Testing Report

This report explains the various activities performed as part of testing the Major Group Project application.

The finished product, Pacto, is a social networking application aimed at students attending universities in the United Kingdom. Pacto consists of two applications: a front-end client application and a server-side API. Both parts of the final application were tested separately and they each have an individual testing suite.

Testing Scope

- **In-Scope**: Functional testing for all features including user-facing and server-side functionality
    - Authentication
    - User Profiles
    - Pacts
    - Posts/Comments
    - Friends
    - Notifications
    - Search

    Running development application on different operating systems (Windows, macOS, Linux)

    User Acceptance Testing (UAT) - Ensure that software solves the problem initially set out to solve

- **Out of Scope**: Performance testing was not done for this application. This includes measuring loading times and operating under heavy loads (large number of users, frequent requests, etc.). It was not possible to test the system for performance due to the limitations of certain third party services that are used (ex. free APIs with limited number of requests/bandwidth allowed).

End-to-end testing was not done as the client and server application were both tested separately.

Metrics

	Test Cases Executed	Test Cases Passed	Code Coverage
Client	374	374	99.35%
Server	297	297	97.22%

Client Testing

The client application was created using React and automated testing was done using Jest. React Testing Library was used in order to test all the components and pages.

In order to run the automated tests, run the following command from within the client directory:

```
$ npm run test
```

Server requests were mocked in order to test the client interaction with the server application. Although we thoroughly tested the server application, testing for the interaction between the client and server was not automated.

Server Testing

The server application was created using Node.js. Server-side testing was done using the Jest JavaScript testing framework as well as supertest.

In order to run the automated tests, run the following command from within the server directory:

```
$ npm run test
```

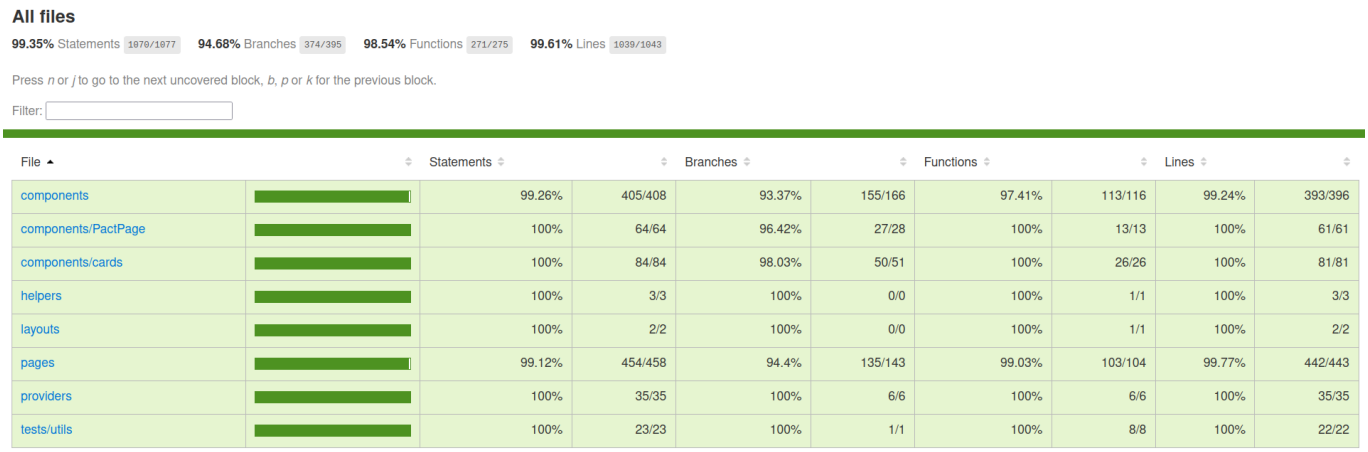
Coverage

In order to view the automated test coverage in the terminal, run the following command from within either the client or server directory:

```
$ npm run coverage
```

Running coverage will create a new directory (within the directory that coverage was run in) called `coverage`. Within this directory, a html report is created that shows the test coverage in detail. The report is located at `/coverage/lcov-report/index.html`

The coverage for the client application currently looks like this:



All files

99.35% Statements `1070/1077` **94.68%** Branches `374/395` **98.54%** Functions `271/275` **99.61%** Lines `1039/1043`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
components	99.26%	405/408	93.37%	155/166	97.41%	113/116	99.24%	393/396
components/PactPage	100%	64/64	96.42%	27/28	100%	13/13	100%	61/61
components/cards	100%	84/84	98.03%	50/51	100%	26/26	100%	81/81
helpers	100%	3/3	100%	0/0	100%	1/1	100%	3/3
layouts	100%	2/2	100%	0/0	100%	1/1	100%	2/2
pages	99.12%	454/458	94.4%	135/143	99.03%	103/104	99.77%	442/443
providers	100%	35/35	100%	6/6	100%	6/6	100%	35/35
tests/utils	100%	23/23	100%	1/1	100%	8/8	100%	22/22

