QA CINEMAS

By Jade Foster, Isabella O’Hara, Mariana Nikolova and Fatima Sheikhnur

Contents

- Intro to QA Cinemas
- Scrum overview
- Jira board overview
- CI/CD (Git)
- Back-end
- Front-end
- Testing
- DEMO
- Reflections/conclusions

Introduction

- QA Cinemas website using the MERN stack
- Started with a planning phase
- Front-end team (Jade+Fatima)
- Back-end team (Bella+Mariana).
- 2 sprints

Scrum Overview

Bella - Scrum master + Developer

Mariana - Product Owner + Developer

Jade - Developer

Fatima - Developer

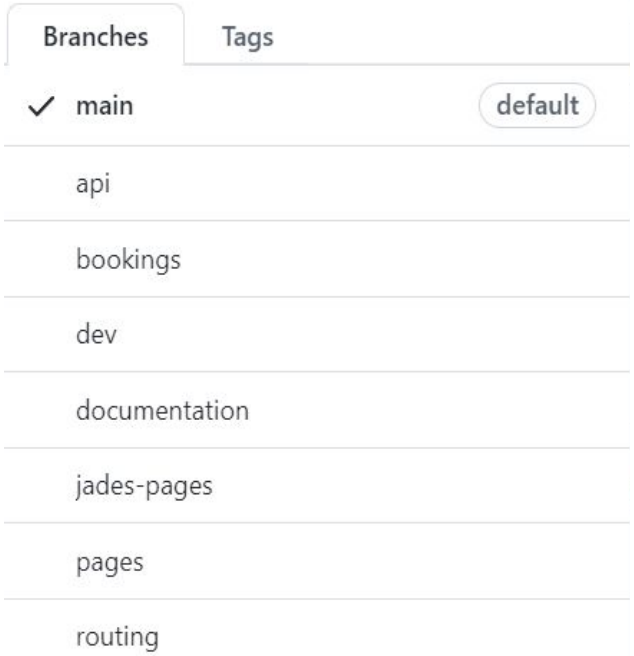
1. Sprint Planning Meeting
2. Daily stand-ups
3. Sprint review
4. Sprint retrospective

Jira Board Overview

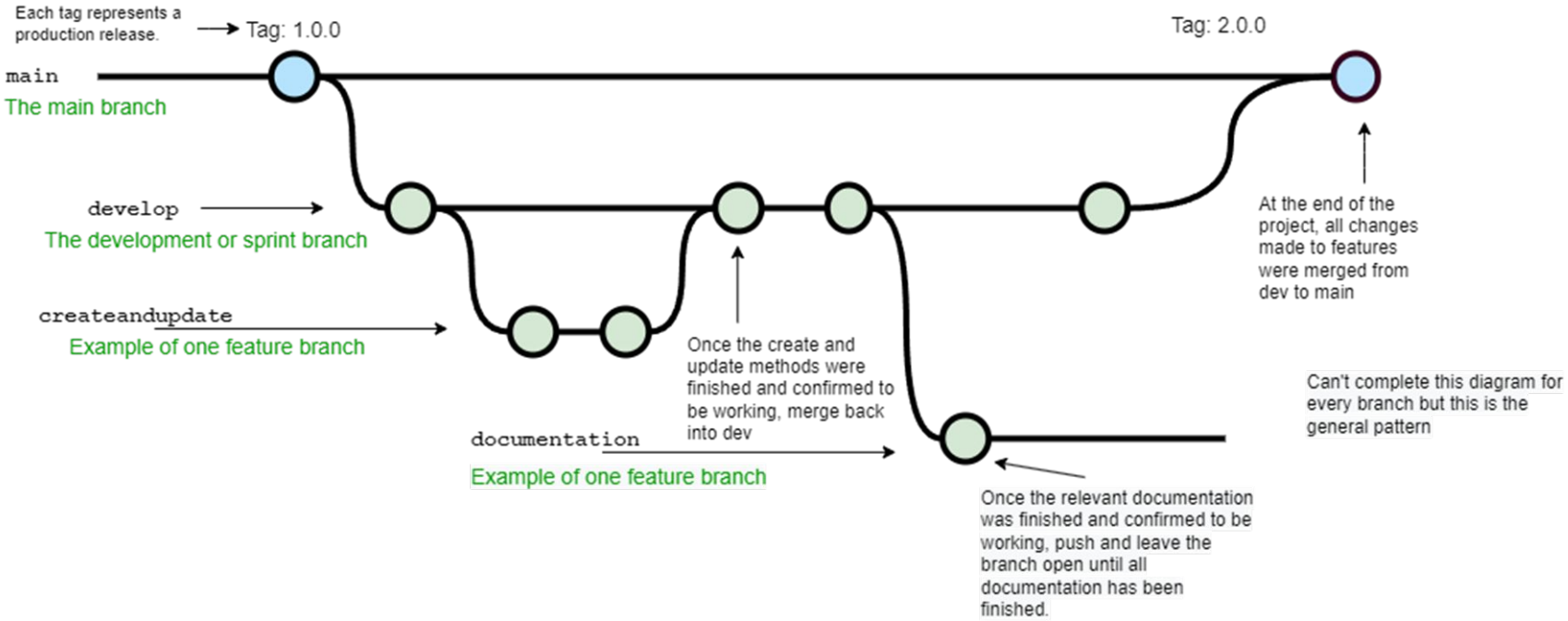
- Utilised a Kanban board format
- Issues from the perspective of the developer
- Organised into 3 user epics :
  - MVP
  - Additional features
  - Testing
- Assigned story points from 2-34
- Organised into 2 sprints:
  - Development
  - Testing

