omni:port

Omniport documentation

Dhruv Bhanushali, Praduman Goyal

Nov 27, 2022

CONTENTS

Omniport is a portal for educational institutes, designed from the ground up to be extensible, customizable, performant and powerful.

Fig. 1: The very classy Omniport wordmark, designed by Kunal Satpal

Omniport also comes with a powerful collection of apps and is simple enough for you to be able to write your own. It's legit magic. Pure, unadulterated magic.

This documentation walks you through the Omniport project. It explains its philosophy and its features. It explains the technologies involved in the project and the ways in which this project can make your institute more technologically equipped than it is.

It also explains how yo can pitch in and support its continued development and upgradation. It explains how you can contribute code, design, documentation, buzz or traffic to the project.

Lastly but most importantly, it explains what it can do for you and what you can do with it.

Happy development!

CHAPTER

ONE

INTRODUCTION

Omniport, as mentioned countless times, is a portal designed to make the lives of every stakeholder in an educational institute easier. From the students to the administration, Omniport is a simple way to usher your institute into the digital age.

Read on to know what Omniport is all about.

1.1 Philosophy

When we set out to build Omniport, we had a bunch of things in mind. Vaguely speaking, it had to be epic and had to blow everyone's mind off.

After a lot of thinking, some clarity. It had to be poetic. It had to evoke in the readers a love that they never thought they could experience again, a love so powerful it would compel them to cheat on their language of preference with Python, to switch to Django from PHP (or even Ruby on Rails, for that matter). On the frontend, there never was an real contender to React anyway.

1.1.1 The zen of Omniport

In a fashion similar to Python's, here is a zen poem that somewhat guides the decisions behind Omniport.

Don't repeat yourself. Ever. Ever. Oh wait, my bad. Sorry.
Focus on the essential. Everything else, like this sentence, is irrelevant.
Bike-shedding is the best way to annihilate time. Let's discuss button colours.
Roles are just temporary hats people wear. Apps put personae on roles.
Generalised is better than hard-coded. Customisable is better than generalised.
Development should be exciting. Deployment should not.

1.1.2 Significance

The zen of Omniport, in addition to PEP8, StandardJS, and our own code style, is the driving force behind our development. Any pull request, even one of the smallest ones, that takes the code towards a state with higher compliance with the above zen shall be considered worthy of merge.

1.2 Features

Omniport prides itself on its impressive and rich feature set. This includes the following non-exhaustive list.

1.2.1 Code quality

- **Idiomatic Python (Django) and JavaScript (React)**

  The entire codebase is written keeping in mind all the philosophies of Python and Django, class-based views and all as well as JavaScript and React, class-based components and all.

- **PEP8 compliant**

  All the postulates of PEP8 have been satisfied, from the sane rules to the preposterous 80 character limit.

- **StandardJS compliant**

  All the postulates of Standard have been satisfied, from the sane rules to the preposterous no semicolon rule.

- **Completely documented**

  Even the most basic and obvious functions and classes have been documented amply, as this documentation testifies.

1.2.2 Cutting-edge

- **Modern technological stack**

  A consistent JSON API by Django REST framework on the backend powers a single-page React application on the frontend.

- **Cutting-edge technology**

  Hope the devs are OK with bleeding because Omniport rolls with the latest in technology as evident from the stack.

- **Containers**

  Docker containers ensure an easy set up and also the peace of mind that there will never be surprises in production.

1.2.3 Ideals

- **Flexibility**

  The entire project consists of mix-and-match, plug-and-play, load-and-forget components that work independently but better together.

- **Power**

  Almost any and every feature that you could possibly think of, and then some, have been baked in and well integrated.

- **Minimalism**

  The entire project is very minimal and is totally devoid of bloat or unwanted software. We call it lean.

- **Simplicity**

  Almost the entire project can be configured via external YAML and ENV files without touching a single line of code.

- **Open-source**

  From the tech stack to the dependencies, Omniport is built on open-source. It only makes sense to give back to the wonderful community.

1.3 The stack

Unless otherwise specified, Omniport runs on the latest versions of all the components in the stack. This means that there is no LTS version or long-term release and Omniport is always a rolling distribution.

Sphere	Sub-sphere	Technology
Orchestration	*Containers*	Docker
NoSQL databases	*Sessions*	Redis
	Communications	Redis
	Channels	Redis
	Temporary app	Redis
	GUI	Redis Commander
SQL database	*Application*	PostgreSQL
	Library	Psycopg2
Cache	*Application*	Memcached
Message broker	*Application*	RabbitMQ
	Library	Celery
Reverse proxy	*Application*	NGINX
Backend	*Language*	Python
	Framework	Django
	WSGI server	Gunicorn
	ASGI server	Daphne
Frontend	*Language*	JavaScript
	Framework	React
	Transpiler	Babel
	Bundler	Webpack

- Tier I (no dependencies on other infrastructure)
  - Message broker
  - Channel layer
  - Session store
  - Notification store
  - Application store
  - Database
  - Cache
- Tier II (depend on and wait for Tier I to be ready)
  - Intranet server
  - Internet server
  - Redis GUI
- Tier III (depend on and wait for Tier II to be ready)
  - Reverse proxy

CHAPTER

TWO

STRUCTURE

To keep the code clean and keep the repository histories from mangling with each other, each distinct component of Omniport lives in its own repository, free from the scaffolding and core codebases, yet dependent on them for functioning.

Read on to learn more about the structure of Omniport.

2.1 Repositories

Omniport is not a single repository. The entire project is made up of dozens of distinct, yet interconnected parts, most of which have their own repositories under the IMGIITRoorkee GitHub organisation page. This means that cloning the Omniport project is not one operation but many.

The essential ones of these repositories are explained here.

2.1.1 Infrastructure

This repository contains the code that drives the Omniport infrastructure. That means that all `Dockerfiles`, environment variables and build scripts are kept in this repository.

2.1.2 Codebase

Omniport backend

This repository contains the code of the Omniport backend architecture, written in Django on Python. This includes all the core functionality of the project and is extensible in terms of choosing your own set of apps to run on it.

This backend can be customised and branded according to your own wishes. All configuration and branding lies in directories outside the code so that you do not have to know development to set the project up.

Omniport frontend

This repository contains the code of the Omniport frontend architecture, written in React on JavaScript. This includes all the core functionality of the project and can be extended with apps, that are independent of the backend.

This frontend takes all the customisations from the backend and is therefore totally branding-ready and tweakable. All changes on the backend are reflected here with very little to almost no effort.

2.1.3 Shell

As we previously mentioned, Omniport is a portal that is epically generic. Any institute around the world, from Fiji to Fiji the other way, can set it up and get going.

But if your institute has the technical know-how that a group like IMG provides IIT Roorkee, you can easily *swap* out models in the `kernel` module of `omniport-backend` with your own. You can also *switch* out the serializers in the `kernel` module with your own.

This swap and switch functionality can be made use of to develop shells for the backend as we have made one for our institute. You are free to refer to it to make your own, or run the fully functional, unshelled versions of the software.

Note: Other than the shell for **Indian Institute of Technology, Roorkee**, any shells made for Omniport are not under the purview of Information Management Group.

IMG couldn't possibly take responsibility for the upkeep, maintenance or upgradation of any random shells made by any random developers for any random institutes of any random country.

2.1.4 Sub-components

Apart from these repositories are Omniport itself, there are a huge number of repostories that are a part of Omniport. These are apps and services.

Every single one of Omniport's many apps and services resides in its own repository. They follow a naming convention as follows.

- `omniport-service-<service>` and `omniport-frontend-<service>`
- `omniport-app-<app>` and `omniport-frontend-<app>`

Totalled, these repositories number in the fifties. But since these are on an open architecture, there is total democracy and freedom regarding who can create apps and what those apps can provide. This number has no upper limit.

2.2 Folder structure

Omniport follows this outline of folder configuration. Omniport Docker houses both sides of the codebase in a `codebase/` folder.

```
omniport-docker
├── docker-compose.yml
├── scripts
│   └── ...
├── django
│   ├── Dockerfile
│   └── ...
├── react
│   ├── Dockerfile
│   └── ...
├── nginx
│   ├── Dockerfile
│   └── ...
└── codebase
    ├── omniport-backend
    │   ├── scripts
    │   │   └── ...
    │   ├── omniport
    │   │   ├── core
    │   │   ├── shell*
    │   │   ├── services
    │   │   ├── apps
    │   │   └── ...
    │   └── ...
    └── omniport-frontend
        ├── scripts
        │   └── ...
        ├── omniport
        │   ├── core
        │   ├── formula_one
        │   ├── services
        │   ├── apps
        │   └── ...
        └── ...
```

2.3 Architecture

We, the developers of Omniport, pride ourselves on Omniport's architecture. It is a smart blend of automation and explicitness. We strive to make Omniport as automated as possible to reduce the amount of code a person has to write to set it up and extend it, but at the same time not be too *magical* for an ordinary person to understand, in detail, its functionality.

Omniport, at its very core, contains a distinction between apps and services. Apps and services are all Django applications, there is but an ideological difference between the two categories.

2.3.1 Services

Services are Django apps that provide critical functionality. Services may depend on other services and on the core. A service may not depend on an app. Services may depend on other services, which is strongly encouraged because it leads to a cohesive system.

All services must be installed for Omniport to function properly. Services cannot be chosen at will. Services do not contain any form of filtering based on user roles, corresponding active statuses, or IP addresses.

Services, and services only, may appear in the Omniport sidebar. The subset of services that appear in the sidebar are defined by their `config.json` file on the frontend.

See also:

More on frontend configuration files *here*.

For an example, consider the service 'Developer', which is used by developers to develop OAuth2 based applications on Omniport.

Services are cloned into `services/` directories inside the src directory (`omniport/`) in both the backend and frontend codebases.

Since there is no mix-and-match capability in Omniport services, the process of cloning services has been merged with the cloning of the codebases using the `clone_codebase.sh` script provided by `omniport-docker`.

See also:

More on the `clone-codebase.sh` script *here*.

2.3.2 Apps

Apps are extensions to Omniport that provide new functionality. Apps may depend on other apps, which is strongly discouraged because it may lead to crashes when dependencies are forgotten, or on services, which is strongly encouraged because it leads to a cohesive system.

Apps can be plugged into the Omniport system as and if needed or wanted. Apps can be chosen at will.

All apps can choose their target demographic on the basis of roles and corresponding active statuses. These filters can be configured in the `config.yml` file on the backend. This leads to every user seeing a set of apps on the backend.

See also:

More on backend configuration files *here*.

$$A = \{\text{apps allowed to a user based on role, active status and orgin IP address}\}$$

Then there is a set of apps whose frontend has been installed in the frontend architecture. This leads to a second set of apps common to every user.

$$B = \{\text{all apps installed on the frontend}\}$$

The net set of apps that any user sees is the intersection of both these sets.

$$Result = A \cap B$$

Apps are cloned into `apps/` directories inside the src directory (`omniport/`) in both the backend and frontend codebases.

2.3.3 Dependencies

```
service ---may depend on--> core
service ---may depend on--> service
app     ---may depend on--> core
app     ---may depend on--> service
```

CHAPTER

THREE

ENVIRONMENTS

The Omniport project has been designed keeping both development and production environments in mind. Although the terms might be generic enough for every single team in the world to have both of them, every single team implements these two workflows in a very unique way.

The Omniport Docker project is highly opinionated on both of these workflows, which means that the out-of-the-box defaults might very likely not be what you expect or want.

Note: If you are a lone-wolf developer, the default production workflow, albeit with debugging enabled, is more likely to suit you rather than the default development one.

Read on to understand what Omniport's assumption of your workflows looks like.

3.1 Development

The development workflow is the workflow suited to teams where multiple developers work on multiple projects at the same time. This is the development flow implemented at IMG, IIT Roorkee.

The postulates of the development workflow are:

- **There is a central development server.**

  It must be highly powerful, but it depends on how many developers are going to be simultaneously developing on and for Omniport.

- **Developers SSH into the remote from a terminal in the maintainers' lab.**

  Each developer gets a unique account on the remote. They clone the codebase (and not the infrastructure) in their accounts, each running a copy and making their changes in said copy.

  All development work happens on the remote. One feasible option is programming on Vim inside SSH. Another is programming on the codebase replicated via SFTP.

- **Application servers share standalone services.**

  The common services such as the database and the message-broker are run in the Docker sandbox under an altogether different user, which is neither the `root`, nor one of the developers.

  At IMG we call this user `apps`. You are free to name it whatever pleases you. Just remember this name for later.

- **All developers run their own application servers.**

  Since application servers hot-load the code from the host, every developer can host their code independently of other application servers.

- **Application servers expose a port on the 0.0.0.0 interface on the server.**

  The Django development server runs on `0.0.0.0:600xx` and the React development server runs on `0.0.0.0:610xx`, proxying requests to the Django server.

These servers can be accessed in a browser on the developer's local terminal by visiting `http://<server_IP>:<port>/`.

If this is not your setup and you are not in a postition for this to be your setup, you are, unfortunately, on your own for this one. See if our production setup suits you more, but that's about as far as we can go.

But given that you are developers, we have complete faith that you'll find a way. *Developers always do.*

3.2 Production

The production workflow is the workflow suited to production where no development takes place and everything runs with maximum automation. Alternatively this workflow is also applicable to individual developers, developing on their own computers. This is the production flow implemented at IMG, IIT Roorkee.

The postulates of the production workflow are:

- **There is the production server.**

  It must be powerful enough to bear the load of all the people who would be using it. Depending on your college size and your own purchasing power, that could mean your laptops or entire farms.

- **One person, the sysadmin, the boss, the chief, opens an SSH session.**

  I wasn't sure if SSHs or SSHes, so I went with *'…opens an SSH session'*. This person must manage an underprivileged user on the server, which still has Docker rights though.

  At IMG, we call this user `apps`. Again you can name it whatever you damn well please, provided you remember this name for future use.

