Read the Docs Template Documentation

Release 1.0

Read the Docs

Jul 14, 2023

CONTENTS

Purpie is an open-source platform that brings together many different solutions, to create a space focused on video-oriented communication and content sharing. If you are interested in hearing more, you can continue with the page below;

- What is Purpie?

If you'd like to give it a go yourself, you can try our Quick Setup guide in the page below, of if you are a developer, you can use Manual Setup.

- Quick Start
- Manual Setup

If you are looking to contribute to this project, you may start with a look at our Contribution Guide and License.

- Contribution Guide
- License

If this document does not answer your questions, you can check out FAQs, or join our community.

- License
- Community

If you'd like to learn more about us, you can visit our website.

- doganbros.com

CHAPTER ONE

WHAT IS PURPIE?

Purpie is an open source social media solution focusing on video oriented communication and content sharing. Purpie orchestrates various media services and solutions for online meetings, live streamings, on demand video delivery and many more. With an agile and hierarchical way of user interaction and content sharing possibilities, Purpie proposes a robust and flexible solution for various video communication and distribution needs.

Purpie is;

- **Unique** social media service prioritizing video communication in the first place.
- **Resourceful** as an orchestrator to integrate and coordinate multiple media solutions and services.
- **Open** as an open source project to be customizable for various business cases.
- **Flexible** with its Zones, Channels and Profiles to cover possible use cases.
- **Elegant** with its look and feel.

CHAPTER TWO

POSSIBLE USE CASES

Purpie can be used as is or customised for any kind of video based social media needs.

Here are some possible use cases:

- Social media platform;
  - For specific business domains such as e-learning, benchmarking etc…
  - For local authorities such as municipalities.
  - For societies, clubs, religious groups etc…
  - To be used by a company internally.
  - So on…
- Gaming and entertainment streams.
- Monetizing video content of a video production company.
- Online auctions platform.
- Business matchmaking solution.
- Event management company to organize online events.
- Online health advisory service.
- Online discussion and debating platform.

CHAPTER

THREE

DESIGN PHILOSOPHY

Purpie has hierarchical content management architecture which lies behind Zones and Channels. Like in any membership based solutions, users have their own profile. Adding to this, in Purpie, any user can create multiple Zones and multiple channels in each Zone.

3.1 What is a Zone?

A zone is the base for creating channels, which means a channel would always belong to a Zone. Conceptually, Channels having similar contents would exist in the same Zone. This similarity can be based on content type, content source or any other categorization. Following key points should be in consideration while interacting with Zones:

- Zones can be private or public.
- Users can not share content directly from a Zone.
- Private Zones can only be accessed by the Zone members.
- Public Zones are open to other Purpie users.
- Users can join existing public Zones in Purpie and join Private Zones only through invites.
- A user can create a Channel inside a Zone if they are the Zone owner or if given a role that has permissions to create Channels by the Zone owner.

3.2 What is a Channel?

Users can create and share content from a Channel, which is always under a Zone. So, the Channel is where you publish your content and let others view it.

Following key points should be in consideration while interacting with Channels:

- Channels can be private or public.
- Users can share content from a Channel (also from their Profile but we will come to that later).
- Private Channels can only be accessed by the Channel followers.
- Public Channels are open to other Purpie users.
- Users can follow existing public Channels and join private Channels only through invites.
- Users can create content inside a Channel only if they are the Channel owner or if given a role that has permissions to create content by the Channel owner.

3.3 What is a Profile?

By default, each user has a profile in Purpie, like it is such in any site having membership functionality Users can add others into their Contacts, which are the only ones who can see the content they publish through their profile. Profiles are very handy if a user does not want to create a Zone or Channel but just wants to share a video content with the people they may know.

Following key points should be in consideration while interacting with Profiles:

- Users can share content from their Profiles only with their Contacts.
- Each user can have a single Profile.
- Users can request to add any other Purpie user to their Contacts.

CHAPTER

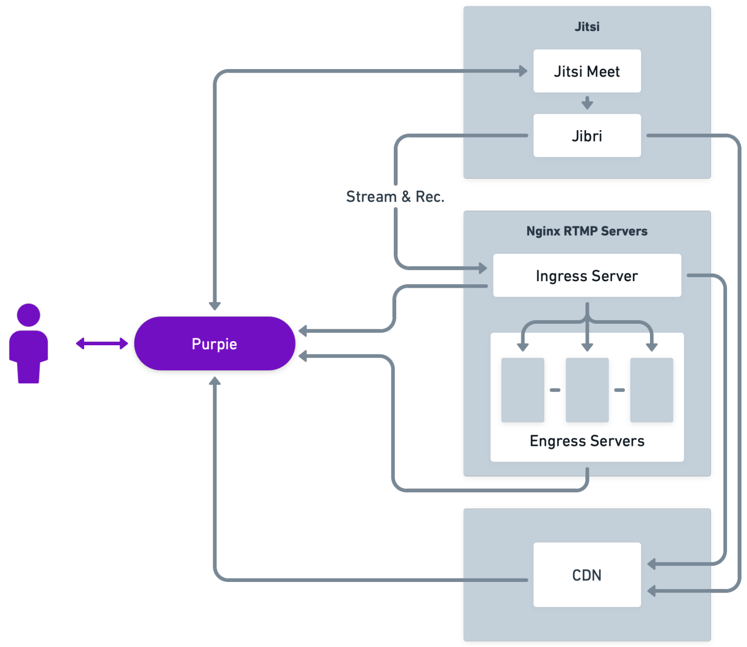
FOUR

HOW DOES IT WORK?

