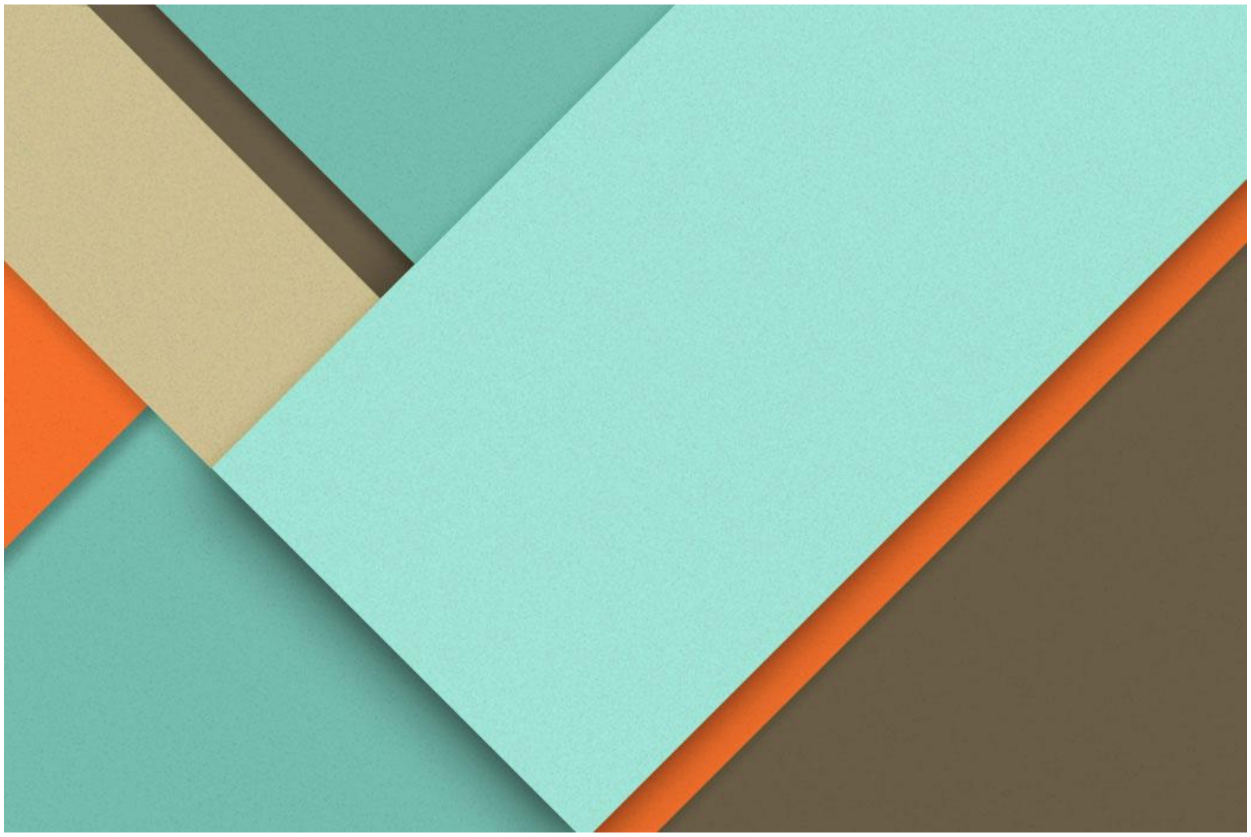


Junior dev technical test

14.01.2022

—

7Mojos
Casino games

Overview

This test aims to get an overview of your programming skills. We want to see how you solve programming issues and how you organize yourself. The goal is to create a todo list application using Typescript and PIXI graphic library. A todo is a reference note to remind you of something to do in the future. In this application you will be able to add, edit, delete and sort todos from the list. You will have to organize the application in a model view controller kind of architecture. If you don't know what this is, research it. Here's a good video to explain simply what it is. What is programming MVC? [Detailed Explanation] I will explain more in the specifications section. You can copy snippets of code you find in your research as long as you understand what it does and adapt it to your application. Don't copy the entire application from the web because we will test you on your implementation. Some requirements are bonuses, to be done if you have time.

