Introduction

This is the documentation for Todoly, a todo list application. With this web app you can do the following activities:

- Create your own account
- Login to your account
- Save all your todos
- Edit your todos
- Delete your todos after you are done

Each todo is formed by a title, description and date of your last edit.

Backend

The backend is written in Node.js using the Express.js framework and it save the todos in a relational MySQL database. Moreover, all the inputs from the users are sanitized to prevent XSS attacks using xss library (https://jsxss.com/en/index.html). The password of the user is hashed using bcrypt (https://github.com/dcodeIO/bcrypt.js#readme).
The credentials to access the database and the session secret it is saved in .env file that has the following structure:

```
HOST="#######"
USR="########"
PASSWORD="##########"
DB="##########"
PORT="########"

SESSION_SECRET="LONG RANDOM STRING"
SESSION_MAX_AGE="INTEGER"
```

The backend is dockerized and deployed on a server [redacted]

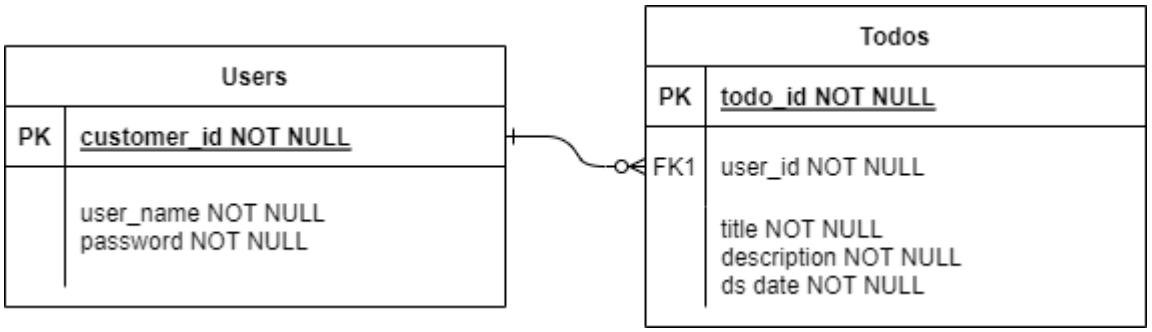
Frontend

The frontend is written in NextJS with TailwindCSS. It uses also UI components from DaisyUI (https://daisyui.com/). The frontend is deployed on Vercel (https://click-app-frontend.vercel.app/) [redacted]

Database

In the database each user and todo is identified using an UUID (https://www.percona.com/blog/2007/03/13/to-uuid-or-not-to-uuid/).

The schema of the database is the following:



API endpoints (Authentication):

- **/login [POST]**: It accepts a JSON body like {"username": "sadman", "password":"sadman"} and it returns in case of 200 OK response:

```
{
        "success": true,
        "message": "Successful Login",
        "user": {
                "username": "sadman",
                "id": "d504be8c-5547-4dd7-9e9e-f3d1a9040e31"
        }
}
```

  otherwise it returns:

```
{
"message": "Password is wrong"
}
```

- **/register [POST]**: It accepts a JSON body like {"username": "user18", "password": "user18"} and It returns status 409 (Conflict) if the user is already registred, otherwise it returns status 200 (OK).
- **/logout [POST]**: It returns status 200 OK when successful. It doesn't require a body.

API endpoints (Todo management)

- **/todos [GET]**: It loads all the todos of a specific user, doesn't require a body and it returns an empty array if no todos exist for that specific user, otherwise it returns:

```
{
	"todos": [
		{
			"id": "3cba02cb-d1ff-4680-af60-dda75e37a9d0",
			"title": "New Todo 2",
			"todoText": "Lorem ipsum 2",
			"userID": "d504be8c-5547-4dd7-9e9e-f3d1a9040e31",
			"ts": "2022-08-09T19:44:18.000Z",
			"dt": "2022-08-09T19:44:18.000Z"
		},
		{
			"id": "e22f25f9-58c6-4c86-8f9b-7c66f583f0a2",
			"title": "New Todo",
			"todoText": "Lorem ipsum",
			"userID": "d504be8c-5547-4dd7-9e9e-f3d1a9040e31",
			"ts": "2022-08-09T19:44:03.000Z",
			"dt": "2022-08-09T19:44:03.000Z"
		}
	]
}
```


- **/createTodos [POST]**: It requires a JSON body like this {"title": "New Todo 2","todoText": "Lorem ipsum 2"} and it returns 200 OK.
- **/updatetodos [POST]**: it requires a JSON body like this {"id": "ac3af3c8-fac8-4b7e-9667-17a921ca3de2","title": "test","todoText": "Lorem ipsum"}. The id is the id of the todo that we would like to modify. It returns 200 OK if successful.
- **/deletetodos [DELETE]**: it requires a JSON body like this: {"todo_id": "c05ba7db-2920-47ff-9c75-ca56224c82ce"} and it returns 200 OK.