

# TodoApp

---

WEB SYSTEMS DEVELOPMENT

Authors: Lucas Gutiérrez Durán  
Yousef AlSarraj

Theory Report  
Date: 4/5/2022

# Contents

1. Introduction to the project .....	2
2. Sprint 1 – Tasks and Tasks Collections management.....	5
2.1. Introduction .....	5
2.2. Presentation .....	5
2.3. Navigation diagram .....	6
2.4. Contents model .....	6
3. Sprint 2 – Users management.....	8
3.1. Introduction .....	8
3.2. Presentation .....	8
3.3. Navigation diagram .....	9
3.4. Contents model .....	10
4. Sprint 3 – Tasks Collections sharing and Notifications.....	12
4.1. Introduction .....	12
4.2. Presentation .....	12
4.3. Navigation diagram .....	13
4.4. Contents model .....	15
Appendix 1. User guide .....	17
1. Project management.....	17
2. Task management .....	18
3. Other functions .....	18

# 1. Introduction to the project

---

## 1.1. Requirements

---

For this section, we refer to the statement provided by the teacher of the subject where everything that our application must contain is explained.

The main goal of the exercise is to develop a web application with Ruby on Rails in which we are going to implement a web system for managing a ToDo Web Application similar to Todoist. In order to solve the exercise is mandatory to use software development good practices, like version control software (git) and code hosting (GitHub) and is advisable to use TDD (Test Driven Development, by using RSpec).

We want to develop a ToDo Web Application, similar to Todoist in Ruby on Rails. This app should fulfil the following requirements:

- It should be accessible from any web browser.
- Application should allow the user the management (CRUD operations) of:
  - o Tasks (Create, Edit, Delete, Check). Tasks have a priority (High, Medium, Low) and they can have deadline date.
  - o Project or Tasks collections (Create, Add and Remove Tasks, Delete and Share).
  - o Users (Sign-in, Login, Logout, Delete).
  - o Tasks collections sharing (Requests, Allowance, Denial, Revoke).
- The app should have a User Management module, which permits users to access the system by authenticating themselves and determining user roles:
  - o Administrator: Can manage users, relationships, tasks collections and tasks
  - o User: Can create tasks and collections and can share task collections.

## 1.2. Methodology

---

The delivery of the practice will be divided into three sprints, in which the main functionalities of the project will be addressed. The planned schedule is as follows:

- Sprint 1. Tasks and Tasks Collections management. From the 8th of February to the 16th of March.
- Sprint 2. Users' management. From the 17th of March to the 13th of April.
- Sprint 3. Tasks Collections sharing and Notifications. From the 14th of April to the 4th of May.

At the end of each sprint, the professors will have a small meeting with each team to check the progress made. In this meeting the students must show the work done and show the updated models of content, navigation and presentation. It is important to attend to these

meetings since part of the final grade will depend on the degree of compliance with the requirements of each sprint.

### 1.3. Technologies used

---

#### - Program language

To do this we have used the Ruby language within the Rails framework. We have decided to use this language because of its simplicity, since, for this context, it is very easy to use. In addition, the teacher provided basic knowledge with which we could start programming in this language.

We tried to do without JavaScript, as we did not find it necessary and everything could be done using Ruby alone. However, as it is obvious, we had to use HTML and CSS for the frontend development. Although, as an exception, we had to use an extension called '.erb' to be able to use Ruby embedded in HTML, which made the connection between the frontend and the backend much easier.

#### - Database

To ensure the persistence of our application we have used MongoDB through MongoDB Atlas. MongoDB is a non-relational document database based on JSON-like file formats. We have found it convenient to use this database because of its simplicity and lightness. It provided us with what we needed without much hassle.

In addition, thanks to MongoDB atlas we have had a server in the cloud totally free and accessible from any computer, so this has made our work much easier at the time of deployment.

Ruby on Rails provided useful gems to be able to manage in an extremely simple way the use of mongo from the backend, so we didn't have to worry about complex statements. We only had to make calls to established methods such as new, save or destroy.

#### - GitHub

For version control we have used GitHub (<https://github.com/LuGuDu/todoapp>) trying to follow the git-flow.

Each component has had its own development branch in which it was making small increments in each Sprint. Once the work was finished, we would agree to make a pull request to another branch called releases, where we would see if everything was working correctly together. When, finally, everything worked correctly, we made another pull request to the main branch, where you can find our complete work.

We tried not to touch the main branch too much. Only when necessary, at the end of each sprint. We did this in order to get into good software development habits using git-flow.

Another advantage of having used github is to be able to visualize how much work each component has done, among other things.

- Deployment

For the deployment of the page, we have used the Heroku service. This service has allowed us to have our website hosted, so that anyone can access it through a browser with internet access.

From the beginning we linked Heroku and GitHub to achieve a continuous deployment, which is a good practice in the world of software development. To do this we told Heroku our GitHub account and we told it that, every time a push was made to the main branch, a deployment of that branch would be made. In such a way that we would always have the web page updated with the latest changes made in each sprint.

This worked correctly until a few weeks ago because, due to a security bug in GitHub, Heroku temporarily stopped deploying with GitHub. This forced us to deploy the last sprint manually using a console and Heroku CLI. Even so, we managed to overcome the problem and we were able to deploy our web application without any major problems.

