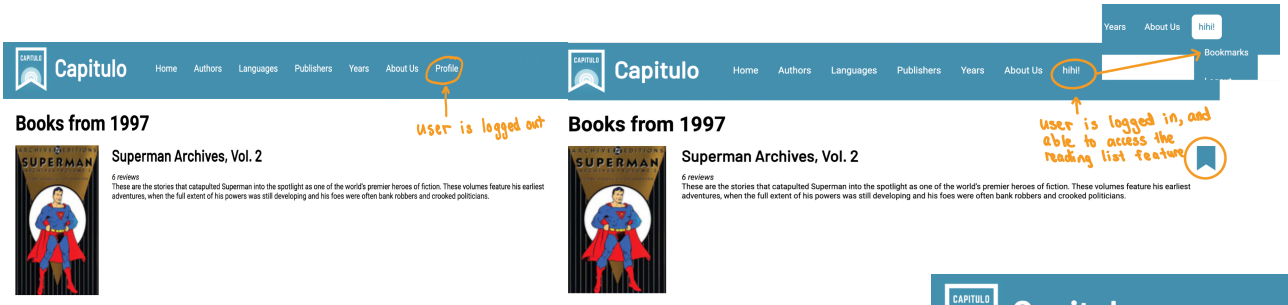
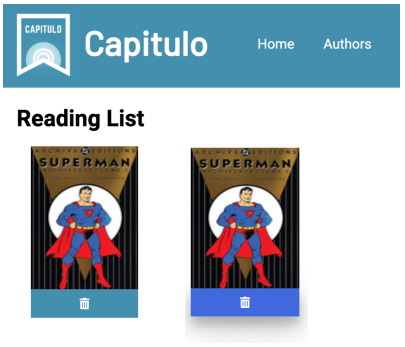
In this report, we will be introducing our cool new feature, which is our reading list. This feature instantly allows users to have their very own reading list, which they can manage by adding any books they want to read, with the option of deleting them as well.





Users must have their own account in order to access their reading list on the website. To achieve this, we decided to slightly modify our domain model by adding a reading list attribute to the User class, so that each User instance that is first created in the website will be initialized with their own reading list that will be available to them once logged in.

```
self.__read_books = []
self.__reviews = []
self.__pages_read = 0
self.__reading_list = []
```

In addition to this, we decided to follow the **application architecture** by using Flask's **blueprints** to implement our feature. A separate blueprint allows us to decouple our application into smaller, reusable components.Doing so benefited our application as it organises our code into a more cohesive manner, allowing us to incorporate further one of the SOLID design principles: **single responsibility**. Therefore our reading list blueprint is only responsible for handling any requests from the user concerning their reading list.

Following this principle allows us to make our class easier to understand, reusable, and most importantly - easier to maintain. Incorporating a blueprint into our design also allowed us to separate the view from its service layer. By doing so, Flask is able to interact with the server without creating a dependency, which will benefit our application in the long run since any changes to the domain model will not necessarily affect the view layer. For example, if we were to change the views component of the blueprint in the future, we would still be able to reuse the services component.

We also ensured the use of our Abstract Repository interface so that our service layer does not have to directly depend on the repository. By doing so, we have also applied the **dependency inversion principle** and **abstract repository pattern**, to further remove the dependencies between our low and high level modules, which could prove to be detrimental to our application if we were to change our code.