Git Branches

- Feature branch model
- Back end branch is “api”, which links to “bookings” branch
- Front end branches split into “routing”, “pages” and “jades-pages”
- Last minute styling changes were done on the dev branch, as it would not threaten the functionality of the rest of the code.



Git Branching Diagram



CINEMA DB

Movies collection

Name: String
Actors: [String]
Director: String
Genre: String
Release date: date
Duration: number
Classification: string
Rating: number

Discussions collection

Movie name: String
Username: String
Comment: String
Rating: Number

Cinemas collection

Name: string
Location: string
Phone number: number
Manager's email: string
Normal screens: {number of screens: number, capacity: number}
Director screens: {number of screens: number, capacity: number}
Movies shown: [movie objects]

Screenings collection

Cinema name: String
screen number:
Screen type: String
Movie: {movie object}
Date/time: Date
Seats left: number

Ticket prices

User type: Adult, prices: {normal screen price: number, director screen price: Number}

User type: Child, prices: {normal screen price: number, director screen price: Number}

User Type: Concession, prices: {normal screen price: number, director screen price: Number}

Back End - Database and API calls

- We started working on our backend by creating a Mongo database, mongoose schemas and planning the necessary API routes
- We then created our Express.js router and API endpoints linking to Screenings,
- Ticket prices and Discussions collections

```
const server = app.listen(3001, () => {
    console.log("Server started successfully on port
    + server.address().port);
```

```
const mongoose = require("mongoose");
const{Schema, model} = mongoose;

const DiscussionsSchema = new Schema({
    "MovieName": String,
    Name: String,
    Comment: String,
    Rating: Number
router.post("/create", (req,res) => {
    DiscussionsModel.create(req.body).then(ds => {
        res.status(201).json(ds);
    }).catch((err) => {
        res.status(500).json(err);
```

