codeburst

Todo: Mental Model for building 80% of any web/business application for Beginner Programmers — Part 1

Rajesh Pillai 10 min read · Oct 11, 2019

157

Building a mental model and metadata-based approach for beginner programmers

Everything begins with a Todo Application (Twitter, Pinterest, Dribbble, Fiverr, Blog Engine, Reddit, Hackernews, eCommerce, etc).

Everyone is a beginner and everyone has to begin with something at some point in time and never underestimate the power of a Todo application.

> *This article will not contain any code, but only ideas that you can learn and apply.*
>
> *This is a live article and more use cases will be updated over a period of time.*

Our object is not only to understand building applications but how to take the learning and build a mental model to apply it to other projects/applications.

Also mastering the below items will help you build almost 80% of applications/projects out there.

- CRUD
- Image / File Upload
- Handling one-to-one relation
- Handling one-to-many relation
- Handling many-to-many relation
- User Authentication and authorization
- Navigation and Menus
- Autocomplete
- Grids and pagination
- Lazy loading of images and data
- Cookies, Local Storage and Session Storage (for web projects)
- Auth provider integration (Google, Facebook, Twitter, etc.)
- Payment Gateway Integration (PayPal, Stripe, etc.)

- DOM Manipulation (for web projects)

Don't take the Todo app lightly and ignore the naysayers. You won't believe what you can learn and build with a simple Todo app model.

This post is technology agnostic. Please feel free to use vanilla JavaScript, React, Angular, Vue, Svelte, etc for frontend and Node.js, .NET core, PHP, Ruby, Python, Go, etc. for backends.

For database free feel to choose any, PostgreSQL, MariaDB, MySQL, MongoDB, etc.

This model is helpful for building desktop apps as well.

> *Excuse my wireframes/mockups as I am not a professional designer.*

NOTE: The UML Diagram is only for representation. You can adjust the design as needed as long as the core functionality remains the same.

CAUTION

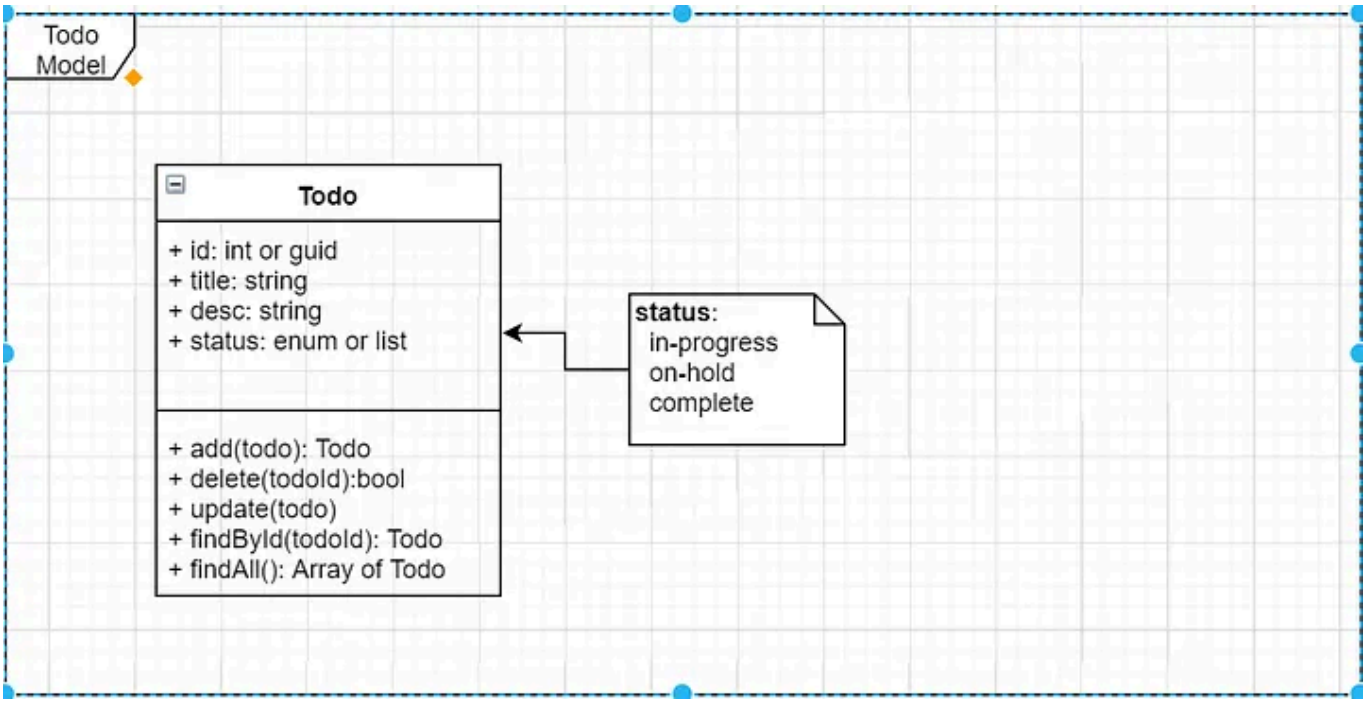
> *Also, do not to use any third party library or component when creating features. Try creating all components using the UI Library/Framework of your choice from scratch as that will make you better at programming complex features as well. Use only primary library or framework if you need to deal with UI like React, Vue, Svelte etc.*

The White Belt

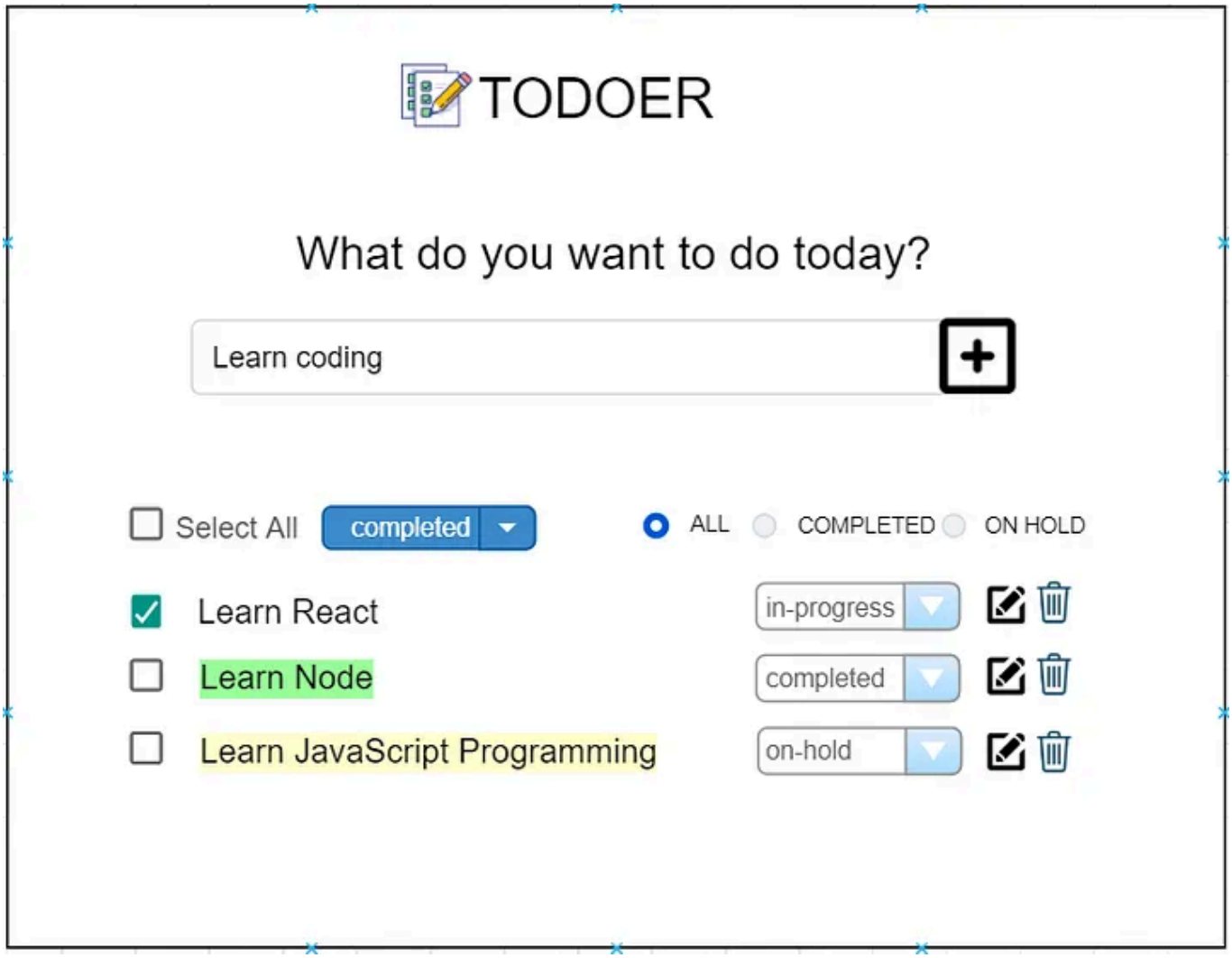
Level 0 — You are just beginning to code

Let's begin with a simple model. I will be showing the model attributes and the operations within the same model diagram. For implementation, you can keep the methods/API within the same model or create a separate API layer for the functions that work on the model.