Purpie is an orchestrator which provides effective and robust integration with various media services. Purpie ties up following media solutions and services:

- Jitsi for:
- Online meetings
- Webinars (Large rooms)
- Streaming & recording meetings
- Nginx RTMP Servers for:
- Live and on-demand streaming
- Recording
- Cloud-storage for:
- On demand video delivery

4.1 Architecture



CHAPTER

FIVE

QUICK SETUP

Quick Setup guide focuses on an easily configured development environment. In this environment, Purpie, RTMP servers and Jitsi each run in their own containers. An FQDN (Fully Qualified Domain Name) will be required.

5.1 Install Docker & Docker Compose

Since all the components will be running in containers, you will need Docker and Docker Compose. You may install Docker by following the guide in this link.

You can follow this link to install Docker Compose.

5.2 Download Purpie from GitHub Repository

```
sudo apt-get install git
git clone https://github.com/doganbros/purpie-docker.git
cd purpie-docker
```

5.3 Configure Purpie

Make a copy of the `.env.example` named `.env`. This `.env` contains environment variables for Jitsi as well as Purpie.

5.4 Generate SSL Certificates

SSL certificates are required for jitsi, purpie and RTMP servers to work correctly. If you have ssl certificates already, copy them to `./docker/ssl` directory. The `fullchain` certificate should be named as `cert.crt` where as the `private` certificate should be named as `cert.key`.

If you do not have SSL certicates already, run the script below to obtain one.

```
./docker/ssl-gen.sh
```

5.5 Generate Strong Passwords

To generate secured and strong passwords in the security section options of `.env` file for Purpie and Jitsi, run the script below.

```
./gen-passwords.sh
```

5.6 Start Purpie

```
docker-compose up -d
```

5.7 Rebuilding the Image

Sometimes, especially after an update, you might want to rebuild your image. Doing so would let you build your frontend again.

First, bring your docker containers down;

```
docker-compose down
```

Then start them up again with the following command.

```
docker-compose up --build -d
```

CHAPTER

SIX

SETUP FROM SCRATCH

Manual Setup installs each component individually to maximize control and configurability. All components can be installed into a single machine, provided the machine has an FQDN, and is strong enough to meet the requirements. Please make sure the certificate for the domain includes not only the domain itself, but all its belonging subdomains as well. You may ensure it by using a wildcard like "*.mydomain.com" when procuring the certificate.

6.1 Setting Up Purpie

6.1.1 Requirements

- Node.js (>= 10.13.0, except for v13) (Windows Build Tools for Windows systems)
- Yarn
- NestCli
- Postgres

6.1.2 Getting Started

```
git clone https://github.com/doganbros/purpie # Clone Repository
cd purpie
```

Install dependencies with

```
yarn install
```

6.1.3 Set Environment Variables Into .env File

You may configure the environment by making a copy of the boilerplate provided. The functionality of each environment variable is documented within the `.env.example` file.

```
cp .env.example .env # Then make changes to the boilerplate provided
```

Table 1: You can find a brief coverage of most prominent variables below.

Variable	Example	Explanation
NODE_ENV	development	The environment your application runs in.
SERVER_PORT	8000	The port the backend server is running on.
PORT	3000	The port the frontend serves from.
HOST	purpie.io	Host address for backend and frontend.
PURPIE_API_KEY	YOUR_API_KEY	Key for access API directly with PURPIE_API_SECRET
PURPIE_API_SECRET	YOUR_API_SECRET	Secret for access API directly with PURPIE_API_KEY
RTMP_INGRESS_URL	ingress.yourrtmp.com	The RTMP server Purpie pushes your streams to.
RTMP_EGRESS_URL	egress.yourrtmp.com	The RTMP server Purpie pulls steams from to display.

6.1.4 Creating Postgres Database

Please follow the steps below to get a development Postgres server running. The easiest way to use docker. If you have running Postgres database server you can skip these steps and simply create an Purpie database.

- Make sure you have docker installed on your computer. If you do not have docker already on your computer, Go to this link, choose your platform and click download. Follow the simple steps to get docker installed on your computer.
- Open your terminal (command prompt or preferably powershell on windows).
- Enter the command `docker run --name purpie-dev -e POSTGRES_PASSWORD=$YOUR_POSTGRES_PASSWORD -p 5432:5432 -d postgres`. Postgres docker image will be downloaded and Postgres Docker container with the name purpie-dev will up and serve from port 5432 after this command.
- Run `docker exec -it purpie-dev psql -U postgres` to connect your Postgres database.
- Inside the docker container, run `CREATE DATABASE 'purpie';` to create your Purpie database.
- Run `\q` to quit from Psql and Docker container.
- Remember to update `DB_USER`, `DB_PASSWORD`, `DB_DATABASE` and `DB_HOST` .env variables to your database user name, database password database name, and database host respectively.