- **Everything is managed and orchestrated by Docker Compose.**

  You execute one command and everything goes up.

We are pretty confident that's how any production would work so you should be fine. If you are smart enough to use an altogether different production setup, you should be smart enough to tweak this to your use. You're on your own for this one, unfortunately.

Remember our advice from *the development section*.

CHAPTER

FOUR

SETTING UP

Getting started with Omniport is easy, especially when going with the Docker route. There are a few steps that are common to both enviroments after which the two flows diverge a bit. Effectively, thanks to Docker and some smart development, we are able to reuse the same code and containers, with minuscule changes, on development and production machines.

We'll try to comprehensively cover all scenarios but know that we will have to restrict ourselves to the two out-of-the-box flows only.

Read on to understand how you can set up Omniport Docker on your own computers, development servers and production servers.

4.1 Server configuration

To start up, use a fresh installation of your favourite operating system. This could be any flavour of Linux, such as Ubuntu, Fedora or RHEL, to name a few.

4.1.1 Ports

Ensure that no other services are running on the ports specified below, especially those marked with an asterisk.

Port	Designated use
80*	NGINX http
443*	NGINX https
5432	PostgreSQL
5672	RabbitMQ
15672*	RabbitMQ management
11211	Memcached
6379	Redis
8000	Gunicorn
8001	Daphne
8081*	Redis Commander
60000 - 60031	Django development
61000 - 61031	React development

In the most common scenarios, 80 and 443 will be occupied on a fresh install of a server distribution like RHEL or Ubuntu Server. Stop Apache2 and prevent it from automatically starting up again.

```
[apps ~]$ sudo systemctl stop apache2
[apps ~]$ sudo systemctl disable apache2
```

4.1.2 Users

Set up a user other than `root` to build and manage the containers. Name him `apps` or whatever you fancy.

In case of a development setup, make user accounts for all your developers, one per person. Let's assume you have two `dev1` and `dev2`. They also get their own directories at `/home/dev1/` and `/home/dev2/`.

```
[dev1 ~]$ whoami
dev1
[dev1 ~]$ pwd
/home/dev1
```

```
[dev2 ~]$ whoami
dev2
[dev2 ~]$ pwd
/home/dev2
```

4.2 Docker

Once you have configured the server, it's time to install the Docker CE suite on it. Setting up Docker is easy. Since that is beyond the scope of the documentation, please refer to the Docker documentation for instructions pertaining to your operating system.

Do note that installing Docker alone will not suffice, you will also need to install Docker Compose.

Note: The Docker documentation contains the links on install Docker and Docker Compose on your system of choice.

4.2.1 Start and enable Docker

After you have installed Docker and Docker Compose on your machine, you will need to make some configuration changes.

Docker has been installed but the daemon is neither running nor has been set to run automatically on a reboot. You can accomplish these changes as follows.

```
[apps ~]$ sudo systemctl start docker
[apps ~]$ sudo systemctl enable docker
```

4.2.2 Groups

The Docker daemon process `dockerd` is only accessible to users that are a part of the Docker group.

Remember `apps` from *Environments* and *server configuration*? So `apps` must a member of the group `docker`, which is automatically created when Docker is first installed.

```
[apps ~]$ sudo usermod -aG docker apps
```

Also in a development environment, your developers must also be members of the group.

```
[apps ~]$ sudo usermod -aG docker dev1
[apps ~]$ sudo usermod -aG docker dev2
```

You will need to restart your user sessions, and in my personal experience in some cases, even restart your computer, after this.

4.2.3 User namespaces

User namespaces are a way to limit the surface area of the Docker sandbox in the event of a security breach. Basically user namespaces map `root` in the Docker sandbox to another UID on the host, thereby stripping away all rights of the root user in a container to cause damage to the host.

> **Warning:** While user namespaces are very cool and good, we at IMG have run into a number of intermittent and irregularly occurring issues when running Omniport under a namespace. This step is therefore advised only for people who know what they are doing.

To enable namespaces, elevate your privileges, open the file `/etc/docker/daemon.json` and type in the following lines.

```
{
  "userns-remap": "apps"
}
```

If you decided to go with an alternative name for the main user, replace `apps` with the username of that user.

You will need to restart the Docker daemon after this change.

```
$ sudo systemctl restart docker
```

4.3 Omniport Docker

Clone Omniport Docker from GitHub and enter the directory.

Inside the directory `omniport-docker/` you must clone the codebase of Omniport, namely the backend and the frontend. Enter the `codebase/` directory and then clone both `omniport-backend` and `omniport-frontend` repositories from GitHub.

Note: You may use the *clone codebase script* to accomplish the same goal.

4.3.1 Configuring the environment

RabbitMQ

In `rabbitmq/`, copy `message_broker_stencil.env` to `message_broker.env` and populate the environment variables. These will be used when the queue is set up so choose a strong password.

You will need to provide these values in the Django configuration, so remember them.

PostgreSQL

In `postgres/`, copy `database_stencil.env` to `database.env` and populate the environment variables. These will be used when the database management system is set up so choose a strong password.

You will need to provide these values in the Django configuration, so remember them.

Django

Django configuration works at two levels:

- base-level configuration
- site-level configuration

Enter `codebase/omniport-backend/configuration/`.

Base-level configuration

Copy `base_stencil.yml` to `base.yml` and populate all the variables there. If you need help, refer to the *documentation on this file*.

Site-level configuration

In `sites/`, copy `site_stencil.yml` to

- `site_0.yml` in development
- `site_1.yml` and `site_2.yml` in production

and populate all the variables there. If you need help, refer to the *documentation on this file*.

NGINX

There is no configuration file to write here. Just keep the domain names for the Intranet site and the Internet site at hand. If you have SSL enabled, which you absolutely should because it enhances security and is absolutely free, place your

- certificate named `omniport.crt`
- private key named `omniport.key`

in the directory `cert/`.

Also, if you choose the SSL route, remember to answer 'yes' for HTTPS in the NGINX image build script.

4.3.2 Build the Dockerfiles

There are quite a few images to be built which, thanks to Omniport's liberal use of scripts, translates to executing just as many shell commands. All these scripts are interactive, which means you will be asked questions, whose answers will determine the end result of the operation.

Maybe one day we will write a script that runs these scripts.

```
[apps omniport-docker]$ ./scripts/build/django.sh
[apps omniport-docker]$ ./scripts/build/react.sh
[apps omniport-docker]$ ./scripts/build/nginx.sh
[apps omniport-docker]$ ./scripts/build/memcached.sh
[apps omniport-docker]$ ./scripts/build/rabbitmq.sh
```

(continues on next page)

(continued from previous page)

```
[apps omniport-docker]$ ./scripts/build/postgres.sh
[apps omniport-docker]$ ./scripts/build/redis.sh
```

That's all. Omniport Docker is ready to roll.

4.4 Development

With everything set up, the flows diverge. This is how to develop for Omniport.

4.4.1 Sysadmin

As `apps`, enter `omniport-docker/`. Set up all the shared services.

```
[apps omniport-docker]$ ./scripts/start/development.sh
```

This should start all shared services such as the database and message-broker. Being heavy and all, every developer shares these services.

To enter the container for any service, execute this command.

```
[apps omniport-docker]$ docker-compose exec <service_name> bash
```

Note: Most Omniport services are built off of Debian and have `bash` installed. In case, you face trouble, try running `sh` instead.

Go as a developer and migrate the services.

Relax.

4.4.2 Developer

As `dev1`, clone the following.

- `omniport-backend`
- `omniport-frontend`
- all services on the backend and frontend
- your apps on the backend and frontend

Then from `omniport-backend/`, start your development server.

```
[dev1 omniport-backend]$ ./scripts/start/django.sh
```

Note the port that you are assigned Let's call it `<django_port>`.

Then from `omniport-frontend/`, start your development server.

```
[dev1 omniport-frontend]$ ./scripts/start/react.sh -d <django_port>
```

Visit these ports from your browser and you should be able to see your app. Changing code in either codebase will automatically reload the servers.

To enter the container for any server, execute this command.

```
[dev1 anywhere]$ docker exec -ti <port> sh
```

Note: Some services, namely those built off of Debian, have `bash` installed. Most don't. So the best way to go about it is to execute `sh` and once in the container, execute `bash`.

From time to time, you will have to enter the Django container to make and run migrations, collect static files and do the occasional housekeeping. You should migrate only your own apps. Apps migrated by `dev2` should not be migrated by `dev1`. In most sane cases, this situation will never even arise.

Have fun!

4.5 Production

With everything set up, the flows diverge. This is how to deploy Omniport.

As `apps`, enter `omniport-docker/codebase/omniport-frontend`. Build the frontend for NGINX to serve.

```
[apps omniport-docker]$ ./scripts/build/frontend.sh
```

This should build your React app and place it a folder for NGINX to serve.

Then just start all services using Docker Compose.

```
[apps omniport-docker]$ docker-compose up -d
```

You should be able to access your app on the domains you specified in NGINX, provided you have the DNS routing properly set up.

To enter the container for any service, execute this command.

```
[apps omniport-docker]$ docker-compose exec <service_name> sh
```

Note: Some services, namely those built off of Debian, have `bash` installed. Most don't. So the best way to go about it is to execute `sh` and once in the container, execute `bash`.

From time to time, you will have to enter the Django container to make and run migrations, collect static files and do the occasional housekeeping.

Relax.

4.6 Omniport Docs

Note: You must have `Python 3.8` and `pipenv` installed before you can set up the documentation locally.

1. Install the project dependencies.

   ```
   $ pipenv install --skip-lock
   ```

   Omit the flag `--skip-lock` if you want the latest packages. Keep in mind that the project has been tested on the versions of the packages specified in the `Pipfile.lock` so its compatibilty with the latest packages might not be assured.

2. Start the `pipenv` virtual environment via running the following command. Or you can prefix the commands for starting the server or for building docs using `pipenv run`.

   ```
   $ pipenv shell
   ```

3. To start the development server, run the following commands inside `/docs`. This will automatically re-build your changes as you make them.

   ```
   $ make dev
   ```

   Pass extra flags like port (default: 8000) by suffixing `SPHINXDEVOPTS="-p 8080"` with the above command.

4. Build the documentation into HTML pages by running the following command inside `/docs`.

   ```
   $ make html
   ```

   This will create the `docs/build` directory. You can preview it using one of Python's module called `http.server`.

5. To build the documentation into some other format, refer to `make help` command for all the options available.

4.6.1 Cannot start the virtual environment?

If you keep getting the following error when you try to run `pipenv shell`:

```
Shell for UNKNOWN_VIRTUAL_ENVIRONMENT already activated.
No action taken to avoid nested environments.
```

Run `exit` to resolve the issue. You will be able to start the virtual environment now.

CHAPTER

FIVE

HOW TO . . .

Omniport presents a sea of possibilities to any aspiring developer. It provides a platform for developing the richest and finest applications that can

- eventually be added into the portal, under the aegis of the local instance maintainers.
- use the most trustworthy authentication source for your campus applications.

However it is natural to face the question of how.

> *"How can I build my Omniport application?"*
> *"How can I use OAuth2 to provide seamless authentication?"*

These, and so many more similar questions, are just indicative of the kind of dilemmas you might face. To make it easy for you, we have compiled a list of tutorials that you can follow to make your journey as a campus developer more comfortable.

5.1 . . . brand Omniport?

Omniport is the working title of the project. Your portal need not be named Omniport, although no one's going to stop you from naming it that if you prefer.

However, for those of us that would like to brand their specific instance of Omniport, branding has been made really really easy. Effortless actually. Omniport supports deep branding of the portal to customise it thoroughly for your institute.

Very fancy.

5.1.1 Destination

All paths referenced here lie within the `omniport-backend/` folder inside the `codebase/` folder.

5.1.2 Instructions

- **Follow the naming convention specified.**

  Omniport is not sentient and will not locate your images if not named and placed according to the pattern. To omit an asset, leave out the file.

- **Follow the size and aspect ratio guidelines.**

  This is so that the assets integrate harmoniously with the rest of the portal. As feature packed as Omniport is, it is not free of assumptions.

- **Follow the file format specification.**

  As a rule of thumb prefer *.svg* and *.ico* when the image needs to scale up and down respectively. Preferred file formats are given, in order of preference, best stick to them.

- **All branding is optional.**

  While a bit of imagery adds to the aesthetics, it is not essential. In its absence, everything will gracefully fallback on textual branding defined in *configuration/*.

5.1.3 Image specifications

File name	Aspect ratio	Height in use	Formats
`favicon.ico`	square	16, 32, 48 px	multisize `.ico` only
`logo.xyz`	square preferred	3.5em *(~49px)*	`.svg`, `.png`, `.jpg`
`wordmark.xyz`	no restrictions	3.5em *(~49px)*	`.svg`, `.png`, `.jpg`

All images should be provided in 2x size if not supplying in the `.svg` format to ensure that they work well on HiDPI displays.

For best results, use RealFaviconGenerator to generate your `favicon.ico` files.

5.1.4 Branding types

Batteries-included

The following files are required to brand Omniport with your custom branding. Defaults will be provided for these images.

- `site/`:

  Since Omniport allows you to run multiple sites (such as Intranet and Internet) from the same portal, we allow you to brand them separately by loading indexed files.

  Indexing works by suffixing `_<side_id>` to the folder name. So a logo for a site with site ID 2 should be placed in `site_2/` for Omniport to recognise it. This is optional and you can skip the indexing portion to use the same asset across your sites.

  Ensure that you brand all your served sites or provide a non-indexed version as fallback. You may use the provided default Omniport branding for your portal or *for reference* when designing your own, if designing your own.

  If not redesigning your own icons, be sure to respect the *Brand usage guide*.

Batteries-not-included

The following files are required to brand Omniport with your custom branding. No defaults will be provided for these images.

- `institute/`:

  Your institutes logo and wordmark should be used here. There are no defaults and no references for this set of imagery.

