APPRENTICE TECHNICAL LOG DOC

Breakable Toy – To Do App

Oscar Rene Garzon Castro

OVERVIEW

A To Do App website was implemented using Java as a back-end language and JavaScript for the front-end. Spring boot and React were used respectively. Java Collections was used for the persistence layer. Maven was used to build the back-end and npm for the front-end. The To Do App allows you to add, edit and delete To Do. Tasks can be sorted by priority or/and due date. Task can also be filtered by name, priority and done/undone. The app also shows metrics for the average time to finish a task.

The persistence layer is an array list that implements a Crud repository and methods to sort and filter tasks. The web controller was implemented using Spring boot methods. The front-end is composed of several React components that communicate between each other and make calls to the back-end Api.

CONTEXT

At the beginning of the project, I’ve never listened about layer architecture, persistence layer, domain, repository and service. I had to study and research all these concepts along the way among several others. I understood these are common design patterns that will help other developers to understand my code.

I had work in the front-end with very basic HTML, CSS and JavaScript. For this project I learned React and how to design a dynamic website.

[Repo to the project.](https://github.com/oscar-garzon/breakable_toy)

SOLUTION

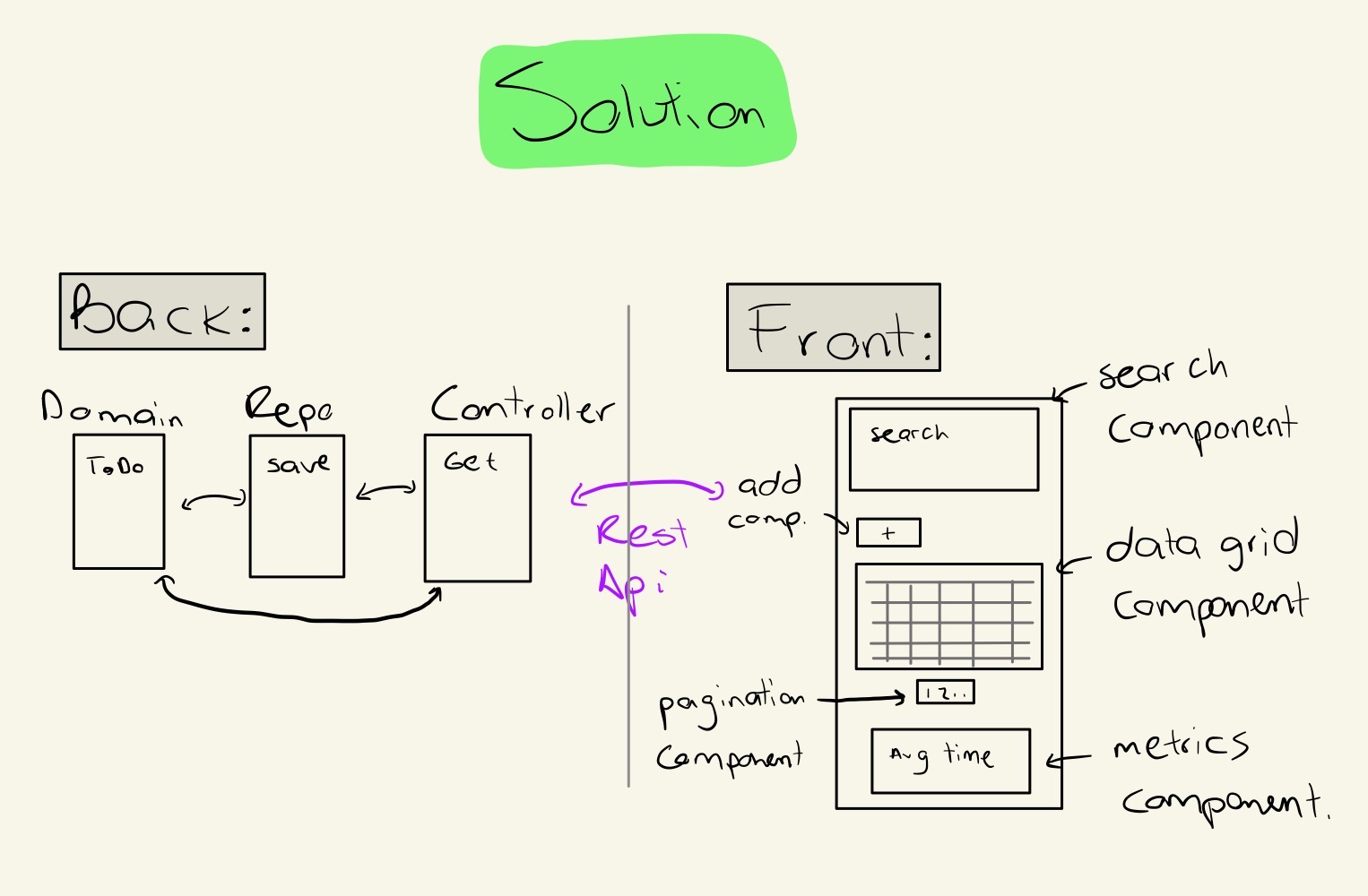
For the back-end we are going to use layered architecture. That means the app will be composed of different layers that talk to each other and each layer will implement one conceptual idea. Hence, we have three different layers:

* **Domain**: what a To Do task is and all its properties
* **Repository**: the way we are going to communicate with the persistence layer (No matter how the tasks are store we will communicate this way)
* **Controller**: this is how we expose the back end to the outer world. A Rest API will be used for this app

For the front-end we are going to use React. Before we move any further it is good to **refresh some concepts**. React works based on **components;** this is a good thing because in that way you don’t have to reload all the page just the part that needs to be reloaded. So, what is a component? It’s a part of a page that wraps only one function. For example, a navbar could be a component because it serves the function of navigating through the page by buttons or links. Technically a component is a function that you export and returns markup.   
How do you store information that you want to keep even after a component is rendered? If you use a normal variable, then React will forgot it after a render. This is when **states** come in handy. States allow you to keep information between different renders.   
How do you communicate with external systems? The **useEffect** function is designed for that. It takes two arguments, the first is the logic function, the second one is an array of dependencies. The first time the component associated to useEffect is rendered the logic function will be executed. Then if any of the dependencies change its value the logic function will run again.

The front-end will have the following components:

* **Search**: User can search tasks by name, priority and state
* **Add: I**t adds task to the list
* **DoneTodo:** Checkbox to mark a task as done or undone
* **Edit:** A modal that allows to edit tasks
* **ToDoList:** This is the main component. It will display ten tasks per page.
* **Pagination:** It has links to the different pages of the list
* **Metrics:**  It will show the average time to finish all task and the average time to finish tasks by priority



**Back-end implementation**

To model the ToDo we are going to create a ToDo class with attributes:

* Long id
* String text, priority
* LocalDateTime done\_date and creation\_date
* LocalDate due\_date
* Boolean done

It will have **two constructors**:

* Parameters: text, priority and due\_date
* Parameters: ToDo object

The second constructor is to return copies of the same ToDo but with a different reference. This is mainly because the repository will send copies of the saved ToDo’s.

**Repository**

The CRUD repository needs to connect with any persistence layer regardless of how the information is stored. For this we will have an interface with the following methods:

* void deleteAll
* void delete
* void deleteById
* boolean existsById
* List findAll
* Optional findById
* <S extends T>S save
* boolean saveAll

For this project it is a requirement to use Java Collections instead of a database. We consider two data structures for it: HashMap and ArrayList. The pros and cons discussion are described in the alternative solutions section.

We need another interface to define methods more specific for the To Do List. The interface will have the following methods:

* ArrayList<ToDo> filterBy(Specification spec )
* ArrayList<ToDo> findAll(Sort sort)
* HashMap<String, String> getMetrics()
* Page findAll(Pageable, Sort)

getMetrics returns a HashMap with the keys as the metric name and the value will be the metric value.

**Specification class** has a predicate method that returns true when the object satisfies the filter that the user is looking for and false in other cases.

**Sort class** handles the required information to sort the tasks. It has the following attributes:

* principalSortBy
* principalSortOrder
* secondarySortBy
* PrincipalSortOrder

Methods to get the attributes and:

* boolean hasPrincipalSort
* boolean hasSecondarySort

**Pageable class** receives two parameters Page and PageSize. Its function is to indicate the page where the front end is and the number of elements it is displaying.

**Page class** is an iterable that satisfies the condition in Pageable and Sort. It also has methods to the next and previous page, first and last page, and the total amount of elements in the whole ToDo list.

Page findAll(Pageable, Sort) was not implemented due to time constraints. The first two weeks of the project were spent learning the concepts of: layer architecture, domain, repository and service. Also, several Spring boot tutorials were taken during the first two weeks. The required methods for a repository interface are heavily based in Spring boot CrudRepository and JpaRepository. After the first two weeks of work only a basic skeleton was coded.