Link to the web page deployed on Heroku: <https://todoappdsw.herokuapp.com/>

## 2. Sprint 1 – Tasks and Tasks Collections management

---

### 2.1. Introduction

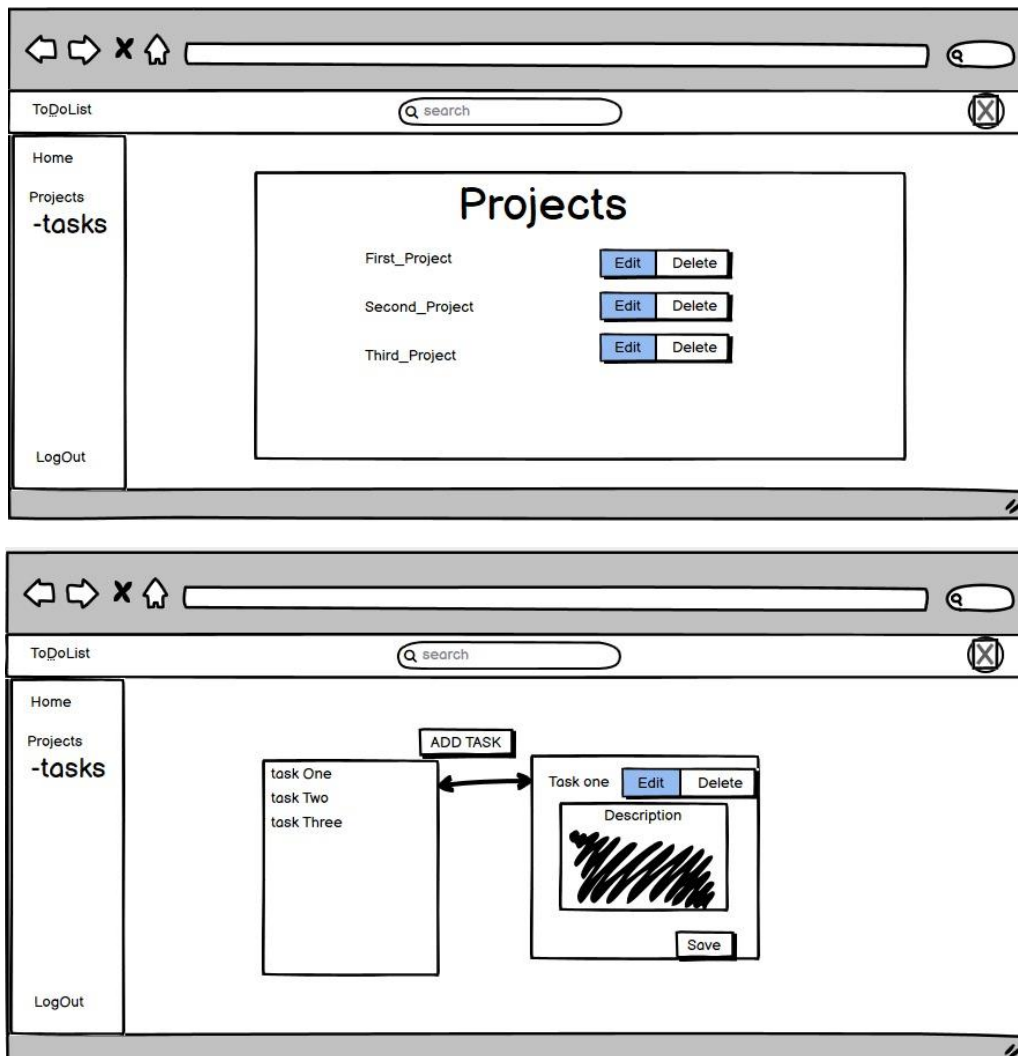
---

During this first sprint we have developed the functionalities related to task and project management. For both elements you can create, delete, and update them. A task can have one or no project assigned to it.

### 2.2. Presentation

---

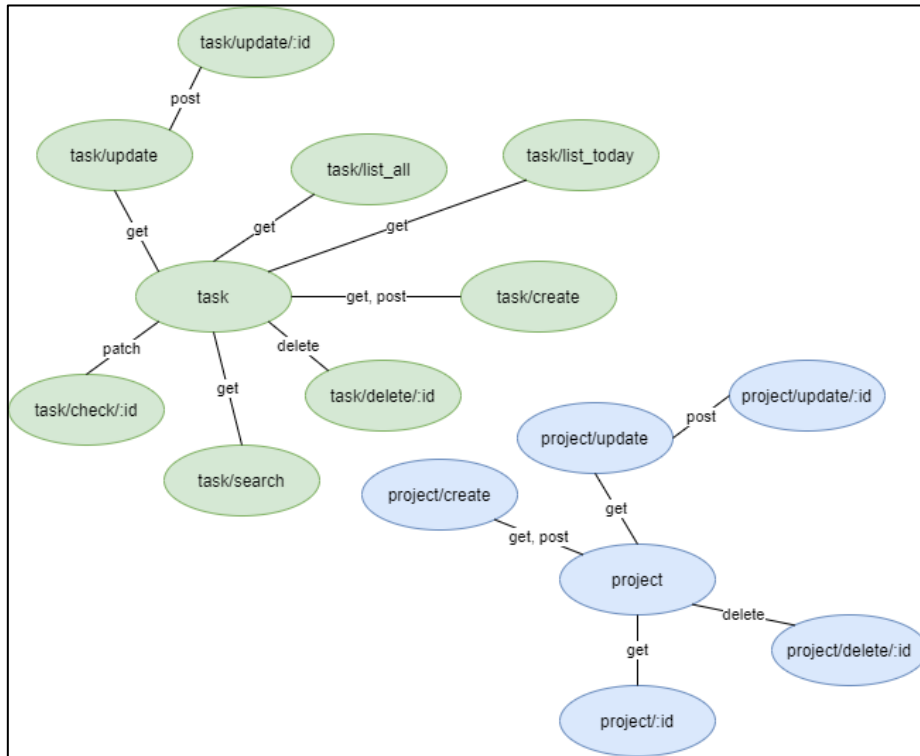
Below, we show the sketches we made as an orientation to the graphic design of the frontend of our web application. The sketches have been made using Balsamiq Mockups.



### 2.3. Navigation diagram

---

In the following figure we show the diagram of all the routes of our web application. In green you can see the routes related to the tasks, and in blue those of the projects.