- `maintainers/`:

  The branding imagery employed by Information Management Group for the Indian Institute of Technology, Roorkee is available *for reference*.

5.2 . . . create an app?

5.2.1 Backend

File structure of a typical app is as follows.

```
placement_and_internship
├── config.yml
├── LICENSE
├── README.md
├── __init__.py
├── http_urls.py
├── apps.py
├── migrations
│   └── __init__.py
├── models
│   └── __init__.py
├── static
│   └── assets
│       ├── favicon.ico
│       ├── icon.svg
│       ├── logo.svg
│       └── README.md
└── views
    ├── hello_world.py
    └── __init__.py
```

Look familiar? That's because it is. The structure of the app is the same, with a few notable exceptions as below.

config.yml

We need to create, or if it exists, populate the `config.json` file with the internal name, the display name, the URLs and the acceptables as shown in the example.

```yaml
nomenclature:
  name: placement_and_internship
  verboseName: Placement and internship
description: Build your resume, apply and get placed
baseUrls:
  http: placement_and_internship/
  static: placement_and_internship/
isApi: true
acceptables:
  ipAddressRings:
  - self
  - specifics
  - maintainers
  - intranet
  roles:
  - name: Student
    activeStatuses:
    - IS_ACTIVE
```

See also:

For more information on `config.yml` see *this*.

http_urls.py

This file is the same old `urls.py` of Django, with the name changed in order to differentiate if from the `ws_urls.py` file that configures the URLconf for WebSockets.

ws_urls.py

This file is the URLconf for Django Channels that guides requests from WebSockets. This file needs to exist if your app makes use of realtime communications. If you define `ws:` in your `baseUrls:` you must have this file in your app's root folder.

static/assets/

This folder, with no variations in the directory names, contains various app assets such as icon, favicon and logo.

5.2.2 Frontend

File structure of a typical app is as follows.

```
placement_and_internship
├── config.json
├── LICENSE
├── README.md
└── src
    ├── components
    │   └── app.js
    ├── css
    │   └── app.css
    ├── index.js
    ├── reducers
    │   ├── exampleReducer.js
    │   └── index.js
    └── urls.js
```

Look familiar? That's because it is. The structure of the app is the same, with a few notable exceptions as below.

config.json

We need to create, or if it exists, populate the `config.json` file with the internal name, the display name and the URLs as shown in the example.

```
{
  "nomenclature": {
    "name": "placement_and_internship",
    "verboseName": "Placement and Internship"
  },
  "baseUrl": "/placement_and_internship",
  "source": "placement_and_internship/src/index"
}
```

See also:

For more information on `config.json` see *this*.

index.js

`index.js` plays an important role in connecting your app to `Omniport`. It act as a gateway between `omniport-core` and your app.

An `index.js` file looks like this.

```
import React, { Component } from 'react'
import { Route } from 'react-router-dom'
import App from './components/app'
import { createStore, applyMiddleware } from 'redux'
import { Provider } from 'react-redux'
import thunk from 'redux-thunk'

import rootReducers from './reducers'

export default class AppRouter extends Component {
  constructor (props) {
    super(props)
    this.store = createStore(rootReducers, applyMiddleware(thunk))
  }

  render () {
    const { match } = this.props
    return (
      <Provider store={this.store}>
        <Route path={`${match.path}/`} component={App} />
      </Provider>
    )
  }
}
```

Here `App` is the normal react app component from which you can proceed similar to a normal react app.

urls.js

`urls.js` contains both navigation URLs for frontend as well as API endpoints for backend.

An example of `urls.js` can be as follows.

```
import appConfig from '../config.json'

// Frontend URLs
export function urlBaseView () {
  return `${appConfig.baseUrl}`
}

export function urlGroupDetailView (slug) {
  return `${urlBaseView()}/${slug}`
}

export function urlGroupTeam (slug) {
  return `${urlGroupDetailView(slug)}/team`
}

// Backend URLs
export function urlBase () {
  return `/api/groups/`
}

export function urlGroupList () {
  return `${urlBase()}group/`
```

(continues on next page)

(continued from previous page)

```
}

export function urlActiveGroupPost () {
  return `${urlBase()}post/`
}
```

5.2.3 Automate all of this

Note: Even if it's too complicated for you, refer to the `create/app.sh` scripts provided by Omniport for both the *backend* and the *frontend*.

Profit.

5.3 . . . use Omniport OAuth2?

The following walkthrough will show you how to integrate Omniport OAuth2 in your applications.

5.3.1 Registering your application

Register your app for Omniport OAuth2 by visiting the developer portal on `/developer/add`. Here you will be asked a few details about your app. You have to

- add your app name, redirect URIs in correct format.
- describe your app in strictly 127 - 511 words.
- select the minimum number of scopes (user information) that you will require for your app.
- read *Omniport developer terms of use* and agree to all of the terms therein.

Click on Add button to proceed. You will be taken to app administration dashboard. Here you can edit a few things about your app, like branding, team members and redirect URIs.

Note down your **client ID** and **client secret key** from the dashboard.

Note: Describe your app adequately and truthfuly. Also keep in mind that selecting as fewer scopes as possible will increase the odds in favour of approval of your app by the administration. After submitting your app request, your approval status can be seen and tracked from the developer dashboard.

Warning: You will be unable to use OAuth2 without said approval. Users will not be able to access your authorise your app to use their data.

5.3.2 Authorising the user

To authorise your users, redirect them to `/oauth/authorise` through a `GET` request with the following parameters.

Parameter	Description
client_id (required)	The client ID you obtained from the dashboard
redirect_uri	One of the redirect URIs you have registered on the dashboard
state	Any string that you want the `REDIRECT_URI` to receive on success

Warning: If the `redirect_uri` does not match any of the listed redirect URIs, the authorisation request will fail and the user will not see an authorisation screen.

Note: For additional information on the query parameters refer to the documentation of the Django OAuth Toolkit PyPI package.

A sample authorisation request URL could therefore look like the one below.

```
/oauth/authorise/?client_id=MY_CLIENT_ID
  <&redirect_uri=REDIRECT_URI>
  <&state=RANDOM_STATE_STRING>
```

5.3.3 The user experience

At this point, you've forwarded the user to Omniport authorisation page, where they will be greeted by the following screen (they might need to log in before they see it).

Neither you or your application can or need to do anything here; Omniport deals with the user, receiving the user's intent to approve or disapprove your app. Based on whether the user approves the app or denies it, the user will be redirected to `REDIRECT_URI` or back to Omniport home.

5.3.4 Handling the Response from Omniport OAuth2

If the user clicks approves the app to access their data on the previous screen, OAuth2 will redirect the user to the `REDIRECT_URI` specified earlier with a code parameter, any state passed to the authorisation URL will be forwarded to `REDIRECT_URI`.

```
REDIRECT_URI?code=AUTHORISATION_CODE
  <&state=RANDOM_STATE_STRING>
```

This one-time authorisation code has a TTL of 1 minute. So the next step, which is anyways supposed to be automatic must be done pretty fast.

5.3.5 Getting the access and refresh token

Once your application has an authorization code, it will now need to exchange the authorization code for a pair of access and refresh tokens from Omniport.

To get these tokens, you'll need to make a `POST` request to `/open_auth/token/` with the following parameters:

Parameter	Description
client_id (required)	The client ID you obtained from the dashboard
client_secret (required)	The client secret you obtained from the dashboard
grant_type (required)	`authorization_code` as it is
redirect_uri (required)	One of the redirect URIs you have registered on the dashboard
code (required)	The authorisation code sent to the REDIRECT_URI

If everything goes right and the request is successful, you'll receive a 200 response containing a `JSON` body like this:

```
{
  "access_token": "vqygrcouTuyAYHZLz3rGcZf5FpPd3K",
  "expires_in": 36000,
  "token_type": "Bearer",
  "scope": "read write",
  "refresh_token": "BbVJsFL1Ks5LkXDe9ZFUIFvIzXKt9M"
}
```

However, if the response is not successful, you'll receive an error response.

```
{
  "error": "some_error_message"
}
```

5.3.6 Using the access token

Now that you have the access token, you can get the user data by sending a `GET` request to `/open_auth/get_user_data/` with the access token in the header as follows:

```
{
  "Authorization": "Bearer vqygrcouTuyAYHZLz3rGcZf5FpPd3K"
}
```

where `vqygrcouTuyAYHZLz3rGcZf5FpPd3K` is the access token.

If the access token is valid, then you will receive a 200 response with a dictionary containing the user's information based on the scopes of the client app, similar to this:

```
{
   "userId": 1234,
   "username": "15xxx009",
   "person": {
      "shortName": "dhruvkb",
      "fullName": "Dhruv Kanti Bhanushali",
      "roles": [
         {
            "role": "Student",
            "activeStatus": "ActiveStatus.IS_ACTIVE"
         },
         {
            "role": "Maintainer",
```

(continues on next page)

(continued from previous page)

```
            "activeStatus": "ActiveStatus.IS_ACTIVE"
        }
    ],
    ...
},
"student": {
    "enrolmentNumber": "15xxx009",
    ...
},
"contactInformation": {
    "instituteWebmailAddress": "xyz@iitr.ac.in"
    ...
}
}
```

However, if the access token is invalid, you will receive the following `401 Unauthorized` error response.

```
{
    "detail": "Authentication credentials were not provided."
}
```

5.3.7 Generating new access token using refresh token

The access token has a short lifetime of **36000 seconds**. If it expires, you'll need to generate new tokens either by re-authenticating the user or using the refresh token.

To generate these new tokens, you'll need to make a `POST` request to `/open_auth/token/` with the following parameters:

Parameter	Description
client_id (required)	The client ID you obtained from the dashboard
client_secret (required)	The client secret you obtained from the dashboard
grant_type (required)	`refresh_token` as it is
refresh_token (required)	The refresh token received in exchange of authorization code.

The newly created tokens were successfully generated if you get a 200 response. The "JSON" response body will appear as follows:

```
{
  "access_token": "cjlgnpwhfuyAYHZLz3rGcZf5FpPd3K",
  "expires_in": 36000,
  "token_type": "Bearer",
  "scope": "read write",
  "refresh_token": "NhdwsFL1Ks5LkXDe9ZFUIFvIzXKt9M"
}
```

You will get an error response, though, if the response is unsuccessful.

```
{
  "error": "some_error_message"
}
```

5.3.8 Logging out the user

You can revoke the tokens to bar access when the user logs out.

To revoke the access and refresh token, you'll need to make a `POST` request to `/open_auth/revoke_token/` with the following parameters:

Parameter	Description
client_id (required)	The client ID you obtained from the dashboard
client_secret (required)	The client secret you obtained from the dashboard
token (required)	The access/refresh token you wish to revoke
token_type_hint (required)	`access_token` or `refresh_token`

5.3.9 Future plans

As of now the OAuth2 flow supports only the `authorisation_code` grant type in the flow. This will eventually be expanded to support most if not all of the multitude of flows backed by the OAuth2 specification.

5.3.10 Further reading

You should read the official OAuth docs for more theoretical information on OAuth2 and its many loosely-regulated forms.

For additional information on the parameters refer to the documentation of the Django OAuth Toolkit PyPI package.

5.4 . . . write logs?

A developer's got to write logs. When things break down, when errors pop up and when something is off, we refer to logs to comprehend what's happening.

But for there to be logs, one needs to write logging in one's app. Here's how to do just that.

5.4.1 Getting a logger

To write logs, start by getting a logger instance. This can be achieved in one line of code.

```
import logging

logger = logging.getLogger(__name__)
```

The value of `__name__` here is a dot separate path upto the current file. In a file named `hello.py` inside the module `views` inside the app `application`, it's value is `application.views.hello`.

Omniport has been configured to automatically configure loggers and handlers via Discovery. Each discovered app has a dedicated logger, configured to work with two handlers: `console` (only active during development and debugging) and one named after the app (only active in production).

The latter of the two writes logs to file, rotated every midnight keeping the last 32 days (nearly a month, except $32 = 2^5$ which is a cool number).

Back to the point, all of it happens in the background, like magic, so there's very little to do on your part.

5.4.2 Writing logs

Writing logs is easy as well.

```
logger.critical('This is a critical message!!')
logger.error('This is an error message!')

logger.warning('This is a warning message')
logger.info('This is an info message')

logger.debug('This is a debug message that no handler will catch')
```

That's practically all there is to it.

5.5 . . . read logs?

Reading Omniport logs is just as easy as writing them is. That's largely because a lot of the work has been done for you out-of-the-box. We're really generous that way.

5.5.1 Enter logs container

From the Omniport Docker root directory, run the script that opens a shell into the logging volumes.

```
[apps omniport-docker]$ ./scripts/start/logs.sh
```

This will drop you into the Docker shell. For more on this script, you can read the *script article*.

5.5.2 Find the log directory

An assortment of events are continuously logged in Omniport. A list of these events and their corresponding log locations are mentioned here.

Reverse proxy logs

Directory /reverse_proxy_logs/

NGINX

Subdirectory nginx_logs/

The following is a map of the sources and log files for NGINX.

Source	Log file
Intranet access log	intranet-access.log
Intranet error log	intranet-error.log
Internet access log	internet-access.log
Internet error log	internet-error.log

Web server logs

Directory /web_server_logs/

Gunicorn

Subdirectory gunicorn_logs/

The following is a map of the sources and log files for Gunicorn.

Source	Log file
Access log	<x>-access.log
Error log	<x>-error.log
Django log	<x>-django.log
Service and app logs	<x>-<name>.log

Here <x> must be replaced with **1** or **2** based on the site IDs. For more information on site IDs, refer to the *site-level configuration docs*.

Also <name> should be the actual name (not verbose name) of the service or app in question.

Daphne

Subdirectory daphne_logs/

The following is a map of the sources and log files for Daphne. This is not exhaustive because Daphne is used exclusively on `ws/` URLs and is not well-documented itself.

