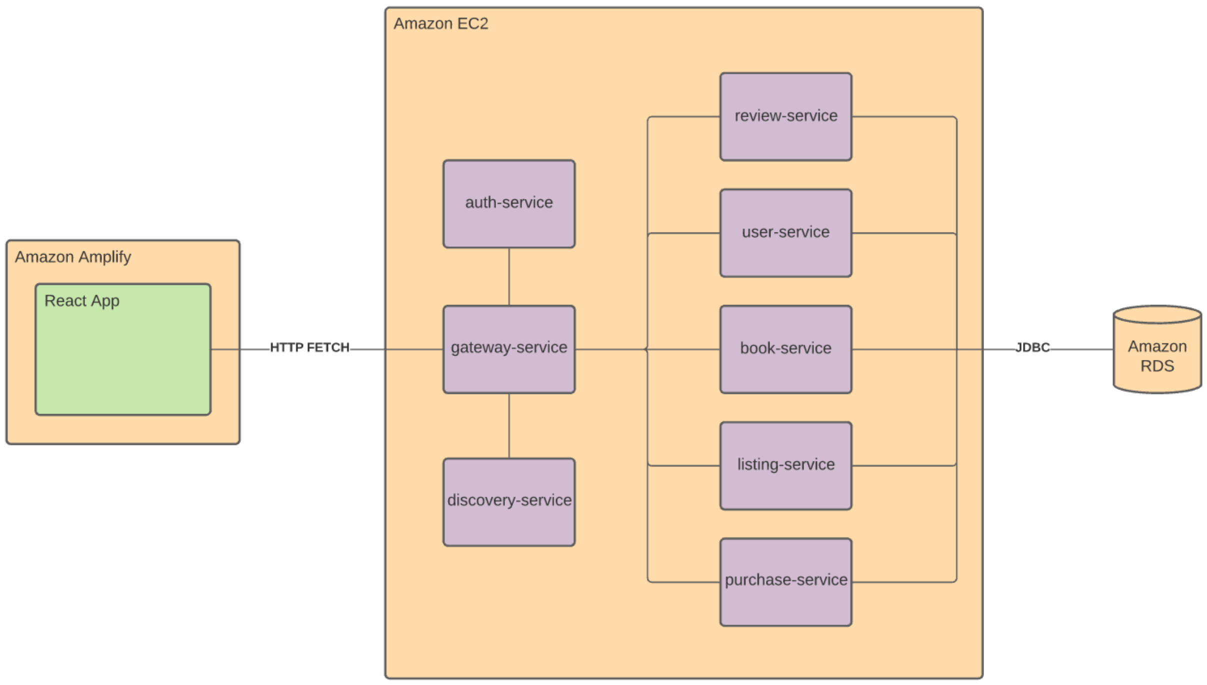
Project Report - MEDS

Maxwell Reid (s3787033), Ewan Breakey (s3845382),
Sefanur Erciyas (s3842307), Thomas Dib (s3838765)

Our Vision

Our vision for the Bookaroo application is to connect sellers of a plethora of books to potential customers. Our interface provides an accessible platform for people of all ages that allows them to pursue their passion of reading and share that very passion with others. Not only can our users view various books (and information regarding them) offered on the site, but they can also share books with their friends, purchase the books, or even sell them. Selling books once you have finished reading them creates an ecosystem that allows readers to recycle books and receive them for a cheaper price than brand new. This promotes reading to a larger audience and helps the environment in the fact that not as many books are being printed.
The ability to swap books with other users reduces the redundancy of having to sell a book and wait for the funds to purchase another book, a feature that has been missed in the current market. Our interface is quick to learn and easy to use, and is thus user friendly for all of our customers!
We believe reading is an essential part of learning and growing as a person, and should be spread far and wide to everyone; the message we are trying to spread to everyone!

System Architecture



https://lucid.app/lucidchart/13449ee8-bd8b-4820-ae08-c510924a1efe/edit?beaconFlowId=33D3E257E1784F03&invitationId=inv_1e2f1826-300b-44a9-9a5a-b0b5372ca341&page=0_0#

Refactoring

Our development process favoured carefully planned architecture over a “move fast and break things” style approach and so there were no major refactors. However, various parts of the application were refactored at times to address anti-patterns and code smells such as duplicated code, bloated components and to increase development ergonomics.

One refactor that occurred was to address a problem with bloated React components and repeated state-handling logic on the frontend. Various container components required logic for re-requesting state from the backend when one of their child components updated. To implement this pattern, the parent maintained state to determine whether the current state was “valid” and then passed a function to set that valid status to false to each child component. This led to a large quantity of repeated code in each container component. To fix this, we refactored the request logic into a react hook called “useAsync” that encapsulated making a HTTP request, cancelling requests for unmounted components and this invalidation logic. Thus, we could shrink our bloated container component code into a few legible lines with all of the messier details hidden in the useAsync hook.