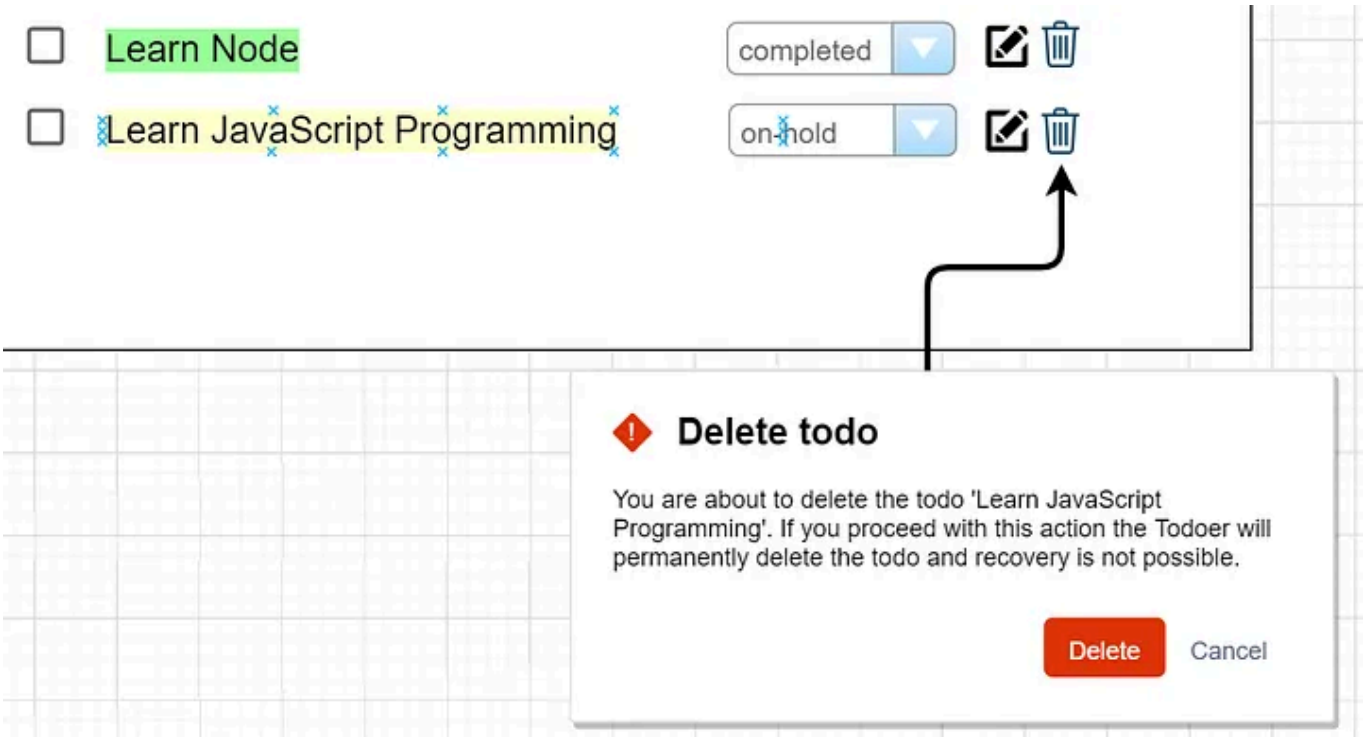
Our first model will look like the below figure.



Your task is to build something that is similar to the screen below.



On deleting a todo a confirmation like below should appear.



Feel free to play with the UI.

The following functionality needs to be implemented.

- Add todo
- Edit todo (on click of the edit icon, the corresponding label should change to a textbox and on the press of ENTER key submit the record for saving.
- Change todo status
- Delete todo (before deleting a confirmation dialog should be popped up)
- Select All (and implement the action listed in the dropdown)
- Filter Todos
- Color code list according to the status.

The lesson to be learned

- Setting up a simple CRUD application
- Understand the fundamentals of application design
- Interacting with services and database
- Showing a modal dialog

Level 0.1 — Dragging and Prioritizing the todos sequence

Implement dragging and rearranging todos in the table. For this feature make the needed model changes

- Introduce a “sequence” column in the todo’s model
- The todos should be listed in order of “sequence” in ascending order by default.
- As you drag and rearrange todos in the table the sequence should be correctly updated.

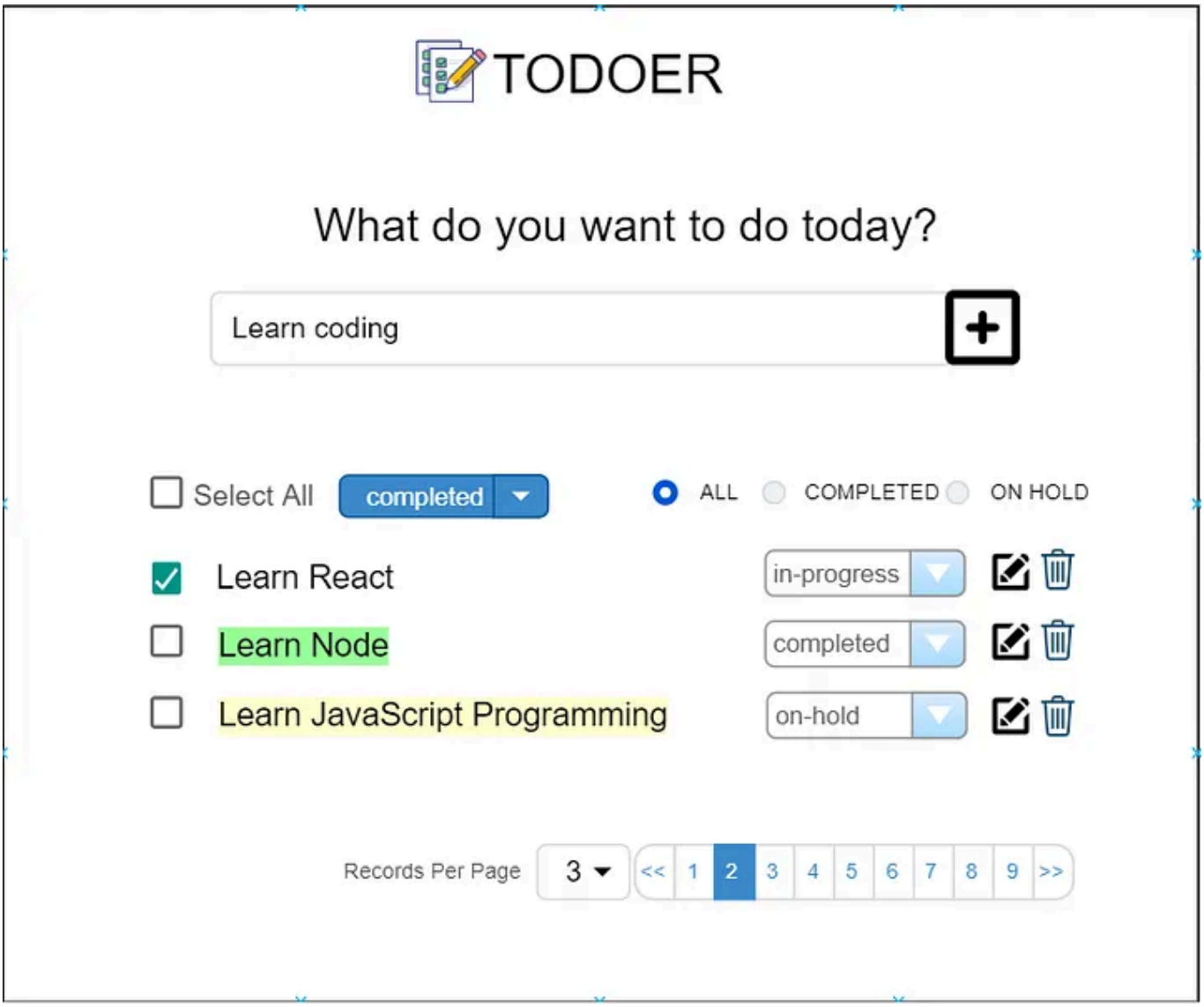
NOTE: Do not use any library for drag and drop. Use HTML5 events for the same.

The Yellow Belt

Level 1— Add Pagination Support

The model being the same and as your product is finding love in the community, the todo list is getting longer. Now to optimize the end user performance implement paging feature as shown below.

There are various ways in which pagination support can be added. For starters refer the below mockup.



The following functionality need to be implemented.