Back End - Database and API calls

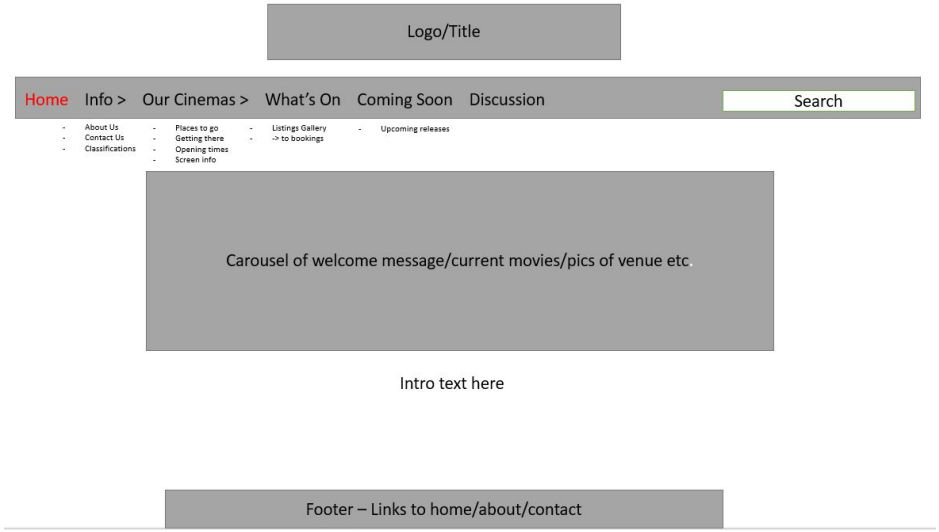
- Fetch API to link to database
- API calls were used on Bookings and Discussions pages

```
const[AllScreenings, setAllScreenings] = useState([]);
```

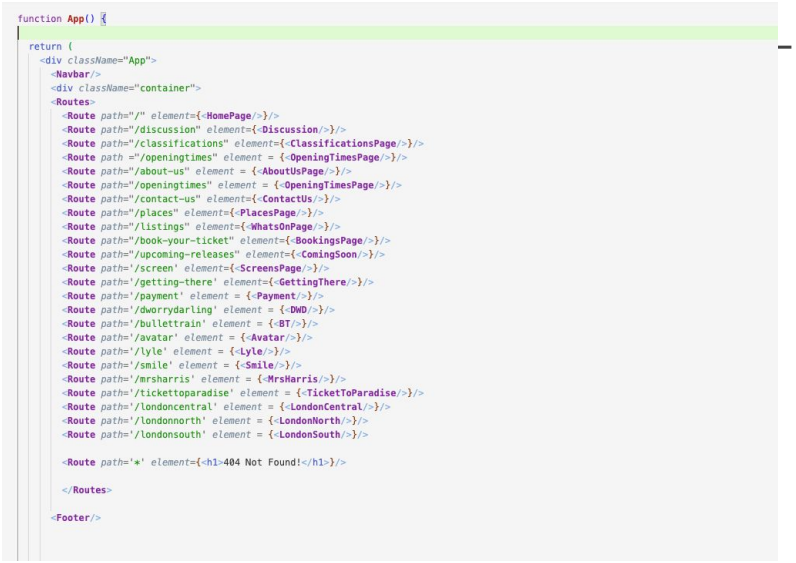
```
//get all screenings from the collection
const getAllScreenings = () => {
    fetch("http://localhost:3001/cinema/screenings/getAll").then(res=>{
        res.json().then(data=>{
            setAllScreenings(data).then(AllScreenings => AllScreenings)
        }).catch(err=>{
            console.log(err)
        })
    })
```

Front End Initial Planning

- To ensure that both front end developers had the same vision, we created a detailed initial plan of each page of the website



