Bot Documentation

> Telegram Bot build for the *BeuthBot*-Project, with easy extensibility and customization in mind.

Table of content

1. Bot Documentation
2. Table Of Content
3. Getting Started
    a. Prerequisites
    b. Installing
4. Overview
5. Structure
6. Functionalities
    a. User Requests
    b. Commands
    c. Functions
    d. BotFather
7. Further Development
8. Further Reading
9. Prerequisites
10. Versioning
11. Authors

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

Prerequisites

You will need a current version of node & npm.

Installing

After cloning the repository, install the dependencies. You can then run the project.

```
# install dependencies
> npm install
```

```
# serve at localhost:8000
> npm start
```

Overview

The bot is basically a *Node-Express*-Backend. Incoming requests are checked and specifically handled.

Structure

The bot is separated into two files. `index.js` contains the fundamental logic. The bot get created with its token and waits for incoming events. For example an incoming message. The bot then calls a handler function.

These handlers can be found in the second file, `commands.js`. This file contains the available commands as an Object. Furthermore does ist contain functions to determine if a message contains a commands and to answer the several requests a user can make.

Functionalities

User Requests

The bot supports three different kinds of user requests:

- **Message**: A user sends a message to the bot. We then check if the message contains a command. Commands are declared with a prefixed '/' in *Telegram*.

- **Callback Queries**: The bot can answer with a question, providing the user a simple interface, using a button matrix. When the user clicks on of these buttons we get a *callback query*.

- **Inline Queries**: Users can call our bot from within another chat by prefixing the botname with an '@'. The user can then send a text to the bot, which results in an inline query. The result that the bot gives back is inserted in the chat, where the user called the bot from.

Commands

The `commands.js`-file contains an `commands`-object. Every entry of this object is a supported command. The Key is always the command string, prefixed with '/', eg: '/help'. The value for these keys is an object containing an description, and options object and the reference to the function that renders the answer for the specific command.

```
const commands = {
    '/help': {
        answer: renderHelpString,
        description: 'Get a helpful list of all available commands and
functionalities',
        options: {
            parse_mode: 'Markdown'
        }
```

```
    },
  '/date': {
    answer: (message => 'What date format do you prefer?'),
    description: 'Get the current timestamp in a chooseable format',
    options: {
      parse_mode: 'Markdown',
      reply_markup: {
        // this initiates a callback query
        // by giving the user two buttons to answer with
        inline_keyboard: [
          [
            {
              text: 'Zulu',
              callback_data: JSON.stringify({
                command: 'date',
                payload: 'zulu'
              })
            },
            {
              text: 'German',
              callback_data: JSON.stringify({
                command: 'date',
                payload: 'german'
              })
            }
          ]
        ]
      }
    }
  }
}
```

Functions

The `commands.js`-file provides several functions. Eg. functions to check if a message contains an command and to find out if the requested command is in the 'commands'-object, which means it is an supported command.

Further are functions provided to handle Messages (containing normal *Commands*), *Callback Queries* and *Inline Queries*.

The bot has the following functionalities, that a user can request and use:

- getTimestamp: Get the timestamp of the moment the message containing this command was send.
- getFormatedTimestamp: Renders the timestamp in Zulu or German format, this is a function used to answer a *Callback Query*.
- renderHelpString: Iterates over the *commands*-object and prints all available commands and there description.
- supportedMarkdown: This function gives the User a list of supported Markdown markup by

Telegram.

BotFather

The BotFather allows so configure our bot. You can just write the *BotFather* with Telegram and the bot will guide you through everything. The *BotFather* enables you among others to change the profile picture, description and about text of your bot.

Further you can register the commands and inline queries your bot supports. This allows a cleaner user experience since the bot will then suggest commands and inline queries while the user types. So absolutely do register them!

The neccessary commands are:

- '/setcommands' - '/setinline'

Further Development

New commands can simply added to the 'commands'-object but have to follow the presented structure under commands.

Further Reading

- Telegram Bot API

Built With

- Node.js
- Express.js
- Node-Telegram-Bot-API

Versioning

We use SemVer for versioning. For the versions available, see the tags on this repository.

Authors

- **Tobias Klatt** - *Initial work* - GitHub

See also the list of contributors who participated in this project.