- Add a basic pagination
- Ability to select records/rows per page

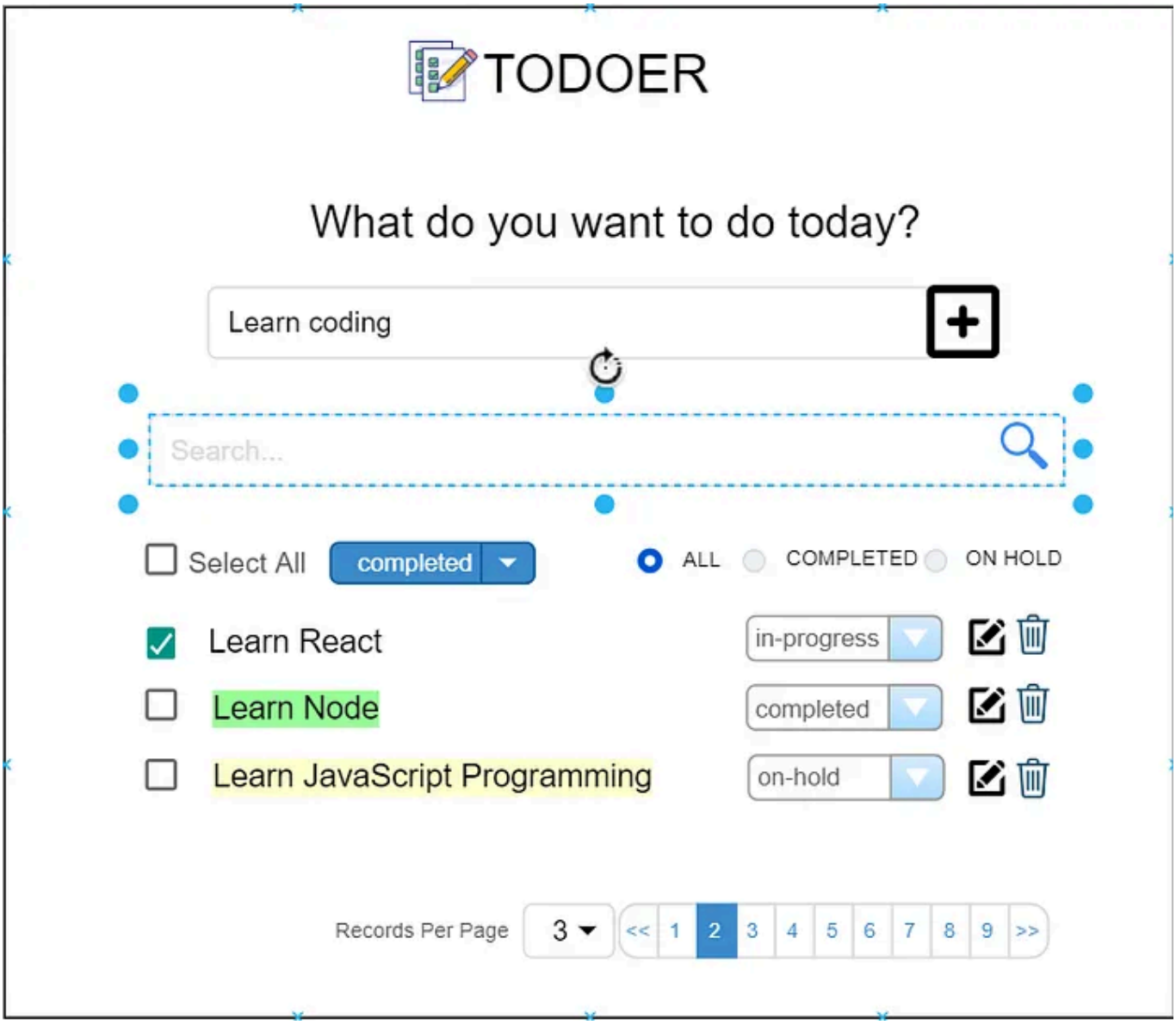
The lesson to be learned

- Understand how to code pagination both from the frontend and the backend perspective.
- Understand the importance of UX and performance.

Level 2— Add Search Feature

Our todos are getting huge and pagination is helping us with great user experience. But getting to a specific item is very difficult. That's where the search feature comes into play.

The model is the same as above, and let's take a look at the wireframe.



The following functionality needs to be implemented.

- On keying in the search term and pressing enter the results should be filtered.

- The pagination should be re-rendered accordingly to the search result.
- Implement caching

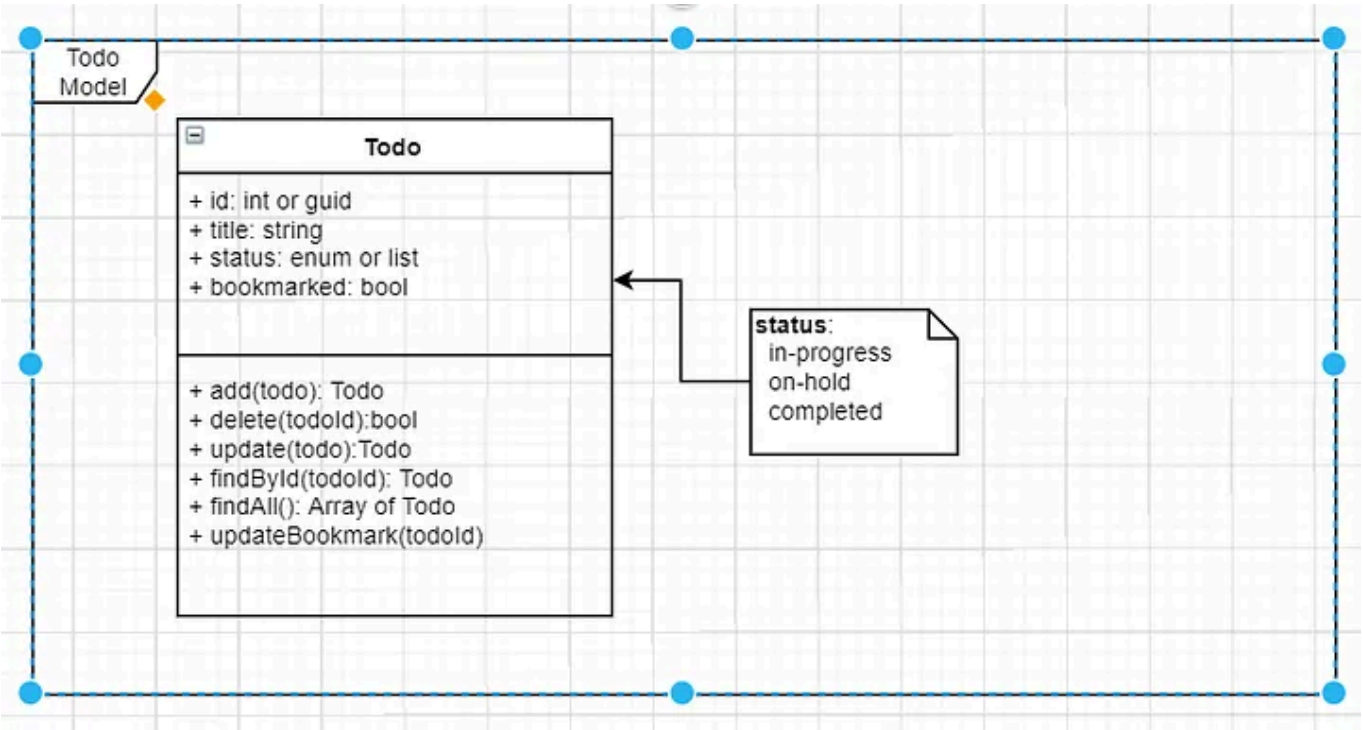
The lesson to be learned

- Implement one of the most common functionalities found in any website or application.
- Importance of caching, challenges and cache invalidation.

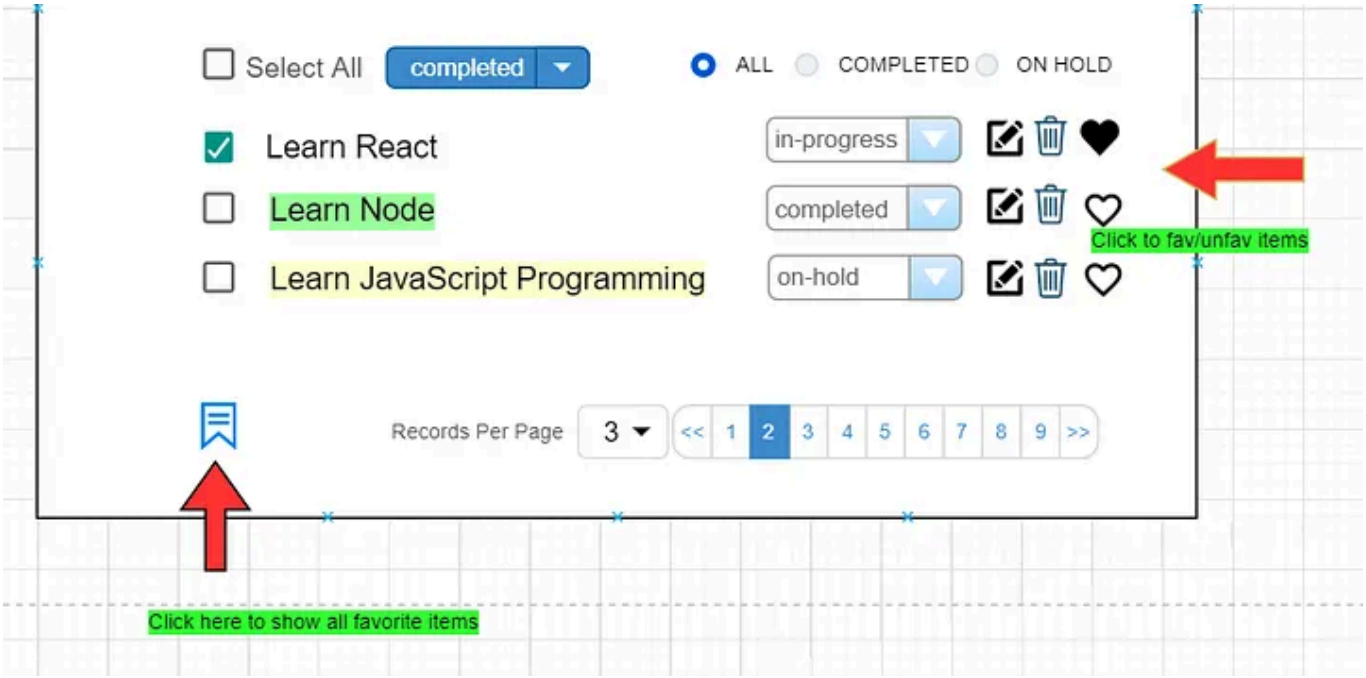
Level 3— Adding a favorite icon and animating it

Let's add a favorite icon (as it's everyone's favorite) and animate it on hover. We need to add a field in our model to store the favorites/bookmark items.