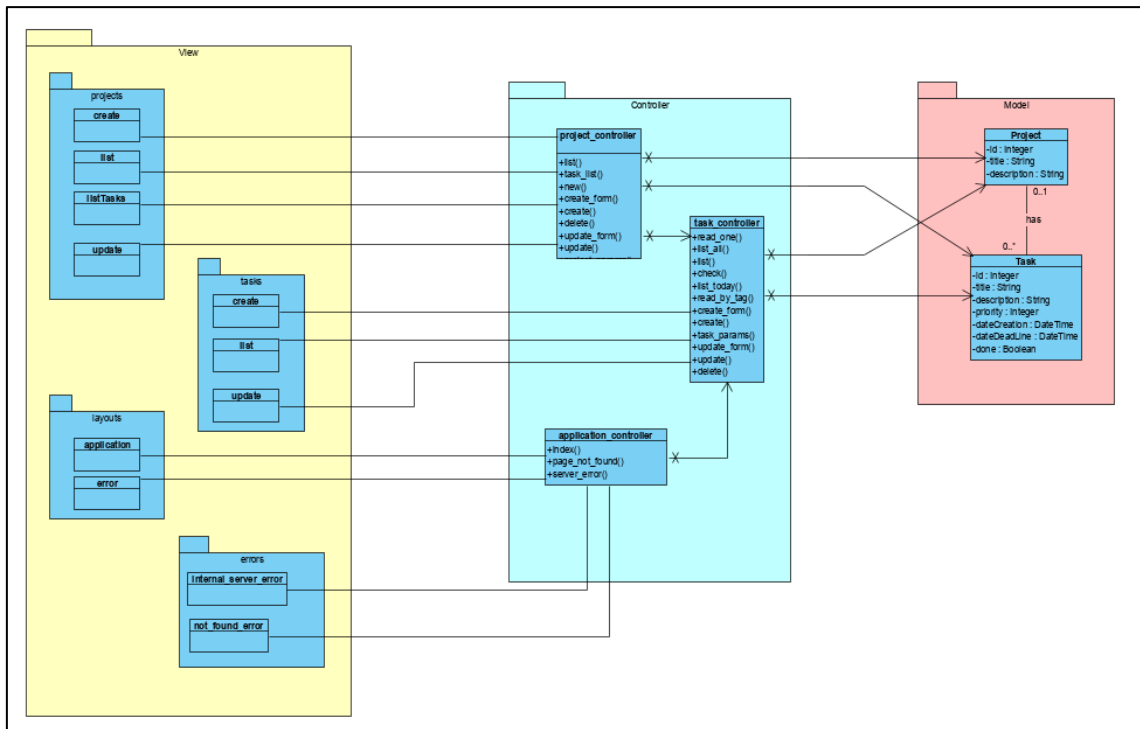
### 2.4. Contents model

---

The following figure shows our class diagram. We have built the web application following the MVC (Model - View - Controller) architectural pattern.

For this we have created two classes in the model layer, project, and task. For each one we have created its corresponding controller. Each one has the corresponding methods to manipulate the information that we have stored in the database, for example, to retrieve all the tasks of today, or to make a search of the tasks that contain a text string in its description.

To use these methods, we have created different views. We have tried to make as few views as possible so that the user is not switching between views all the time. We have also added views for 404 and 500 errors, so that the user has feedback in case of some errors on the web page.





### 3. Sprint 2 – Users management

---

#### 3.1. Introduction

---

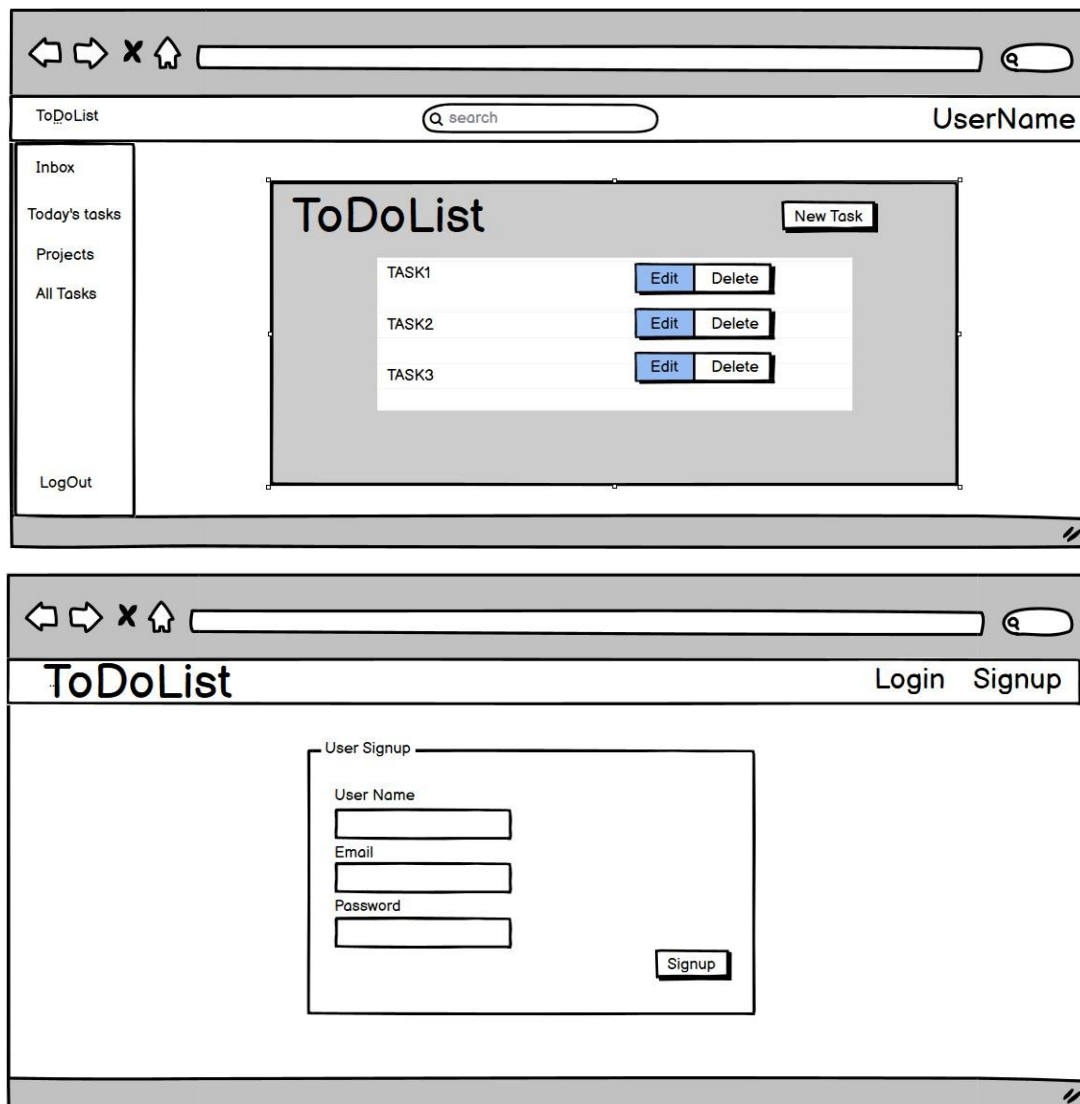
During the second sprint we have developed the functionalities related to the task management. We added two roles: normal and admin. The admin can manage all the tasks, projects, and users.

