LAB11: EJS + MySQL

Objective: Students will practice

- Dynamic template with EJS
- Express routing parameter
- MySQL

Lab instruction

- The LAB11 instruction and lab resources are posted on MS Teams channel LAB11 – ToDoList Application of subject 953262 (your section).
- Download the zipped file of resources-lab11.
- There are 7 steps according to the LAB11 sheet posted on the channel.
- The LAB11 is worth 10 points in total.
- Score criteria: full point (for output correct); -1 (for output does not correct); -1 (for not follow problem constraint)
- **Assignment Submission:**
    - Upload your solutions to MS Team assignments. The submission later than the 'due date' will get 50% off your score. At the 'close date', you cannot submit your assignment to the system.
    - You should also be prompt for TA calling to verify your work on your computer.

A ToDoList Project

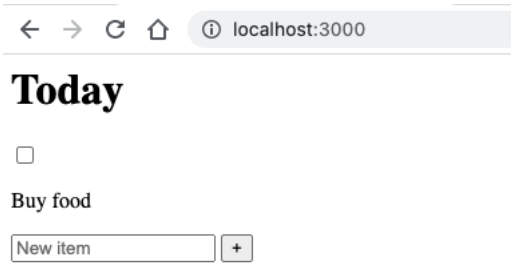
This project is to build a Nodejs application, connecting to MySQL, that allows a user to maintain important tasks to a to-do list. Users can add new tasks and also delete tasks from the Web application.

Step 0: Check MySQL service (1 point)

1. Start the MySQL service via terminal: mysqld start
2. Connect to your MySQL database via your selection of DBMS tool e.g. MySQL Workbench. Ensure that the connection is successful.
3. Create a database name `todolistDB`

Step 1: Set up a to-do-list project (1 point)

1. Download the starting files from the LAB11-resource file. In the **todolist** folder, you will find **app.js** and **index.html** and 5 **folders (public, views, config, control and model)**.
2. Create package.json file of the npm package
3. Install **express**, **body-parser**, **ejs, lodash and mysql2** packages into the project with NPM. Check package.json file again to ensure the installations.
4. In app.js, create **an express root(/) route for the GET**
    a. Then, add command `res.render('list')` to call EJS to **render** the **views/list.ejs** inside app.js.
5. Run the server with **nodemon app.js** and test a result on your browser (http://localhost:3000/). You should see the expected output as follows.

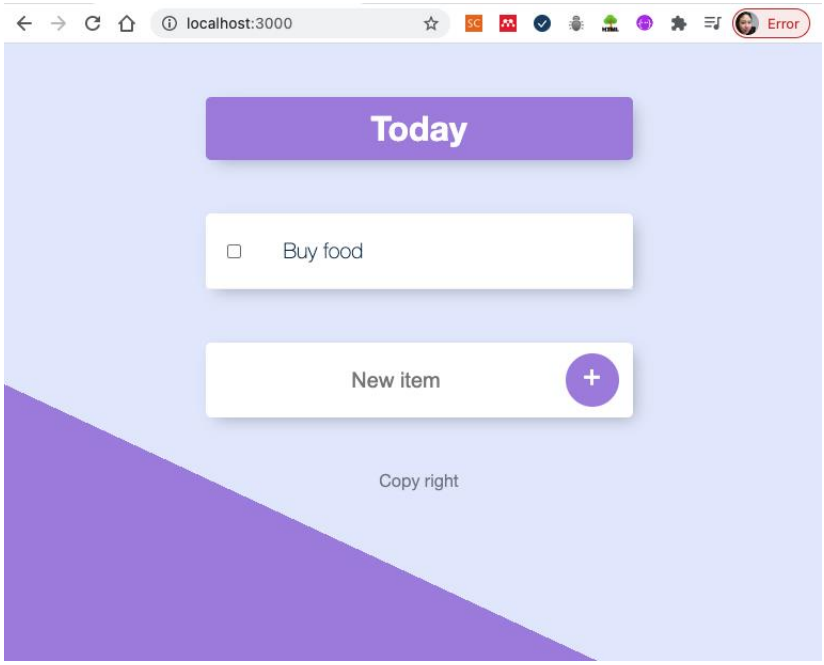


Step 2: Modify the list template with its partials (2 points)

Currently, the list page does not contain header and footer of HTML part yet. So, you need to replace the header and footer with the ejs partials and include them into the list.ejs.