Source	Log file
Access log	<x>-access.log

Here <x> must be replaced with **1** or **2** based on the site IDs. For more information on site IDs, refer to the *site-level configuration docs*.

Supervisor

Subdirectory supervisord_logs/

The following is a map of the sources and log files for Supervisor.

Source	Log file
Self	self-<x>.log
Gunicorn stdout	gunicorn-<x>-stdout.log
Gunicorn stderr	gunicorn-<x>-stderr.log
Daphne stdout	daphne-<x>-stdout.log
Daphne stderr	daphne-<x>-stderr.log

Here <x> must be replaced with **1** or **2** based on the site IDs. For more information on site IDs, refer to the *site-level configuration docs*.

5.5.3 Tail the log

For real-time monitoring we go back to the classic `tail` utility whose capabilities have withstood the test of time.

```
docker@logs:/<logs_directory>$ tail -n <line_count> -f <log_file_name>
```

And there you have it. You are now armed with a comprehensive understanding of where to find logs to troubleshoot any and every error that could hit Omniport.

5.6 . . . migrate database?

Before you can make any use of Omniport, you'll need to migrate the database. This can be accomplished with the usual `migrate` command. from inside the intranet or Internet server container.

```
[apps omniport-docker]$ docker-compose exec intranet-server bash
docker@intranet-server:/omniport$ python manage.py migrate <app_name>
```

In a regular classic Django setup you could very easily skip <app_name> and migrate every installed app automatically. Unfortunately, our use of swappable models makes that a little complicated.

Warning: Because of the same reason as stated above, the use of a shell must be thoroughly debated and determined beforehand. After the migration, a shell cannot be inserted or removed without a complete reset of the database.

You will need to migrate apps individually and in a strictly defined order. This is as follows.

1. kernel (will automatically migrate contenttypes, auth, base_auth and shell)
2. auth
3. session_auth
4. sessions
5. open_auth
6. oauth2_provider
7. admin
8. guardian

After the above have been migrated, you have a fully functional Omniport core. But that is not all, you'll also have to migrate the following before the installation can be of any use.

1. *any remaining services*
2. *any remaining apps*

At any time during the migration process, run the following command to see what has been migrated and what is left to be migrated.

```
docker@intranet-server:/omniport$ python manage.py showmigrations
```

That's it. Now you have a fully migrated database and are ready to populate it.

5.7 . . . examine database?

While the Django ORM is beautiful in its capabilities and allows you to interact with the database in wonderfully Pythonic ways, there are always times when you have to get your hands dirty and enter the belly of the beast known as Postgres.

That's also simple to do in Omniport. Just enter the following command into the shell and you'll be dropped into the PostgreSQL prompt where you can issue SQL queries to your heart's content.

```
[apps omniport-docker]$ docker-compose exec database psql -U <user> -d <db>
```

Here <db> and <user> must be replaced with the values of POSTGRES_DB and POSTGRES_USER respectively, as written in the file `postgres/database.env`.

Here on out, refer to the PostgreSQL documentation for help.

5.8 . . . access app registry?

Discovery and Configuration allow Omniport to be extensible and tailored to the needs and requirements of a particular institute. But this comes at the cost of integrability as one app can never fully be sure of the existence of another.

Consider for example, the service Helpcentre which allows users to ask queries which may have to do with a particular app. To enable this functionality, Helpcentre needs to know, dynamically, the list of apps that have been installed in the Omniport ecosystem.

This is where accessing the app registry comes into the picture. This functionality enables developers of services like Helpcentre and Notifications to be aware of the apps installed and configured by the sysadmin.

Fortunately Discovery provides this feature. To use it, first import the function `available_apps`.

```
from discovery.available import available_apps
```

Then in a function where the request object is available, such as a view or a DRF viewset, invoke the `available_apps` function.

```
available_app_list = available_apps(request=request)
```

5.9 . . . monitor health?

All Omniport containers (except NGINX, so far) have healthchecks built into them to provide an easy way to monitor the health of containers in real time. These healthchecks run at 16 minute intervals.

The results of the healthchecks can be seen using the `ps` command afforded by Docker Compose.

```
[apps omniport-docker]$ docker-compose ps
```

And there you have it. Container health is mentioned in brackets in the 'State' column. The state can be one of several values.

Up (health: starting) if the health check was run on an upcoming container

Up (healthy) if the health check was run and reported a success

That's all you need to know to stay updated on the health of your containers!

CHAPTER

SIX

REFERENCES

Omniport provides a huge ocean of ready-to-use components and functions and classes. It's natural to feel lost every once in a while. And when you are lost, references are what will guide you from one built-in feature to the next.

These features of Omniport that are intended to make lives of developers' easier have been documented here. The documents here are not meant for sysadmins or users of the software but rather explain, in painful detail, the various aspects of Omniport that developers would want to incorporate into their apps.

Warning: These guides are *volatile* and are *bound to change* as we update Omniport.

The nature of Omniport's rolling-release development cycle prevents us from versioning these documents.

6.1 Formula 1

Formula 1 is a collection of compound components and frequently-used functions that aims to make it easier to develop apps for Omniport.

By pre-preparing these components and functions, Formula one makes it supremely fast and extremely easy to develop apps that look great and integrate cohesively with the rest of the platform.

Read more about the offerings of Formula 1 in these documents.

6.1.1 Components

Components make React apps what they are. Modular and reusable, they make frontend development easy. "What's better than components?", you ask? Components that are ready-made for your development joy!

Formula 1 provides a number of commonly used components so that your experience building Omniport apps is as easy as possible. They also eliminate the need to rewrite components for everything as well ensure that the apps look and feel consistent with the core as well as each other.

Read on to know what these components are, what they do, how you can use them and how the props you pass to them determine how they look and behave.

App header

The *app header* is always attached to the top of the app and shows information about the app or site such as its logo, name and user information.

Props

1. `mode`

   Type `enum (string)`

   Default `site`

   Description

   defines if `AppHeader` should display app-related branding or site-related branding
   Enums: `site`, `app`

2. `appName`

   Type `string`

   Conditions for use prop `mode` has value `app`

   Default ***required**

   Description

   used to get corresponding assets and app `baseUrl` from backend as well as frontend configs
   Note: **NOT** `appVerboseName`

3. `sidebarButton`

   Type `Boolean`

   Default `false`

   Description

   whether to display sidebar button to the left of the logo

4. `sideBarVisibility`

   Type `Boolean`

   Conditions for use prop `sideBarButton` has value `true`

   Default `false`

   Description

   whether the sidebar is opened (visible, `true`) or closed (hidden, `false`)

5. `onSidebarClick`

   Type `function()`

   Conditions for use prop `sideBarButton` has value `true`

   Default `null`

   Description

   the function that will be called when sidebar button is clicked

6. `hamburgerOptions`

   Type `[string]`

   Conditions for use prop `sideBarButton` has value `true`

Default

```
[
  'hamburger--minus',
  'hamburger--spin',
  'hamburger--squeeze'
]
```

Description

types of hamburger animations
Accepts any subset of the default list
For more information, click here

7. `userDropdown`

   Type `Boolean`

   Default `false`

   Description

   whether to display sign in button or user details and notification bell
   Displays institute logo, wordmark or name depending on availability

8. `middle`

   Type `<ReactNode>`

   Default `null`

   Description

   element, if any, to be rendered in the middle of the header

9. `right`

   Type `<ReactNode>`

   Default `null`

   Description

   element, if any, to be rendered on the right side of the header

Example usage

```
import { AppHeader } from 'formula_one'
```

Importing `AppHeader` from `formula_one`

```
const hamburgerOptions = [
  'hamburger--minus',
  'hamburger--spin'
]
```

Defining an array of strings to pass in `hamburgerOptions`

```
<AppHeader
  mode='app'
  appName='placement_and_internship'
  sideBarButton={true}
  sideBarVisibility={this.state.sideBarVisibility}
  onSidebarClick={this.handleSidebarClick}
  hamburgerOptions={hamburgerOptions}
```

(continues on next page)

(continued from previous page)

```
  userDropdown
/>
```

Using `AppHeader` in `app` mode

It will render page title, favicon, app name and app logo by querying the backend and handling all possible cases to render the best possible assets with a user dropdown or 'Login' button based on whether a user is logged in.

```
<AppHeader
  mode='site'
  userDropdown
/>
```

Using `AppHeader` in `site` mode

It will render site page title, site favicon, site name and site logo by querying the backend and handling all possible cases to render the best possible assets with a user dropdown or 'Login' button based on whether a user is logged in. .. code-block:: jsx

    <AppHeader />

Using `AppHeader` with absolutely no props

It will render the header in `site` mode and the institute logo in place of the user dropdown.

App main

`<AppMain>` is a wrapper around the container in between your `<AppHeader>` and `<AppFooter>` which applies some default style to the app.

Example usage

```
import { AppMain } from 'formula_one'

import style from '../css/app.css'
```

Importing `AppMain` from `formula_one`

```
<AppHeader {...props} />

<div styleName='style.app-container'>
  <AppMain>
      ...
  </AppMain>
</div>

<AppFooter {...props} />
```

Using `AppMain` in an app

App footer

The *app footer* is always attached to the bottom of the screen and shows copyright and maintainer information. The footer is also some the coolest code in the entirety of Omniport's frontend.

Props

1. `creators`

   Type

   ```
   [{
     name: string,
     role: string,
     <link: string>
   }]
   ```

   <> denotes optional field

   Default ***required**

   Description

   the creators who deserve credit for their app in the footer

Example usage

```
import { AppFooter } from 'formula_one'
```

Importing `AppFooter` from `formula_one`

```
const creators = [
  {
    name: 'Dhruv Bhanushali',
    role: 'Backend developer',
    link: 'https://dhruvkb.github.io/'
  },
  {
    name: 'Praduman Goyal',
    role: 'Frontend developer',
    link: 'https://pradumangoyal.github.io/'
  }
]
```

Defining an array of objects to pass in `creators`

```
<AppFooter creators={creators} />
```

Using `AppFooter`, passing creators list

Default DP

The *default DP* component displays the first letter of the prop `name` in the shape of circle with the theme colour as background. It can be used whenever display image is not provided by user.

Props

1. `name`

   Type `string`

   Default ***required**

   Description

   the string whose first letter will be shown in the component

2. `size`

   Type `string`

   Default `'1.5em'`

   Description

   the `font-size` property of the letter

Example usage

```
import { DefaultDP } from 'formula_one'
```

Importing `DefaultDP` from `formula_one`

```
<DefaultDP name='Praduman Goyal' size='3em' />
```

Using `DefaultDP` passing name and size

User card

A user card is a component which displays the user's information in an organised way, in an aesthetic card.

Props

1. `image`

   Type `string`

   Default `null`

   Description

   the source to pass in the image tag's `src` attribute

2. `name`

   Type `string`

   Default ***required**

   Description

   the name of a user, used as heading in the card as well as the prop for *the default DP component*

3. `size`

   Type `string`

   Default `'1.5em'`

   Description

   directly passed as a prop to *the default DP component*

4. `username`

   Type `string`

   Default `''`

   Description

   the sub-heading for the user card

5. `roles`

   Type `[string]`

   Default `''`

   Description

   an array of role names as strings, that are joined with `', '`

6. `right`

   Type `<ReactNode>`

   Default `null`

   Description

   the element to be shown on the right side of the component
   This is generally used to pass icons like 'cross', 'close' or 'check'.

Example usage

```
import { UserCard } from 'formula_one'
```

Importing `UserCard` from `formula_one`

```
const roles = ['Student', 'Frontend developer']
```

Defining an array of strings to pass in `roles`

```
<UserCard
    name='Praduman Goyal'
    username='pradumangoyal'
    roles={roles}
    image='/path/to/profile.jpg'
    size='3em'
    right={<Icon name='check' />}
/>
```

Using `UserCard`, passing all the props

Tile card

A *tile card* is a component that shows information about one unit of many similar units in an organised way, in an aesthetic card.

Props

1. `name`

   Type `string`

   Default ***required**

   Description

   the heading to be used in the card

2. `desc`

   Type `string`

   Default ***required**

   Description

   the description that is shown below the heading

3. `imageUrl`

   Type `string`

   Default `iconName`, see below

   Description

   the associated image to be display in the card

4. `iconName`

   Type `enum (string)`

   Default `group`

   Description

   the icon to display in case an image URL is not passed in the props
   Enums: For a comprehensive collection of valid inputs, refer to the icon list.

Example usage

```
import { TileCard } from 'formula_one'
```

Importing `TileCard` from `formula_one`

```
<TileCard
  name='Alohomora'
  desc='Reset passwords, no-questions asked'
  imageUrl='/path/to/image.jpg'
  iconName='cube'
/>
```

Using `TileCard`, passing all the props

Tiles

Tiles are grids of tile cards.

Props

1. `tiles`

   Type

   ```
   [{
     name: string,
     desc: string,
     iconName: string,
     imageUrl: string,
     link: link
   }]
   ```

   Default ***required**

   Description Refer to the props for *the tile card component*, except for `link`.

   link Defines URL to which corresponding `TileCard` should point.

Example usage

```
import { Tiles } from 'formula_one'
```

Importing `Tiles` from `formula_one`

```
const tiles = [{
  name: 'Alohomora',
  desc: 'Reset passwords, no-questions asked',
  imageUrl: '/path/to/image.jpg',
  iconName: 'cube',
  link: '/alohomora'
}]
```

Defining an array of objects to pass in `tiles`

```
<Tiles
  tiles={tiles}
/>
```

Using `Tiles`, passing all the props

Maintainer view

`MaintainerView` is a component that wraps components which are meant to be viewed only by maintainers with special permissions.

Props

1. `which`

   Type `string`

   Default ***required**

   Description

   the permission that the user, who must be a maintainer, is supposed to have

Example usage

```
import { MaintainerView } from 'formula_one'
```

Importing `MaintainerView` from `formula_one`

```
<MaintainerView which='helpcentre'>
  ...
</MaintainerView>
```

Using `MaintainerView`, passing prop `which`

Non-maintainer view

`<NonMaintainerView>` is a component that wraps components which are strictly useless for maintainers if they have certain permissions.

