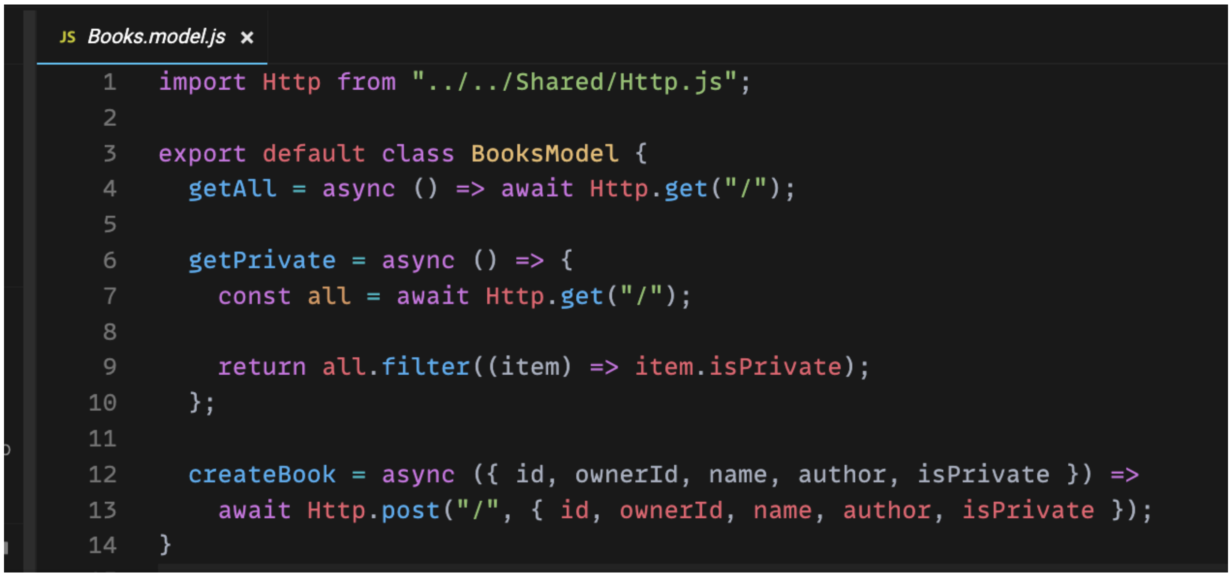
Code Review #2

Books.model.js



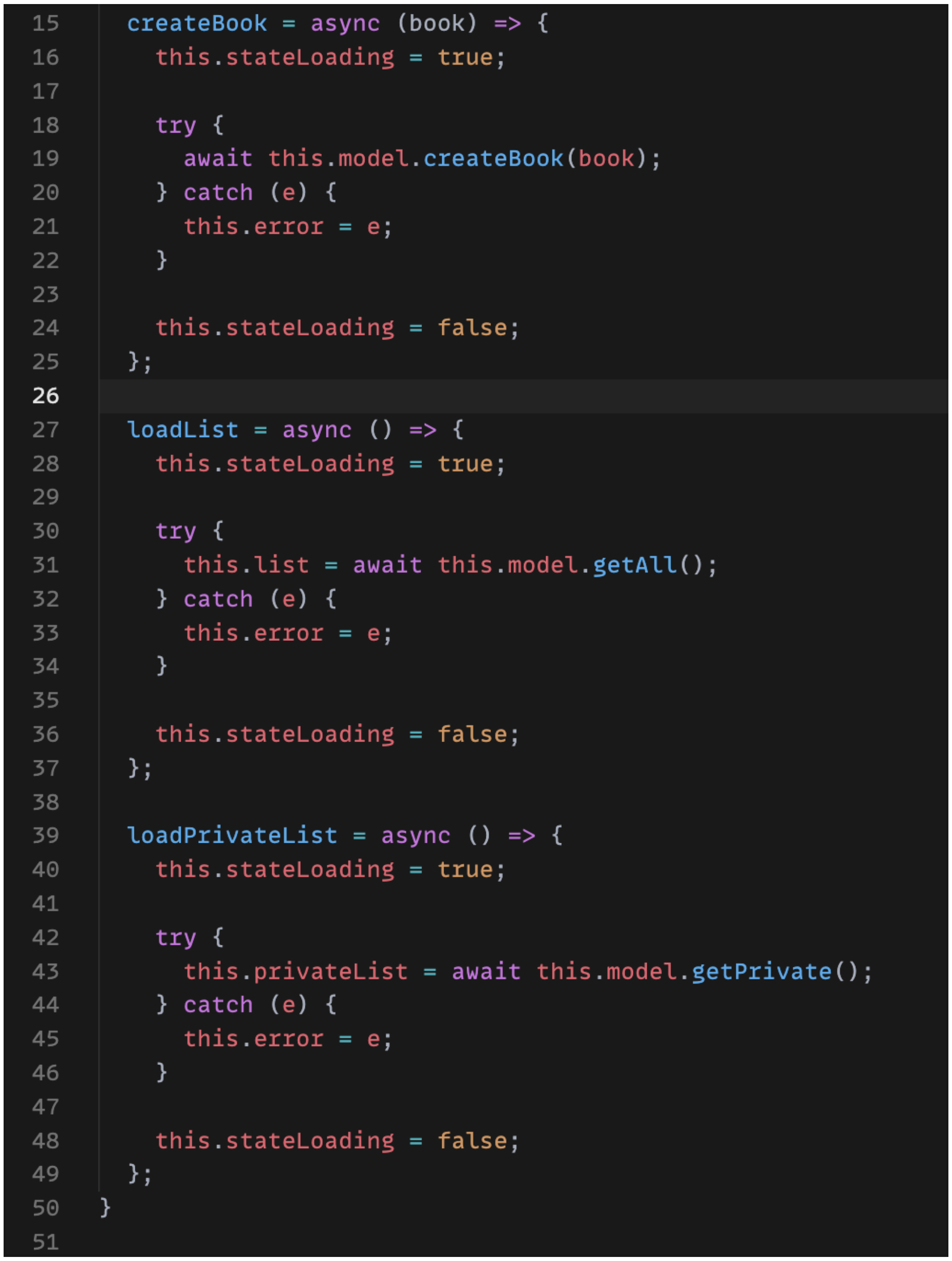
Books.model.js

```js
import Http from "../../Shared/Http.js";

export default class BooksModel {
  getAll = async () => await Http.get("/");

  getPrivate = async () => {
    const all = await Http.get("/");

    return all.filter((item) => item.isPrivate);
  };

  createBook = async ({ id, ownerId, name, author, isPrivate }) =>
    await Http.post("/", { id, ownerId, name, author, isPrivate });
}
```

- according to swagger, private list is a separate request and it does not rely on books' object keys

-----

Books.ctrl.js

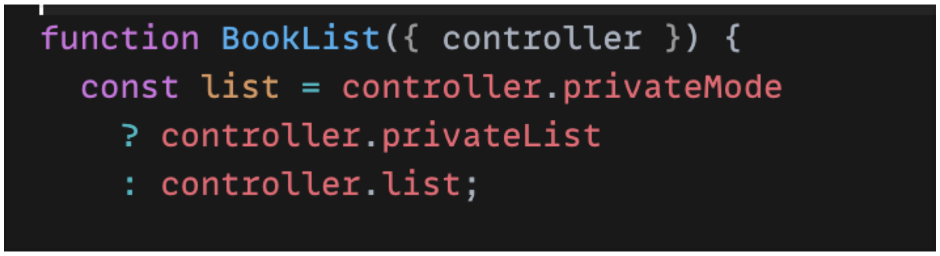


```
  createBook = async (book) => {
    this.stateLoading = true;

    try {
      await this.model.createBook(book);
    } catch (e) {
      this.error = e;
    }

    this.stateLoading = false;
  };

  loadList = async () => {
    this.stateLoading = true;

    try {
      this.list = await this.model.getAll();
    } catch (e) {
      this.error = e;
    }

    this.stateLoading = false;
  };

  loadPrivateList = async () => {
    this.stateLoading = true;

    try {
      this.privateList = await this.model.getPrivate();
    } catch (e) {
      this.error = e;
    }

    this.stateLoading = false;
  };
}
```

- as in first review, all further actions that occurred after async requests should be wrapper in runInAction