Let's take a look at the updated model. We have added a field called bookmarked which is bool (true /false) and also a method to update the bookmark (or you can also call it as the favorite).



Now, let's take a look at the wireframe (an only specific extract is shown).



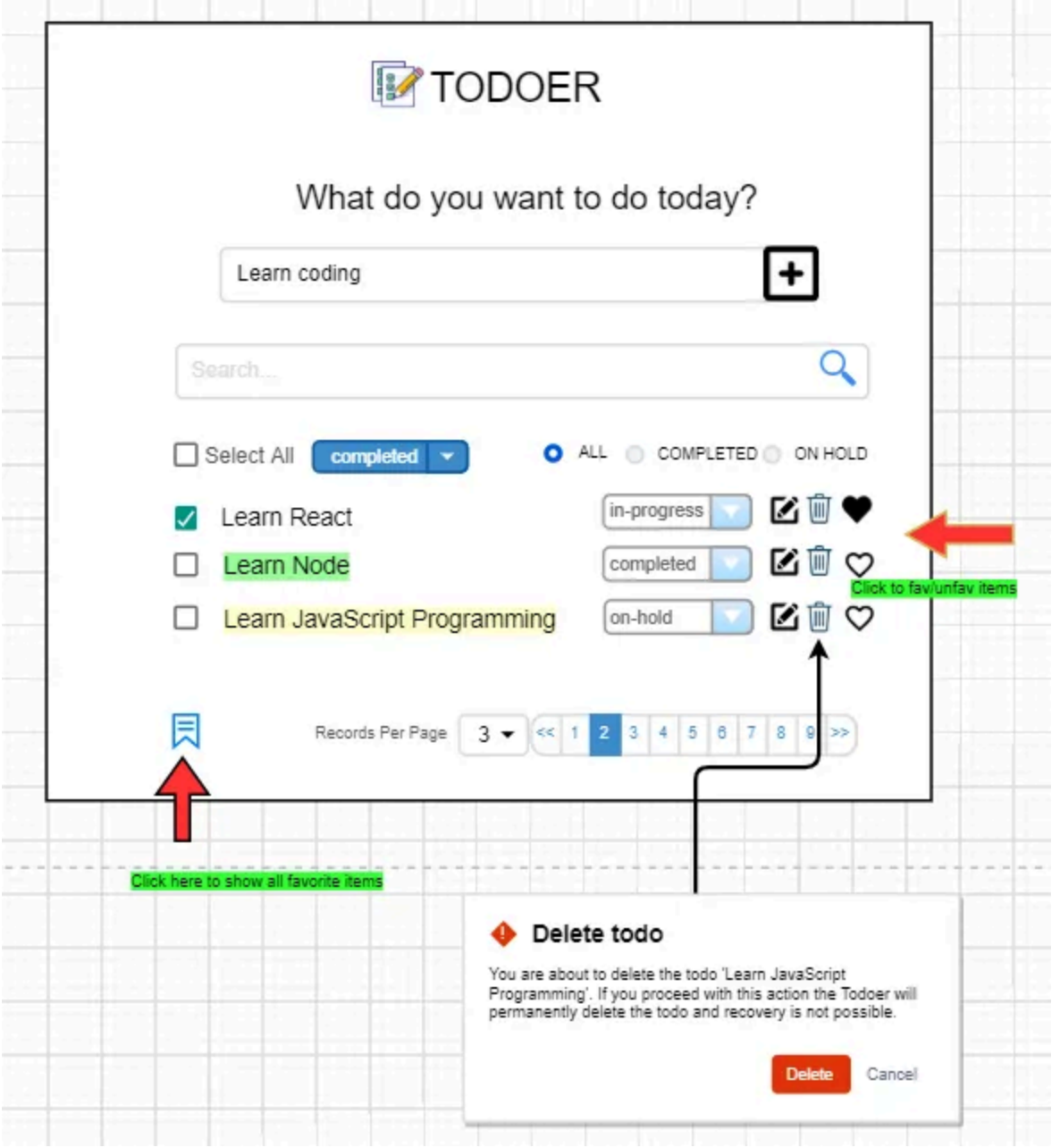
The following functionality needs to be implemented.

- On click/unclick of heart icon, the item should be added/removed from the favorite list.
- On click of the bookmark icon at the bottom of the screen, only the Favorited items should be listed.
- Animate when hovering over the heart icon ❤️

The lesson to be learned

- How CSS transition/ animation works

The completed UI so far should look like the below figure.

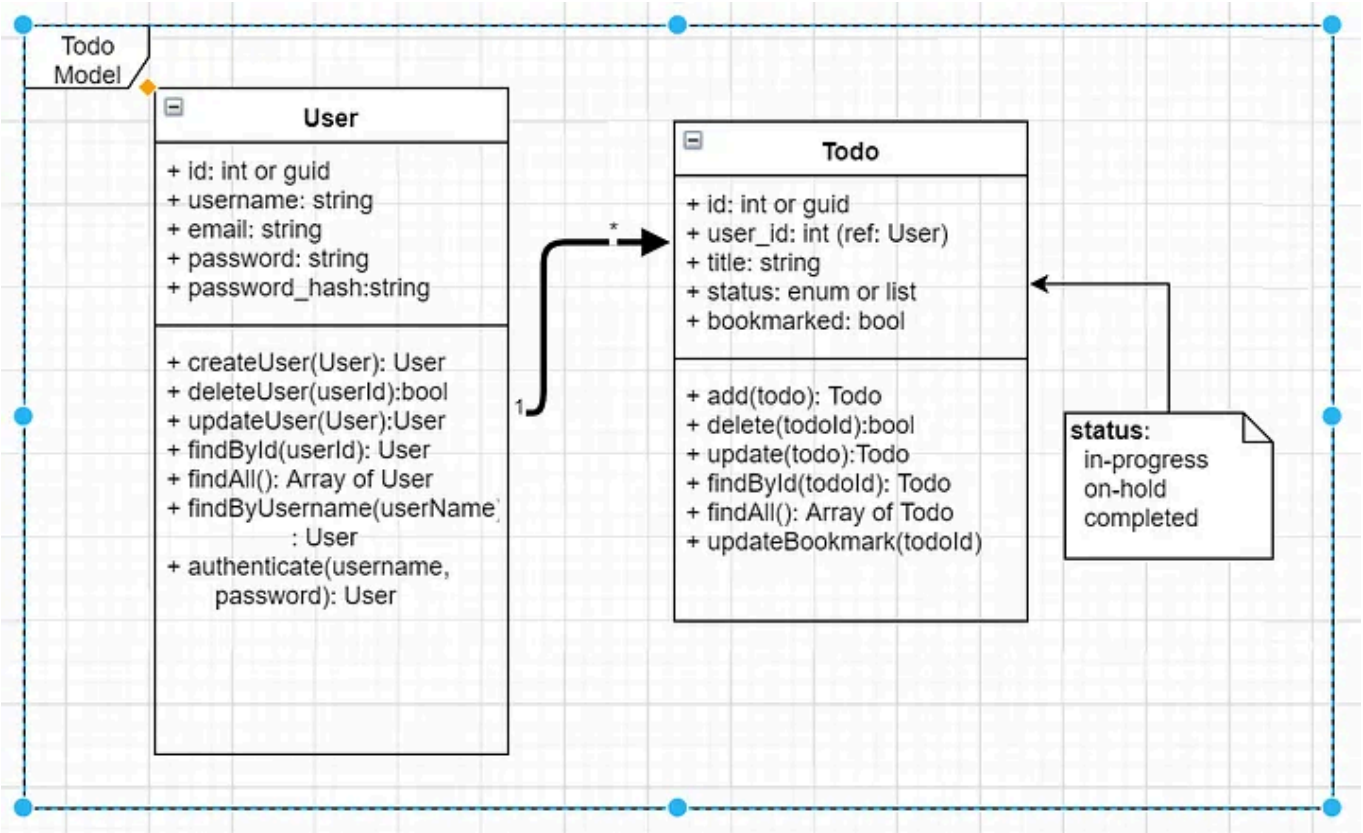


Congratulations and now you have built an app that works great for a single user. In the next levels, we will highlight the features to make our Todo app work for multiple users.