Props

1. `which`

   Type `string`

   Default ***required**

   Description

   the permission that the user, who must not be a maintainer, or is supposed not to have

Example usage

```
import { NonMaintainerView } from 'formula_one'
```

Importing `NonMaintainerView` from `formula_one`

```
<NonMaintainerView which='helpcentre'>
  ...
</NonMaintainerView>
```

Using `NonMaintainerView` passing prop `which`

Masonry layout

Masonry layouts are those that display staggered grids where cells may be of variable heights. This component is a wrapper around the similar components so that all the children are positioned in a way that the columns have the minimum difference between their bottom ends.

Props

1. `children`

   Type

   ```
   [
     <ReactNode
       ...
       image={Boolean}
     />
   ]
   ```

   Default *required

   Usage This prop differs from standard usage. See below.

   Description

   the list of elements that is to be shown in a masonry layout

2. `columns`

   Type `number`

   Default `2`

   Description

   number of columns into which the list of elements is to be divided

3. `gap`

   Type `number`

   Default `10`

   Description

   gutter in between the columns of the layout

Example usage

```
import { MasonryLayout } from 'formula_one'
```

Importing `MasonryLayout` from `formula_one`

```
<MasonryLayout columns={isBrowser ? 2 : 1} gap={28}>
  {feedList.map(feed => {
    return (
      <FeedCard
        key={feed.id}
        feed={feed}
        image={Boolean(feed.image)}
      />
    )
  })}
</MasonryLayout>
```

Using `MasonryLayout`, by passing number of columns and gap between them

Note: `children` is a special prop that must be passed as actual children in the React shadow DOM. Notice here that `children` actually refers to the `map` on `feedList` returning a list of `FeedCard` nodes.

Custom cropper

The *custom cropper* component is an exact copy of the `<ReactCrop>` component that we had to be included separately to apply styles on it. This is due to our CSS loading process which uses a hashed `styleName` prop instead of `className` like standard React.

Props

Refer to react-image-crop for all information.

Example usage

```
import { CustomCropper } from 'formula_one'
```

Importing `CustomCropper` from `formula_one`

Refer to react-image-crop for all information.

No match

The `<NoMatch>` component is a very beautiful full page 404 component with its own header and footer and is mobile responsive.

Props

There are absolutely none!

Example usage

```
import { NoMatch } from 'formula_one'
```

Importing `NoMatch` from `formula_one`

```
<Route component={NoMatch} />
```

Using `NoMatch` in main routes

```
<Redirect to='/404' />
```

An app can redirect to main 404 when there is no path route matched.

Note: The standard way to use it is to redirect to it. Using it within one's header and footer can unleash a world of ugly and nasty.

6.1.2 Functions

Formula 1 provides a number of commonly used functions out-of-the-box so that developers may share them and not have to reinvent the wheel. Another sweet advantage is that advancements to the metaphorical wheel can be reaped by all users automatically.

Get cookie

`getCookie()` is a function which makes consuming cookies *a piece of cake*. It returns the value of the cookie corresponding to name passed to it as parameter `cname`.

Signature

```
function getCookie(cname) {
    ...
    return value
}
```

Parameters

1. `cname`

   Type `string`

   Default ***required**

   Description the name of the cookie whose value is being retrieved

Return

Type `string`

Data

- value of the cookie whose name is `cname`
- `null` if the specified cookie is not found

Examples

```
import { getCookie } from 'formula_one'
```

Importing `getCookie()` from `formula_one`.

```
> getCookie('csrftoken')
'4OgZaiZcwEHAAeqnzqrn2jFubHe7IemgS07ZJJ0CltuQD3e0MSHIKaAtqeBb7HhD'
```

Using `getCookie()` to read the cookie named 'csrftoken'.

If role

`ifRole()` is a function which returns the status of a role corresponding to a passed array of a user's roles.

Signature

```
function ifRole(roles, role) {
    ...
    return status
}
```

Parameters

1. `roles`

   Type `[object]`

   Default ***required**

   Description the array of **role objects** obtained by requesting `who_am_i/`

   ```
   [
       {
           "role": "Student",
           "activeStatus": "ActiveStatus.IS_ACTIVE"
       },
       {
           "role": "Maintainer",
           "activeStatus": "ActiveStatus.IS_ACTIVE"
       }
   ]
   ```

2. `role`

   Type `string`

   Default ***required**

   Description the role whose existence is being checked

Return

Type `enum` (`string`)

Data

- `NOT_ROLE` if required role is not matched with any role in the array.
- **Enums:** `IS_ACTIVE`, `HAS_BEEN_ACTIVE`, `WILL_BE_ACTIVE`, `NOT_ROLE`

Examples

```
import { ifRole } from 'formula_one'
```

Importing `ifRole()` from `formula_one`

```
> const roles = [
            {
                "role": "Student",
                "activeStatus": "ActiveStatus.IS_ACTIVE"
            },
            {
                "role": "Maintainer",
                "activeStatus": "ActiveStatus.IS_ACTIVE"
            }
        ]

> ifRole(roles, 'Maintainer')
'IS_ACTIVE'

> ifRole(roles, 'Faculty')
'NOT_ROLE'
```

Using `ifRole()` to get the status of various role of a person

Common apps

`commonApps()` is a function which returns the list of apps obtained by the intersection of two lists, one being the API list obtained by requesting on `apps/app/` and one being set in `config.json` at frontend.

Signature

```
function commonApps(apiList) {
  ...
  return commonList
}
```

Parameters

1. `apiList`

   Type `[object]`

   Default **required**

   Description the array of **app objects** obtained by requesting `apps/app/`

```
[
  {
    "nomenclature": {
      "name": "one",
      "verboseName": "One"
    },
    "baseUrls": {
      "http": "one/",
      "ws": null,
      "static": "one/"
    },
```

(continues on next page)

(continued from previous page)


```
    "assets": {
      "favicon": "assets/favicon.ico",
      "icon": null,
      "logo": "assets/logo.svg"
    },
    "description": "App number one"
  },
  {
    "nomenclature": {
      "name": "two",
      "verboseName": "Two"
    },
    "baseUrls": {
      "http": "two/",
      "ws": null,
      "static": "two/"
    },
    "assets": {
      "favicon": "assets/favicon.ico",
      "icon": "assets/icon.svg",
      "logo": "assets/logo.svg"
    },
    "description": "App number two"
  }
]
```


Return

Type `[object]`

Data intersection of the two list with the object in the following form


```
{
  "assets": {
    "favicon": "assets/favicon.ico",
    "icon": "assets/icon.svg",
    "logo": "assets/logo.svg"
  },
  "baseUrl": "/one",
  "baseUrls": {
    "http": "one/",
    "ws": null,
    "static": "one/"
  },
  "description": "App number one",
  "nomenclature": {
    "name": "one",
    "verboseName": "One"
  },
  "source": "one/src/index"
}
```

Examples

```
import { commonApps } from 'formula_one'
```

Importing `commonApps()` from `formula_one`

```
commonApps(apiList)
```

Using `commonApps()`, returns an array of object of type described above

App details

`appDetails()` is a function which returns the config object stored at the frontend of the requested app according to their `appName` (**not** `appDisplayName`) which is generally of the form `placement_and_internship`.

Signature

```
function appDetails('placement_and_internship') {
  ...
  return {
    present: {bool},
    details: {object}
  }
}
```

Parameters

1. appName

   Type `string`

   Default ***required**

   Description the app whose config object is required (**not** `appDisplayName`)

Return

Type `{object}`

Data Object containing `present` and `details`

present `bool` that specifies whether the app exists

details App config object defined in `config.json` of the app

Examples

```
import { appDetails } from 'formula_one'
```

Importing `appDetails()` from `formula_one`

```
> appDetails('placement_and_internship')
{
  present: true,
  details: {
    nomenclature: {
      name: "placement_and_internship",
      verboseName: "Placement and Internship"
    },
    baseUrl: "/placement_and_internship",
    source: "placement_and_internship/src/index"
  }
}
```

Using `appDetails()` to get config of an app named `placement_and_internship`

Get theme

`getTheme()` is at the heart of `Omniport` themes. It returns the current theme colour out of available Semantic colours.

Signature

```
function getTheme() {
    ...
    return theme
}
```

Return

Type `string`

Data

- can be passed directly to `Semantic UI` components
- returns currently active theme colour as `string`
- returns `'blue'` if no colour is set and also sets theme colour to `'blue'`
- **Enums:** `red`, `orange`, `yellow`, `olive`, `green`, `teal`, `blue`, `violet`, `purple`, `pink`, `black`

Examples

```
import { getTheme } from 'formula_one'
```

Importing `getTheme()` from `formula_one`

```
> getTheme()
'orange'
```

Using `getTheme()` to get currently selected theme

Get theme object

`getThemeObject()` is the function that returns colour description object of currently selected theme.

Signature

```
function getThemeObject() {
    ...
    return themeObject
}
```

Return

Type `{object}`

```
{
    name: {string},
    verboseName: {string},
    description: {string},
    hexCode: {string}
}
```

Data

- returns an object describing active theme colour
- returns the object corresponding to the colour `'blue'` if no colour is set and also sets theme colour to `'blue'`
- can be used when active theme is to be passed to JSX elements' `style`

Examples

```
import { getThemeObject } from 'formula_one'
```

Importing `getThemeObject()` from `formula_one`

```
> getThemeObject()
{
    name: "orange",
    verboseName: "Maverick Orange",
    description: "Unorthodox and chaotic",
    hexCode: "#f2711c"
}
```

Using `getThemeObject()` to get current selected theme object

6.1.3 URLs

`urls.js` is a file in `formula_one` that contains some URL functions that could be potentially useful in making API calls to various endpoints.

You can check read and go through the entire file on GitHub.

Usage

```
import { urlAppBranding } from 'formula_one'
```

Importing one of the URLs from `formula_one`

```
> urlAppBranding('placement_and_internship')
'/api/apps/app/placement_and_internship/'
```

Using one of the URLs from `formula_one`

6.2 Scripts

Omniport comes with a number of Bash scripts to perform a number of actions. From setting up the entire Omniport codebase to starting an app, everything that can be scripted will be scripted. That's the Omniport way of things!

Read more about the scripts to know about their function, their customisation flags and options.

6.2.1 Docker scripts

The Docker distribution of Omniport comes loaded with a bunch of scripts to help you smoothly get your own instance of the portal up and running with minimal hassle, as little pain as can be caused, negligible to zero need for cursing, no crying or banging your head in anguish.

Clone scripts

These scripts pertain to the process of cloning various repositories from GitHub and setting up the codebase on the local machine.

Everything

`clone/everything.sh` aims to set up whole Omniport codebase in one command with the basic services that Omniport expects to run smoothly on both the frontend as well as the backend.

The script takes you through the steps of cloning the codebase.

> **Warning:** Running this script will remove any existing `codebase/` folder in the root directory.

Usage

```
[apps omniport-docker]$ ./scripts/clone/everything.sh
```

Working

Basically, all this script does is call two sub-scripts which in turn clone the *backend* and *frontend* codebases.

Backend

`clone/backend.sh` sets up `omniport-backend` inside the `codebase/` folder with the `services` required at the backend.

> **Warning:** Running this script will remove any existing `omniport-backend/` folder inside the `codebase/` directory.

Usage

```
[apps omniport-docker]$ ./scripts/clone/backend.sh
```

Working

The script clones the repository `omniport-backend` and then defers the cloning process to the `scripts/clone/everything.sh` *script* in that repository.

Frontend

`clone/frontend.sh` sets up `omniport-frontend` inside the `codebase/` folder with the `services` required at the frontend.

> **Warning:** Running this script will remove any existing `omniport-frontend/` folder inside the `codebase/` directory.

Usage

```
[apps omniport-docker]$ ./scripts/clone/frontend.sh
```

Working

The script clones the repository `omniport-frontend` and then defers the cloning process to the `scripts/clone/everything.sh` *script* in that repository.

Build scripts

These scripts pertain to the process of building images for various services involved in the project.

PostgreSQL

This script builds the PostgreSQL container that acts as the primary database for all Omniport apps and services.

```
[apps omniport-docker]$ ./scripts/build/postgres.sh
```

You will get the choice regarding the recreation of the database environment file. Affirm with a `Y` if you intend to rewrite it, else decline with any other character, primarily `N`.

> **Warning:** If this is your first time building the container, choose `Y`. Choosing `N` can have adverse unwanted consequences.

If you choose to recreate the environment file, you will have to answer a questionnaire. Enter a database name, a username and a password for the database.

Remember to populate this database name, username and password in the *Django project-level configuration file*.

RabbitMQ

This script builds the RabbitMQ container that acts as the message-broker for all Omniport apps and services.

```
[apps omniport-docker]$ ./scripts/build/rabbitmq.sh
```

You will get the choice regarding the recreation of the message broker environment file. Affirm with a `Y` if you intend to rewrite it, else decline with any other character, primarily `N`.

> **Warning:** If this is your first time building the container, choose `Y`. Choosing `N` can have adverse unwanted consequences.

If you choose to recreate the environment file, you will have to answer a questionnaire. Enter a username and a password for the message broker.

Remember to populate this username and password in the *Django project-level configuration file*.

NGINX

This script builds the NGINX container that is responsible for operating the reverse proxy to the API, that also serves all our static and media files.

```
[apps omniport-docker]$ ./scripts/build/nginx.sh
```

You will get the choice regarding the recreation of NGINX configuration files. Affirm with a `Y` if you intend to rewrite them, else decline with any other character, primarily `N`.

> **Warning:** If this is your first time building the container, choose `Y`. Choosing `N` can have adverse unwanted consequences.

If you choose to recreate the configuration files, you will have to answer a questionnaire. Enter the domain names that you want to refer to the intranet and Internet sites of the project.