Other component-related refactors also occurred throughout the development of the frontend. Often we would underestimate the scope of a component and decide to refactor it into several other components or identify that several components shared a common element and extract that element into a shared general-purpose component.

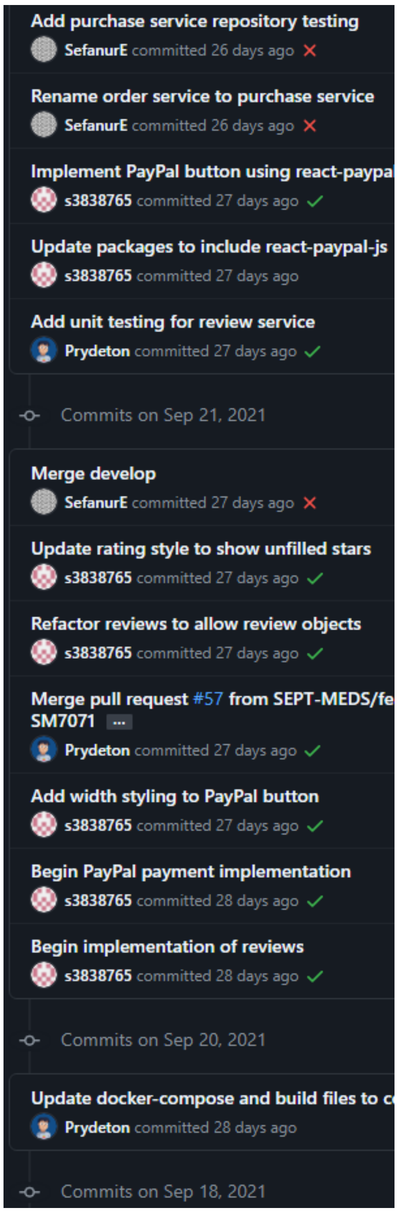
There was one piece of refactoring somewhat motivated by a change of design patterns. Initially, the logic for guarding frontend routes was performed by the **page** component which would redirect unauthorized users back to the homepage after the **page route** component rendered it. This represented an inside-out approach to authorisation in which every page separately handled checking the current user and their role before determining their access level. Identifying that this was suboptimal, we refactored this system to follow a more outside-in principle. We utilised a compositional pattern in which a variant of the **page route** component would perform the authorisation logic and then either render the original **page** component or render an “403 unauthorized” page instead. Thus we could create the authorisation rules declaratively and while re-using existing components. This cut down on code repetition in each protected page, reduced the number of redirects and led to increased separation of concerns as the routing code could handle authorisation instead of the page, which could then focus on rendering it’s content.

The backend of the application is primarily static/stateless and follows very closely to spring conventions and so there were few moving parts that required refactoring during the development process. The few dynamic components, such as the authentication logic, didn’t require refactoring as they were very small and cohesive, right from the initial implementation.

GitFlow

For this project, we strictly followed the GitFlow standard. We maintained a *main* branch and a *develop* branch in order to simultaneously have a production server deployed from *main* while we continued to implement new user stories. When a new feature was being developed, a dedicated branch was branched off of the *develop* branch. Once the feature was fully implemented, we utilised pull requests which needed to be approved by at least one other member of the team before it was merged into *develop*. *Hot fix* and *refactor* branches were also used where appropriate. At the end of each sprint, the *develop* branch was merged into the *main* branch along with the creation of a release tag. Typically commits were made after progress was made on a branch that the developer was confident with on their individual branches. Commits to the *develop* branch were only completed once a review of the merging branch was completed, typically every 2-3 days. Commits to *main* were only completed at the end of the sprint, other than several commits made when attempting to set up continuous deployment. A total of over 450 commits and 90 branches were completed over the project.

Below are 3 screenshots of different periods of the commit history on the *develop* branch to demonstrate consistent commits. First screenshot shows Sept 22, 21, 20 and 18. Second screenshot shows Sept 11, 8, 6, 5 and 3. Third screenshot shows Aug 23, 22, 21 and 19.
Please keep in mind the red crosses are due to running out of free credits on CircleCI. Also, because our repository is part of an organisation on GitHub, we do not have access to the repository insights hence, why we gave these screenshots.