The coverage for the server application currently looks like this:

All files

97.22% Statements `1017/1046` **97.12%** Branches `203/209` **96.51%** Functions `83/86` **97.27%** Lines `1001/1029`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
server	100%	31/31	100%	0/0	100%	1/1	100%	31/31
server/controllers	96.54%	614/636	96.68%	146/151	97.91%	47/48	96.62%	602/623
server/helpers	93.75%	105/112	100%	20/20	90%	18/20	93.63%	103/110
server/middleware	100%	112/112	100%	32/32	100%	10/10	100%	110/110
server/models	100%	69/69	83.33%	5/6	100%	7/7	100%	69/69
server/routes	100%	86/86	100%	0/0	100%	0/0	100%	86/86

Manual Testing

The vast majority of testing has been automated, however, some parts of the project could not be tested automatically. Manual testing was employed for certain parts including:

- Email verification - There was no way to automatically test that a user receives an email and is able to click on a link to verify their account. Manual testing was used to check two important features. First a user should receive an email containing a verification code sent to the email address they used to register for an account. Secondly, a user should be able to use the verification code to verify that they are a student and activate their account. Both these steps were done manually by checking our own inbox (or spam folder) and using the link to create a new account. The verification email that a user receives looks like this:

P

pacto.noreply@pacto.co.uk

Wed 3/30/2022 8:05 PM

To: Treebus, Thom

[You don't often get email from pacto.noreply@pacto.co.uk. Learn why this is important at http://aka.ms/LearnAboutSenderIdentification.]

https://eur03.safelinks.protection.outlook.com/?url=https%3A%2F%2Fu26088800.ct.sendgrid.net%2Fls%2Fclick%3Fupn%3DKzK6uZioLan2QcgT1GEK7HSLHpfVJHgYBKAEjEu3H0F7-2Fy202OLYJIRMYa4I9l0tSeuXghyFB8w4zGeEgKqTORXKMbnUKG9cTcgP7k-2FXfJ8uBcfiZLtxGEPhX7K8IMMHWmxvX3OGhu-2FfKs032rlcdQ-3D-3DmB5h_ZPtEGWCfVZL4gqPxBStf2SkkRUNpFvXiXwdeacRri5BPc1Oj3I57paSe9XV7TQoH1OvxMPzwmiY9Gst2v8SoO99Q0oXhZdm1ovZwoD5Sd4F3mOGG9AqLiHS4HQJhk7F6THqcBvYTxVcI3gFuKuM3MuGoEhGtGT7Tj-2FT55IGEGhbE4-2FyMZQqDwwh-2B4LCS6AXwO2KdYr6BMEw4tpDDyRqlzAXxxQ7GeJyDRcc5NPz2ILU-3D&data=04%7C01%7Cthom.treebus%40kcl.ac.uk%7C065304a2376e46f3dde108da12802faf%7C8370cf1416f34c16b83c724071654356%7C0%7C0%7C637842639175333829%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=9atOVs0kwmNZfPxOm5s8j4UdH%2F392wiCrkTfLhFsJ9w%3D&reserved=0

Reply | Forward

- Seeder - The seeder that creates fake users, pacts, posts and comments is manually tested by looking at the seeded objects via the client application or by checking the database with a tool like MongoDB Compass. First we would confirm that the `unseed` command would delete all database pages, and then we would check that the `seed` command created new objects.

- Page layouts - Automated testing was used to ensure a page contained all necessary components but manual testing was used to check that the layout of a page was correct.

- Different screen sizes - Testing how the application looks/behaves on different screen sizes was done manually using PolyPane, an application that lets you view an app in multiple different screen sizes at the same time. We used PolyPane to view the client on different types of screens ranging from small mobile screens all the way up to 4k monitors. Each page of the user interface was tested manually on different screen sizes to ensure that the components displayed correctly and behaved as expected.

- Collapsible components - The React Testing Library is unable to set the screen size or height of a component which means we were unable to automate the testing of collapsible components. Some components would change depending on the size of the component (large text bodies would be collapsible, etc.) and this had to be tested manually. We would test this manually by for example, creating a post with a very long text body and making sure the post would not show all the text and instead show the first few lines and then an option to expand the text to reveal it entirely.