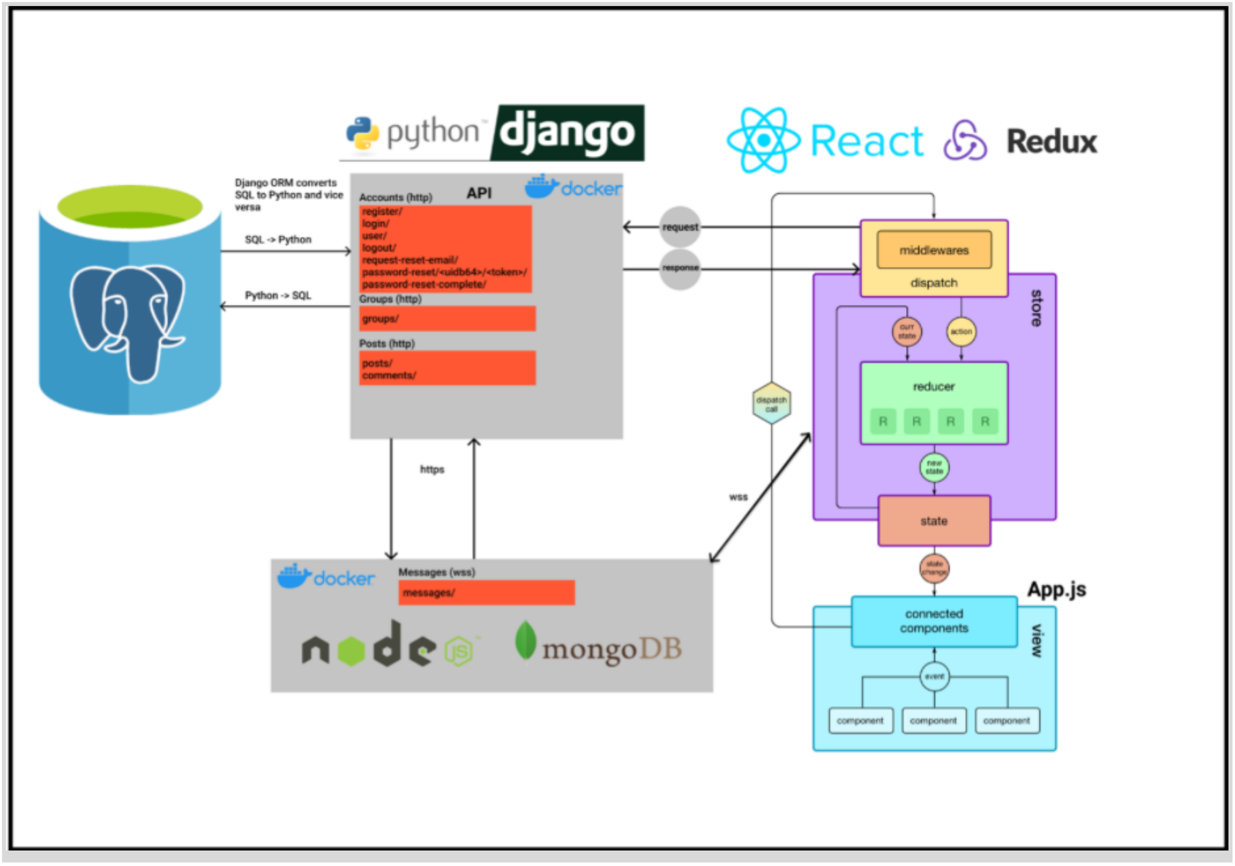
<u>Stöd</u>


python
django
React
Redux
Django ORM converts SQL to Python and vice versa
SQL -> Python
Python -> SQL
API
docker
Accounts (http)
register/
login/
user/
logout/
request-reset-email/
password-reset/<uidb64>/<token>/
password-reset-complete/
Groups (http)
groups/
Posts (http)
posts/
comments/
request
response
middlewares
dispatch
store
curr state
action
reducer
R
R
R
R
new state
state
dispatch call
wss
https
state change
App.js
connected components
view
event
component
component
component
docker
Messages (wss)
messages/
node js
mongoDB

Project Components:

/Resources

Contains the User’s Manual, Test Plans, Test Reports, and the block diagram for our project.

