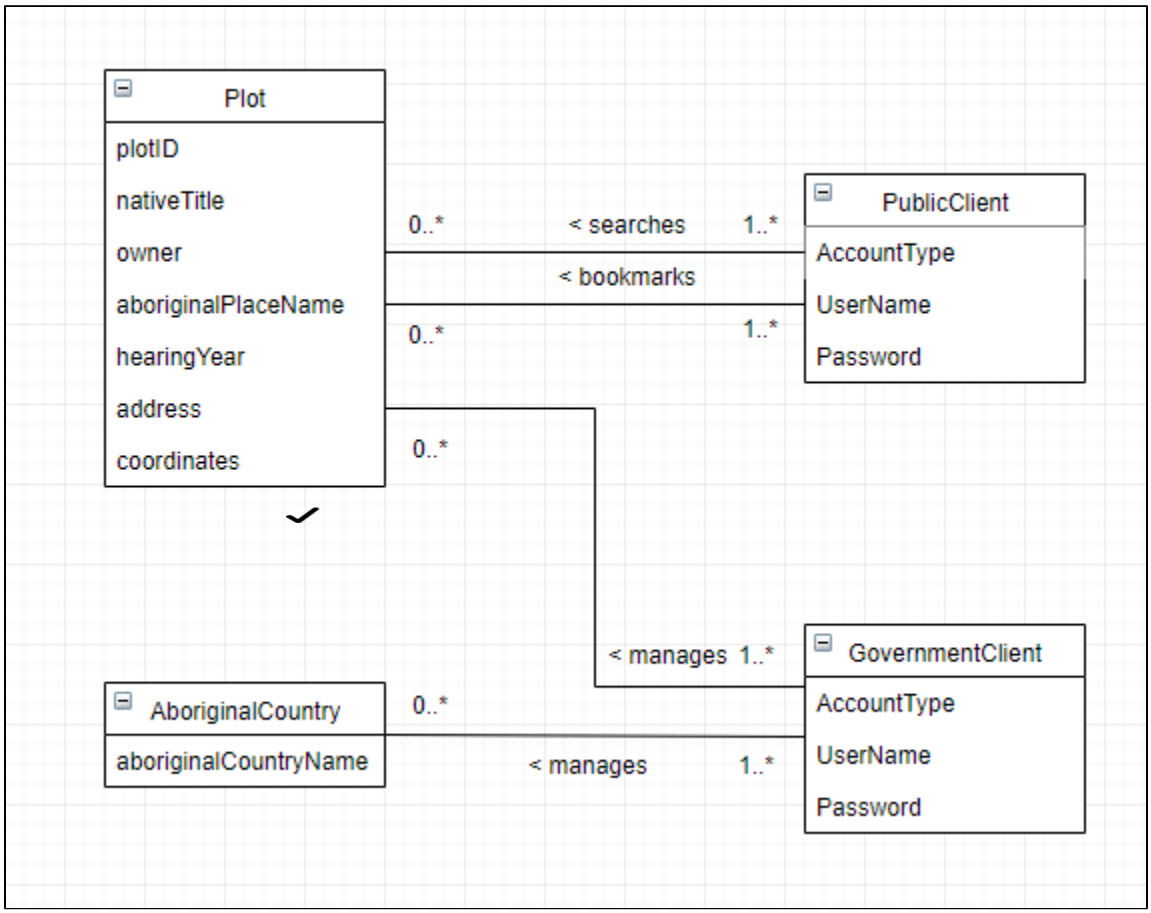
Architectural Documentation

4 + 1 View Model

Logical View (Domain Class Diagram):