For example, at IIT Roorkee, the hostnames for Omniport are `channeli.in` and `channeli.iitr.ac.in` from inside and outside the campus respectively.

Also choose whether to enable SSL with a `Y` if you have SSL certificates or with a `N` if you don't. The configuration files will be regenerated and the image built.

Django

This script builds the Django container that is responsible for operating the API powering Omniport.

```
[apps omniport-docker]$ ./scripts/build/django.sh
```

You will get the choice regarding the installation of developer tools. Affirm with a `Y` if you are going to be using the image for debugging, else decline with any other character, primarily `N`. If you are unsure, press `Y`, although this is not the default for image size concerns.

React

This script builds the React container that is responsible for operating the frontend powering Omniport.

```
[apps omniport-docker]$ ./scripts/build/react.sh
```

You will get the choice regarding the installation of developer tools. Affirm with a `Y` if you are going to be using the image for debugging, else decline with any other character, primarily `N`. If you are unsure, press `Y`, although this is not the default for image size concerns.

Start scripts

These scripts pertain to the process of cloning various repositories from GitHub and setting up the codebase on the local machine.

Development

This script starts all containers that are required to set up the Omniport development environment as outlined in *development environment*.

```
[apps omniport-docker]$ ./scripts/start/development.sh
```

This will start all shared containers required for the functioning of the development Django containers.

Logs

This script starts the Bash container needed for examining the logs generated by various services in production.

```
[apps omniport-docker]$ ./scripts/start/logs.sh
```

This will drop you into a familiar but foreign Bash shell. Executing `ls` here reveals the log directories.

```
# ls
reverse_proxy_logs
web_server_logs
...
```

The `web_server_logs` directory further contains three directories.

```
# cd web_server_logs
# ls
daphne_logs
gunicorn_logs
supervisord_logs
...
```

All these directories contains various logs that can be examined. The container also supports the `tail` binary fortunately enabling you to tail and detect errors.

For explanations regarding the directory and file structure, refer to the *guide on how to read logs*.

6.2.2 Backend scripts

The backend codebase of Omniport comes loaded with a bunch of scripts to help you get all the services you need up and running as well as to get started with your app.

Clone scripts

Cloning the numerous services and packages required by Omniport is not easy. Or, should I say, *was* not easy. Now that cloning scripts have been written setting up Omniport backend is child's play.

Read on to learn how you can use these clone scripts to remove the effort from setting up the codebase.

Everything

This script clones all the necessary services, as well as the shell for IIT Roorkee (with the developer's consent), in their appropriate directories in the `omniport-backend` repository.

Usage

```
[dev omniport-backend]$ ./scripts/clone/everything.sh
```

You will be asked if you wish to install the IIT Roorkee shell. You can approve the installation with `Y` if you are an IIT Roorkee resident or otherwise wish to experience the customisation Omniport allows. You can deny this installation with any other key, primarily `N`.

Services

This script clones all the necessary services in the `omniport/services` directory of the `omniport-backend` repository.

Usage

```
[dev omniport-backend]$ ./scripts/clone/services.sh
```

Formula 1

This script clones the IIT Roorkee shell in the `omniport/` directory of the `omniport-backend` repository.

Usage

```
[dev omniport-backend]$ ./scripts/clone/shell.sh
```

Start Django

Note: This process requires the Django container to have been built beforehand. If that is not the case, read *how to build the Omniport Django container*.

This script starts a Django container with the code mounted in it making it easy to deploy the development version of the app in a container and pre-configure it to find the service containers and connect to them.

Usage

```
[dev omniport-frontend]$ ./scripts/start/django.sh
```

Flags

-p

The port to which the Django developer server must bind

`number`

Create app

Note: After you have an up and running API make sure to create a React app for the frontend of the project.

Although creating a new app on `omniport-backend` is really easy, the main reason being that the process of creating one and the file structure are both pretty much identical to a normal Django app, we have provided a convenient method to make this process effortless for the developer.

Usage

```
[dev omniport-backend]$ ./scripts/create/app.sh
```

You are asked for the name of your app that you have to provide in lower case letters, separating words with spaces. For example, for an app named 'Placement and internship' you must enter

```
placement and internship
```

That's about it! It clones a pre-defined template from GitHub, makes necessary replacements with the app name you have provided and creates a new folder in the `omniport/apps/` folder.

Restarting the server will make Discovery aware of your new project. The root URL of your newly created API can be deduced by replacing spaces in the app name you provided with underscores, _. This is, needless to say, customisable via the `config.yml` file.

You should be greeted with a JSON API, greeting you on your success.

6.2.3 Frontend scripts

The frontend codebase of Omniport comes loaded with a bunch of scripts to help you get all the services you need up and running as well as to get started with your app. Finally when it's all done, you can build your frontend and push it to the NGINX container for deployment.

Clone scripts

Cloning the numerous services and packages required by Omniport is not easy. Or, should I say, *was* not easy. Now that cloning scripts have been written setting up Omniport frontend is child's play.

Read on to learn how you can use these clone scripts to remove the effort from setting up the codebase.

Everything

This script clones all the necessary services, as well as Formula 1, in their appropriate directories in the `omniport-frontend` repository.

Usage

```
[dev omniport-frontend]$ ./scripts/clone/everything.sh
```

Services

This script clones all the necessary services in the `omniport/services` directory of the `omniport-frontend` repository.

Usage

```
[dev omniport-frontend]$ ./scripts/clone/services.sh
```

Formula 1

This script clones Formula 1 in the `omniport/` directory of the `omniport-frontend` repository.

Usage

```
[dev omniport-frontend]$ ./scripts/clone/formula_one.sh
```

Start React

Note: This process requires the React container to have been built beforehand. If that is not the case, read *how to build the Omniport React container*.

This script starts a React container with the code mounted in it making it easy to deploy the development version of the app in a container and pre-configure it to find the API container and connect to it.

Usage

```
[dev omniport-frontend]$ ./scripts/start/react.sh -d <DJANGO PORT>
```

Flags

-d

The Django port where the API is up and running
`number`, **required**

-p

The port to which the React developer server must bind
`number`

Create app

Note: Make sure you have the corresponding backend ready before you start the frontend to avoid any nasty surprises.

Although creating a new app on `omniport-frontend` is really easy, the main reason being that the process of creating one and the file structure are both pretty much identical to a normal React app, we have provided a convenient method to make this process effortless for the developer.

Usage

```
[dev omniport-frontend]$ ./scripts/create/app.sh
```

You are asked for the name of your app that you have to provide in lower case letters, separating words with spaces. For example, for an app named 'Placement and internship' you must enter

```
placement and internship
```

That's about it! It clones a pre-defined template from GitHub, makes necessary replacements with the app name you have provided and creates a new folder in the `omniport/apps/` folder.

Restarting the server will make Discovery aware of your new project. The root URL of your newly created app can be deduced by replacing spaces in the app name you provided with underscores, `_`. This is, needless to say, customisable via the `config.json` file.

You should be greeted with a pleasant homepage.

Build frontend

Note: This process requires the React container to have been built beforehand. If that is not the case, read *how to build the Omniport React container*.

This scripts builds the production React app from the frontend codebase and places the built files in a folder named `build/`.

Usage

```
[dev omniport-frontend]$ ./scripts/build/frontend.sh
```

6.3 Configuration files

Configuration files are the heart and soul of Omniport. At the core of Omniport's flexibility and customisability is a set of configuration files which are either convenient YAML files on the backend or natively parseable JSON files on the frontend.

Read on to know more about these assortment of files and the various aspects of Omniport they control in order to tweak Omniport exactly to your needs.

6.3.1 Project configuration

Omniport allows for a lot of customisability on the project level. In fact, almost everything about Omniport, including the name, can be changed without writing or editing a single line of code. We take great pride in this creation of ours and are always adding more and more items that can be customised externally.

Read on to find the various aspects of the Omniport project that you can customise.

`base.yml` on the backend

Purposes

`base.yml` is the top-level of customisation that can be achieved with Omniport. The values specified here determine a lot of things about how Omniport behaves and responds.

Branding

The branding part of the config file is used to define the branding of the Omniport project, mainly the textual aspect of branding. It requires the `branding` key to be defined in `base.yml`.

branding

Branding related fields of the project
`{object}` which has the following subkeys

institute

Attributes pertaining to the brand of the institute hosting the portal
`{object}` which has the following subkeys

acronym

Acronyms of institutes which have long names
`string`

name

The full name of the institute
`string`

homePage

The URL to the home page of the institute on the web
`string`

maintainers

Attributes pertaining to the brand of the maintainers behind the portal
`{object}` which has the following subkeys

acronym

Acronyms of maintainer groups which have long names
`string`

name

The full name of the maintainer group
`string`

homePage

The URL to the home page of the maintainer group on the web
`string`

Internationalisation

The internationalisation part of the config file is simply to customise the language and timezone of the project.

i18n

Internationalisation related fields of the project
`{object}` which has the following subkeys

languageCode

The languge code of the portal
`string`

timeZone

The languge code of the portal
`string`

Secrets

Secrets pertaining to the Omniport project, such as passwords, client IDs and most importantly the Django secret key go here.

secrets

Secrets of the Omniport project
`{object}` which has the following subkeys

secretKey

The Django secret key of the project
`string`

Services

As Omniport depends on a lot of services for optimum functioning, we made a Docker project to make the setup easy. If you use the Docker project, a lot of these settings have to be their default values as shown in the `base_stencil.yml` file. If you have tweaked them, fill these sections out.

services

Credentials to connect to all Omniport dependencies
`{object}` which has the following subkeys

database

Attributes pertaining to database service
`{object}` which has the following subkeys

host

The IP/name of the host machine or container running this service
`string`

port

The port on which the service can be reached
`integer`

user

The username that can be used to log in to the service
`string`

password

The password for the given user
`string`

name

The name of the database to use
`string`

messageBroker

Attributes pertaining to message broker service
`{object}` which has the following subkeys

host

The IP/name of the host machine or container running this service
`string`

port

The port on which the service can be reached
`integer`

user

The username that can be used to log in to the service
`string`

password

The password for the given user

`string`

channelLayer

Attributes pertaining to Redis container supporting Django Channels
`{object}` which has the following subkeys

host

The IP/name of the host machine or container running this service
`string`

port

The port on which the service can be reached
`integer`

sessionStore

Attributes pertaining to Redis container storing sessions
`{object}` which has the following subkeys

host

The IP/name of the host machine or container running this service
`string`

port

The port on which the service can be reached
`integer`

communicationStore

Attributes pertaining to Redis container storing communications
`{object}` which has the following subkeys

host

The IP/name of the host machine or container running this service
`string`

port

The port on which the service can be reached
`integer`

verificationStore

Attributes pertaining to Redis container storing verifications
`{object}` which has the following subkeys

host

The IP/name of the host machine or container running this service
`string`

port

The port on which the service can be reached
`integer`

notificationStore

Attributes pertaining to Redis container storing notifications

`{object}` which has the following subkeys

host

The IP/name of the host machine or container running this service
`string`

port

The port on which the service can be reached
`integer`

applicationStore

Attributes pertaining to Redis container storing data for omniport applications which are temporary in nature
`{object}` which has the following subkeys

host

The IP/name of the host machine or container running this service
`string`

port

The port on which the service can be reached
`string`

cache

Attributes pertaining to the Memcached cache
`{object}` which has the following subkeys

host

The IP/name of the host machine or container running this service
`string`

port

The port on which the service can be reached
`integer`

Integrations

Useful applications that can connect to Django, like Sentry, can be integrated into the project.

Emails

The emails part of the config file is used to define settings for sending emails from your project.

emails

Email configurations for your project
`{object}` which has the following subkeys

emailBackend

The backend used for sending emails
`string`

emailHost

The email service provider used
`string`

emailUseTls

Whether an email uses a TLS connection or not
`boolean`

emailPort

The port used by smtp server
`integer`

emailHostUser

The email address from which all emails will be sent
`string`

emailHostPassword

The password of the host's email account
`string`

IP address rings

Omniport being the web portal serves a number of clients on different levels of the network. There is the server itself, a close cluster of servers working together, certain cabinet computers that can connect to the server, the lab of the maintainer group, the intranet and then the Internet at large.

These clients can be arranged into rings, where the innermost rings have the highest priority in terms of privilege to access certain APIs and resources.

These rings are described here, in the `ipAddressRings` key.

ipAddressRings

`[{object}]` each of which has the following subkeys

name

The name of this IP address ring, the identifier for this ring
`string`

patterns

The regex of the IP pattern that falls in the level
`[string]`

Examples

A fully equipped `base.yml` file, with all settings populated looks quite like this.

```yaml
branding:
  institute:
    acronym: IIT-R
    name: Indian Institute of Technology Roorkee
    homePage: https://iitr.ac.in/
  maintainers:
    acronym: IMG
    name: Information Management Group
    homePage: https://channeli.in/img/
i18n:
  languageCode: en-gb
```

(continues on next page)

(continued from previous page)

```
  timeZone: Asia/Kolkata
secrets:
  secretKey: '2)@2klj=@a(*o9kyt7u^!g4jbqrqo3$ju^o_g6n*lh-d$$#zdy'
services:
  database:
    host: database
    port: 5432
    user: omniport_user
    password: omniport_password
    name: omniport_database
  channelLayer:
    host: channel-layer
    port: 6379
  sessionStore:
    host: session-store
    port: 6379
  communicationStore:
    host: communication-store
    port: 6379
  verificationStore:
    host: verification-store
    port: 6379
  notificationStore:
    host: notification-store
    port: 6379
  applicationStore:
    host: application-store
    port: 6379
  cache:
    host: cache
    port: 11211
  messageBroker:
    host: message-broker
    port: 5672
    user: omniport_user
    password: omniport_password
emails:
  emailBackend: 'django.core.mail.backends.smtp.EmailBackend'
  emailHost: 'smtp.example.com'
  emailUseTls: True
  emailPort: 587
  emailHostUser: 'no-reply@omniport.com'
  emailHostPassword: 'img@password'
ipAddressRings:
- name: self
  patterns:
  - '^172\.18\.0\.1$'
- name: specifics
  patterns:
  - '^172\.25\.55\.101$'
  - '^172\.25\.55\.219$'
- name: maintainers
  patterns:
  - '^172\.25\.55\.\d{1}$'
- name: intranet
  patterns:
  - '.*'
- name: internet
  patterns:
  - '.*'
```

`site_<id>.yml` on the backend

Purposes

`site_<id>.yml` is cascaded onto base-level customisation on a per-site basis. The values specified here determine a lot of things about how each sub-site of Omniport behaves and responds.