/Chat-server

The chat-server of Stod is built using Node.js with MongoDB.
Our modules are all contained in the /chat-server directory:

- /lib
- /src

These modules both contain an index file. The file in /src is written in typescript and then compiled into javascript and put into /lib. The javascript files are then used for the actual chat-server. A developer can modify the index file in /src to add/change/fix functionality with the chat-server.

/src/events.ts

Unused in the current state of the chat-server.

/src/index.ts

This file contains the connections to the MongoDB database and to the Socket.io sockets of the two users and the relaying of messages between them.

/Stod-Backend

The backend of Stod is built using Python, Django, and Django Rest Framework.
Our backend modules are all contained in the /stod-backend/stod/ directory:

- /accounts_api
- /friends
- /groups
- /posts
- /tags

These modules follow the same general structure with some important files being: models.py, serializers.py, urls.py, views.py. To create/modify a model the developer can edit the models.py file. To add/modify business logic the developer can edit the serializers.py and views.py files. To choose what routes connect to what functionality the developer can edit the urls.py file.

/accounts_api

This module handles all logic related to user accounts, including account registration, logging in, logging out, resetting passwords, validating the user's token, etc.

/friends

This module handles all logic related to friend relations and requests among users. It contains functionality for users to view their friends, send a friend request, and accept a friend request.

/groups

This module handles all logic related to groups. It contains functionality for users to view all the existing groups, subscribe/unsubscribe to groups, and for admins to create/delete groups.

/posts

This module handles all logic related to posts and comments. It contains functionality for users to create posts, view posts, filter posts, create comments, reply to comments, etc.

/tags

This module works in conjunction with the posts module to provide functionality for tags on posts. This module creates an endpoint so that an admin can create a set of tags that users can choose from to tag their post with, and then filter posts by.

/Stod-Frontend

The front end is built using ReactJS, Redux, TypeScript, and yarn.
/public contains all of the static files for our website such as various images. _redirect is used to ensure Netlify loads the page correctly.
package.json is generated by yarn to hold metadata relevant to the packages used in the project
Tsconfig.json is used by TypeScript to specify the root files and compiler options required to compile the project
Yarn.lock is autogenerated by yarn

/src

contains all the source code for our project

- /components

    - Contains multiple directories, each corresponding to a project feature. Within each directory are components related to the features, which are pieces of UI used throughout the react project.
- /actions
    - Contains all the redux actions used by all the various components. Each action is an event that leads to a /reducer
- /reducers
    - Contains all the reducers for each feature of our app. Each function updates states across our project as a result of an event.
- /contexts
    - Contains a context to manage the state for the sockets across various components
- /mui-components
    - Contains a couple Material UI components used throughout the app
- **App.tsx** is the first component that loads when the site is visited and loads the sub components. It also loads the components in /src/components depending on the current URL route.
- **font.css**: Imports the Comfortaa font from google fonts using its CDN.
- **index.tsx**: Creates Material UI theme given the JSON configuration and wraps around the main component tree. Also uses React DOM and hooks App.js into the root div of the DOM to create the entire component tree.
- **react-app-env.d.ts**: Global TypeScript declarations to be ignored.
- **store.ts**: Creates the main data store for Redux. Combines the rootReducer which contains all the other reducers designed by the developer. Also adds the initial state which is an empty JSON object.
- **styles.ts**: Contains a few global Material UI styles frequently used by some components.

Production Branches:

There are 2 production branches which contain a few adjustments to allow for seamless deployment.

Prod:

The front end branch contains all the same files as above but with modified code to ensure requests are made to the production back end servers.