Add purchase service repository testing
SefanurE committed 26 days ago ✗

Rename order service to purchase service
SefanurE committed 26 days ago ✗

Implement PayPal button using react-paypa
s3838765 committed 27 days ago ✓

Update packages to include react-paypal-js
s3838765 committed 27 days ago

Add unit testing for review service
Prydeton committed 27 days ago ✓

Commits on Sep 21, 2021

Merge develop
SefanurE committed 27 days ago ✗

Update rating style to show unfilled stars
s3838765 committed 27 days ago ✓

Refactor reviews to allow review objects
s3838765 committed 27 days ago ✓

Merge pull request #57 from SEPT-MEDS/fe
SM7071 ...
Prydeton committed 27 days ago ✓

Add width styling to PayPal button
s3838765 committed 27 days ago ✓

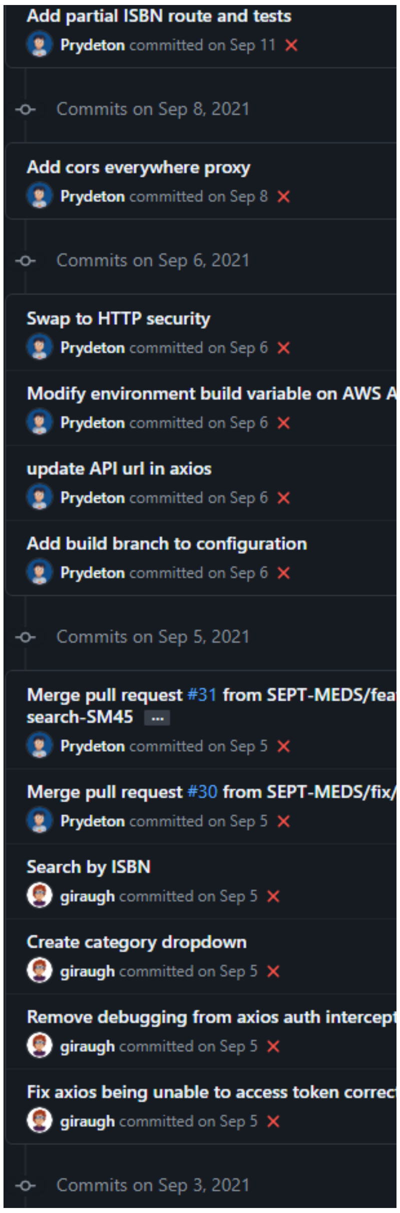
Begin PayPal payment implementation
s3838765 committed 28 days ago ✓

Begin implementation of reviews
s3838765 committed 28 days ago ✓

Commits on Sep 20, 2021

Update docker-compose and build files to c
Prydeton committed 28 days ago

Commits on Sep 18, 2021



Add partial ISBN route and tests
Prydeton committed on Sep 11 ✗

Commits on Sep 8, 2021

Add cors everywhere proxy
Prydeton committed on Sep 8 ✗

Commits on Sep 6, 2021

Swap to HTTP security
Prydeton committed on Sep 6 ✗

Modify environment build variable on AWS A
Prydeton committed on Sep 6 ✗

update API url in axios
Prydeton committed on Sep 6 ✗

Add build branch to configuration
Prydeton committed on Sep 6 ✗

Commits on Sep 5, 2021

Merge pull request #31 from SEPT-MEDS/fea
search-SM45 ...
Prydeton committed on Sep 5 ✗

Merge pull request #30 from SEPT-MEDS/fix/
Prydeton committed on Sep 5 ✗

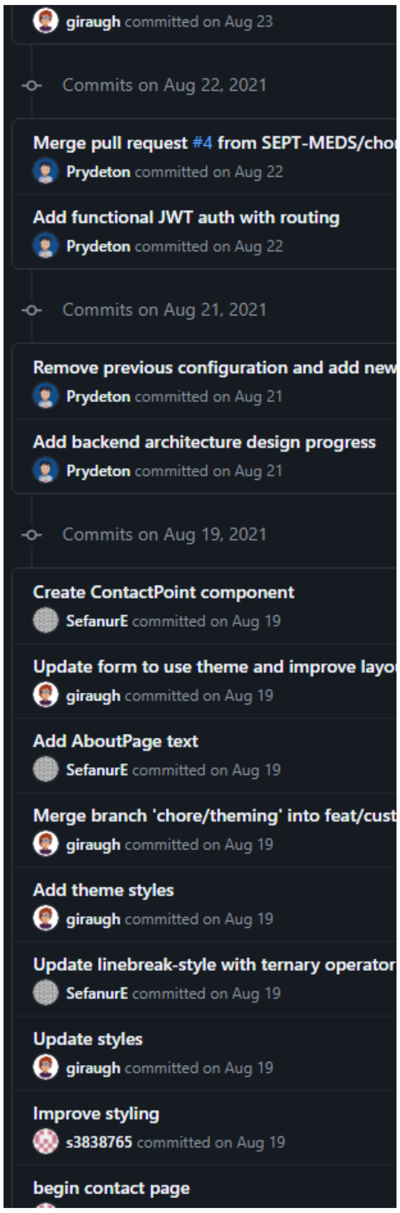
Search by ISBN
giraugh committed on Sep 5 ✗