#### 3.2. Presentation

---

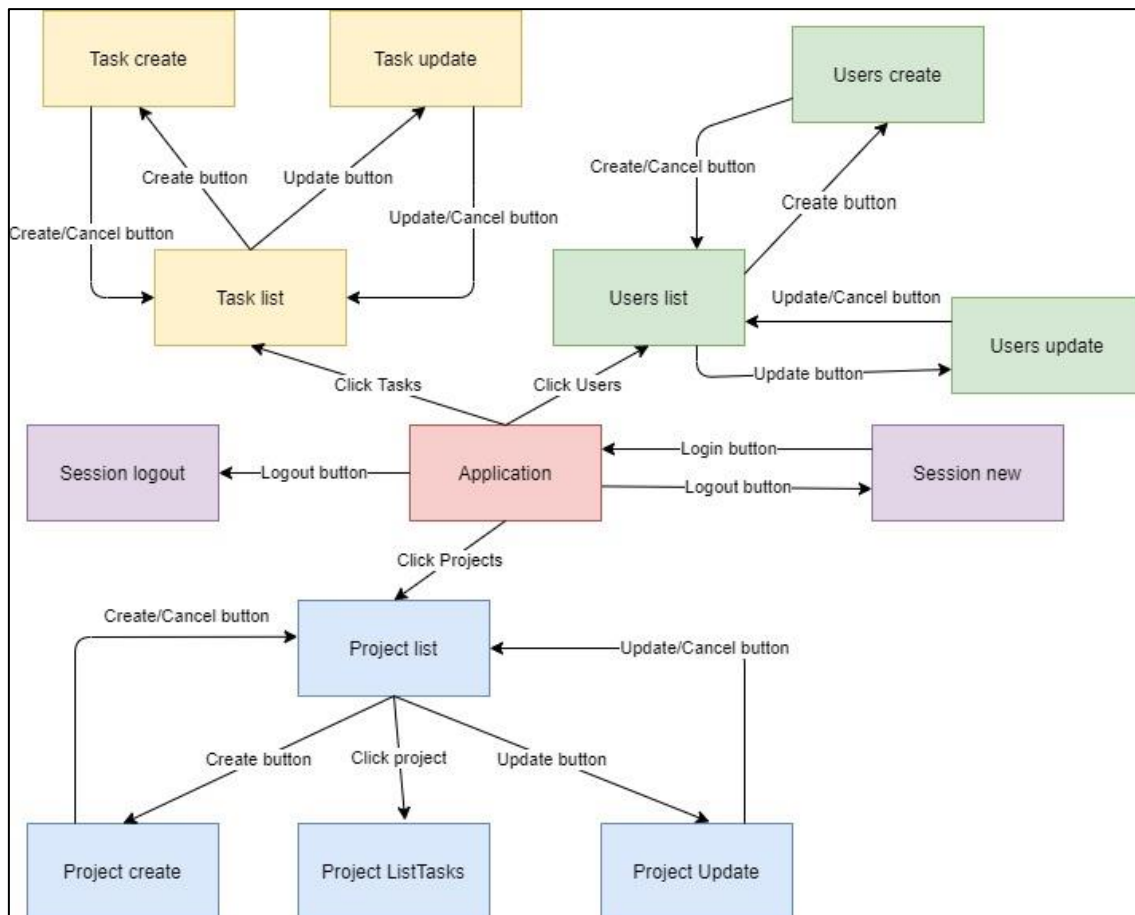
Below, we show the sketches we made as an orientation to the graphic design of the frontend of our web application. The sketches have been made using Balsamiq Mockups.





### 3.3. Navigation diagram

In the following figure we show the diagram of all the views of our web application. We can also see how to change between the views.