6.2 Running Purpie

```
yarn server:start:dev # Runs backend side in dev mode
yarn start:server # Runs backend in production
yarn start:web # Runs frontend side
```

You may refer to this sample NGINX config to set up your Purpie web server.

6.3 Setting up Jitsi

6.3.1 Installing Jitsi with JWT support

Purpie mmakes use of Jitsi for video streaming and live meeting purposes. It requires a Jitsi installation with JWT token support. You can refer to this document by us to install Jitsi. Remember to update the `.env` variable by setting `JWT_APP_ID`'s value to `YOUR_APP_ID`, `JITSI_SECRET` to `YOUR_SECRET` and `JITSI_DOMAIN` to the domain where you set up jitsi.

Installing Purpie Jitsi Module

This module is a React Module that applies on top of Jitsi to provide visual and functional differentiations. To avoid issues regarding version mismatch, start with the customized Jitsi repo below;

```
git clone https://github.com/doganbros/purpie-jitsi-meet.git
cd purpie-jitsi-meet
```

The module is included in the package.json file within the repository. Therefore, you are able to simply follow the promtps as shown;

```
npm install
export WEBPACK_DEV_SERVER_PROXY_TARGET=https://yourdomain.com
make
```

At this point, your customized Jitsi is built and ready to serve. Make sure to edit the NGINX configuration accordingly to make use of your new directory.

6.3.2 Integrating Jitsi with Purpie

- This .lua module for Jitsi Meet manages the integration between purpie and Jitsi. It enables Jitsi to send reports to Purpie. To enable it, follow the instructions below.

```
cp ./external/prosody-modules/mod_purpie.lua /usr/share/jitsi-meet/prosody-plugins
```

Configuring Prosody

- Edit your prosody configuration at `/etc/prosody/conf.d/your.domain.com.cfg.lua`, and add the following lines.

```
purpieApiKey = "yourAPIkey";
purpieApiSecret =  "yourAPIsecret";
purpieAPIBaseUrl = "https://your.baseAPI.url";
```

- In the same file, add purpie to the list of enabled modules of the conference component like below;

```
Component "conference.meet.doganbros.com" "muc"
    restrict_room_creation = true
    storage = "memory"
    modules_enabled = {
        "muc_meeting_id";
```

(continues on next page)

(continued from previous page)

```
        "muc_domain_mapper";
        "polls";
        "purpie";
        "token_verification";
    }
    admins = { "focus@auth.meet.doganbros.com" }
    muc_room_locking = false
    muc_room_default_public_jids = true
```

6.3.3 Installing Jibri

To install Jibri you can follow this tutorial.

Customizing the Finalize Script

- Find the following in your `/etc/jitsi/jibri/jibri.conf` file, and replace path with `/srv/finalize/purpie-finalize.sh`

```
jibri {
    recording {
        finalize-script = <path>
    }
}
```

- Copy the included finalize.sh file to your finalize script directory. This .sh file runs upon the completion of recording, and both uploads the recorded file to an S3 bucket, and manages Jibri's integration with Purpie.

```
cp {./external/jibri/purpie-finalize.conf, purpie-finalize.sh} /srv/finalize
```

- Edit `/srv/finalize/purpie-finalize.conf` accordingly, and restart Jibri.

6.4 Setting up the Ingress RTMP servers

For Purpie, you will need at least one ingress server, and as many additional egress servers as you need for ease of autoscalability. To install NGINX rtmp server, you can follow the guide in this page. Remember to update the `RTMP_INGRESS_URL` and `RTMP_EGRESS_URL` .env variables to your ingress and egress server url respectively.

6.4.1 Integrating the RTMP server with Purpie

- Head into the RTMP server, and use the following to copy the required scripts. `sample-nginx.conf` contains a working example for an RTMP server, who calls upon `purpie.sh` in its workflow. `purpie.sh` is the script that maintains the integration between the RTMP server and Purpie. `purpie-sh.conf` contains customizable variables for the script.

```
mkdir /home/purpie
cp {./external/rtmp/purpie.sh, ./external/rtmp/purpie-sh.conf} /home/purpie
cp ./external/rtmp/sample-nginx.conf /etc/nginx
```

- Navigate to `/home/purpie` and edit the contents of `purpie-sh.conf` accordingly. Then restart nginx.

```
systemctl restart nginx
```

CHAPTER

SEVEN

TESTING PURPIE

After your setup of Purpie is complete, here are a few steps you can test on your installation.

7.1 Testing Meetings in a Development Environment

We provide https://meet.purpie.io as a server that handles all meetings for development. Currently every developer must connect to this server in order to test how meeting is created, video is streamed etc. Since there is only one server, there is a need to identify each development environment while making requests to server in order to create a meeting. To set up your local environment to support this flow, follow the steps below.