You can override the values defined in *the base config file*. In addition, there are certain configuration knobs that make sense to be defined on a site-level, which are explained below.

Site

Every site in Omniport can be individually customised.

Note: So far Omniport has support only for two sites. Work is ongoing to expand this to as many sites as needed.

Omniport allows you to customise the name, description as well as the debugging status of every site in the project. This is accomplished by the `site` key in the config file.

site

The set of fields for branding a site
`{object}` which has the following subkeys

id

The unique ID of the site
`number` where 0, 1, 2 denote development, intranet and Internet sites

nomenclature

The nomenclature pertaining to the site
`{object}` which has the following subkeys

name

The code name of the site
`string`

verboseName

The displayed name of the site
`string`

debug

Whether to display debug info, should be `false` on production sites
`boolean`

description

The description of the site, as an aid to help developers find their way
`string`

Allowances

Allowances determine what parts of Omniport each site is allowed to access. Fields such as the names of apps that are allowed on the site and the hostnames that the site should respond to go here. This section also determines the network rings each site is allowed on.

allowances

The set of allowed entities for any given site
`{object}` which has the following subkeys

hosts

List of domain names that this site is responsible for serving
`[string]`

apps

List of app names that this site exposes to the users
`[string]`

ipAddressRings

List of network rings that are served by the site
These names are ring names from *the base config file*
`[string]`

Examples

A site-level configuration for a typical development site `site_0.yml` looks like this.

```yaml
site:
  id: 0
  nomenclature:
    name: development
    verboseName: Omniport Dev
  debug: true
  description: Development site for Omniport
allowances:
  apps: __all__
  ipAddressRings:
  - self
  - specifics
  - administrators
  - maintainers
  - intranet
  - internet
```

A site-level configuration for a typical production site, more specifically the Intranet site `site_1.yml` looks like this.

```yaml
site:
  id: 1
  nomenclature:
    name: intranet
    verboseName: Omniport Intranet
  debug: true
  description: Intranet site for Omniport
allowances:
  hosts:
  - omniport.intranet
  - intranet.channeli.in
```

(continues on next page)

(continued from previous page)

```
  ipAddressRings:
  - intranet
  - maintainers
  - specifics
```

6.3.2 App configuration

App configuration files are one file each on the backend and frontend that collectively make up the customisation framework on the app level.

Read on to find the various aspects of an app that you can customise.

`config.yml` on the backend

Purposes

`config.yml` is the crux of Omniport's plug-and-play architecture. It primarily serves the following purposes.

These configurations are processed by the modules `discovery` and `configuration` and their processed data can be accessed as follows.

```
from django.conf import settings

DISCOVERY = settings.DISCOVERY

...  # Do something with DISCOVERY
```

This `DISCOVERY` object of `class Discovery` contains the fields `service_configuration_map` and `app_configuration_map` which can be used in your apps.

Nomenclature

The nomenclature part of the config file is used to define the internal and display name of the app, which in turn, defines the app nomenclature all around the code. It requires the `nomenclature` key to be defined in `config.yml`.

nomenclature

Nomenclature related fields of an app
`{object}` which has the following subkeys

name

The name that is used internally to refer to an app
`string`

verboseName

The display name of the app with spaces and proper punctuation
`string`

Description

This is the short app description that populates a number of places where the app description is shown such as on the `apps/` page of service `apps`.

description

A brief description of the app
`string`

Is API

This boolean field toggles whether the root URL of the app must be at `/` or at `/api/`. For all modern apps, this field must be set to `true`.

Only legacy apps or special apps that need to run from the root URL path must set this field to false.

isApi

Decides whether the app is an API or not
`boolean` set to `true`

Acceptables

Note: This section only applies to apps.

The modules `discovery` and `configuration` while processing the config files also determine the various target demographics for any given app. The `acceptables` key defines just that. The filters can be applied on the basis of user roles, their corresponding active statuses and the IP address of the request's origin.

This means that different users, having different roles, coming from different IP address rings will be able to see different apps.

acceptables

The filters deciding the target demographic of an app
`{object}` which has the following subkeys

ipAddressRings

The list of strings naming the IP address rings this app will serve
Should be a subset of `ipAddressRings` defined in `base.yml`.
`[string]`

roles

The roles this app will serve
`[{object}]` where each has the following subkeys

name

The name of any allowed role
Should be one of the roles defined in the `kernel` module.
`string` such as `'Student'` or `'FacultyMember'`

activeStatuses

The active statuses this app will serve
Should be one of the following.
`IS_ACTIVE`
`IS_INACTIVE`

HAS_BEEN_ACTIVE
WILL_BE_ACTIVE
`[string]` where each string is one of the above values.

Examples

Service

An example `config.yml` for a service can be as follows.

```yaml
nomenclature:
  name: apps
  verboseName: Apps
description: Find all the apps that make up Omniport
baseUrls:
  http: apps/
isApi: true
```

App

An example `config.yml` for an app that allows active maintainers to reset people's passwords from within their workplace can be as follows.

```yaml
nomenclature:
  name: alohomora
  verboseName: Alohomora
description: Reset passwords, no-questions asked
baseUrls:
  http: alohomora/
  static: alohomora/
acceptables:
  ipAddressRings:
  - self
  - specifics
  - maintainers
  roles:
  - name: Maintainer
    activeStatuses:
    - IS_ACTIVE
isApi: true
```

An example `config.yml` targeting alumni, and alumnae, living anywhere in the world can be as follows.

```yaml
nomenclature:
  name: alumni_connect
  verboseName: Alumni connect
description: Talk to alumni and learn from their experience
baseUrls:
  http: alumni_connect/
  static: alumni_connect/
acceptables:
  roles:
  - name: Student
    activeStatuses:
    - HAS_BEEN_ACTIVE
isApi: true
```

`config.json` on the frontend

Purposes

`config.json` is one important pillar of Omniport's plug-and-play architecture. It primarily serves the following three purposes.

These configuration files are used to generate `omniport\core\configs.json` and `omniport\core\primarySidebarConfigs.json` which can be used in your apps.

Nomenclature

The nomenclature part of the config file is used to define the internal and display name of the app, which in turn, defines app nomenclature all around the code. It requires the `nomenclature` key to be defined in `config.json`.

nomenclature

Nomenclature related fields of an app.
`{object}` which has the following subkeys

name

The name that is used internally to refer to an app.
`string`

verboseName

The display name of the app with spaces and proper punctuation.
`string`

Dynamic routes

`omniport/core/configs.json` created during build using `discovery.js` is used to create dynamic `Route` component which act as an entry point in Omniport for the service or app. It requires two keys in config i.e. `baseUrl` and `source`.

baseUrl

The URL on which app has to be rendered.
`string`

source

The source to the `index.js` file of the app relative to the `omniport` directory.
`string`

Sidebar configuration

Note: This section only applies to services.

`discovery.js` also creates `omniport/core/primarySidebarConfigs.json` using config files in sub-folders of services. This `primarySidebarConfigs.json`, as the name suggests is used to create the sidebar the appears in Omniport core. This functionality allows Omniport to pin some of services to the sidebar at a position determined by its priority.

primarySidebar

primarySidebar related fields of a service.

`{object}` which has the following subkeys.

icon

The name of the icon from the collection provided by Semantic UI.
`string`

priority

Defines the placement of the button in the sidebar.
Services with higher priority appear higher in the sidebar.
If not provided, it is assumed ~infinite.
If two services have the same priority, they are sorted lexicographically.
`integer`

Examples

Service

An example `config.json` for a service **having its button** in sidebar can be as follows.

```
{
  "nomenclature": {
    "name": "developer",
    "verboseName": "Developer"
  },
  "baseUrl": "/developer",
  "source": "developer/src/index",
  "primarySidebar": {
    "icon": "code",
    "priority": 6
  }
}
```

An example `config.json` for a service, which **doesn't require its button** in sidebar can be as follows.

```
{
  "nomenclature": {
    "name": "auth",
    "verboseName": "Auth"
  },
  "baseUrl": "/auth",
  "source": "auth/src/index"
}
```

App

An example `config.json` for an app can be as follows.

```
{
  "nomenclature": {
    "name": "placement_and_internship",
    "verboseName": "Placement and Internship"
  },
  "baseUrl": "/placement_and_internship",
  "source": "placement_and_internship/src/index"
}
```

CHAPTER

SEVEN

LEGAL

The use of Omniport as a user, a developer and/or as a local instance maintainer is bound by a number of rules. This section is devoted to explicitly outlining these terms.

Do note that above and beyond all these rules is a rule of ethics that we are all bound by as developers, sharing code and knowledge in the fair market that is open-source.

7.1 Privacy policy

The privacy policy of Omniport is still being drafted. Please be patient.

7.2 Developer terms of use

7.2.1 Preamble

These Omniport developer terms of use (the “developer terms”) are between the Information Managment Group (“IMG”), the individual or organisation managing the installation of Omniport in a given institution (the “local instance maintainers”) and the individual or organisation (the “developer”) agreeing to these terms.

These terms govern the developer’s access to and usage of services and products provided by IMG including APIs, SDKs, scripts, tokens, sessions, data, information and documentation (the “platform”).

By choosing to use one or more of the platform’s features in your application (the “app”), the developer agrees to be bound to these terms as a developer. If the developer uses the platform on behalf of another individual or organisation, the developer must have the authority to bind that entity to these terms, otherwise cease the use of the platform and its features entirely.

7.2.2 OAuth2 applications

Use of the platform

The developer will comply with all Omniport policies. The maintainers of the running instance of Omniport will, at their discretion, approve or deny any application submitted for review.

Prohibited actions

The developer warrants that it will not (attempt to)

- violate, encourage or facilitate the violation of Omniport's acceptable use policy.
- use or alter the portal, or any element thereof without explicit consent from the maintainers.
- mislead users to collect, alter, use or delete user data without the explicit consent of the user.

Platform usage

The maintainers in their infinite wisdom, may, in their own discretion, choose to

- review and monitor the platform including detailed analytics of various sections of the platform, but are not obligated to do so. Omniport is not responsible for data and content accessible via the platform.
- restrict any developer's use of the platform. The developer may reach out to the maintainers in case the restrictions are not fair. However the developer is not to circumvent these restrictions.
- issue updates to the platform from time to time, not all of which will be backwards compatible. Any applications using the API endpoints may need to be reworked.

Open-source

The platform and all of its services are open-source. The source code of these are accessible online. However there are a few components and apps of the platform that are not open-source and will never be. The developer must respect the licenses of all of these components.

7.2.3 Security

To the extent the developer possesses or has access to any token issued by the platform, the developer must

- prevent loss, theft, damage or unauthorised access to any bit of user data and tokens using no less than the industry standard security measures.
- maintain a comprehensive security program based on reasonable organizational, physical, and technical security controls.
- use secure communication protocols such as SSL or TLS and Hypertext Transfer Protocol Secure (HTTPS) enabled by default for any data that is transmitted to and from the platform.
- promptly report to the maintainer any known or suspected security breach involving the platform and provide reasonable assistance to the maintainers to patch the breach. n the event of a security breach, prior to issuing any public statements or responses to third party inquiries, the developer will work in good faith with the maintainers to coordinate a statement or response, unless prohibited by law.
- require the user to be authenticated via the OAuth2 flow before accessing the data.
- not copy, use, or store any login credentials (including name, email address, password, and access tokens) except as necessary in connection with the initial user authentication; provided that developer will only store login credentials within the app and will promptly delete login credentials once the user has completed the initial authentication.
- not collect any login credentials from or allow users to input any login credentials into any user interface other than the platform login page, as described in the documentation.
- not impersonate the Omniport login page in any way, not use a custom login page as a proxy or use a layout or design anywhere on your site that may lead the user to believe they are on the platform.

7.2.4 User data

The developer must

- ensure that data is collected, used, processed, transmitted and maintained in compliance with a privacy policy that is made available to users and that clearly and accurately describes the data that is collected and how it is used by the developer and by any third party that is privy to this information or any additional information generated from it.
- notify the users of its responsibility in maintaining the privacy, security and integrity of the data that is collected or accessed by the developer.
- never override the users' instructions, preferences or authorisations in the use or disclosure of their data.

7.2.5 Other platforms and users

The developer is responsible for

- respecting the terms and conditions for all the platforms and operating systems on which the app is distributed and used.
- ensuring that all queries, complaints and feedback of the users pertaining to the app is addressed by the developer and is not targeted or redirected to the maintainers.
- making it clear that the app, although integrated with the platform is not a part of it and that the maintainers do not, in any way, endorse or condone any action undertaken by the app or the developer.

7.2.6 Intellectual property

Omniport trademark license

The maintainers grant the developer a non-exclusive, non-transferable, non-sublicenseable, revocable license to use the Omniport marks solely to promote the app, provided that developer may not imply that the maintainers created, support, or endorse the app in any way pertaining to *Other platforms and users*.

This license is subject to these terms, applicable law, and the Omniport branding guide. All goodwill derived from the developer's use of the Omniport marks will inure to the sole benefit of the platform.

The developer also agrees not to contest or aid in contesting IMG's rights in, or the validity of, the Omniport marks.