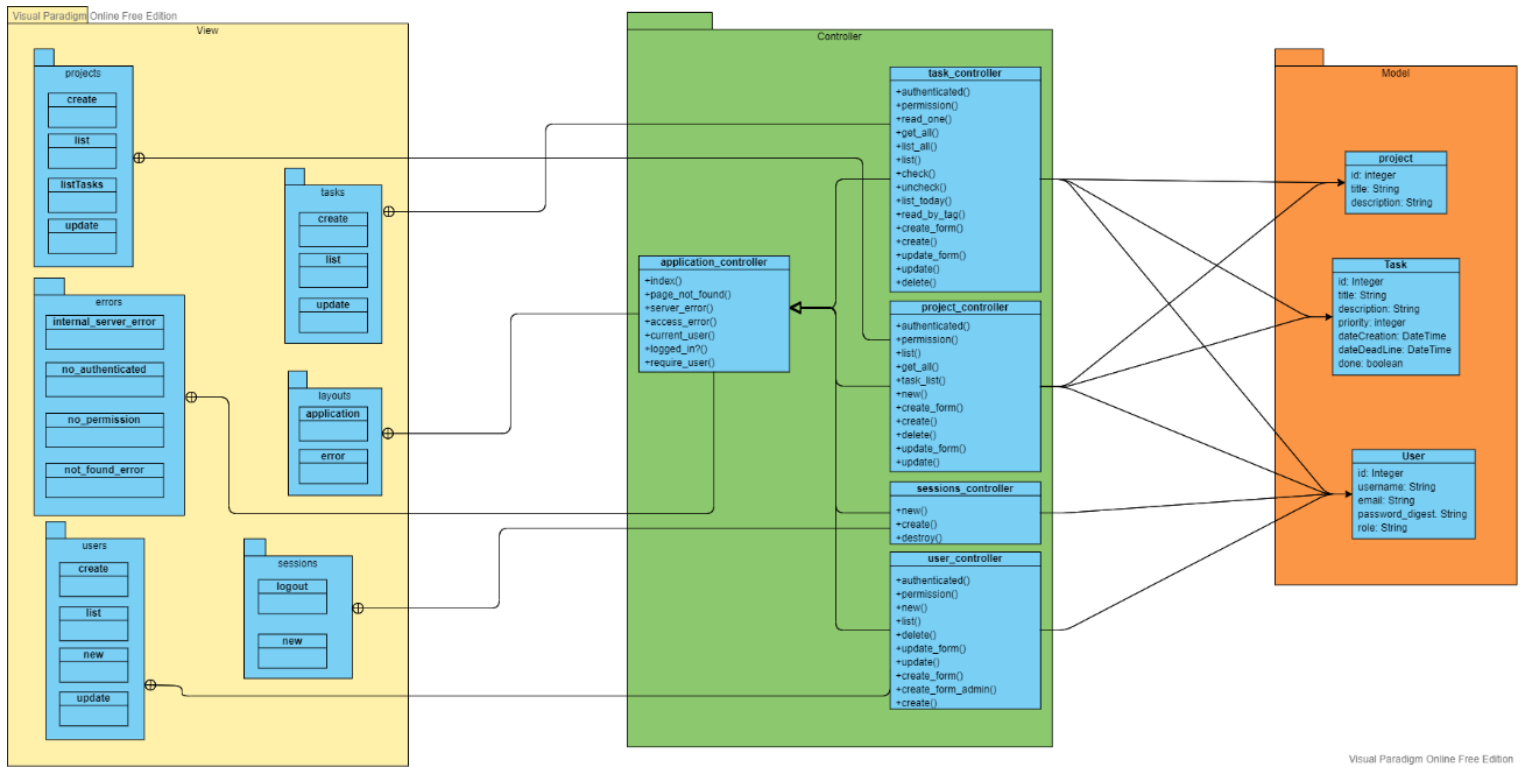


### 3.4. Contents model

---

The following figure shows our class diagram. We have built the web application following the MVC (Model - View - Controller) architectural pattern.

In this sprint, we add a user class to the model layer with the correspondent attributes. Also, we create a User\_controller in order to manage the user (create, delete and update methods). We created a session controller to access to the system, because we also added a login/register form to the system. We also created the related views to see all the functions implemented on the backend.