- Set up your jitsi domain env variable to point to the jitsi server (*JITSI_DOMAIN=https://meet.purpie.io*).
- Set up a local tunnel to your localhost so that the server can make the request to you. Use the command *npx localtunnel –port 8000 –subdomain yourpreferredsubdomain* where *yourpreferredsubdomain* would be a unique address that will be used to identify you local server later.
- Add the environment variable *MEETING_HOST=yourpreferredsubdomain.loca.lt* to your *.env* file. This is the endpoint the jitsi server will be making requests to. Note that you shouldn't include (https://)
- Make sure you have the correct *JITSI_SECRET*, *OCTOPUS_API_KEY*, *OCTOPUS_API_SECRET* and *JWT_APP_ID* env variables set already. If you don't have these already, contact the octopus channel for that.
- Add the environment variable *REACT_APP_STREAMING_URL=https://egress.purpie.io:1980/hls* so that streaming works correctly.
- You are all set! You can now create, record and stream a meeting using the https://meet.purpie.io server.

7.2 Testing Recording and Streaming in a Development Environment

To stream meetings using the https://meet.purpie.io server, follow the instructions below:

- Create a meeting
- Click on the three dots, and on start live stream
- A window will appear, enter *rtmp://ingress.purpie.io/live/<meeting-slug>?uid=1* as a live stream key. (Replace *<meeting-slug>* with the real meeting slug).
- The stream should start in few minutes

CHAPTER EIGHT

ROAD MAP

8.1 2023, Q1

- Alpha Release
- Superuser administration
- Roles and permissions
- User management

8.2 2023, Q2

- Statistics
- Webinar App
- One-on-one messaging
- Voice chat

8.3 2023, Q3

- Open API
- Streaming Studio App
- Purpie Mobile (Alpha Release)
- Multi-language Support

8.4 2023, Q4

- Beta Release
- Purpie Mobile (Beta Release)

CHAPTER

NINE

DEVELOPER GUIDE

This document focuses frontend and backend development of Purpie project and gives information about Purpie Tech Stack.

9.1 Frontend

9.1.1 Requirements

- Node.js. (>= 10.13.0, except for v13) (Windows Build Tools for Windows systems)
- Yarn.

9.1.2 Clone Repository

```
git clone https://github.com/doganbros/purpie
cd purpie
```

9.1.3 Install dependencies

```
yarn install
```

9.1.4 Set environment variables into .env file:

```
cp .env.example .env
```

Then make changes to the boilerplate provided

9.1.5 Setting web server and routing

If Purpie is installed on your local computer, you will need to add the following line to your hosts file. The hosts file for Unix based system including MacOs is `/etc/hosts` where as on Windows, it is `C:\windows\system32\drivers\etc\hosts`.

```
127.0.0.1      purpie.localhost
```

9.1.6 Run project

```
yarn start
```

9.1.7 Try Purpie

- Visit http://purpie.localhost:3000 (3000 is the default port) and create your super admin user.

9.2 Backend

9.2.1 Requirements

- Node.js. (>= 10.13.0, except for v13) (Windows Build Tools for Windows systems)
- Yarn.
- NestCli.
- Postgress.

First steps are same with fronted setup.

9.2.2 Create Postgres database

Please follow the steps below to get a development Postgres server running. The easiest way to use docker. If you have running Postgres database server you can skip these steps and simply create an Purpie database.

- Make sure you have docker installed on your computer. If you do not have docker already on your computer, Go to https://www.docker.com/get-started, choose your platform and click download. Follow the simple steps to get docker installed on your computer.
- Open your terminal (command prompt or preferably powershell on windows).
- Enter the command

```
docker run --name purpie-postgres-dev -e POSTGRES_PASSWORD=YOUR_DB_PASSWORD -p 5432:5432␣
↪-d postgres
```

- Postgres docker image will be downloaded and Postgres Docker container with the name purpie-postgres-dev will up and serve from port 5432 after this command.
- To connect your Postgres database.

```
docker exec -it purpie-postgres-dev psql -U postgres
```

- To create your Purpie database.

```
CREATE DATABASE purpie;
```

- Update your `.env` file with `YOUR_DB_PASSWORD` .
- Run `\q` to quit from Psql and Docker container.

9.2.3 Run project

To run backend server in production

```
yarn start:server
```

To run backend server in development

```
yarn start:server:dev
```

9.2.4 API Testing

Visit http://purpie.localhost:8000/swagger/ to try out some backend APIs.

9.3 Software Spec

9.3.1 Frontend Features

- Typescript (Strict Mode)
- ESNext
- Airbnb Coding Style Guide
- Prettier
- eslint
- yarn is used for package management
- React is the main framework (with hooks)
- React Router is used for client side routing
- Redux is used for managing application state
- Grommet is the main css framework

9.3.2 Backend Features

- Typescript (Strict Mode)
- Airbnb Coding Style Guide
- ESNext
- CORS enabled
- yarn for package management
- Handlebars for rendering email templates
- NestJS is the main framework
- Postgresql is the database used
- TypeORM is the database ORM used
- Class Validator is used to validate request body.
- helmet is used to set http headers correctly.
- dotenv is used to load .env variables
- compression
- eslint
- morgan
- Swagger
- Monitoring with pm2

Open Source Technologies used

- Jitsi

9.3.3 Requirements

- Node.js (>= 10.13.0, except for v13)
- Yarn

9.3.4 Glossary

- represents client side
- represents server side

Architecture

This is a single page web application, that is it handles routing at the client-side without the need to refresh the entire page. All http requests are done using `Asynchronous Javascript and XML (AJAX)`. The data exchange format used between this app and the server is `JSON`.