1. Create a new subfolder named **partials** inside the **views** folder.
2. Move **header.ejs** and **footer.ejs from views folder** into **partials folder.**
3. Use `<%- … %>` to include the 'partials/header' and 'partials/footer' into the **list.ejs**.

4. The expected output is as follows when you run http://localhost:3000/ on the browser.



Step 3: Create an exported method of database configuration (1 points)

This step is to create an exported method of database configuration to be used in the app.js.

1. Open the **'config'** folder; you will find the **db.js** file that already import the **mysql2 package.** So that you can connect to MySQL.
2. Config your database connection to '**todolistDB**'
3. **Create an exported method of mysql connection** to Database '**todolistDB**' using the following code.

```
let config =  { host: "localhost",
user: "...",
password: "...",
database: "todolistDB"
};

module.exports = mysql.createConnection(config)
.then(()=> console.log('Database is connected'))
.catch((e) => console.log(e));
```

4. Then, you can use model **'./config/db.js'** to connect to database from anywhere e.g. **app.js**
    i. Add command `require("./config/db");`
5. in app.js, in order to connect the database through the module of **'./config/db.js'**:
6. Run the server with **nodemon app.js.** You should see the log of 'Database is connected' on your terminal.

Step 4: Create MySQL model() (1 point)

This step is to create the class of `TodoItemsModel` as a data model for managing TodoItems in the database.

First, use your DBMS to create table in the database

```
CREATE TABLE `todolistdb`.`todoitems` (
        `id` INT NOT NULL AUTO_INCREMENT,
        `name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`));
```

In the file 'listItem.js', the class `TodoItemsModel` is extended from `BaseSQLModel` ('baseSQLModel.js').

- `BaseSQLModel` establishes connection to the database as in the ./config/db.js
- `BaseSQLModel` consist of basic SQL statement available to apply with all mySQL table such as
    - `findAll() – to get all data columns from the table`
    - `findByColumn(column) – to get only the specified 'column' from the table.`
    - `create(data) – to insert 'data' into the table`
    - `update(id, data) – to update 'data' to the specified record 'id'`
    - `delete(id) – to delete the specified record 'id'`

- The entity class which extends `BaseSQLModel` must specify
    - `'tablename'` in the constructor to specify table for the data handling of the model
    - You can define addition functions for the entity such as getter and setter for the data table
- File 'listItem.js' is the example of how to define data model
    1. `BaseSQLModel` must be include using require( ) for the class extend
    2. Constructor is specified as 'todoitems' for db table connection
    3. The table 'todoitems' is to define list of todo items for later showing on UI for user to choose their wishing todo list. So, four functions are defined as for your example showing how to call method from the `BaseSQLModel` and how the class `TodoItemsModel` can be exported and used from 'app.js'

        - `defineInitialItems()` is to check if there is a record of todoItems in the database. If not, it will call `setinitialItems` () to insert initial todoItems into db. The functions call this.findAll() and this.create(item) defined by

`BaseSQLModel`. This function is called when you browse to the '/' root GET for the first time.