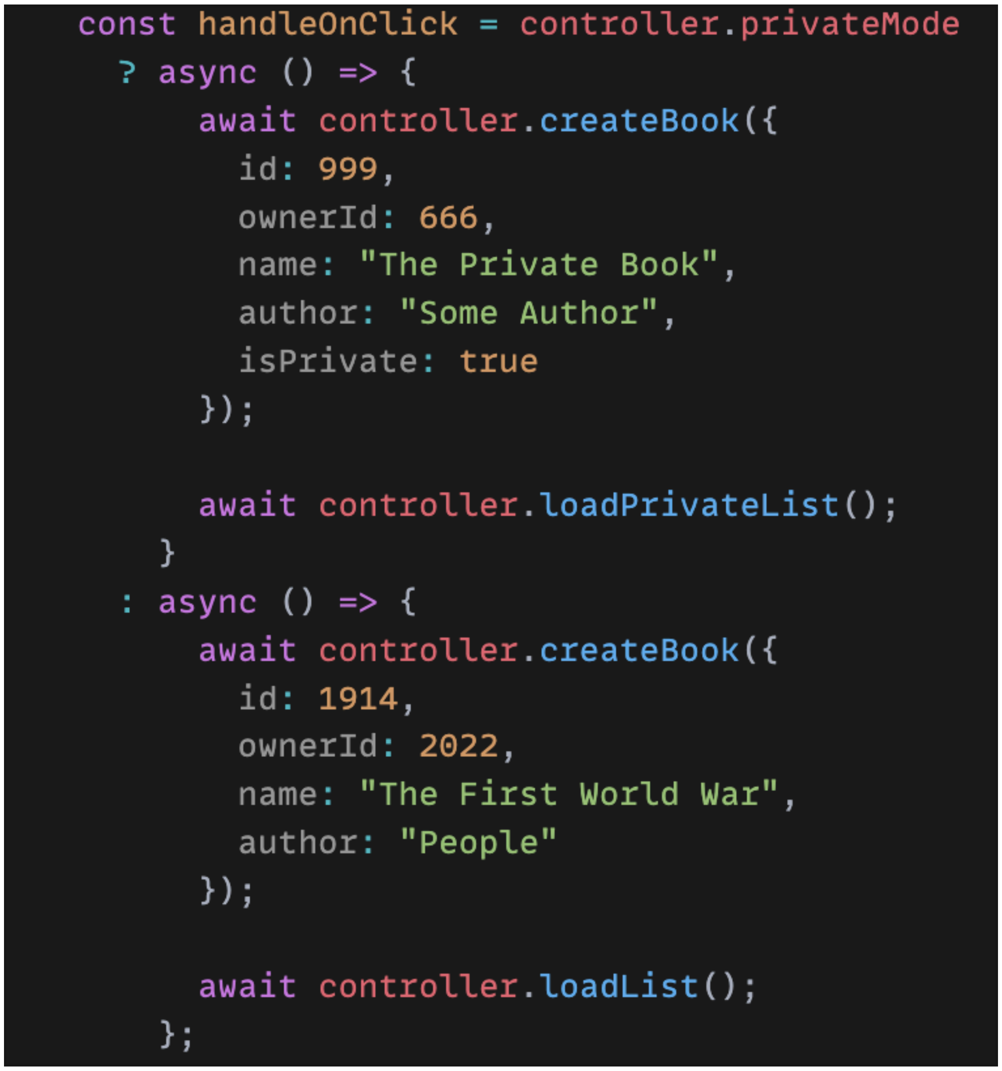
-----

BookList.js



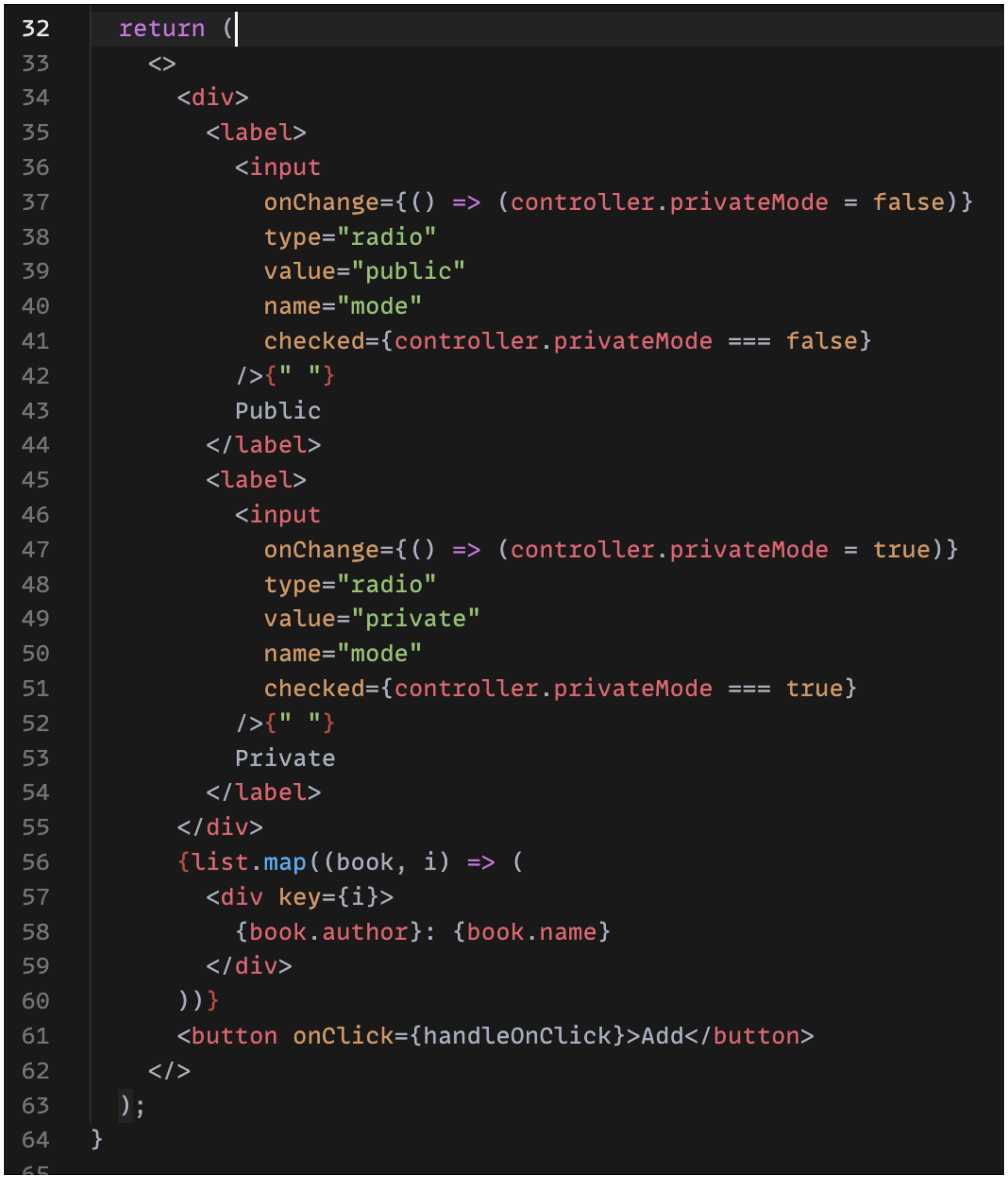
```
function BookList({ controller }) {
  const list = controller.privateMode
    ? controller.privateList
    : controller.list;
```

- such calculations should be moved to controller. use mobx computed values instead



```
  const handleOnClick = controller.privateMode
    ? async () => {
        await controller.createBook({
          id: 999,
          ownerId: 666,
          name: "The Private Book",
          author: "Some Author",
          isPrivate: true
        });

        await controller.loadPrivateList();
      }
    : async () => {
        await controller.createBook({
          id: 1914,
          ownerId: 2022,
          name: "The First World War",
          author: "People"
        });

        await controller.loadList();
      };
```