Programming Languages

HTML

`HTML` is rarely used in this app. It is primarily used to setup the main index file that is responsible for loading the main javasript of the app. It loads the css and display the initial title of the app.

TypeScript

This app uses no `Javascript` (Although it compiles to javascript). `Typescript` is the main programming language used on the server and for building the user interface.

Frameworks and Libraries

NestJS

Nestjs is a progressive Node.js framework for building efficient, reliable and scalable server-side applications. It works well with typescript and follows the SOLID principle

TypeORM

TypeORM is a NodeJS database ORM that supports the latest JavaScript features and provide additional features that helps in developing any kind of application that uses databases - from small applications with a few tables to large scale enterprise applications with multiple databases. It works well with typescript.

OpenAPI (Swagger)

The OpenAPI specification is a language-agnostic definition format used to describe RESTful APIs. Nest provides a dedicated module which allows generating such a specification by leveraging decorators.

Handlebars

Handlebars is used to render email templates before they are sent to clients.

SendGrid

SendGrid is the main service used for sending emails.

Class Validator

Allows use of decorator and non-decorator based validation. Internally uses validator.js to perform validation.

Axios

`Axios` is a promise based HTTP client used in this app. All AJAX requests are handled with `axios`. Their interceptors really help to avoid redundancy in most part of the app.

SCSS

This app uses no `CSS` (Although it compiles to css in the long run). `SCSS` is rearely used in this app. It is used to style a large portion of the app. `SCSS Modules` is recommended if `SCSS` is used. `node-sass` is the library responsible for compiling the app's `scss` to `css`

React

This app uses the latest version of `React` Framework (Library) in collaboration with `Typescript`. `JavaScript XML` is used to develop all the components. **Only Functional Components** are allowed for writing all React Components.

Grommet

Grommet is a `React styled-component` library that helps in building responsive and accessible mobile-first projects for the web. Since this framework provides lots of styled-components, writing `scss` is often not required at all. Developers are required to use most of the features of Grommet without writing lots of `scss` .

React Router DOM

`React Router` (Its DOM binding `React Router DOM`) is the library used to for handling all the client side routing of this app. **Note** that instead of using the library's main `Link` and `NavLink` components, AnchorLink and NavLink are used respectively. This is to make it compatible with the Grommet library. To navigate to other paths of the app inside a component, the `useHistory` hook is used. Routing done in other parts of the app app (especially in a Redux action) uses the `appHistory` helper function insead.

Redux

`Redux` is a predictable state Container for Javascript (Typescript) Apps. This is the main state management library used in the app. Mostly states that are shared across multiple components of the app use redux. Also all network-related states are handled here. `react-redux` is the library that helps in binding redux to react. `redux-thunk` provides the redux middleware that helps the app to deal with asynchronous dispatches in redux actions.

React-i18next

`React-i18next` is a powerful internationalization framework for React / React Native which is based on i18next. Our goal is to support as many languages as possible with the help of this framework and community.

Development Dependencies

Eslint

`Eslint` statically analyzes the application code to quickly find problems. It helps in maintaining the usage of Airbnb coding style guide and the similarity of code written by different develops at a time. Run `yarn analyze` or `npm analyze` to let eslint analyze and report all errors made. If you are using editors like vscode please install the eslint extension to help you in automatically detecting errors.

Prettier

`Prettier` is an opinionated code formatter that helps the app to format the code written to comform to the rules of eslint. Run `yarn format` or `npm format` to do a quick format of the entire app.

Jest

Jest is a delightful JavaScript Testing Framework with a focus on simplicity.

NestJS

While using nestjs at the server-side, One must follow these guidelines.

- NestJS pattern must be followed strictly. For example controllers should be used to handle only http requests, services must be used to generate data or communicate with the database, guards must be used for securing routes etc.
- controllers and providers should reside in controllers and services directories respectively.
- Implement global providers if they are needed only. This will help other developers know from which modules those services are imported from. Example authentication and exceptions would be needed in the entire application but zone service wouldn't.
- the `@IsAuthenticated()` decorator should be used to validate the current user's token. Also permissions could be passed in as paremeters if they are needed.
- Document the controllers written extensively (using decorators provided by Nestjs for OpenAPI). This helps other developers to make requests very easily without reading the source code.
- The built in NestJS exceptions must be used accross the entire application. The first paramter must be a message about the error. And the second parameter must be an error code. For example while generating an error for invalid bearer authentication token, the example below is used.

```
throw new UnauthorizedException(
      'You not authorized to use this route',
      'NOT_SIGNED_IN',
);
```

TypeORM

While using TypeORM at the server-side, One must follow these guidelines.