Backend-prod:

This branch is the production branch for the backend server. It has additional files only needed for the back end server. It has docker-compose.prod.yml files within /stod-backend /chat-server which contain modified code for production deployment. It also contains an nginx conf file within /nginx as Nginx is only used in the back end.

Note: The environment variables used by the production deployment of docker are stored in .env.prod but NOT the ones actually being used in the production environment, as the file has been added to the .gitignore for security purposes. Those downloading for their own use should edit this file with their own variables.

Dev-Build Tool Information

Docker Information

Version: 20.10.5, build 55c4c88

- Python
    - Docker compose version: 3.3
    - Python container version: python:3.7-alpine
    - PostgreSQL container version: postgres:12.0-alpine
- Nodejs
    - Docker compose version: 3
    - Nodejs container version: node:10
    - MongoDB container version: default mongo container
- Nginx
    - Nginx container version: nginx:1.19.0-alpine

Django Information

- Python version: 3.7
- Package versions:
    - asgiref: 3.2.10
    - cffi: 1.14.5
    - cryptography: 3.2.1
    - Django: 3.1.2
    - django-cors-headers: 3.5.0
    - django-crispy-forms: 1.11.1
    - django-rest-framework: 0.1.0
    - django-rest-knox: 4.1.0
    - djangorestframework: 3.12.1
    - drf-pretty-exception-handler: 0.1.1
    - psycopg2: 2.8.6
    - pycparser: 2.20

- pytz: 2020.1
- signals: 0.0.2
- six: 1.15.0
- sqlparse: 0.4.0

Nodejs (Chat Server) information

- Node version: 10
- Yarn version: 1.22.10
- Express version: 4.17.11
- MongoDB version: 3.6.9
- Socket.IO version: 2.1.13
- Typescript version: 4.2.3
- Package versions:
    - @types/express: 4.17.11
    - @types/mongodb: 3.6.9
    - @types/socket.io: 2.1.13
    - body-parser: 1.19.0
    - mongoose: 5.11.19
    - tsc-watch: 4.2.9
    - typescript: 4.2.3

React Information

- Javascript (compiled) version: EcmaScript 5
- TypeScript version: 4.0.3
- React and React DOM version: 17.0.1
- Material UI version: 4.11.0
- Yarn version: 1.22.10
- Redux version: 7.2.2
- Package versions:
    - @material-ui/icons: 4.9.1
    - @material-ui/lab: 4.0.0-alpha.56
    - @popperjs/core: 2.6.0
    - @testing-library/jest-dom: 5.11.4
    - @testing-library/react: 11.1.0
    - @testing-library/user-event: 12.1.10
    - @types/jest: 26.0.15
    - @types/node: 12.0.0
    - @types/react: 16.9.53
    - @types/react-dom: 16.9.8

- @types/react-router-dom: 5.1.6
- add: 2.0.6
- axios: 0.21.0
- es5-ext: 0.10.53
- react-icons: 4.1.0
- react-router-dom: 5.2.0
- react-scripts: 4.0.0
- redux-devtools-extension: 2.13.8
- redux-thunk: 2.3.0
- socket.io-client: 3.1.2
- web-vitals: 0.2.4

Installation Procedure to rebuild the Web App from Scratch:

Backend

- Download the backend-prod branch to a publicly accessible server
- Use Letsencrypt to generate an SSL certificate for the user. The default directory it is placed into will work with the docker compose files
- Update .env.prod to use secure names/passwords
- In chat-server run `docker-compose -f docker-compose.prod.yml up --build`
- In stod-backend run `docker-compose -f docker-compose.prod.yml up --build`
- Create an Admin user for the backend in stod-backend by running `python3 manage.py createsuperuser`
- To reset either database run `docker-compose down -v`

Frontend

- Simply connect the prod branch code to Netlify to deploy the front-end
- Edit {DOMAIN} on line 4 of /stod-frontend/src/actions/types.ts to the IP address of the [backend server]:8000 so requests can be sent to Django
- Edit /stod-frontend/src/components/Users/UserView.tsx line 61 to [backend server]:4000