The Orange Belt

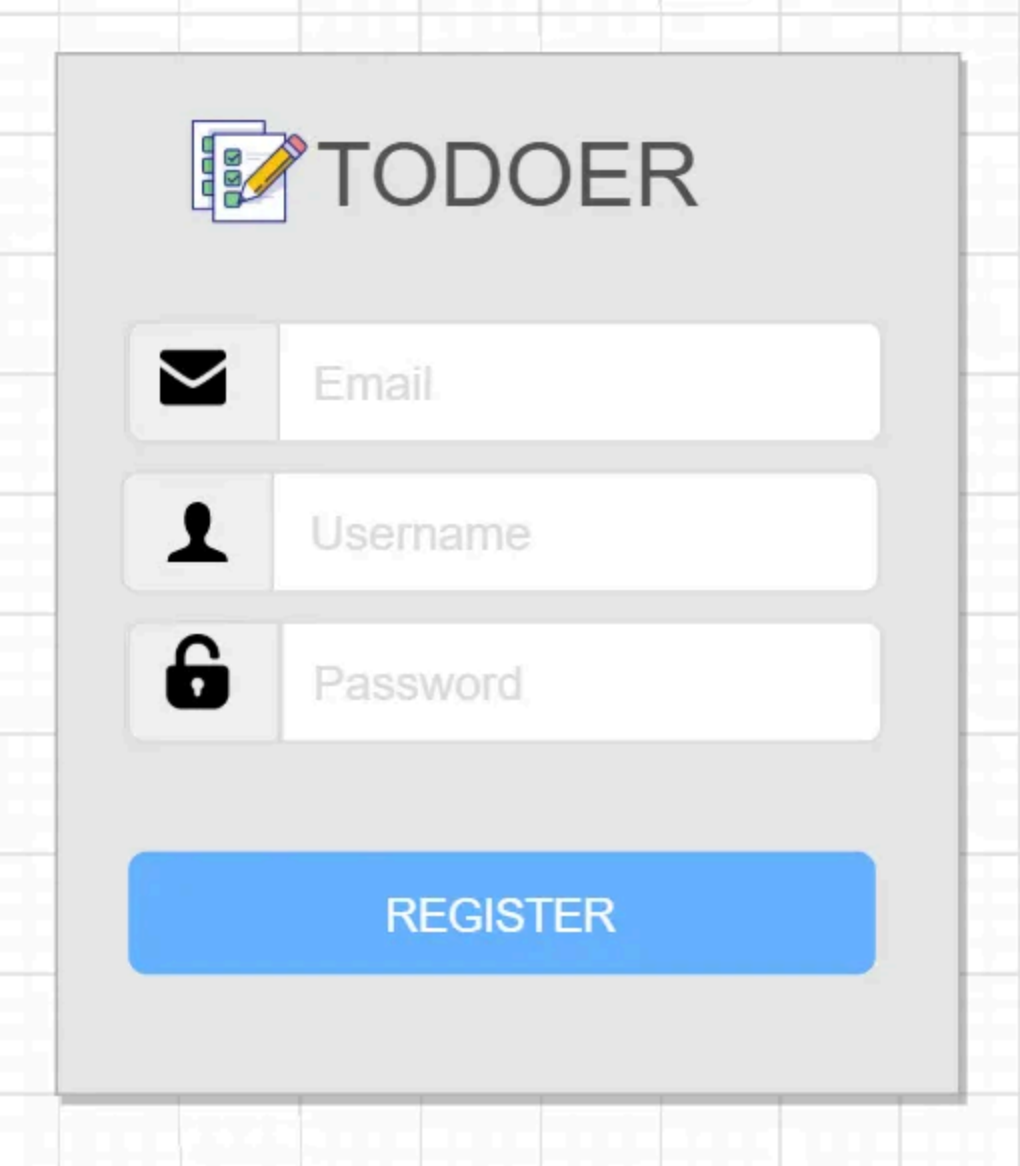
Level 4— Adding the register and login feature

Let's take a look at the model diagram.



So now, every Todo item is associated with a user model, i.e. there is a one to many associations between User and Todo.

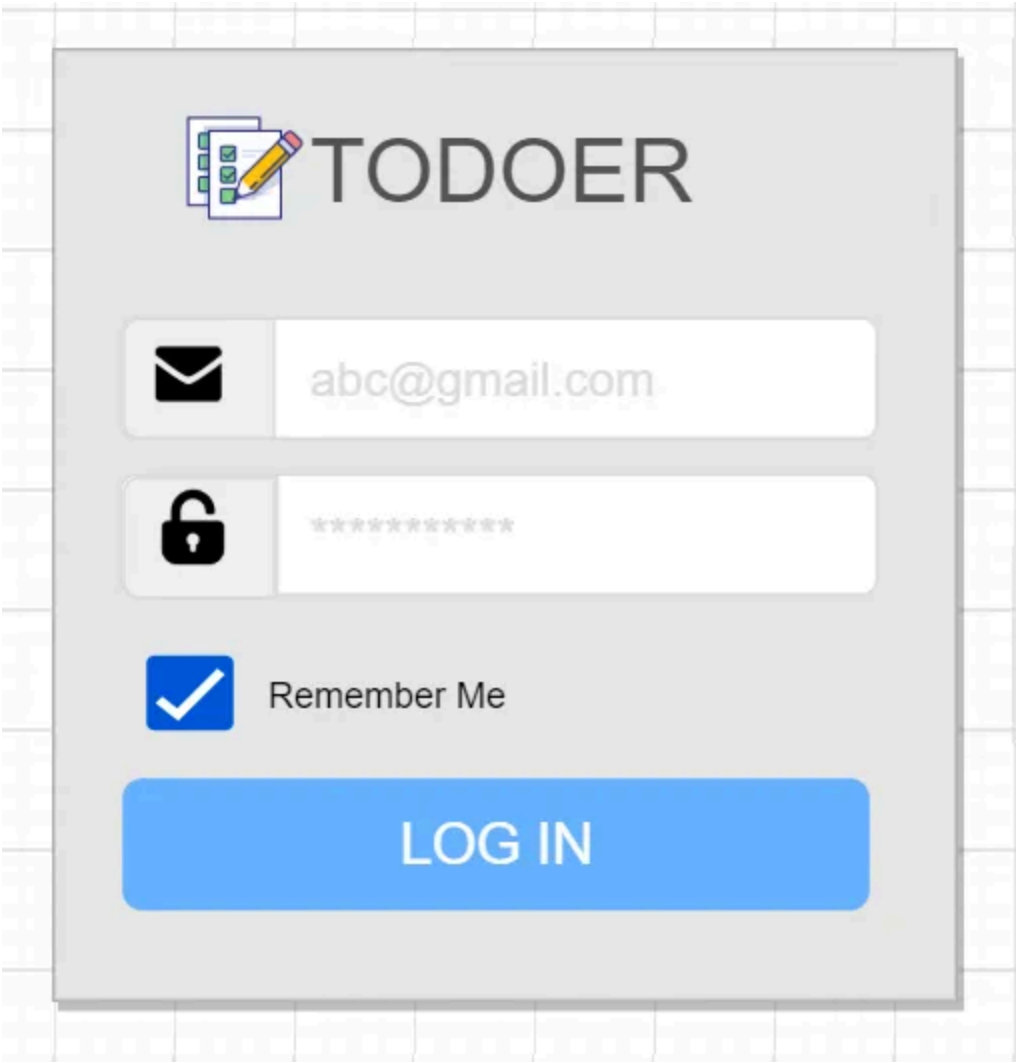
Let’s create a user registration screen.



The following functionality needs to be implemented.

- On click on “REGISTER”, the user details should be stored in the database.
- The password should be hashed and stored in the database.
- If the user already exists with the same email/username, then an error message should be displayed to the client.
- If the user is successfully created, the client should be redirected to the login page.