- The models designed must be relational. That means you must use `OneToOne`, `ManyToOne`, `OneToMany` or `ManyToMany` relation when it is necessary.
- When models, fields, column, etc. are added a migration script must be written in respect of that. This is because we are not using syncronization as it not good for production. **Note** that nestjs will run pending migrations when the application is booted automatically.

Guards In This Application and their usage

This section introduces the main guards used in this application

- #### AuthGuard

  The AuthGuard validates the current bearer token passed to the server when making requests. It sets the payload of the user to `req.user`. It also thows an `UnauthorizedException` exception when the token is invalid.

- #### UserZoneGuard

  The UserZoneGuard validates the current user's authorization to the zone that he/she is requesting. It sets the user zone to `req.userZone`. Other permissions can be passed in using the `SetMetadata` decorator. It also throws an `NotFoundException` exception when the user is not authorized.

Pipes in this application and their usage

This section introduces the main pipes used in this application

- #### ParseTokenPipe

  The ParseTokenPipe is used to parse a JWT. If it succeeds it passes the payload to the parameter. Otherwise it will throw an `UnauthorizedException`.

Decorators in this application and their usage

This section introduces the main decorators used in this application

- **IsAuthenticated**

  The IsAuthenticated decorator wraps over the AuthGuard to avoid writing lots of boilerplates while passing permissions to the AuthGuard.

- **UserZoneRole**

  The UserZoneRole decorator wraps over the UserZoneGuard to avoid writing lots of boilerplates while passing permissions to the it. It also extends the IsAuthenticated decorators so if you do not need to specify it while using it on a route.

- **UserChannelRole**

  The UserZoneRole decorator wraps over the UserChannelGuard to avoid writing lots of boilerplates while passing permissions to the it. It also extends the IsAuthenticated decorators so if you do not need to specify it while using it on a route.

- **CurrentUser**

  The CurrentUser decorator is helper to retrieve the current user's jwt payload

- **CurrentUserZone**

  The CurrentUserZone decorator is helper to retrieve the current user zone. Notice that it zoneId or userZoneId must be set as params in order to retrieve this.

- **CurrentUserChannel**

  The CurrentUserChannel decorator is helper to retrieve the current user channel. Notice that it channelId or userChannelId must be set as params in order to retrieve this.

Middlewares Used in this application

This section introduces the main middlewares used in this application.

- **PaginationMiddleware**

  The PaginationMiddleware parses all get requests' pagination query paramters. All get requests pass through this middleware. This means that, the pagination query parameters `req.query.limit` and `req.query.skip` are passed to controllers automatically (Global middleware for get requests). It can in turn be used in paginating records. When no values for limit and skip query parameters are passed by the user, limit is set to a default of 30 and skip is also set to a default of 0. Limit cannot be greater that 100. The type `PaginationQuery` can help in intellisense.

Authentication

This app interacts with a stateless http server. Authentication is realized by sending a JSON Web Token (By the way this is one of my favorite technologies) to the server. The steps for authenticating users are listed below.

1. When it is the first time the user is visiting the app or the returning user is not authenticated, React Router will redirect the user to the login page.
2. The User will either login or create a new account
3. The app sends the authentication information to the server
4. If the server successfully authenticates the user, a json web access token and its refresh token is created on the server and sent as an http only cookie to the client
5. By default the access token only lasts an hour. After this if the refresh token is still valid, the server will generate a new access and refresh tokens to the client
6. In subsequent requests, the app will send the access token stored in the cookies to the server to identify the user making the request.
7. If the token expires or becomes invalid the user will automatically be redirected to the login page. Thanks to the `axios` response interceptor.
8. If the user returning to the app is already authenticated react router will redirect the user to the main application page.

Authentication persistence through subdomains

Since this app allows users to create subdomains, it needs to persist authentication through the main domain and subdomains. This is one of the main reasons why cookies are been used. For cookies to persist authentication through domains and subdomains, the main domain parameter supplied while creating them must be valid. One of the rules for its validity is that it must have at least one dot. Due to this, localhost will not work. Read this article to learn more. Even though developers can still use localhost but if another subdomain is visited, authentication would be required again. Developers can therefore set a different domain other than localhost in `/etc/host` (or `C:\Windows\System32\Drivers\etc\hosts` for windows) file. The domain recommended is octopus.localhost. This is because it allows all subdomains to see the cookie as well. # Application Structure

```
├── README.md
├── SOFTWARE-SPEC.md
├── appspec.yml
├── package-lock.json
├── package.json
├── scripts
│   ├── after_install.sh
│   ├── before_install.sh
│   └── start.sh
```

(continues on next page)