Front End Routing

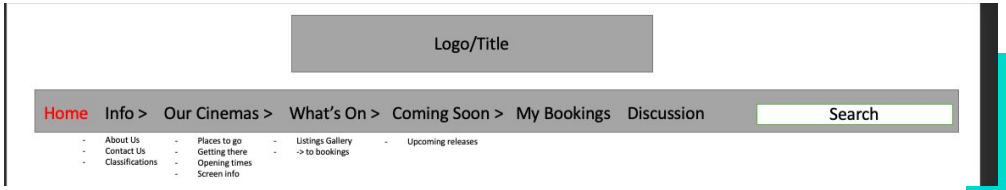


```
function App() {

  return (
    <div className="App">
      <Navbar/>
      <div className="container">
      <Routes>
        <Route path="/" element={<HomePage/>}/>
        <Route path="/discussion" element={<Discussion/>}/>
        <Route path="/classifications" element={<ClassificationsPage/>}/>
        <Route path ="/openingtimes" element = {<OpeningTimesPage/>}/>
        <Route path="/about-us" element = {<AboutUsPage/>}/>
        <Route path="/openingtimes" element = {<OpeningTimesPage/>}/>
        <Route path="/contact-us" element={<ContactUs/>}/>
        <Route path="/places" element={<PlacesPage/>}/>
        <Route path="/listings" element={<WhatsOnPage/>}/>
        <Route path="/book-your-ticket" element={<BookingsPage/>}/>
        <Route path="/upcoming-releases" element={<ComingSoon/>}/>
        <Route path='/screen' element={<ScreensPage/>}/>
        <Route path='/getting-there' element={<GettingThere/>}/>
        <Route path='/payment' element = {<Payment/>}/>
        <Route path='/dworrydarling' element = {<DWD/>}/>
        <Route path='/bullettrain' element = {<BT/>}/>
        <Route path='/avatar' element = {<Avatar/>}/>
        <Route path='/lyle' element = {<Lyle/>}/>
        <Route path='/smile' element = {<Smile/>}/>
        <Route path='/mrsharris' element = {<MrsHarris/>}/>
        <Route path='/tickettoparadise' element = {<TicketToParadise/>}/>
        <Route path='/londoncentral' element = {<LondonCentral/>}/>
        <Route path='/londonnorth' element = {<LondonNorth/>}/>
        <Route path='/londonsouth' element = {<LondonSouth/>}/>

        <Route path='*' element={<h1>404 Not Found!</h1>}/>

      </Routes>

      <Footer/>
```

- We began Front-End development with the routing of our website. Ensuring all pages were part of averall navigation. The Navbar was the first component made, where we included dropdowns, ensuring navigating the site runs as smoothly as possible.

Front End (Evolution of the Navbar)

- We used modules such as reactstrap on the navbar for styling and interactivity with the user- we ran into some problems which we later resolved including the dysfunction of automatic close of the navbar.

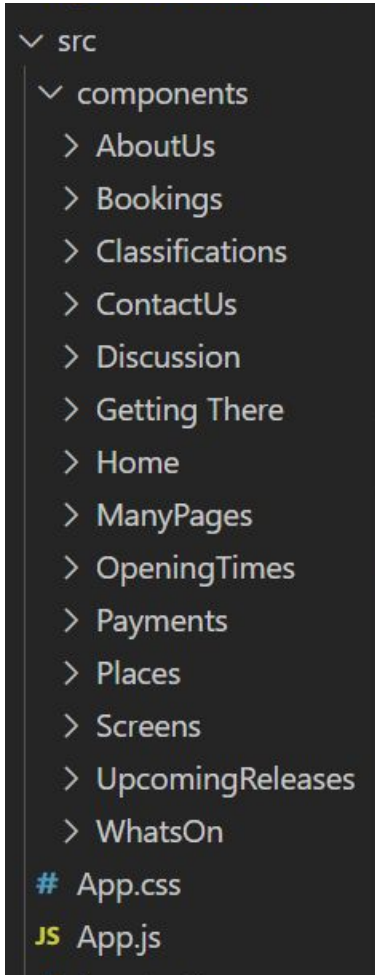


Front End - Components

- Each page has its own folder containing:
  - Main jsx page
  - Any other components related to the page
  - Other assets/individual css styling

- “ManyPages” contains elements used on multiple pages, e.g the header, footer and navbar

- The homepage contains extra components that create a carousel, which have extra settings to better the user experience



Front End - Link to Back End

- The bookings page links directly to the cinema database

- Front end user can also filter through the database using .filter

```
import React, { useEffect } from 'react';
import { useState } from 'react';
import {useNavigate} from "react-router-dom";

const BookingsPage = () => {
    document.title = "Bookings"

    const[AllScreenings, setAllScreenings] = useState([]);
    const[movie, setMovie] = useState("");
    const[cinema, setCinema] = useState("");
    const[visible, setVisible] = useState(false);

    //get all screenings from the collection
    const getAllScreenings = () => {
        fetch("http://localhost:3001/cinema/screenings/getAll").then(res=>{
            res.json().then(data=>{
                setAllScreenings(data).then(AllScreenings => AllScreenings)
            }).catch(err=>{
                console.log(err)
            })
        })
    }

    useEffect(() => {
        getAllScreenings()
    },[])

    const submitHandler = (evt) => {
        evt.preventDefault();
```