And the login screen

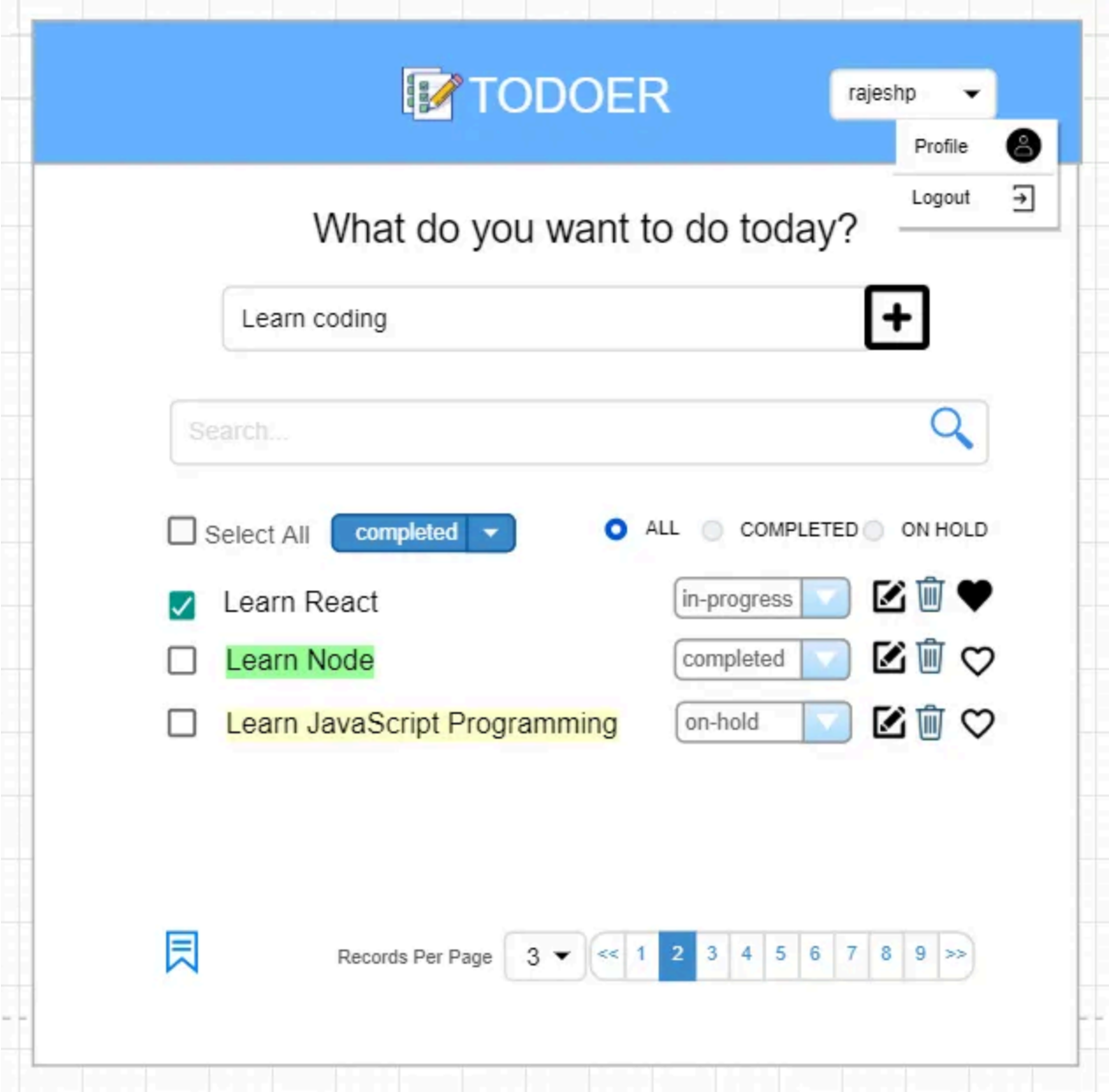


The following functionality needs to be implemented.

- On click of “LOG IN”, the email/password should be validated.
- If the user is successfully validated, then a token should be sent from the server to the client.
- This token will be used to authorize the user for subsequent requests.
- The client should be redirected to the home page (where his list of todos are displayed)

- The user profile name should be displayed on the screen.
- A dropdown menu should be displayed with “logout” and “profile” option.

The new home page once the user successfully login should be as per the wireframe.



The lesson to be learned

- Local authentication
- Implementing routes and navigation
- Understanding session management
- Hashing
- Redirection to page
- Token-based authentication

Level 5— Logout feature

Refer to the above wireframe. When clicked on the logout option, the users session data should be cleared, (any cleanup that is required, needs to be done) and the user should be redirected to the login page.

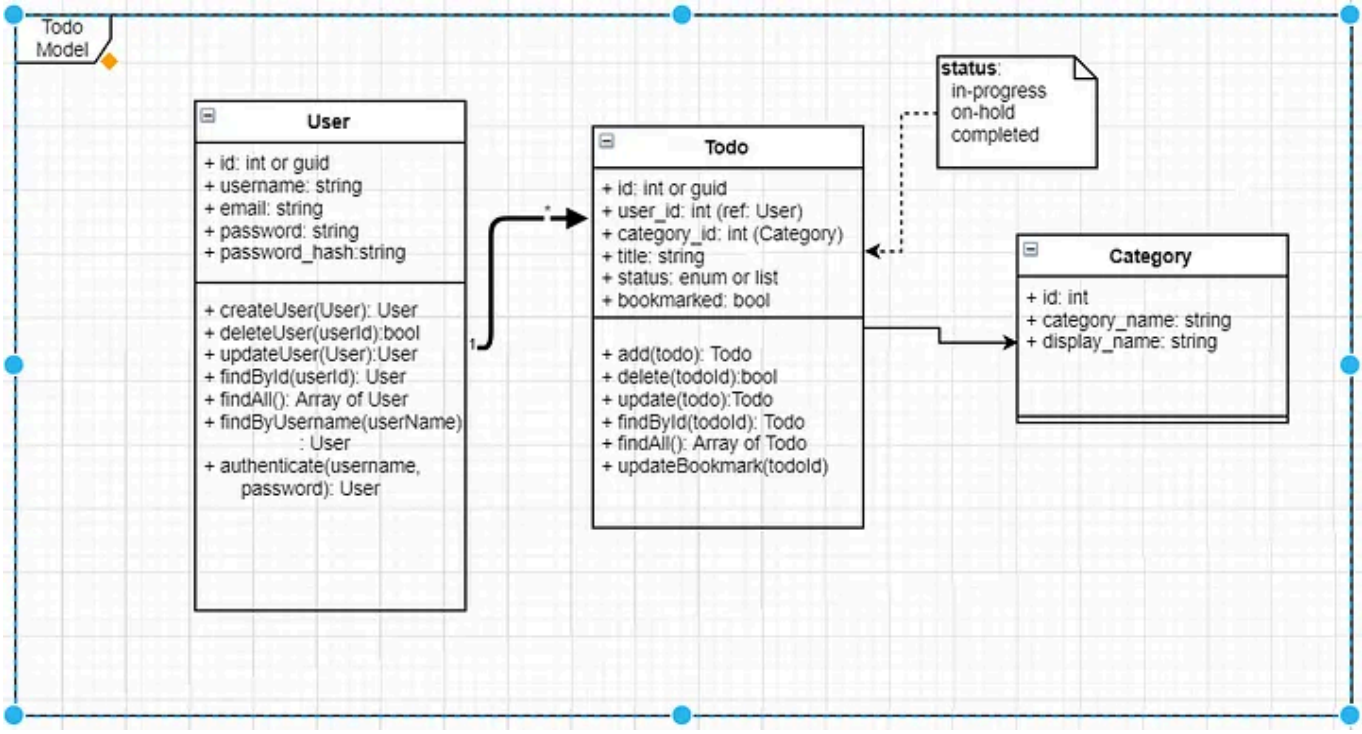
By implementing the above features, now every signed in user gets to manage his/her todo list.

The Green Belt

Let's implement one-to-many associations. Now, it may happen that our one todo item can be subdivided into multiple steps. So, let's model one-to-many associations between a todo item and it's subtasks.

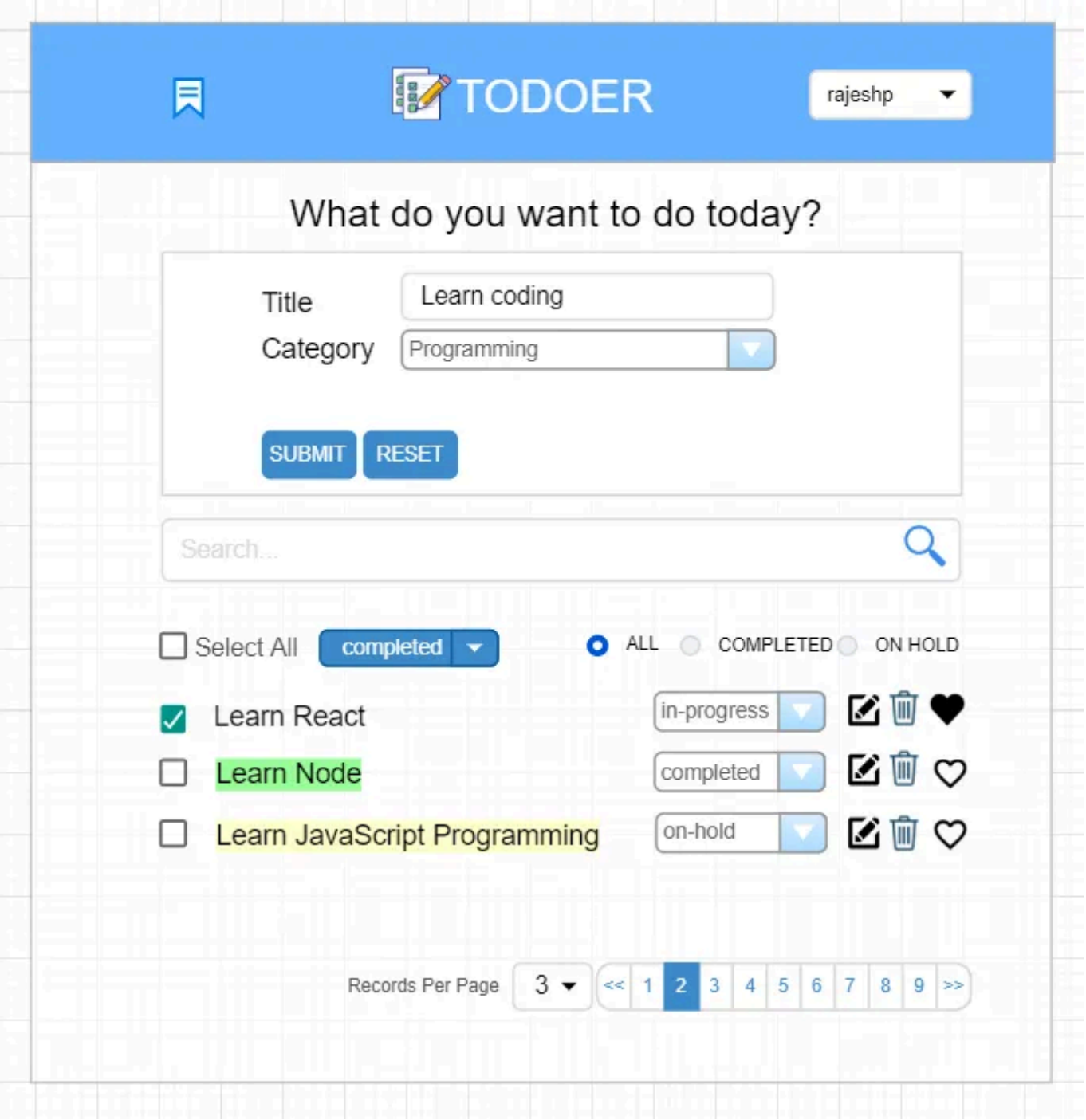
Level 6— One to one association

Let's add a category to every todo. The model looks like the figure below.



