Syte todo-list Technical Design

Table of Contents

[Abstract 3](#_Toc130221812)

[Frontend 3](#_Toc130221813)

[Components 3](#_Toc130221814)

[Backend 5](#_Toc130221815)

[API 5](#_Toc130221816)

[Database 6](#_Toc130221817)

[Schema 6](#_Toc130221818)

# Abstract

A basic todo list

* items consist of a check box and a name
* items can be added
* marked as completed or unmarked
* items can be deleted
* support user registration and login

# Frontend

Single page react application.

Consume APIs using axios.

## Components

App

The root object manages the object components

Implements

1. Login API  
   Using email/password  
   Fetch a token from the backend and store it in the browser’s local storage.
2. Registration API
   1. Attempt to create a new user with a username and password
   2. If the user already exists, ignore it already exists and attempt to login using the provided username and password.
   3. Login newly created user.
3. Logout  
   null the token and remove it from the local storage.

Login

UI

1. Button – LOGIN  
   Bound to the App login function
2. Button – REGISTER  
   Bound to the App register function
3. Field – Username
4. Field – Password

Once logged in, forward the user to HomePage.

HomePage

The component where the todo list is managed.

Implements

1. Create todo API: Create a new todo item with an optional name and unchecked by default.  
   Using the Auth Bearer header with the token received during login
2. Update todo API: Toggle the completed state of the todo item.  
   Using the Auth header
3. Delete todo API: Delete a todo item from the list  
   Using the Auth header
4. Get todos list: Fetch all existing todo items for the user  
   Using the Auth header

UI

Field – todo name

Button – add item bound to the create todo function

HTMLUList – with an item for each todo item in the data

* Checkbox – bound to the Update function
* Button – delete bound to the delete function

Button – logout bound to the App logout founction

# Backend

Serve APIs for managing users and users’ todo items.

## API

Register – POST [<host>/register]

* Body
  + email: string – email
  + password: string
* Returns
  + Status 201
  + Result
    - error: soft error to update user already exists
    - object: the newly created user object (id, email, password)

Login – POST [<host>/auth/login]

* Body
  + email
  + password
* Returns
  + Fail
    - status 401
    - Unauthorized
  + Success
    - status 201
    - access\_token: the login token to be used in the API headers

all-tems – GET [<host>/todo-items]

* No parameters or body
* Returns
  + Fail
    - status 401
    - Unauthorized
  + Success
    - status 200
    - [{id: int, owner\_id: int, done: bool, name: string}, …, {…}]

create-item – POST [<host>/todo-items]

Items are created un-completed by default.

* Body
  + name: string – optional
* Returns
  + Fail
    - status 401
    - Unauthorized
  + Success
    - status 200
    - {id: int, owner\_id: int, done: bool, name: string}  
      (newly created item)

edit-item – PATCH [<host>/todo-items/:id]

* Parameter
  + id: int - item id to update
* Body
  + name: string
  + done: bool

delete-item – DELETE [<host>/todo-items/:id]

* Parameter
  + id: int – id of the item to be deleted
* Returns
  + Fail
    - status 401
    - Unauthorized
  + Success
    - count: int (0 – nothing deleted, 1 – one todo item deleted)

# Database

PGSQL database with two tables, user and todo\_item.

Edit the database schema by defining new ORMs in the backend and then migrating the DB.

## Schema

* user
  + id: int – PK
  + email: string – Unique
  + password: string
* todo-item
  + id: int – PK
  + owner\_id: int – FK (-> user.id)
  + name: string – nullable
  + done: bool – default: false