- `getTodoItemsName()` is to get the list of item name defined in the db
- In the last line of code, the instant of TodoItemsModel is exported as a Node.JS module. `module.exports = new TodoItemsModel();`

Step 5: Get MySQL data model from app.js and render on EJS (2 point)

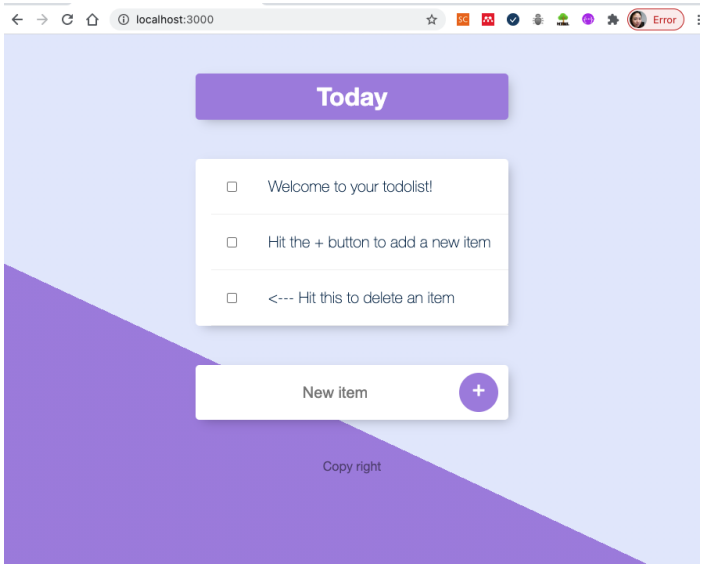
1. Open the **app.js ,**
   a. Add `const listItem = require("./model/listItem");` to add the export module of data model defined above
2. **in the GET '/' method**
   a. Add `listItem.defineInitialItems()` to route to the model for checking and defining initial list of todoItems

   b. Add `const items = await listItem.getTodoItemsName()` to get the list of todoItems from db

   c. In your res.render('list'), add these **two fields** { `listTitle: "Today", newListItems: items }` for pass data to the EJS for UI display in later step

   d. In **list.ejs**, modify the <h1> element with EJS to print out the value of `listTitle` and the <p> element with EJS to print out all `newListItems` value.

The result should be like this:



Step 6: Create items in the database from the app.js module. (1 point)

1. Open the **app.js**; **Require the listItem.js** module to be used in the app.js module.
```
const listItem = require("./model/listItem"); //(already done)
```

2. **Create** `const newitem` variable for preparing data to insert into the db model. The item should be defined using table name with its value as follows.

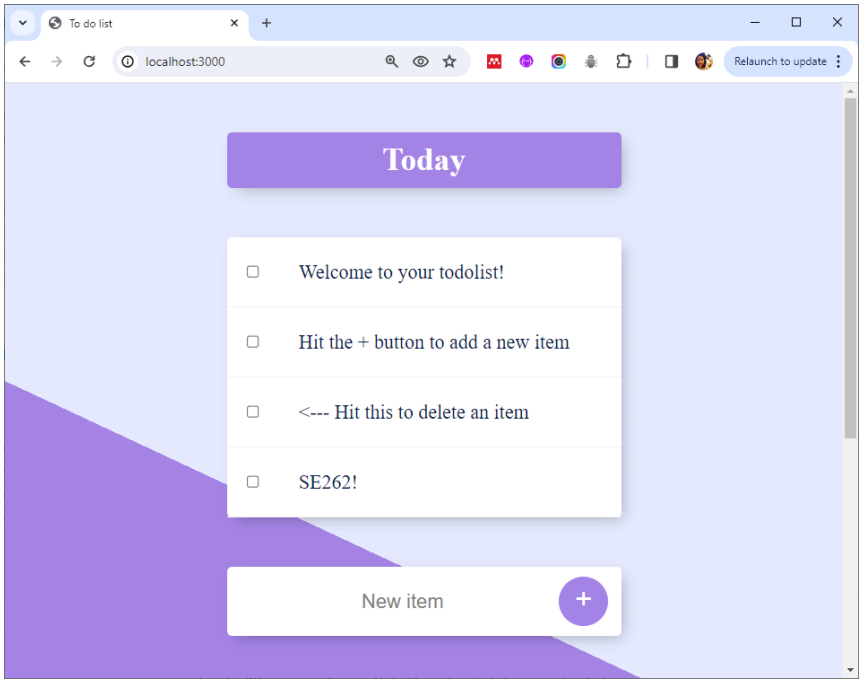
```
const newitem = {
    name: "SE262!",
  };
```

3. **Insert the** `newitem` **into db through the model** using the code as follows.