## 4. Sprint 3 – Tasks Collections sharing and Notifications.

---

### 4.1. Introduction

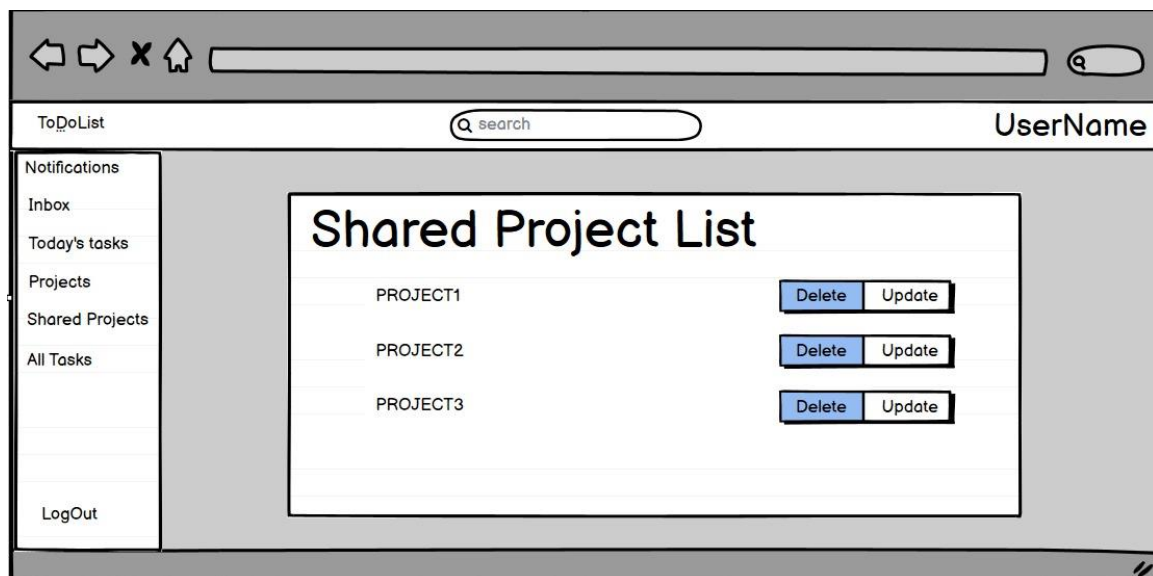
---

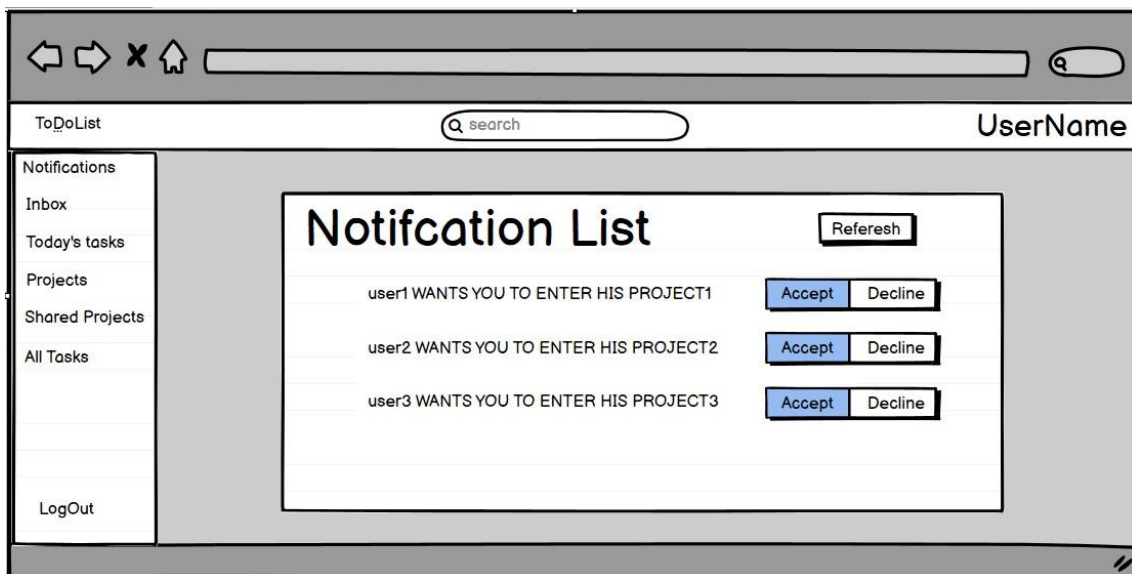
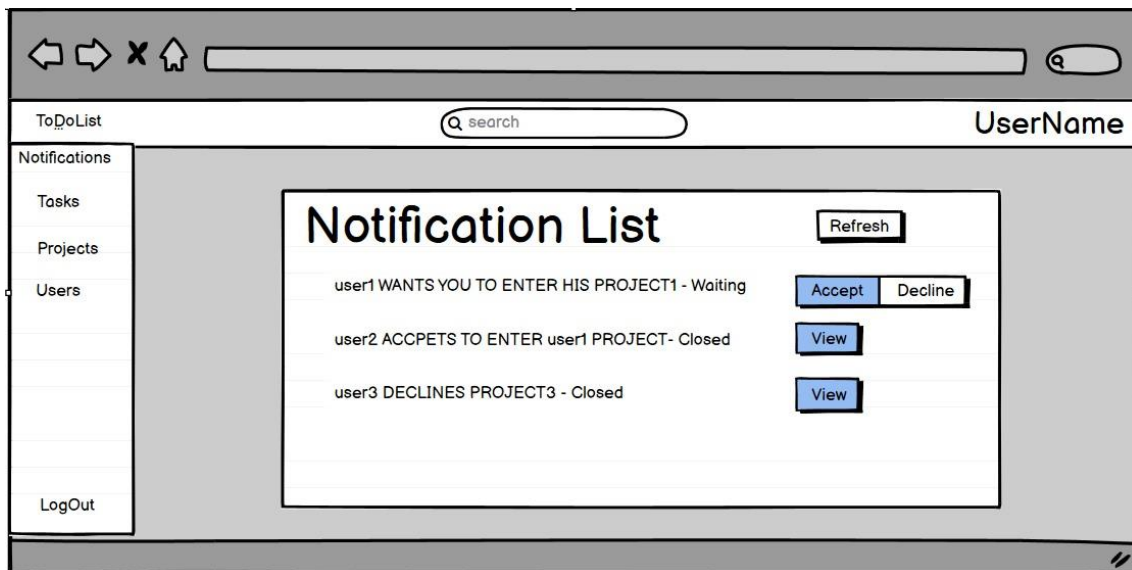
During the third sprint we have developed the functionalities related to the notifications. Now we can invite other users to collaborate in our projects, and it could be done due to the notifications we implemented.

### 4.2. Presentation

---

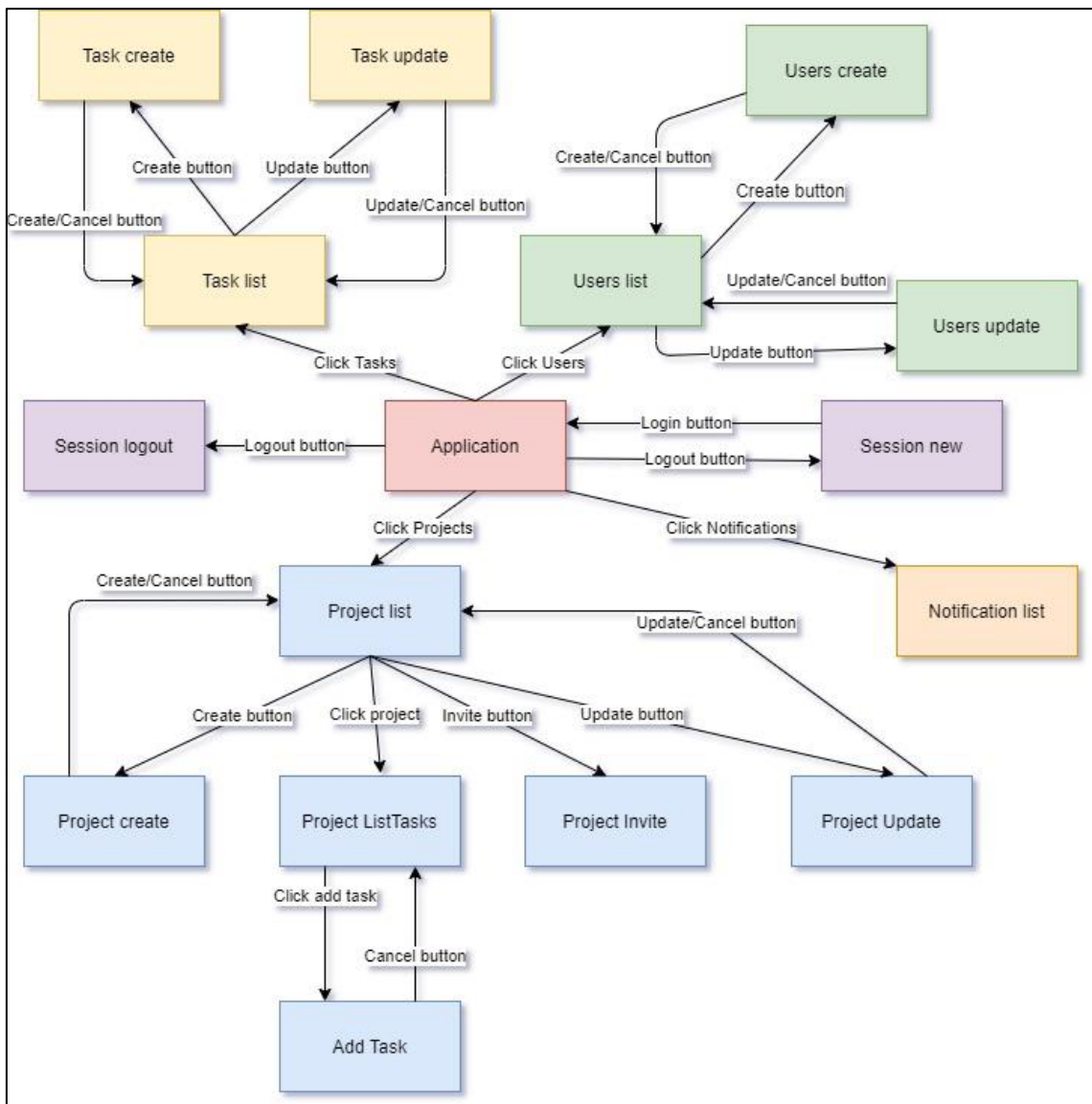
Below, we show the sketches we made as an orientation to the graphic design of the frontend of our web application. The sketches have been made using Balsamiq Mockups.





