps*: This folder contains all of the tools that appear on the sidebar of our application - Announcements, Calendar, Contacts, Teams. Each of these contains its own JS and CSS file. Similar tools that are added to the sidebar should be placed in this folder
    - *components*: This folder contains items in the webpage that appear on multiple pages, namely the Navigation and Widget code. Each of these contains its own JS and CSS file. Similar items that need to be used by multiple pages should be placed in this folder.
    - *pages*: This folder contains the code for the major pages within our application that aren't specific to a single tool - About, Home, Login, NotFound, Profile. This folder only contains JS files.
    - *services*: This folder contains code that does not render HTML using React on the page, but instead is used for other purposes, like contacting the API or handling forms. We currently have services for the API, Authentication, Browser Data, Descriptions, Forms, and Testing. This folder only contains JS files.
    - *static*: This folder contains assets that do not require any changes, such as images or helper code from third-party sources. Any website images should be placed here.
    - *stylesheets*: This folder contains all of the CSS code for the JS files in Pages. When adding a new Page, the stylesheet should go here.
    - *App.js*: This file contains the router for our application and handles changes in state between nested components. When adding a new component to the application, add a route to this file to render that component.

Files:

- The standard *.gitignore*, *README.md*, *package-lock.json*, and *package.json* files also appear at the top level. These are provided by NPM by default, and we follow standard conventions for each of them. We assume that developers are familiar with these tools when working with the codebase.
- *.eslint* and *.prettier* files describe our linting rules throughout the application. Developers can edit these files to change formats if desired. More information can be found at ESLint and Prettier if developers desire.

Code Structure: Backend

Our backend code follows standard Django conventions for file organization and code structure. For more information, consult the Django Rest Framework documentation at https://www.django-rest-framework.org/.

Resources

1. General

1.1. Home Page: http://opusteam.us
1.2. User Docs: https://docs.opusteam.us/
1.3. GitHub Team: https://github.com/opustm

2. Frontend

2.1. Frontend Repository: https://github.com/opustm/frontend
2.2. React: https://reactjs.org/docs/getting-started.html

3. Backend

3.1. Backend Repository: https://github.com/opustm/backend
3.2. Django: https://docs.djangoproject.com/en/3.2/
3.3. Swagger: https://django-rest-swagger.readthedocs.io/en/latest/