Create category dropdown
giraugh committed on Sep 5 ✗

Remove debugging from axios auth intercept
giraugh committed on Sep 5 ✗

Fix axios being unable to access token correc
giraugh committed on Sep 5 ✗

Commits on Sep 3, 2021



giraugh committed on Aug 23

Commits on Aug 22, 2021

Merge pull request #4 from SEPT-MEDS/cho
Prydeton committed on Aug 22

Add functional JWT auth with routing
Prydeton committed on Aug 22

Commits on Aug 21, 2021

Remove previous configuration and add new
Prydeton committed on Aug 21

Add backend architecture design progress
Prydeton committed on Aug 21

Commits on Aug 19, 2021

Create ContactPoint component
SefanurE committed on Aug 19

Update form to use theme and improve layo
giraugh committed on Aug 19

Add AboutPage text
SefanurE committed on Aug 19

Merge branch 'chore/theming' into feat/cust
giraugh committed on Aug 19

Add theme styles
giraugh committed on Aug 19

Update linebreak-style with ternary operator
SefanurE committed on Aug 19

Update styles
giraugh committed on Aug 19

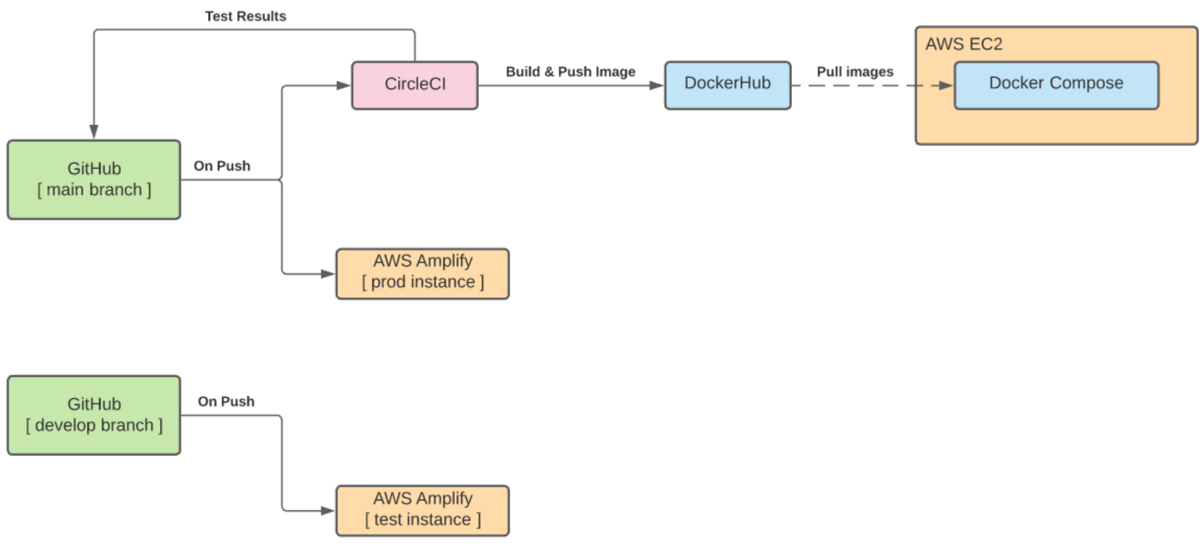
Improve styling
s3838765 committed on Aug 19

begin contact page

Scrum Process

Typically the team met up formally 2-3 times per week in which each meeting was documented. The scrum master (Maxwell Reid) led the team to review what had been completed since the previous meeting before discussing issues encountered and future plans. This was in addition to constant communication across the team through the use of Slack, Discord and in-person discussions. This ensured the team remained aware of the status of the project at all times. User stories were updated as tasks were completed and as requirements changed.

Deployment Pipeline



https://lucid.app/lucidchart/96f1556c-4995-4567-82cf-30b3c869d52f/edit?viewport_loc=-11%2C152%2C2219%2C1076%2C0_0&invitationId=inv_0db563d7-47f1-455e-812e-4ec83a59018e

Testing

Each of the services, particularly those that had direct interaction with the Database, were tested using Unit Testing. This included the Repository, Service and Controller, each of which were tested using Mockito. Prior to deployment, these tests were run via CircleCI which was reported by GitHub.

Additionally, each user story was tested manually through Acceptance Testing. This was done prior to the end of each sprint to ensure that each user story behaved appropriately. These were reported in the Acceptance Testing document and if an error was identified, the developer responsible for that User Story was identified and informed of the bug. Once the acceptance criteria was met for a user story, the user story was moved from *Feature Complete* to *Done* and the team was notified. Issues identified through acceptance testing were also reported utilising the GitHub issues page.

Evidence of testing was provided across each of the milestones.