Developer trademark license

The developer must grant to the maintainers a non- exclusive, non-transferable license to use the app marks and descriptive materials that developer publishes about the app or the developer's use of the platform. This license is limited to the maintainers promoting their products or services and acknowledging or promoting developer's use of the platform.

The license is subject to applicable law and any trademark usage guidelines that the developer provides to the maintainers, except to the extent the trademark usage guidelines require further permission for the uses described above or conflict with these terms.

Reservation of rights

Omniport reserves all right, title and interest in the marks pertaining to the platform. Equivalently the developer reserves all right, title and interest pertaining to the app.

Except as explicitly set forth herein, the terms do not grant anyone any right in another entity's marks or other intellectual property.

7.2.7 Confidentiality

The maintainers and the developer may disclose confidential information to each other. The receiving party may use the disclosing party's confidential information only to exercise its rights and perform its obligations under these terms.

The receiving party must use a reasonable degree of care to protect confidential information. The receiving party will not disclose confidential information to any third party except to its employees, agents, or third party contractors who need to know it and if they are bound by terms at least as restrictive as those in these terms.

Confidentiality obligations do not apply to the extent the information

- was known to the receiving party without restriction before receipt from the disclosing party
- is publicly available through no fault of the receiving party
- is rightfully received by the receiving party from a third party without a duty of confidentiality
- is independently developed by the receiving party without access to Confidential Information.
- A party may disclose confidential information to the extent it is compelled to do so by law if it provides reasonable prior notice to the other party, unless a court orders that the other party not be given notice. Upon written request, the receiving party will promptly return all confidential information and copies to the receiving party, or certify in writing that it has destroyed all such materials. Breach of this section could cause the disclosing party irreparable harm, and the disclosing party may seek immediate equitable relief, in addition to other rights and remedies it may have.

7.2.8 Termination

These terms will remain in force unless terminated as stipulated in the subsections below.

Termination by IMG

IMG may terminate these terms or suspend developer's access to all or any part of the platform

- if the developer is in material breach of these terms and fails to cure that breach within 30 days after receipt of written notice.
- if IMG is required to do so by the law of the land or by ethical and moral principles.
- if ING ceases to offer any products or services covered by these terms.
- if IMG determines or has reason to believe the developer or the app may cause harm or loss to the platform or to any of the platform's users, or the developer or the app is or will be a threat to to the platform or to any of the platform's users.
- for any other reason with 30 days prior written notice to the developer.
- in order to assess or address any imminent or potential security threat.

Termination by the developer

The developer may terminate these yerms at any time by ceasing all use of the platform (including use by the apps) and by either

- deleting the app if the developer is the sole developer.
- leaving the team if there are others on the app team.
- Leaving the app running without any active member left on the team does not constitute termination on part of the developer because the app will still be active and will be under the responsibility of the developer.

Effect of termination

If these terms are terminated

- the rights granted by IMG to the developer will cease immediately.
- the developer will cease all use, operation, support, promotion, and distribution of the the app and the platform.
- the developer may lose all access to any content, material or information that the developer has provided to IMG regarding the app or the platform.

The following terms will survive the termination of the agreement

- *Prohibited actions*
- *Platform usage*
- *User data*
- *Intellectual property* - *Definitions*

7.2.9 Warranties

The developer represents and warrants that

- all information that the developer provides to the maintainers is true, accurate and complete.
- the developer has the full right, power and authority to make, distribute and operate the app, use the platform and to enter into these terms.
- the developer, the app, its use and its use of the platform will not violate the intellectual property rights, or other rights of others, or violate any laws.

7.2.10 Indemnity

The developer will indemnify, defend and hold Dropbox and its affiliates harmless from all costs and expenses arising from any third party claim relating to any breach or omission on the developer's part in upholding these terms.

7.2.11 Disclaimer

THE OMNIPORT SERVICE, PLATFORM AND SOFTWARE ARE PROVIDED "AS IS", AT YOUR OWN RISK, WITHOUT EXPRESS OR IMPLIED WARRANTY OR CONDITION OF ANY KIND. IMG AND THE LOCAL INSTANCE MAINTAINERS DISCLAIM ANY WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT.

7.2.12 Limitation of liability

TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL OMNIPORT, IMG, THE LOCAL INSTANCE MAINTAINERS, THEIR AFFILIATES, OR AGENTS BE LIABLE FOR

ANY INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, EXEMPLARY OR CONSEQUENTIAL (INCLUDING LOSS OF DATA, USE, BUSINESS OR PROFITS) DAMAGES, REGARDLESS OF LEGAL THEORY. LOSS OF DATA, USE, BUSINESS OR PROFITS (IN EACH CASE WHETHER DIRECT OR INDIRECT) EVEN IF IMG AND THE LOCAL INSTANCE MAINTAINERS KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY OF SUCH DAMAGES. TO THE FULLEST EXTENT PERMITTED BY LAW, THE MAXIMUM LIABILITY OF AGGREGATE LIABILITY OF ALL AFOREMENTIONED PARTIES WILL NOT EXCEED 0 (ZERO, ZILCH, NADA, NIL) IN THE CURRENCY OF YOUR CHOICE.

7.2.13 Modification

IMG may revise these terms from time to time. If, in the maintainers' sole discretion, a revision is material, they will notify the developer by emailing the email address associated with the developer's account. In the event of a material revision, if developer does not agree to the revised terms, it may terminate the terms within 30 days of receiving notice of the revision. If a revision is not material, the maintainers will post the revised terms on its website, and the developer is responsible for checking these postings regularly. By continuing to access or use the platform after revisions become effective, developer agrees to be bound by the revised terms.

7.2.14 Disputes

None of the parties on the side on IMG including, but not limited to, affiliates, agents and local instance maintainers will ever indulge a dispute unless explicitly expressing an intention to do so which has about the same chance as a snowball in hell given that IMG is a student organisation running an awesome non-profit project to make every college on a planet a technology-enhanced place.

7.2.15 Miscellaneous

These developer terms constitute the entire and exclusive agreement between the developer and IMG with respect to the platform, and supersede and replace any prior or contemporary agreements, terms, and conditions applicable to the platform. These terms do not create third-party beneficiary rights. IMG's failure to ensure a provision is not a waiver of its right to do so later.

IMG and the developer are not partners, affiliates or agents but the relationship is deemed to be that of independent contractors.

Notices to IMG are to be sent via email, courier or mail and are deemed given when received. Notices to the developer are to be sent via email, courier or mail are and deemed given when sent.

IMG can be reached at

```
INFORMATION MANAGEMENT GROUP,
INSTITUTE COMPUTER CENTRE,
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE,
ROORKEE - 247667, HARIDWAR DISTRICT,
```

(continues on next page)

(continued from previous page)

```
UTTARAKHAND, INDIA (IN)
ATTN.: CHIEF COORDINATORS
```

7.2.16 Definitions

In addition to the definitions in the *Preamble* of the terms, the following definitions are to be understood in context of the terms expressed above.

Terms	Definition
user	Any customer or user of the platform and the app
marks	All trademarks, service marks, logos, icons, trade names or stylisations used to identify the party, its products or services
policies	All policies and requirements set forth on the Omniport website or portal or documentation
user data	Any data that users of the platform upload to or create on it
confidential information	If referring to the developer, the app confidential information and if referring to IMG, the Omniport confidential information
Omniport confidential information	Any code, inventions, know-how, user data, or business, technical or financial information that Omniport discloses to developers
app confidential information	Any information that the developer discloses to Omniport that a reasonable person would consider confidential under the circumstances

7.3 Brand usage guide

This document guides on the use of the logos and brand imagery of both Omniport and the Information Management Group.

All branding assets are available on GitHub. Feel free to use them when referring to Omniport or to IMG. Your use of these branding assets is subject to a few basic rules.

Read on to understand what these rules are.

7.3.1 Omniport

Typography

The font used in the logotype is Raleway.

Textual use

In all texts, the name must be typed as Omniport. If using the logotype, the name must be stylised as omni:port.

Logo usage

Note: All `em` units are on a `14px` baseline.

Fig. 1: Omniport logo at `3.5em`

Fig. 2: Omniport wordmark at `3.5em`

Clear spacing

To maintain logo sovereignty, we require the logo to have a minimum spacing of `1em` on all sides at height of `3.5em` around it.

Size

To maintain logo legibility the logo should never be scaled lower than `1.75em`. This may vary but you have to use your sensibilities.

Proportions

The logo must always be scaled so as to maintain the proportions and aspect ratio. That's usually achieved by using the `Shift` key.

Background

The logo must always be used on a background where it can stand out. Extremely detailed and colourful backgrounds are not recommended. Only in the cases where your background has a palette similar to the logo, you are to use our grayscale variant.

Alterations

You are not to alter the logo in terms of color, shape, shadows, borders, typeface or tilt. Our designers put a lot of love into it. Respect that.

7.3.2 Information Management Group

Typography

The font used in the logotype is Bauhaus.

Textual use

In all texts, the name must be typed as Information Management Group or IMG. If using the logotype, the name must be stylised as information management group.

Logo usage

Note: All `em` units are on a `16px` baseline.

Fig. 3: Information Management Group logo at `3em`

Fig. 4: Information Management Group wordmark at `3em`

Clear spacing

To maintain logo sovereignty, we require the logo to have a minimum spacing of `1em` on all sides at height of `3em` around it.

Size

To maintain logo legibility the logo should never be scaled lower than `1.5em`.This may vary but you have to use your sensibilities.

Proportions

The logo must always be scaled so as to maintain the proportions and aspect ratio. That's usually achieved by using the `Shift` key.

Background

The logo must always be used on a background where it can stand out. Extremely detailed and colourful backgrounds are not recommended. Only in the cases where your background has a palette similar to the logo, you are to use our grayscale variant.

Alterations

You are not to alter the logo in terms of color, shape, shadows, borders, typeface or tilt. Our designers put a lot of love into it. Respect that.

7.4 Open-source

Omniport is a product of, runs on, and therefore is, open-source. All aspects of Omniport, *including this documentation for some reason*, are licensed under the GNU licenses.

Omniport FOSS

7.4.1 GNU General Public License version 3

Fig. 5: All code is licensed under GNU GPLv3.

All code pertaining to Omniport, including but not limited to its core, its services and apps ecosystem and snippets thereof is licensed under the GNU General Public License or GPL. The license can be read in its entirety on the GPL page on the GNU website.

7.4.2 GNU Free Documentation License

Fig. 6: All documentation is licensed under GNU FDL.

All documentation pertaining to Omniport, including but not limited to this manual, excerpts thereof and other technical articles, is licensed under the GNU Free Documentation License or FDL. The license can be read in its entirety on the FDL page on the GNU website.

7.4.3 Other licenses

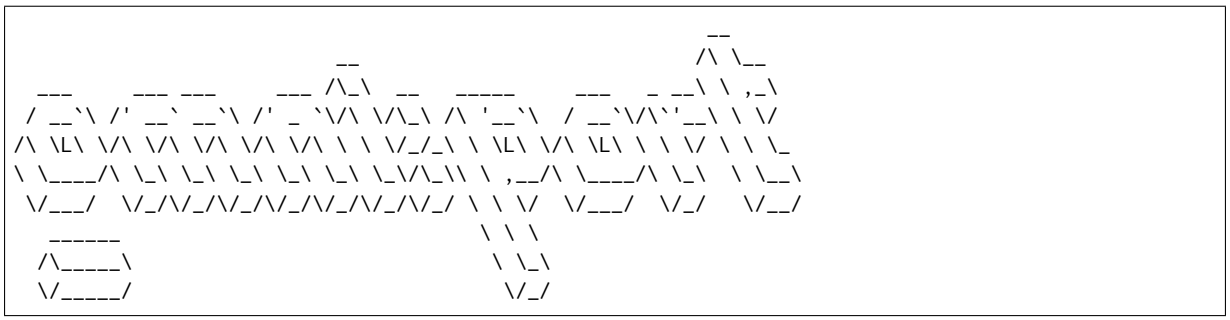
We don't really understand how license cascading works but we sure do hope that Omniport does not violate the licenses of any of its dependencies. All dependencies of Omniport retain their individual licenses and Omniport does not in any way overrule the licensing of any of its dependencies, or dependencies thereof.

If you get how licensing and the cascading of licensing works, feel free to get in touch. If Omniport is in violation of any terms of its dependencies' licenses, please bring this to our notice. We'll be happy to resolve it.

CHAPTER

EIGHT

CREDITS



Omniport is a team-effort. The sheer size of the project means that no one person is capable of creating, maintaining or upgrading the project. That said, some people deserve credit for their exceptional contributions towards the project. These are mentioned below.

8.1 Docker

The Dockerised distribution was made possible by Dhruv Bhanushali (dhruvkb) with intermittent help of Pranjal Tale (pranjaltale16).

8.2 Backend core

The backend was developed and Dockerised by Dhruv Bhanushali (dhruvkb) with the idea of swappable models incepted by Rohith Asrk (rohithasrk).

8.3 Frontend core

The frontend core was developed by Mohit Virli (mohitvirli), and then prepped for deployment by Praduman Goyal (pradumangoyal). It was Dockerised by Dhruv Bhanushali (dhruvkb).

8.4 Documentation

This documentation was written by Praduman Goyal (pradumangoyal) and Dhruv Bhanushali (dhruvkb), the latter being the editor as well.

8.5 Services

The full set of services for Omniport were developed by Dhruv Bhanushali (dhruvkb) on the backend and Praduman Goyal (pradumangoyal) on the frontend.

8.6 Apps

The entire team of the Information Management Group worked on developing apps for Omniport. They can be found on GitHub (IMGIITRoorkee) and on our website.

8.7 Orchestra

Like we said earlier, the orchestra deserves a fair share of credit.

Fig. 1: All hail the greatest campus development group of all time!

CHAPTER

NINE

MUSIC ♪

If Omniport was a symphony, which it is if you look closely enough, the setup would be this.

Composers

Dhruv Bhanushali
Praduman Goyal

Orchestra

Information Management Group,
IIT Roorkee