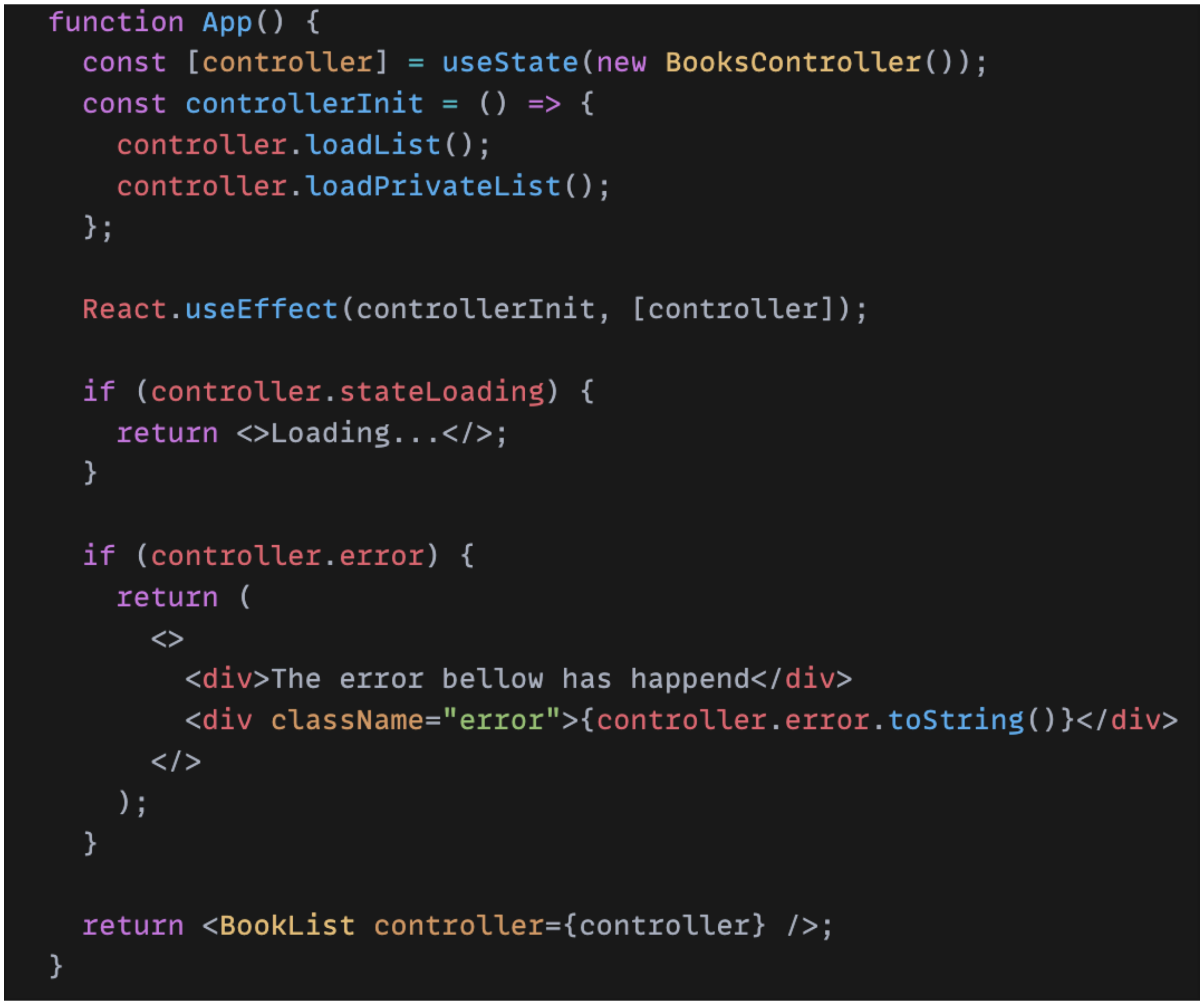
- same goes here. considering that all values are store-related, we can move this logic to controller and just use a plain function



```
  return (
    <>
      <div>
        <label>
          <input
            onChange={() => (controller.privateMode = false)}
            type="radio"
            value="public"
            name="mode"
            checked={controller.privateMode === false}
          />{" "}
          Public
        </label>
        <label>
          <input
            onChange={() => (controller.privateMode = true)}
            type="radio"
            value="private"
            name="mode"
            checked={controller.privateMode === true}
          />{" "}
          Private
        </label>
      </div>
      {list.map((book, i) => (
        <div key={i}>
          {book.author}: {book.name}
        </div>
      ))}
      <button onClick={handleOnClick}>Add</button>
    </>
  );
}
```

- semantically, it is better to use inputs along with labels (with "htmlFor" attr), not wrapped in them
- book list could be wrapped in its own block tag for readability. better to use UL as a wrapper and LI as an iterable tag
- index as a key is considered a bad practice (read reconciliation). use book properties (id) instead

-----
index.js



```
function App() {
  const [controller] = useState(new BooksController());
  const controllerInit = () => {
    controller.loadList();
    controller.loadPrivateList();
  };

  React.useEffect(controllerInit, [controller]);

  if (controller.stateLoading) {
    return <>Loading...</>;
  }

  if (controller.error) {
    return (
      <>
        <div>The error bellow has happend</div>
        <div className="error">{controller.error.toString()}</div>
      </>
    );
  }

  return <BookList controller={controller} />;
}
```

- when state changes happening, we will not be able to see anything but "Loading..." or error message, even though the list is all that will be changed. consider splitting this logic and keep the toggle control (from all books to private) apart from BookList, so we can always see it.

-----

tests

- same arguments as in first review
- API requests should be mocked (jest.mock) and no real requests should be sent in unit tests. for real-life scenarios, e2e should be used