(continued from previous page)

```
├── server
│   ├── README.md
│   ├── dist
│   ├── entities
│   │   ├── Channel.entity.ts
│   │   ├── Invitation.entity.ts
│   │   ├── Post.entity.ts
│   │   ├── User.entity.ts
│   │   ├── UserChannel.entity.ts
│   │   ├── UserChannelPermission.entity.ts
│   │   ├── UserZone.entity.ts
│   │   ├── UserZonePermission.entity.ts
│   │   ├── Zone.entity.ts
│   │   ├── base
│   │   ├── data
│   │   └── repositories
│   ├── helpers
│   │   ├── jwt.ts
│   │   └── utils.ts
│   ├── migrations
│   │   └── 1625561314952-InitialMigration.ts
│   ├── ormconfig.ts
│   ├── src
│   │   ├── app.module.ts
│   │   ├── auth
│   │   ├── mail
│   │   ├── main.ts
│   │   ├── typeorm-exception.filter.ts
│   │   ├── utils
│   │   ├── views
│   │   └── zone
│   ├── test
│   │   ├── app.e2e-spec.d.ts
│   │   ├── app.e2e-spec.js
│   │   ├── app.e2e-spec.js.map
│   │   ├── app.e2e-spec.ts
│   │   └── jest-e2e.json
│   ├── tsconfig.build.json
│   ├── tsconfig.json
│   ├── tsconfig.tsbuildinfo
│   └── types
│       ├── Post.ts
│       ├── PaginationQuery.ts
│       ├── UserPayloadRequest.ts
│       └── UserZoneRequest.ts
├── src
│   ├── App.tsx
│   ├── assets
│   │   ├── background.png
│   │   └── logo.png
│   ├── components
│   │   ├── layouts
```

(continues on next page)

(continued from previous page)

```
│   │   └── utils
│   ├── config
        ├──i18n
        │   └── [language].json
│   │   ├── app-config.ts
│   │   └── http.ts
│   ├── helpers
│   │   ├── history.ts
│   │   ├── utils.ts
│   │   └── validators.ts
│   ├── hooks
│   │   └── useTitle.ts
│   ├── index.tsx
│   ├── layers
│   │   ├── meeting
│   │   └── zone
│   ├── models
│   │   ├── form-submit-event.ts
│   │   └── response-error.ts
│   ├── pages
│   │   ├── Private
│   │   └── Public
│   ├── react-app-env.d.ts
│   ├── routes.ts
│   ├── scss
│   │   └── index.scss
│   └── store
│       ├── actions
│       ├── constants
│       ├── reducers
│       ├── services
│       ├── store.ts
│       └── types
└── tsconfig.json
```

- `README.md`

  This is the main readme file of the application

- `package-lock.json`

  This is automatically generated for any operations where npm modifies either the node_modules tree, or package.json. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.

- `package.json`

  Lists all the dependencies, author, version, etc of the app.

- `public`

  This is where the main index.html file that loads the react app lives.

- `server`

  This is where most backend work is done in this app.

  – `dist`

Typescript compiles to this directory.

- `entities`

  This is directory hosts all the typeorm model definitions. All typeorm entities must end with `.entity.ts`

  * `base`

    This directory hosts the typeorm models that will be inherited by other models. For example the `RecordEntity` defines most of the repeating fields in records such as `id`, `createdOn`, `updatedOn` etc.

  * `data`

    This directory hosts the default data used by some entities.

- `helpers`

  This directory hosts all the utilities functions of the application.

- `migrations`

  This directory hosts all the migration scripts used by typeorm. To create a new migration please use the script `yarn migration:create`. NestJS automatically runs all pending migrations when it is booted. While creating migrations, typeorm driver must be prefered to raw sql. This helps in migrating to other databases in the future.

- `types`

  This directory hosts all the utility typescript types used in the application.

- `test`

  This is the directory that hosts all end-to-end testing scripts.

- `src`

  This is the directory where most of the work is done. It hosts all the NestJS controllers, modules, services, pipes, guards, middlewares etc. Note that, scripts other than NestJS specific shouldn't be put here.

  * `app.module.ts`

    The root module of the application. All other modules are imported into this file.

  * `main.ts`

    The entry file of the application which uses the core function NestFactory to create a Nest application instance.

  * `mail`

    This directory hosts the module used for sending mails in this application. To send a mail, a view is created inside the views directory. The `MailModule` is imported into the current module and the `MailService` is injected into the current service. Using the `sendMailByView` method of the mail service emails can be sent using sendgrid.

  * `[module_name]`

    The src directory hosts all the nestjs modules in this application. To create a new module, a new directory with the same name is created. It is recommended that the nest cli is used to generate modules, controllers, services etc. The nest cli command `nest g module [module_name]` generates a new module. This creates a new directory inside the src folder and a new module named `[module_name].module.ts`. All directories created inside this must not be empty.

    · `decorators`

      All decorators for this module is created in this directory.

      · `dto`

        All dtos for this module is created in this directory. A DTO is an object that defines how the data will be sent over the network. This is especially useful in `POST` and `PUT` requests. The class validator decorators can also help in validating payload fields. All dtos must end with `.dto.ts`.