Requirements

1. Use Typescript as the programming language.
2. Use PIXI for the graphic library.
   https://www.npmjs.com/package/pixi.js/v/5.0.0-rc
3. Use Greensock tween library for movement and graphic effect on the UI.
   https://www.npmjs.com/package/gsap
4. Target browser is Chrome desktop.
5. For event management system use PIXI.Signal.
   https://pixijs.download/release/docs/PIXI.Signal.html

Specifications

You will not only have application architectural constraints but also programming constraints. We want the application divided into a specific number of classes that will be part of your model view controller system. We want an **O**bject **O**riented **P**rogramming style in your implementation.

Classes description

We want the application to have no dependencies between systems. For example a button which is part of the view side of **MVC** does not require to have a direct reference to the controller part of the system. Systems should only communicate with each other via events (**PIXI.Signal**). When a button is clicked it only dispatches an event that other classes are listening to.

The list below is debatable and flexible. You can argue this model and we can debate it. You cannot remove classes but you can add as much as you need for the application to work.

- **Application**

  This class is the entry point of your application. It should extend **PIXI.Application**.

- **ToDoView**

  This view should display this information.

  - Date of creation.
  - Due date.
  - Title
  - Description text.
  - Priority. (color)

  It should have these interactions.

  - Selectable with mouse click.
  - When in edit mode all the texts are editable.

  It should have these states.

  - Enabled/Disabled
  - Displayed/Hidden
  - Edit mode
  - Overdue

- **ToDoListManager**

  This class is the **ToDoView** container. It should handle the scrolling logic of the list of todos. It should also handle the sorting, the add and the delete logic.

- **BaseButton**

  This class should be the base class of all your buttons. It should display a label text by default.

  It should have these states.

  - Enabled/Disabled
  - Mouse over
  - Mouse click
  - Idle

  It should have mouse interactions and it should send an event when clicked.

- **Model**

  The model is the representation of the data. It is where the controller gets the data to pass to the view to be displayed. The model is the bridge between the local storage data and the controller. It should not have any application logic. It only cares about retrieving data from the local storage and giving it to the controller.

- **Controller**

  This is where all the application UI logic resides. The controller listens to the UI and tells the todo list what to do. It also retrieves data from the model, processes it and passes it to the view.

Features requirements

A todo should display the following information.

- Title
- Description
- Creation date
- Due date
- Priority Should be displayed using colors. Green (Low), Yellow (Medium), Red (High). You should think of a way to change the priority. Radio buttons maybe.

The todos should be displayed in a vertical list. A scrollbar should be displayed if there are more todos to fit the viewable area. A todo can be added, modified, deleted and sorted. A todo can be selected with mouse interactions. It can be modified when in edit mode. When in edit mode all the texts are editable. When a todo is expired it is no longer editable, it can only be deleted. In this state it should be greyed out. Independently of the sorting mode a new todo should appear on top of the list. When saved, it will be sorted in the proper order.

When the application first starts there are no todo in the list. The only button active should be the **add** button. At the initialization of the list, no todos are selected by default. If you select a todo, the **edit** and **delete** button becomes enabled. When in edit mode clicking outside a currently selected todo will automatically save the todo new data to the model. Remember the model is not the data, it is a gateway to the data. If you delete a todo we must have a confirmation popup with the option to cancel or accept. The list is persistent so you will save the data in local storage.

You also have sorting options. You can sort them by title, creation date, due date and priority. You can also have these sorting options in ascending and descending order. By default they should be sorted by title in alphabetical ascending order. The sorting options are disabled if there are no todos in the list. When a sorting option is selected the list is automatically sorted.

Bonus feature

As a bonus if you have time you can add the following features.

- **Handle multiple users**

  You will have to set up a database of users in the local storage where you save the data. This means you would have a user authentication system to implement. You should display the current user name in the upper left corner of the application. You would also need extra popups for the login and the account creation. The logic of these popups should not be implemented in the popups classes. You should implement an account creation controller and a user login validation controller. You should also create a base popup class.