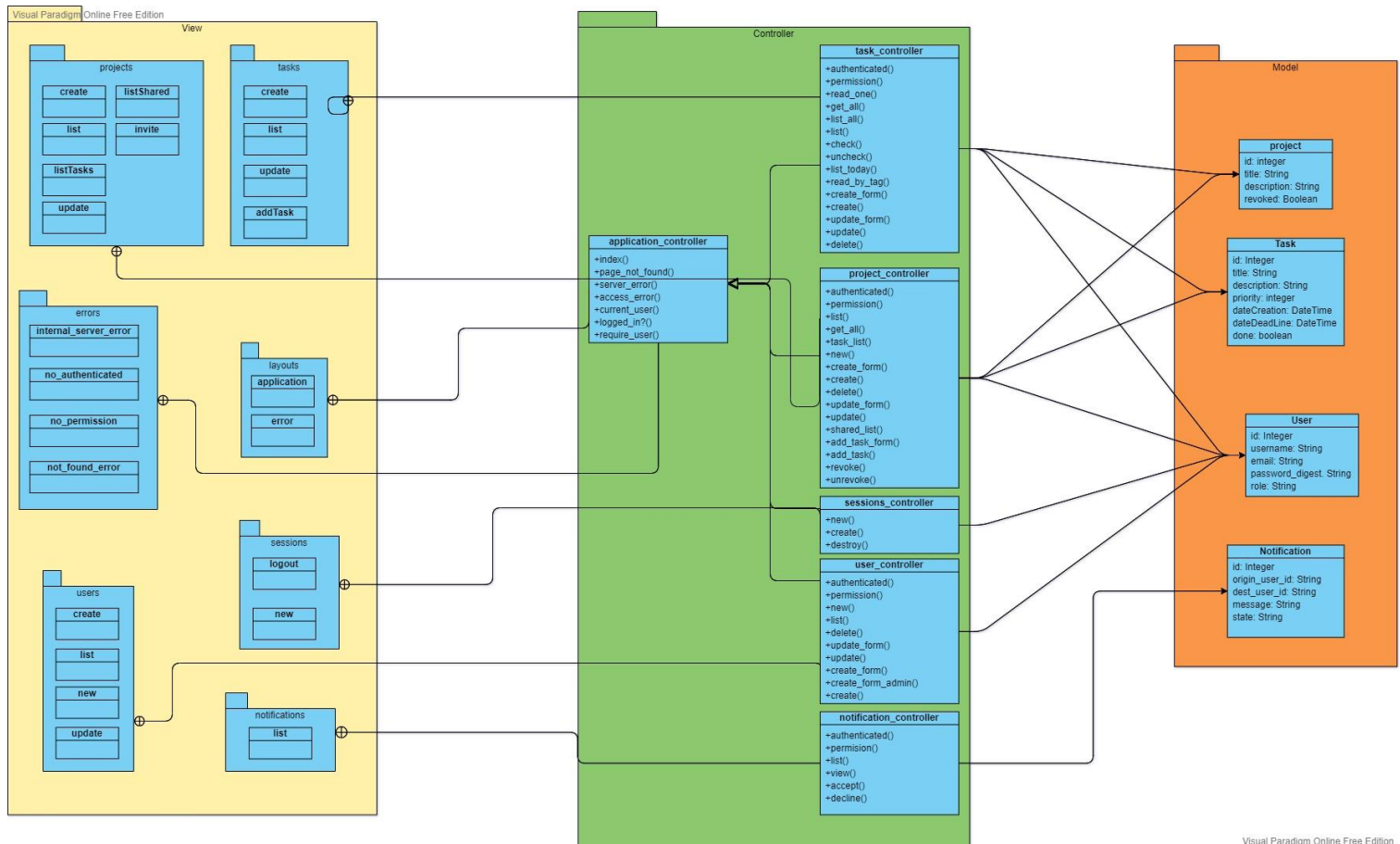
#### 4.3. Navigation diagram

In the following figure we show the diagram of all the views of our web application. We can also see how to change between the views.



#### 4.4. Contents model

The following figure shows our class diagram. We have built the web application following the MVC (Model - View - Controller) architectural pattern.



In this sprint we added a new controller called `notification_controller`. This controller will be in charge of managing the notifications: their listing, marking them as seen, accepted or rejected.

With these notifications we are able to know if a project has been shared with a user or not.

If we look at the model of the notification, we can see that among other attributes it has the id of the source user and the target user. In addition, a notification belongs to a project, so it will also incorporate, among other things, the id of the project we are interested in sharing.

If there is a notification in 'Accept' status, this means that the assigned project has been shared to the target user of the notification. This is how we can share projects.



Obviously we have also had to add those views corresponding to notifications and shared projects.

We have added a view that lists the shared projects that a user has. So there you can see the projects that other users have shared with you. From this list you can access different options, such as adding tasks to that project, deleting or updating it, or inviting new users or revoking access to more users from now on.