NOTE: The Todo model has a reference to the Category Model.

The wireframe can be implemented as shown below. I have slightly changed the UI to accommodate the new field.



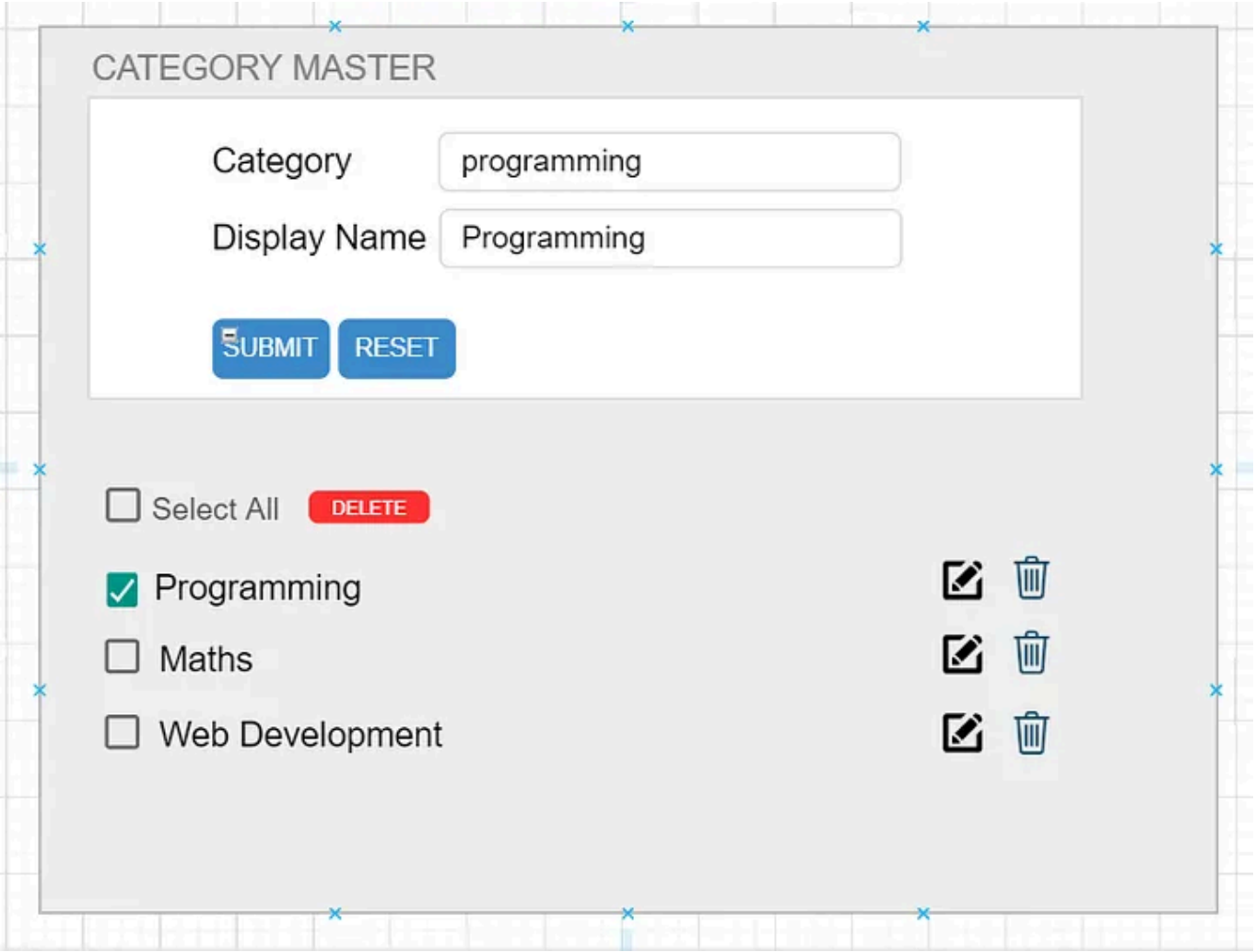
The following functionality needs to be implemented.

- The category field should be populated from the category table in the database (make an ajax call). The display_name should be populated in the dropdown list.

- On saving the todo, the category_id should be saved along with the todo model.

Level 6.1— Building the Category Form (CRUD)

Here is a quick wireframe.

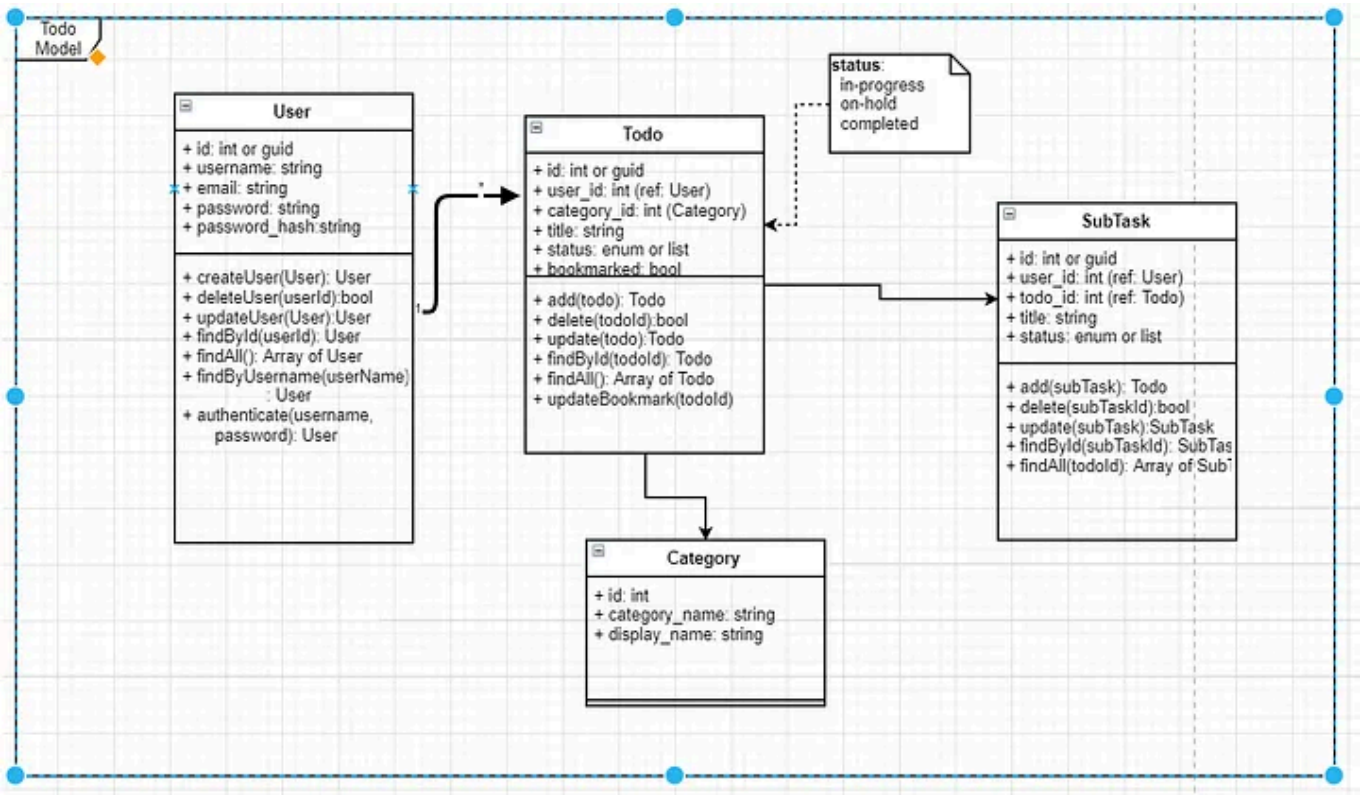


The following functionality needs to be implemented.