Front End - Link to Back End

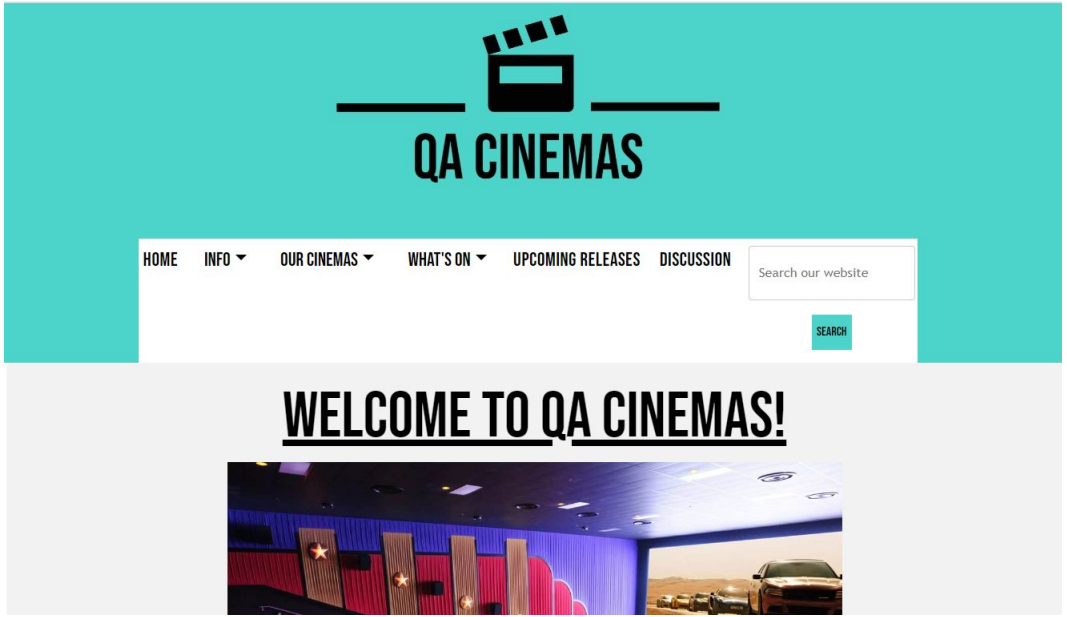
- The Discussions page links to our database for fetching the latest discussions and adding new ones
- Discussions are filtered by movie
- New posts are checked for profanity using bad-words node module before being posted

```
const Filter = require('bad-words'),
    filter = new Filter();
const[movie, setMovie] = useState("");
const[comment, setComment] = useState("");
const[rating, setRating] = useState(0);
const[name, setName] = useState("");
```

```
const postDiscussion = () =>{
    const newDiscussion = {"MovieName":movie, "Name":name,
    "Comment":comment, "Rating":rating}

    //checking for profanity before posting n
    if (filter.isProfane(comment) || filter.isProfane(name))
        return alert("Profanity is not allowed!");
    } else{
    fetch("http://localhost:3001/cinema/discussions/create",
        method: 'POST',
        headers: {'Content-Type':'application/json'},
        body: JSON.stringify(newDiscussion)
    }).then(console.log(newDiscussion)).then(res=>{
        res.json().then(newDiscussion).then
        (alert("Comment added successfully!")).catch(err=>{
        console.log(err)
```

Front End - Styling

- Styled using our logo's colour and font as a basis for the entire website



- Main features:
  - Bold theme for titles/navbar, softer and more readable theme for main body text
  - Aqua blue, grey/white theme for brand recognisability
  - Titles and images centered on the page for better aesthetics
  - Dropdown menus in navbar to stop overcrowding

Front End - Styling

```
.title{
    font-family: 'Bebas Neue', cursive;
    font-size: 80px;
    color: black;
    text-decoration-line: underline;
    text-align:center;
}

.subtitle{
    font-family: 'Bebas Neue', cursive;
    font-size: 60px;
    color: black;
    text-align:left;
}


.nav {
    flex-direction: column;
    background-color: #4CD3C4;
```

- General styling formatted in app.css, and pages that require individual styling are styled in their own css file
  - So if changes need to be made, the source code for them is easy to locate.
- Added dynamic styling such as changing the colour of buttons on hover
- Added document titles for each page and changed the react app logo in the public folder so that the tab info reflects the nature of the website.