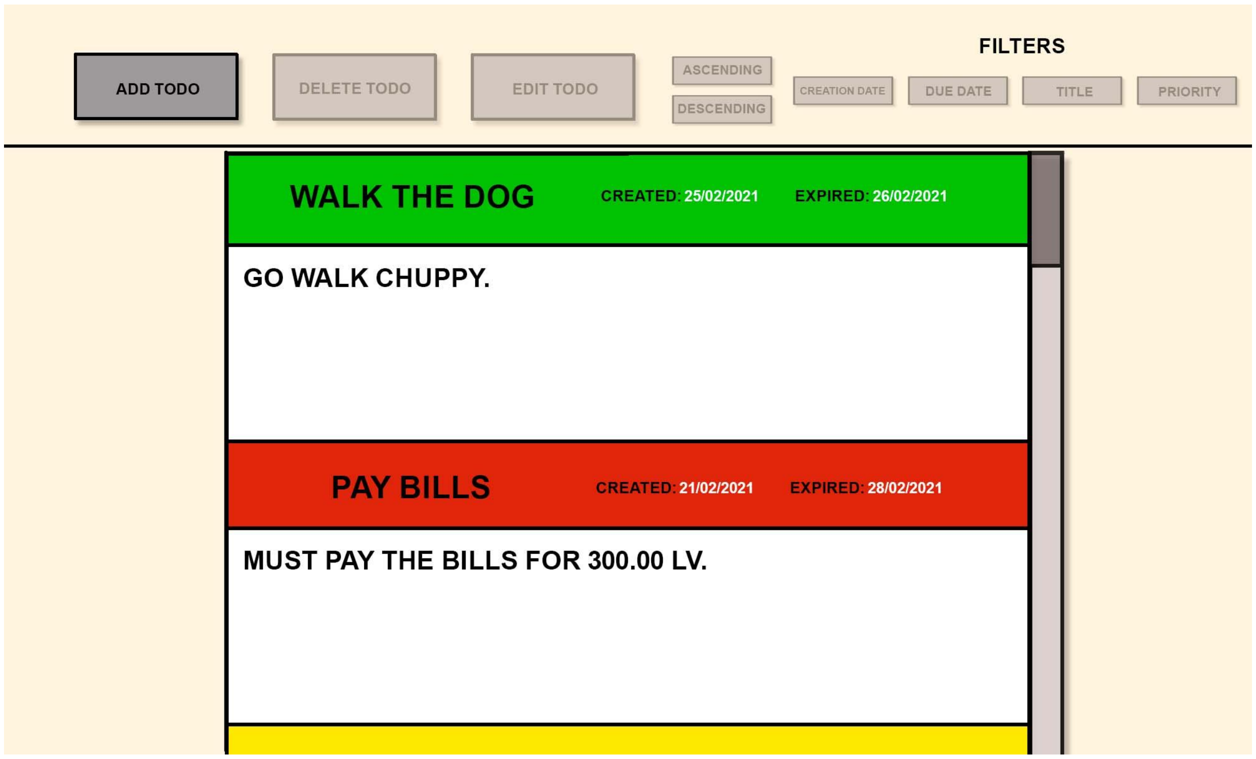
  - **Login popup**

    This popup should have an input text for the username, a login button, a cancel button, a create account button, a message text for login status.

  - **Account creation popup**

    This popup should have a username input text, a create account button and a cancel button. It should also display an account creation status message text.

Design mockup


ADD TODO
DELETE TODO
EDIT TODO
ASCENDING
DESCENDING
FILTERS
CREATION DATE
DUE DATE
TITLE
PRIORITY
WALK THE DOG
CREATED: 25/02/2021
EXPIRED: 26/02/2021
GO WALK CHUPPY.
PAY BILLS
CREATED: 21/02/2021
EXPIRED: 28/02/2021
MUST PAY THE BILLS FOR 300.00 LV.