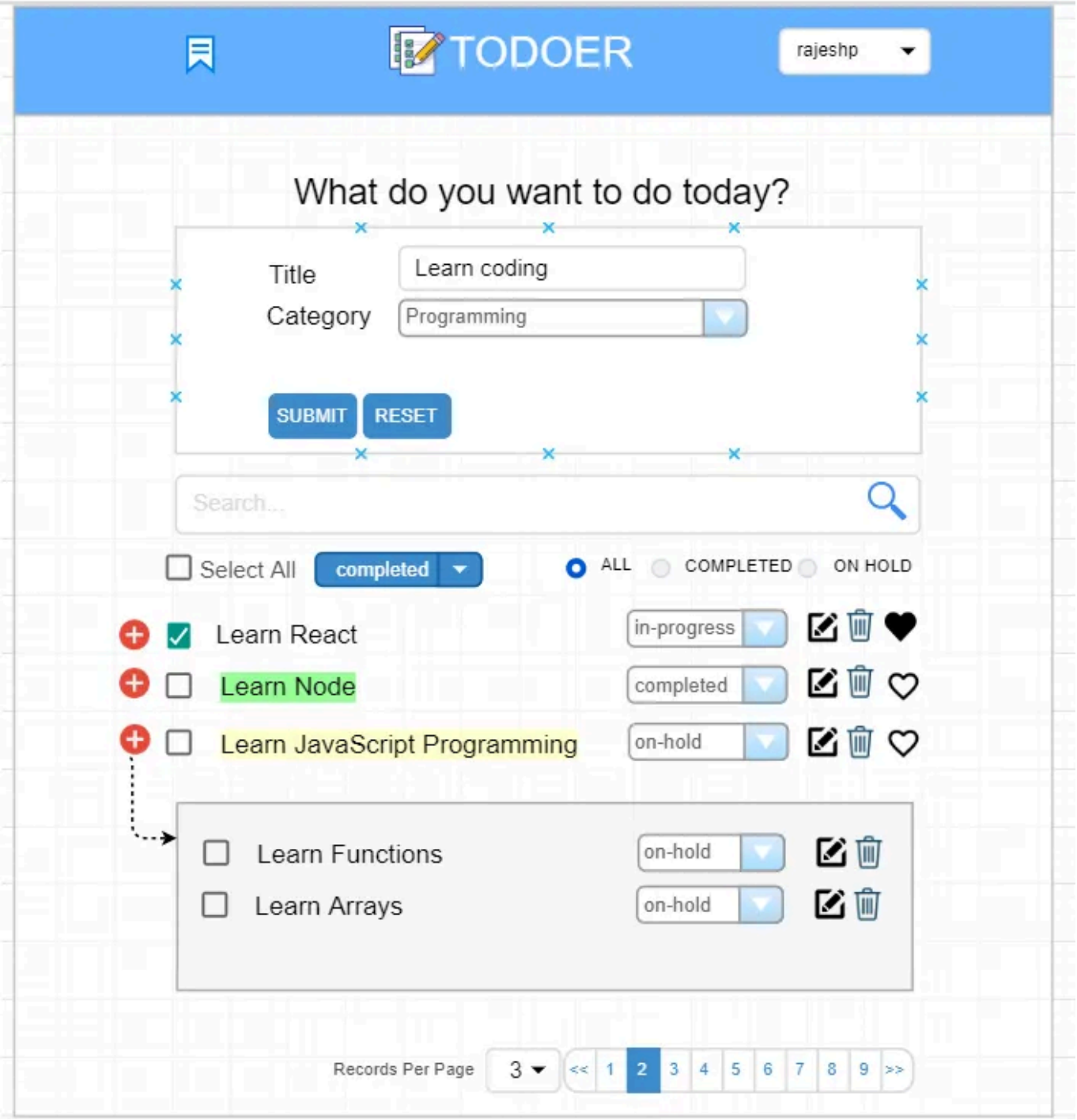
- Implement CRUD feature for category master

Level 7— One to many associations

The updated model is shown below. We now have a new SubTask model that has a reference to the Todo model.



Now, let's take a look at the updated wireframe and how you would go about implementing this feature.



The following functionality needs to be implemented.

- Add an add icon to every todo item.
- On click of the add icon a new panel/dialog/box to be open that contains input fields for adding subtasks.

- All features remain same within subtasks (except we don't have a favorite option here, but you can add it if you want)
- On saving the subtasks, the corresponding todoId should be saved as part of the subtask item.
- BONUS: When all the subtasks are marked as completed, you can automatically mark the main todo as completed as well. (This can be part of an application setting, for e.g. Automatic mark main task as complete when all subtasks are completed".

Level 7— Many to many association

Add a many to many relation between Todo and Tags.

Requirements:

- Add a Tag Model (tagname (string)
- Add a Junction Model to represent many to many relations betwen todo and tags (TodoTags -> (todo_id, tagname)

In the User Interface make the follow changes in the Todo Form

- Add a Custom Input component that allows the user to type tags, separated by comma

- Add this input component in the todo form
- If the tag is new, automatically create it and assign to todo_tags table. Otherwise just use the existing ones.

Level 8 — Reusable Custom Components

Star Rating Component

Add a star rating components to each todo (update the model as needed). Support half starts as well. A sample UI representation is given below. Note design it so that the user can change the number of stars dynamicall, set colors etc.

HINT: Use render props pattern (Let's say instead of star some other symbols to be used).

NOTE: For React devs

- Dont' use external state management library. If you need global state, use context
- Demonstrate if HOC or Renderprops pattern can be applied here if needed.
- Optimize the component structure and if needed, use useCallback and useMemo with justifications of their use.
- Make use of custom hooks as required to keep logic DRY.

That's it for part 1. Watch out for part 2 where we will add more features to the app and then in subsequent parts will create the meta-model to simulate

common applications like e-commerce, blog, twitter, Pinterest, Hackernews, Reddit, etc.

Ideas for Part 2

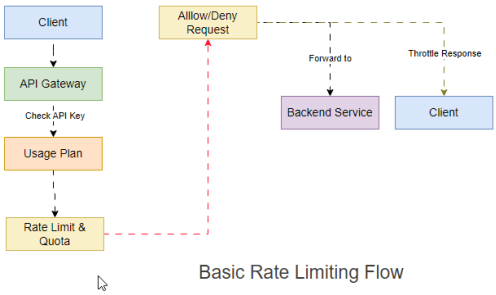
- Task Collaboration
- Real time chat (WebSocket, one on one chat, group chat etc.)
- Video Collaboration (WebRTC, streaming etc.)
- Brainstorming Board

Apps Projects Beginner Model Bootcamp

Rajesh Pillai

What are your thoughts?

More from Rajesh Pillai and codeburst



Basic Rate Limiting Flow

In codeburst by Rajesh Pillai

API Rate Limiting

These articles/stories are part of a series on #tinysystemdesign. You can follow on my...

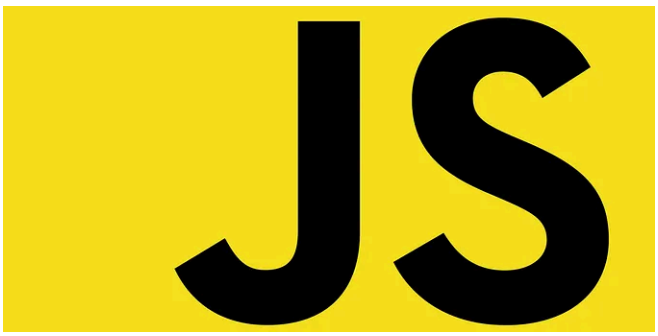
Sep 24, 2023 64

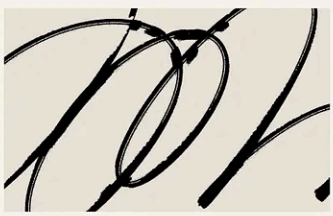
In codeburst by Zuzana K

Digging my way out of tutorial hell

I wouldn't call myself a web developer.

Oct 13, 2017 8.8K 38





In codeburst by Arnav Aggarwal

The Simple Rules to ‘this’ in Javascript

A few rules determine what the 'this’ keyword is inside a function

Jun 27, 2017 3.1K 15

In Unlearning Labs by Rajesh Pillai

HttpClient — A educational rewrite of core features provided by Axio...

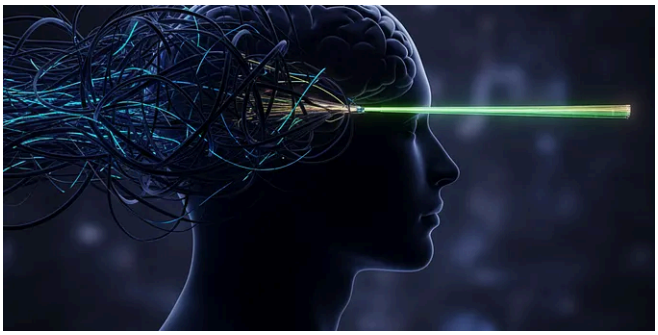
We know NPM packages are subject to vulnerabilities. The recent one is the Server-...

Aug 13, 2024 2

See all from Rajesh Pillai

See all from codeburst

Recommended from Medium



In Beyond Localhost by The Speedcraft Lab

I Failed 47 System Design Interviews—Then One Netflix...

A single shift in how I framed scalability turned rejection into three competing offers...

Nov 5 907 25

Nidhi Ashtikar

Day 5—Python Programming—LOOPS

Loops:

Jul 7 3

In The Software Journal by Software Journal

20 Git Command-Line Tricks Every Developer Should Know

Stop memorizing commands. Start using Git like a power tool that saves time, prevents...

3d ago 95

In Level Up Coding by Ewa Barczykowska

How to Master Sorting in Python Using the sorted() Function

This article is for beginner or intermediate programmers who want to learn about sortin...

Sep 10 14

In Towards AI by coding with tech

I Tried Learning Data Science in 30 Days—Here's the Reality

What actually happens when you try to compress a multi-year skill into one brutal...

4d ago 79 2

Simone Carolini

Modern Python Code Quality Setup: uv, ruff, and mypy

Here's a practical guide to setting up Python code quality tooling that's fast, automated,...

Dec 8

See more recommendations