# Quick Start Guide

### Step 1:

Install “docker” and “docker compose”.

Instructions available ont their official website: https://docs.docker.com/engine/install/

### Step 2:

Install “git”.

Instructions here for various platforms: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

### Step 3:

Clone the github repository to where you want to deploy the solution with the command:

```
git clone https://github.com/somik123/todo-list.git
```

### Step 4:

Run these two commands from inside the root folder where the github repo was cloned. The folder contains the “docker-compose.yml” file. (On linux, may need to run these commands with “sudo” prefix).

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

To see all running containers use the command:

```
$ docker ps
```

To stop the running containers, use the command:

```
$ docker compose stop
```

# Frontend with React.js

## App.js

The app is broken into two parts, The Main view with Todo list and add form and the view which allows for editing and deleting of then Todo tasks.

App.js contains the <Switch> where both of these views can be loaded based on the URL. Default path "/" loads the main view, while the path "/edit/{id}" loads the edit view for the task with id of {id}.

The views are imported from their respective components in lines 6 and 7 on the App.js.

## Services Folder

The services folder contains a single file, "TodoDataService.js" which is used for communication with the REST API.

API's base URL is defined as a constant API_BASE_URL and the individual API calls are defined as asynchronous methods in the TodoDataService class.

## Components Folder

The component folder contains the two views that are used for the display.

**TodoList**

This contains the display for the list of Todo tasks, option to mark them as complete/incomplete as well as the form for adding more tasks to the list.

Items of note here are the:

- retrieveTodoList() method that gets the list of Todo tasks from the API using the "TodoDataService" and saves it to state variable array "todoList". Once the component is mounted, this method is called.
- saveTodo() method, that gets called once the user has typed in a new Todo task and clicks the "save" button. It validates the user input and sends it to the backend server through "TodoDataService". Once success, it calls "retrieveTodoList()" to refresh the list of Todo tasks, on failure it sets the error flag to display the error message.
- completeTodo() method that is called when the user clicks on the checkbox to mark/unmark the task as complete/incomplete.

The rest of the methods are usual setter methods for the state variables.

**EditTodo**

This contains the Todo task edit form as well as the task delete button.

Items of note here are the:

- getTodo(id) method that uses the TodoDataService to retrieve the Todo task by it's id. Once the component is mounted, the id provided in the URL is used to get call this method.
- updateTodo() method, that gets called once the user has modified the Todo task and clicks the "update" button. It validates the user input and sends it to the backend server through "TodoDataService". Once success, it redirects the user to the list of Todo tasks view, on failure it sets the error flag to display the error message.
- deleteTodo() method, that deletes the Todo task through "TodoDataService". It will always redirect the user to the list of Todo tasks view.

The rest of the methods are usual setter methods for the state variables.

## Tests

The test package for this React App contains the following tests:

- App.test.js
    - Confirm title is displayed.
        - This is done as a sanity check as it should always be displayed regardless.
- __test__/TodoList.test.js
    - Checks if the "Add Task" button is rendered.
        - This, as the above, is done as a sanity check
    - Checks if the "Add Task Form" is displayed on button click.
        - This ensures the form is displayed when the anyone clicks on the button.
    - Save a new task.
        - It clicks the "Add Task" button again, and unlike the previous time, it fills it up with a new Todo task.
        - Once done, it saves it. No validation is done in this test.
        - After adding, it waits for 2s to allow API to receive and save the data.
    - Ensure the task was added successfully.
        - This method looks for the task that was added in the previous step and ensures it exists.
    - Tick the checkbox for the task that was added.
        - This method will try to tick the checkbox for the new task that was added during this test phase. Before clicking, it will confirm that the currently displayed checkbox is NOT checked. This is done by looking at the URL of the image for checkbox.

- Confirm it shows as checked.
    - This method will check whether the previous method was successful in checking the checkbox. If it was, the URL of the image should be changed to reflect that. At the end, it'll again activate the checkbox to make it unchecked.
- Confirm it shows as unchecked.
    - This method will check whether the previous method was successful in unchecking the checkbox. If it was, the URL of the image should be changed back to the original URL to reflect that.

## Dockerfile

A simple Dockerfile is used to compile and launch the React App in docker using node:alpine image. The Dockerfile is located inside the todo_frontend folder and is usually launched by the docker-compose.yml to create a docker stack.

# Backend with Dropwizard and MySQL Database

## Database Configuration

Database details needs to be updated both in “docker-compose.yml” as well as inside the “simpletodo-backend/config.yml”. Default settings are:

**Database Type:** MySQL
**Database Name:** todo
**Database User:** todo
**Database Password:** 68yqU1eqg0JfvFTO

The JDBC3 plugin for Dropwizard was used to communicate with the database.

When launched using the “docker-compose”, the database host is “db” while if running on same machine, it can be changed to “localhost”. This needs to be defined in the “simpletodo-backend/config.yml” as well.

## org.somik.todo

The entry-point for this app is the simpletodoApplication.java file located in this. Here, in the run() method, the “cross-origin resource sharing” filter was enabled and configured to allow all origins, as well as to allow methods used with REST calls. Without this, the React App will not be able to communicate with the backend server.

This Java application uses a “database access object” (DAO) called TodoDao. Once the connection to the MySQL is established using JDBI3 and TodoDao class is registered as the DAO, the “data service layer” TodoService is initialized and using dependency injection, injected into the “TodoResource”, which is the Controller that receives the API calls and processes them.

## org.somik.todo.api

The entity/bean is defined in this package as “Todo.java”. It contains the attributes for the Todo and its constructors, getters, setters. The entity is then mapped to the database using the “TodoMapper”. This class implements the “RowMapper” interface and maps each row of the database table to one “Todo” object.

## org.somik.todo.dao

This class is our registered DAO and is used for all database calls. This DAO contains all method signatures and their respective SQL queries/updates.

## org.somik.todo.service

The “TodoService” interface contains the method signatures used by the service layer to communicate with the DAO interface. Different implementations of this interface can be used for different business logics. This provides lose coupling between the

The “TodoServiceImpl” is one of the implementations of the “TodoService” interface.

- insertTodo() method validates the length of title (name), description and dates are valid before passing these to the “todoDao” to save to the database. On success it returns the latest added Todo task, on failure, it returns null.
- updateTodo() method does the same validations as “insertTodo()” method as well as ensures the “id” is valid and on update, returns the Todo task with that id.
- findById() returns the Todo task that matches the id, as long as the id is valid.
- completeTodo() method ensures the task exists and marks it as checked/unchecked depending on the input.
- findAll() returns all Todo tasks in the list.
- createTodoTable() is used to create the tables in the database. This is only done if the tables do not exist. If they do, the tables needs to be dropped manually for recreation.
- deleteTodo() deletes the Todo task that matches that id.
- validateDate() method confirms the date string provided is between 1st Jan 2022 and 31st Dec 2032. It returns true if valid, false if invalid.

## org.somik.todo.service

The service is responds to the API calls. It uses “TodoService” interface as it’s “data service layer” and makes data calls through it to respond to the API calls.

| HTTP Request | API Endpoint | Description |
|---|---|---|
| GET | /todo/install | Setup tables in the database |
| POST | /todo | Insert in todo list |
| GET | /todo | Get all existing rows from todo list |
| GET | /todo/{id} | Get one row from todo list by id |
| PUT | /todo/{id} | Update one row in todo list by id |
| GET | /todo/{id}/1 | Mark a task as completed |
| GET | /todo/{id}/0 | Mark a task as incomplete |
| DELETE | /todo/{id} | Delete one row from todo list by id |