If you click on this last option, the invite button will be disabled and you will not be able to click it again unless you contact an administrator, because they have the option to undo the revoke option. If they do this, you will be able to re-invite, and therefore revoke access to the project again.

One improvement we have had to implement is to be able to add tasks from the project view. Before we could select a specific project and view its tasks, but we could not add them from there, so we had to add a new add task function to be able to add tasks from there in a more comfortable way.

This functionality can be found both in the list of own projects and in the list of shared projects.

Notifications can have different states: View, Accept, Decline, Close, Waiting.

When a notification is launched for the first time, it remains in waiting status until the receiving user decides to accept or decline it. Once it is marked, a new notification will be created in response to the original sender user. This response will have the status of Close, indicating that the invitation action is already closed, so now, in order not to occupy unnecessary space in your notification tray, you can mark it as View.

We have also added an attribute to the project in order to know if a project has been marked with the revoke option. A Boolean that, if false, indicates that the project is open to user invitations, but, if true, then the project is closed and no one else can be invited. Only the users who had access until the time of the revoke will be able to access the project.

## Appendix 1. User guide

---

During the first sprint, a small user guide was requested and is attached here. In this guide we only see what is related to project management, tasks and some more functionality.

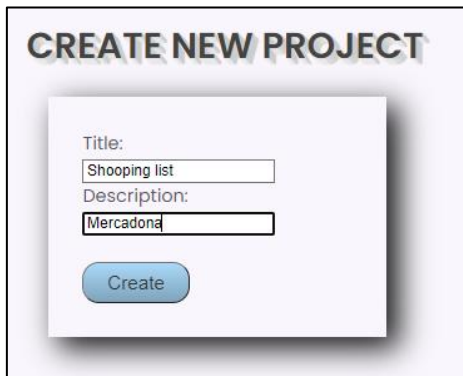
As we say, this was in the first sprint. There are many more features since then. However, this is the basics of the program. These functionalities explained here remain the same since then, leaving aside graphical improvements or data that the user of the program can ignore because they are code details.

### 1. Project management

---

In order to access to the project list, you have to click on the left menu on the Projects button. Then the list will be shown on the webpage.

Now you can do some functions. If you click in the title of one project, you will access to the list of the tasks assigned to this project. Also, if you move your mouse over the '...' a menu will be shown, in which you can choose between delete or update a method.



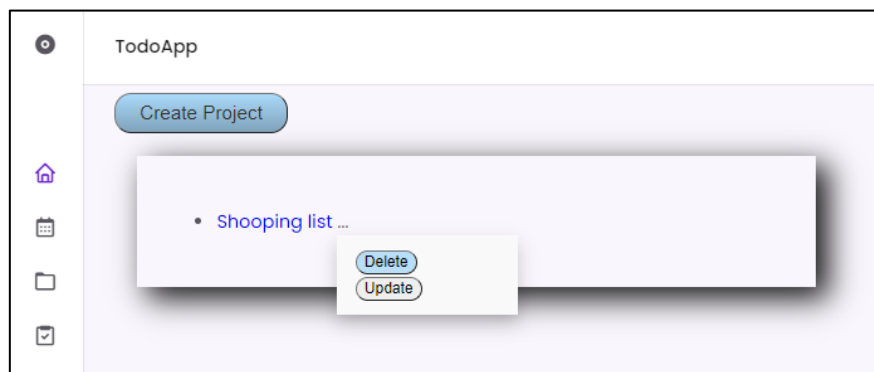
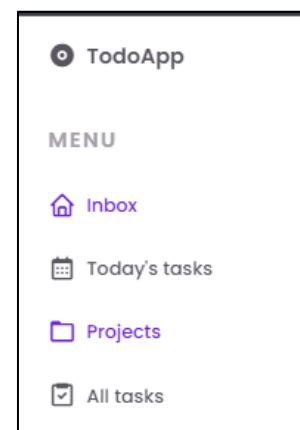
**CREATE NEW PROJECT**

Title:

Description:

*Important: If you delete a project, all the tasks assigned to this project will be deleted too.*

If you click on 'Create project' a form will be displayed. You can specify the title and the description of the project you want to create.



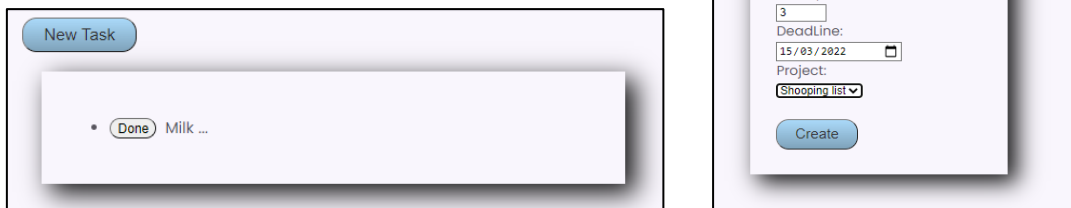
## 2. Task management

---

In a similar way that with the projects, you can click on 'Inbox' button to access to the list of all the tasks hasn't been done yet. The *inbox* view is the root path of the project; other views redirect to this.

On the task list view, you can create a new task. A form will be display in which you can specify the title, description, priority (1..4), deadline and project assigned of a task.

All the tasks have a *Done* button, unless not completed. If you click *Done* button, the tasks will be removed from the inbox list. You can see again this task on the *All tasks* view.



The left screenshot shows a 'New Task' button and a task list with a 'Done' button next to a task titled 'Milk ...'. The right screenshot shows the 'CREATE NEW TASK' form with fields for Title, Description, Priority, Deadline, and Project, and a 'Create' button.

Also, we provide an extra view *Today's tasks*, in which you can see a list of the tasks for today and a date before today. Only not completed will be shown.

## 3. Other functions

---

### Search by string

We added a little browser. So, the user can search tasks introducing a string. If a task contains the string on his title or description, it will be shown on a list. It only works for tasks.



The screenshot shows the 'TodoApp' header bar. It features the 'TodoApp' logo on the left, a search input field with the placeholder text 'Search' in the center, and a magnifying glass icon on the right.