Testing

- API testing with mocha and chai + istanbul for coverage
- ~81% coverage on each file
- Selenium automation testing on the front end.

```
    WebElement listingsGallery = navbar.getListingsGallery();
    Assert.assertEquals(true, listingsGallery.isDisplayed());
}

@Test
Run | Debug
public void clickComingSoon() {
    navbar = new Navbar(driver);
    navbar.comingSoon();
    WebElement upcomingReleases = navbar.getUpcomingReleases();
    Assert.assertEquals(true, upcomingReleases.isDisplayed());
}
```

```
> qa-cinemas@0.1.0 test
> mocha 'backend/tests/*.js'

Server started successfully on port 3001


  discussionsAPI test
DB connected!
Test DB Connected
    ✓ /getAll
    ✓ /getFive
    ✓ /create

  screeningsAPI test
Test DB Connected
    ✓ /getAll
    ✓ /getDateAndTime
    ✓ /getScreeningTimes
    ✓ /getSeatsLeft
    ✓ /put/:tickets

  ticketPricesAPI test
Test DB Connected
    ✓ /getAll
    ✓ /getFiltered


  10 passing (433ms)

-------------------------|---------|----------|---------|---------|-------------------
File                     | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------|---------|----------|---------|---------|-------------------
All files                |   85.71 |      100 |    87.5 |   85.71 |
 backend                 |     100 |      100 |     100 |     100 |
  index.js               |     100 |      100 |     100 |     100 |
 backend/routes          |   81.63 |      100 |   66.66 |   81.63 |
  discussionsAPI.js      |   81.81 |      100 |   66.66 |   81.81 | 8,17
  screeningsAPI.js       |   81.48 |      100 |   66.66 |   81.48 | 8,17,25,38,54
  ticketPricesAPI.js     |   81.81 |      100 |   66.66 |   81.81 | 8,22
 backend/schema          |     100 |      100 |     100 |     100 |
  discussions.js         |     100 |      100 |     100 |     100 |
  screenings.js          |     100 |      100 |     100 |     100 |
  ticketPrices.js        |     100 |      100 |     100 |     100 |
 backend/tests           |    83.6 |      100 |     100 |    83.6 |
  discussionsTest.js     |   83.78 |      100 |     100 |   83.78 | 49-50,59-60,74-75
  screeningsTests.js     |   81.81 |      100 |     100 |   81.81 | 36-37,48-49,59-60,70-71,86-87
  ticketPricesTest.js    |   86.66 |      100 |     100 |   86.66 | 34-35,45-46
-------------------------|---------|----------|---------|---------|-------------------
```

Time for a live demo!

Sprint review and retrospective

Sprint review

- Back-end demo + tests
- Front-end walk-through

Sprint retrospective

- Smaller unit work going well
- Improve on communication between teams

Evaluation - What would we do differently next time?

- Separate front end and back end into different projects
  - Would help avoid lots of the bugs/issues we came across.
- Avoid importing node_modules and package-lock.json into GitHub
  - Would help avoid compatibility issues
- Not writing a request body for GET request to act as a filter.
  - Postman accepts it but the fetch API doesn’t - have to write the filtering methods after.

Conclusion

- We used full stack programming to create a functional website where the user experience is centred, and the MVP was fulfilled.
- Skills obtained over the last 6 weeks were implemented from agile scrum methodology to using full stack development technologies.
- We collaborated well in the end on the dev branch, and managed to complete the tasks assigned to us on the jira board.
- If there were no time constraints we would add more styling, making the website more appealing, as well as improving the user experience
- Additional functionality, e.g. working login system so that the user can see their existing bookings, and a functional payments page

Thank you!