      · `pipes`

        All pipes for this module is created in this directory. Pipes are used to transform input data coming from `req.body`, `req.query` or `req.params` etc. All pipes must end with `.pipe.ts`

      · `guards`

        All guards for this module is created in this directory. Guards determine whether a given request will be handled by the route handler or not, depending on certain conditions (like permissions, roles, ACLs, etc.) present at run-time. All guards must end with `.guard.ts`

      · `interfaces`

        All interfaces for this module is created in this directory. **Note**: All interface must be declared using `class` but not the `interface` keyword. This is because Typescript removes all interfaces when it is compiling to Javascript. All interfaces must end with `.interface.ts`

      · `exceptions`

        All exceptions for this module is created in this directory. Nest comes with a built-in exceptions layer which is responsible for processing all unhandled exceptions across an application. When an exception is not handled by your application code, it is caught by this layer, which then automatically sends an appropriate user-friendly response. All exceptions must end with `.exception.ts`

      · `controllers`

      If multiple controllers are used in this module, it is recommended to put them in the controllers directory. Otherwise there is no need to create this directory for them. All controllers must end with `.controller.ts`

      · `controllers`

      If multiple services are used in this module, it is recommended to put them in the services directory. Otherwise there is no need to create this directory for them. All services must end with `.service.ts`

      `[module_name].module.ts`

      This is the file that all providers, controllers etc of this module are imported into. This is then imported into the `app.module.ts`

  – `ormconfig.ts`

    This is the file that contains all the configuration of the application's database. It is used by typeorm to create migrations and connect to the database.

  – `tsconfig.json`

    This is the file that contains the typescript configuration for the server. The configuration used in this app is in strict mode.

- `src`

  This is where most frontend work is done in this app.

  – `App.tsx`

    This is the main component that loads the app routes and run initial scripts (eg. retrieving current user)

- `assets`

  This directory contains all the static assests used in the app
- `components`

  This directory contains most of the helper components used in the app
- `config`

  This directory contains all the configuration files of the app
- `helpers`

  This directory contains all the utilities functions of the app
- `hooks`

  This directory contains all the general react hooks used in the app
- `index.tsx`

  This is the main script and starting point of the app responsible for bootstrapping the react app
- `layers`

  This is the directory where layers (modals) used in the app are stored
- `models`

  This is the directory where typescript types used accross the entire app are declared.
- `pages`

  This is the directory where pages served in the browser are stored

  * `Private`

    All Privates Pages are stored in this directory.
  * `Public`

    All Public Pages are stored in this directory.
- `react-app-env.d.ts`

  This is a generated file coming with create react app
- `routes.ts`

  This is the file where all public and private routes are decalared. All public and private routes live in the publicRoutes and privateRoutes array respectively. Make sure you put the route in the correct context. All private routes require that users are authenticated, otherwise they will be redirected to the login page
- `scss`

  The directory that hosts all the scss for the app
- `store`

  This is the directory that is used to handle everything to do with the app's redux store.

  * `actions`

    All actions of the store are declared in this directory. Every action ends with `.action.ts`. This is to make all actions easier to search. Also all action functions end with `Action`.

    * `constants`

      All constants used in the store is declared in this directory. End all constants with `.constant.ts`. This is to make all constants easier to search.

    * `reducers`

      All reducers of the store are declared in this directory. Every reducer ends with `.reducer.ts`. This is to make all reducers easier to search.

    * `services`

      All services of the store are declared in this directory. Every service ends with `.service.ts`. This is to make all services easier to search. The `http` helper function must be used to make http requests

    * `store.ts`

      This is the script that creates the main store of the app.

    * `types`

      All typescript types of the stored are declared in this directory. Every type file ends with `.types.ts`. This is to make all types easier to search.

    * `tsconfig.json`

      This is the file that contains the typescript configuration for the app. The configuration used in this app is strict

  - `scripts`

    Server run and build commands are included in this folder files for installing requirements and ci cd auto deployment.

  - `appspec.js`

    file contains scripts files calls for ci cd auto deployment into aws instance

CHAPTER

TEN

CONTRIBUTION

10.1 How do you contribute to Purpie

We would very much like your request to improve Purpie so please read and follow this contributions manual before you start working.

10.2 Create and Report Issues

You should give us as much as possible details about the problems and how to reproduce these issues. Also, give us more detail about version of the Purpie you are using and development environment.

10.3 Contributor License Agreement

The Purpie projects are licensed under the Apache License 2.0 so you need to sign our Apache-based contributor license agreement as either an individual or a corporation to continue making these projects available under an Open Source license. **If you do not accept this agreement then sadly, we cannot accept your contribution.**

- If you are an individual please sign the form here.
- If you are a corporation, please sign the form here.

10.4 Pull Request Strategy

- Make sure your code passes the linter rules that are executing automatically when creating pull request.
- Purpie is a monorepo project so perform only frontend or backend change with one logical operation per pull request.
- Cleanly message your commits, squash them if necessary.
- Rebase your working branch on top of the develop branch before starting the coding.

10.5 Coding style

10.5.1 Comments

- Comments documenting the source code are required.
- Comments should be formatted as proper English sentences.

10.5.2 Duplication

- Copy-paste source code is not allowed, you can reuse it.

10.5.3 Formatting

- There are some prettier packages with eslint in the codebase, so you need to adjust your editor with Purpie settings.

10.5.4 Naming

- Util function names camelCase, file names kebab-case and react file and component names PascalCase format in the Purpie.
- The names of global constants (including ES6 module-global constants) should be written in uppercase with underscores to separate words. For example, `BACKGROUND_COLOR`.

CHAPTER

ELEVEN

LICENSE

Purpie is open-source and is released under the Apache License 2.0.