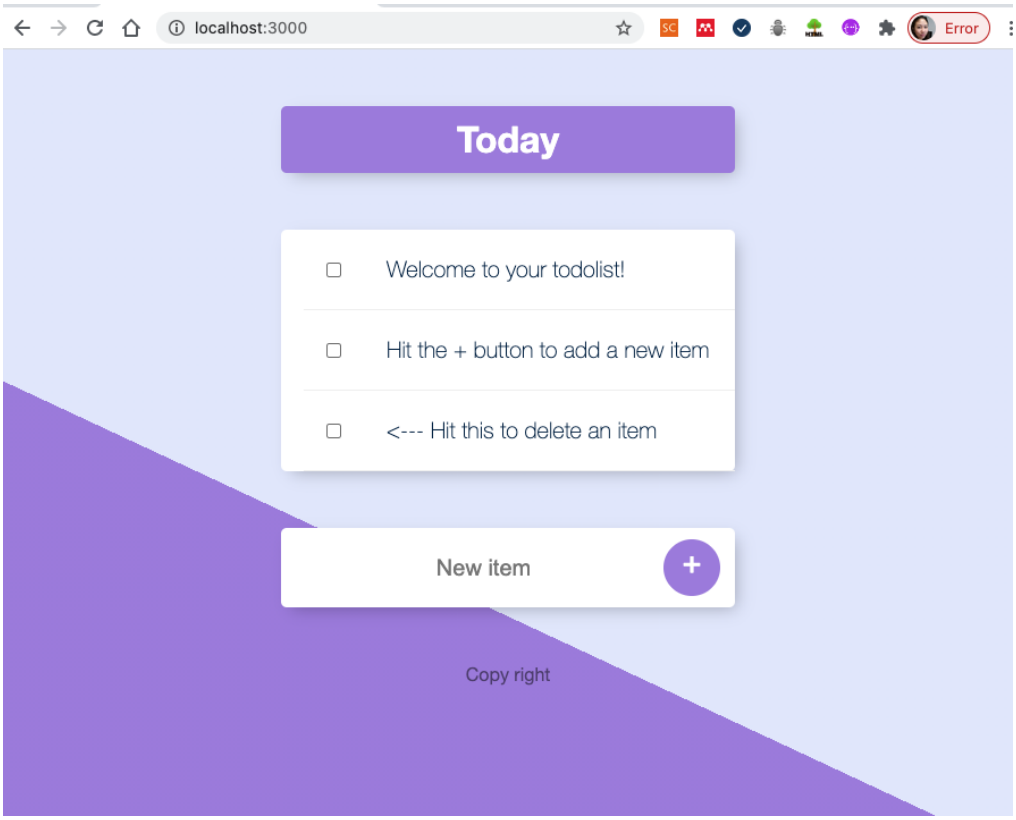
```
listItem.create(newitem);
```

4. Reload http://localhost:3000

5. Check db table 'todoitems', now you should have 4 records.



Step 7: Deleting items from the ToDoList Database (2 points)
This step is to delete an item from the items collection when the user checks a box on the item. The example is as follows when the user checks a box on 'SE262!' item.



1. To finish this step, you have to send the item's id back to the server and then delete the item from the database.
2. In list.ejs, **fill in action and method attributes of the 1st <form> element** with "`/delete`" and "`post`" respectively.
3. Add attribute `onChange="this.form.submit()"` into **the checkbox input element** to send the value of variable `checkbox` (item's id) back to server. Use EJS to embed an id of the item.
4. In app.js, **create a '/delete' route POST method** to send the item's id from the checkbox form to the server. Log it to see a result.
5. Then **use async and await** to delete the item using the following code.

```
const result = await listItem.delete(deleteItemId);
      //if result found, print out successful message in the
console and redirect to ('/')
}
```

6. Finally, check the items in your database.