# AUX-BOT Documentation

## <Факултет по математика и информатика, Софийски университет>

**Курс**: Разработка на клиент-сървър (fullstack) приложения с Node.js + Express.js + React.js, летен семестър 2019/2020

**Изготвил**: Александър Бориславов Канджички,

**фн**: 81427

Дата: 25.07.2020

# Requirements:

## Functional requirements – use cases

* As a user, I can register, login and view my profile
* As logged user, I can browse all available commands on the bot
* For selected command, I can select action to take - (Run, Watch result, Schedule, Help)
* I can execute any listed command, configuring all parameters
* For commands that are over-time, I can live-watch results
* As administrator, I can manage command parameters and default values
* As administrator, I can see history of command runs

## Non-functional requirements

* The application should support live-updates of data
* The application should be integratable with any bot sharing the same interface as AppBot (compatible with other platforms)
* The commands should be easily maintainable in a configuration
* New commands should be easy to integrate without risking previous commands’ functionality bugs
* Failures in the third-party Bot should be reported and handled by the application without it failing

## Used modules and packages

* React.js - <https://www.npmjs.com/package/react>
* React application example - <https://github.com/Yog9/SnapShot>
* ws - <https://www.npmjs.com/package/ws>
* Axios - <https://www.npmjs.com/package/axios>
* RxJs - <https://www.npmjs.com/package/rxjs>
* Discord.js - [https://www.npmjs.com/package/discord.js](https://www.npmjs.com/package/discord.js?)
* Express - [https://www.npmjs.com/package/express](https://www.npmjs.com/package/discord.js)
* Mongoose -[https://www.npmjs.com/package/mongoose](https://www.npmjs.com/package/discord.js)
* Passport - [https://www.npmjs.com/package/passport](https://www.npmjs.com/package/discord.js)

## Architecture overview

## Routing

* Frontend routing is implemented with sub-modules for /users and /commands to allow pre-loading, lazy-loading, internal navigation and context separation
* Backend is using express-link routing for command handlers to ensure each command has independent scope and handler and attaching/detaching specific command does not influence other ones

## Authorization and permissioning

* The Bot permissioning is managed by Discord.js and the provided developer API – the backend provides secretly stored in the environment variables TOKEN that the bot consumes to communicate with the discord server
* Authorization of frontend users is implemented with JWT, using interceptor on frontend for each request, and authorization-midlewares on backend for handling the sent JWT

## Live updates

* Live-updates communication is implemented with web-socket, opened on connecting to the app
* Backend provides API with exporting standard command interfaces. Communication is required JSON strings through the websocket
* For data separation of users and commands, rxjs is used to provide proper filtering, data mapping and throttling of updates
* Updates on the client are handled with Context effects and data setters, standard in React

## Command storage

* Commands configuration is saved in Mongo Db, as json so it’s easy to communicate both through web-socket and HTTP
* Each saved entry is following strict command schema, implementing command configuration interface to ensure error-proof frontend rendering of the dynamic configuration of parameters

# Installation and operation manual

## Installation

* Run `npm install` on both frontend-react and backend folder
* Create discord server, follow this guide to give access to the bot to the discord -<https://discordpy.readthedocs.io/en/latest/discord.html>
* Install mongo db or use external one – add the connection string in the config to connect to mongo (can run `start-db.bat` on windows if you already have local DB)
* Add commands configuration in the database
* On backend run “npm run build” to build the server, `npm start` to start the backend application
* On frontend run `npm start` to run react

## Operation

* You need to register user and then login
* For each command you have to select a valid channel in order to run
* WS needs to be enabled (it should be enabled by default) on your browser, and your server needs to have firewall rule to allow WS port access and http upgrade
* To watch a command – simply type the ID that you have given it when you ran it and you will instantly receive updates

# Difficulties encountered & mitigation

* **App restart will disband all active observables and subscriptions on the discord bot** for live updates, so the user will have to re-run the command. Mitigating this can be done by saving message ID in the DB with expiration date, and on start – fetch all saved messages that haven’t expired and re-subscribe to reactions / events on them
* **React router does not support basename routing path-match**: Couldn’t implement easily the logic where /commands and /users will be completely separate routes with just Router class – needed to wrap them in a router and use a switch. This is partially mitigated. Further improvement can be done by making separate routing modules and importing them directly in the App.js so it looks ordered and pretty

# Used resources

* Example react app to build upon - <https://github.com/Yog9/SnapShot>
  + The repository is provided directly by reactjs official website - <https://reactjs.org/community/examples.html>
* Discord developer portal + API documentation - <https://discord.com/developers>
* React-router github repository issues - <https://github.com/ReactTraining/react-router/issues/5237>. Solution for the root basename navigation of modules
* The Moodle courses - <https://learn.fmi.uni-sofia.bg/course/view.php?id=6082>