The idea to implement Page findAll(Pageable, Sort) is to return the chunk indicated in Pageable. That is Pageable.page \* 10 is the first element to return and Pagination.page \* 10 + Pageable.PageSize will be the last one. We are multiplying by 10 because each page has ten elements according to the project requirements.

**Controller**

A Rest API will be used to communicate with the front-end. We are going to use Spring boot web controller to handle the request. An essential part of a Rest Api is to provide links so that the user can navigate through the Api with links instead of asking each time for different elements. The Api will have the following endpoints:

Get: /todos to get all the todos. It has parameters to handle fiilters and sorting

Get: /todos{id} to get only one task

Get: /todos/metrics to return the metrics of the To Do list

Delete: /todos/{id} to delete an the ToDo with the specified id

Post: /todos to create a new task

Post: todos/{id}/done to mark a To Do as done

Post: todos/{id}/undone to mark a To Do as undone

Put: todos/{id} to edit a To Do

**Metrics**

This class is in charge of computing metrics. It’s an attribute of the ToDo Repository implementation.

**Testing**

Repository, ToDoRepository, ToDo and Metrics must have unit testing. Also, the Rest Api must have testing. Due to time constraints the tests for the Rest Api were not implemented. The planned way to implement them was using Mockito.

**Front-End implementation**

To implement the front-end, create-react-app was used. The intention at the beginning was to use a third-party data grid component because I believed it was going to be the hardest component to implement (at the end it wasn’t that hard). After asking the project mentor if this was valid, he said it was better to implement it myself because the purpose of this project is to learn and create engineers that can build their own frameworks and libraries. I estimated not finishing all the features of the projects on time was likely if I implemented the table myself, but I still choose to implement it myself because I consider learning as the most important part in my career right now and I think that’s also one of the Apprentice objectives.

**Pagination Component**

To implement this component an unordered list will be use, where each item in the list is a button with an event listener onClick that will make a Get request with the number associated to the button as the page and pageSize as 10 due to the project requirements. In the CSS part we are going to use a Flexbox to line the elements horizontally.

Due to time constraints this component was not implemented. The third week of the project was dedicated to learning react, Flexbox and Grid and the four week was used to actually code all the concepts and ideas from the first three weeks.

**Search Component**

The search component is going to be implemented using states, select and input attributes. It will make a GET request to the endpoint /todos with arguments indicating which kind of filter will be used. In the CSS side Grid is used to give the component the required style.

Again, this component was not implemented due to time constraints.

**ToDolist Component**

This is the basic feature of the To Do list. It displays the To Do’s and has buttons to edit and delete the task. This component has states to store the todos, boolean values to open/close the edit and delete modals. It also has functions that uses fetch to make the GET, POST, PUT and DELETE requests. It uses a table tag to render the To Do list. The table uses a map and JSX to render the tasks row by row

**Add Component**

Add component uses a modal to add the information related to a new ToDo. The modal has two buttons: save and cancel. If the user clicks save, then an event is fired, and the handling function calls the Api with a POST request. If the user clicks cancel, then the modal is closed without calling the back-end.

**Metrics Component**

This is a simple component that uses Flexbox to display the general average time to finish tasks and the average time by priority. Each time a task is marked as done/undone or deleted this component is re-rendered to display the new metrics.

**DoneTodo**

It has a checkbox where the user can mark a task as done/undone. The component has a state to keep record if the task is done or undone. It makes calls to the Api to mark the tasks as done/undone.

**Edit**

This component is very similar to Add component. It has a modal that shows the information on the task to edit. The user modifies the field to be updated. As with the Add component it has two buttons, save and cancel. Save calls the Api to update the task. Cancel closes the modal.

ALTERNATIVE SOLUTIONS

For this project it is a requirement to use Java Collections instead of a database. We consider two data structures for it: HashMap and ArrayList. HashMap has the ability to find objects in constant time if you save the id as a key and the ToDo as the value, the issue is when you need to iterate through all the ToDo’s. The ArrayList makes it easier to iterate through the elements, but you may need to iterate all the list to find an element. For this project we consider that the number of times we need to iterate through the list is greater than the number of times we need to find one element. Therefore, we are going to use an ArrayList. Another advantage of ArrayList is that to implement most of the methods in the Repository interface we are going to use the methods provided in ArrayList.