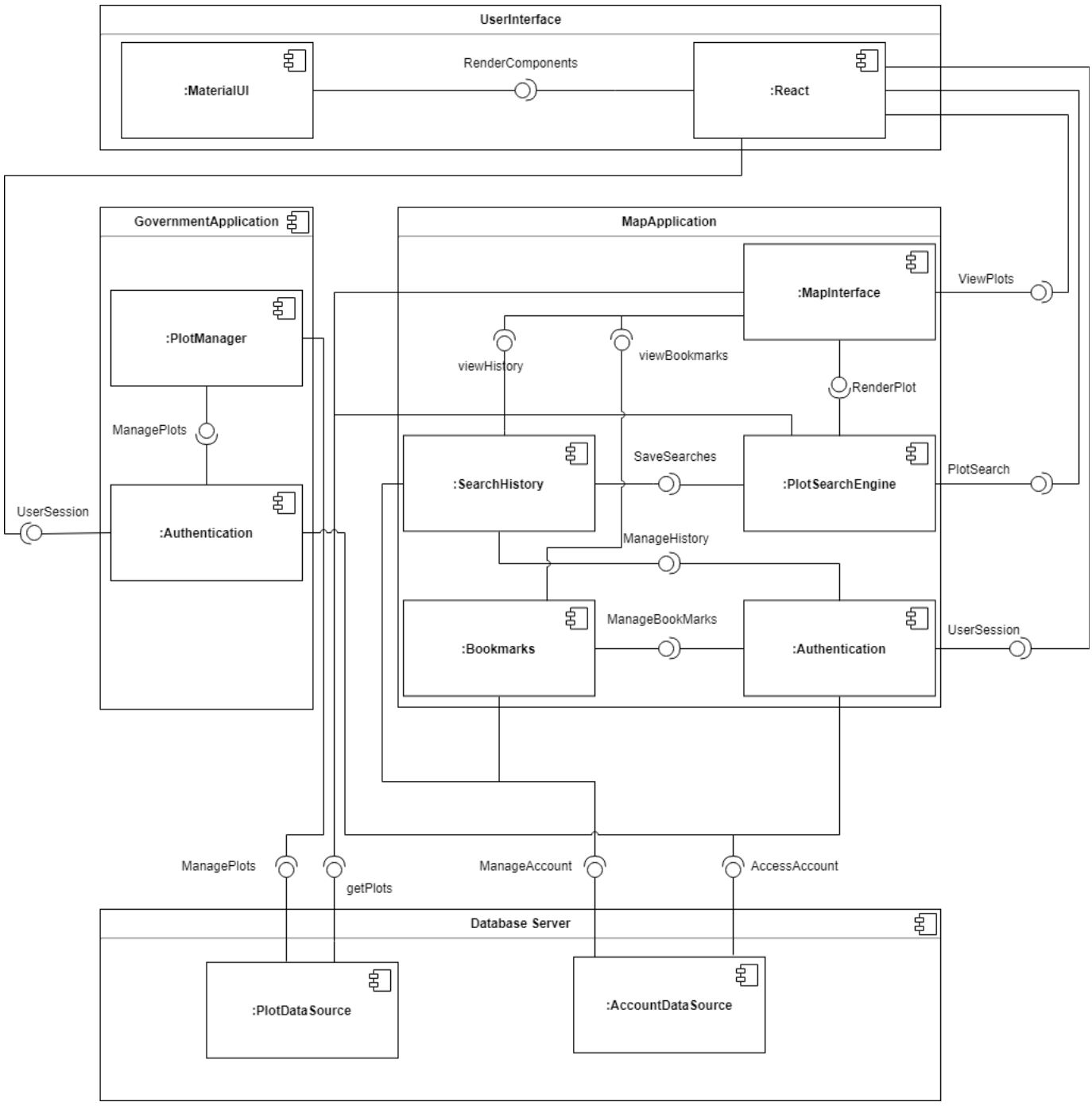
The main purpose of this system is to produce a map interface for public users to view and search plots, and to see details of these plots, with a focus on the Native Title status of the plot.

In order to store and manage these plots, another interface is implemented. Administrative users, such as government employees can use this interface to create new plots, as well as delete from a list of plots.

In general, public users can search all plots, and maintain a history of their searches. They can also save plots to their list of bookmarks.

Government users have the same privileges plus write privileges for all plots.

Implementation View (Component Diagram):

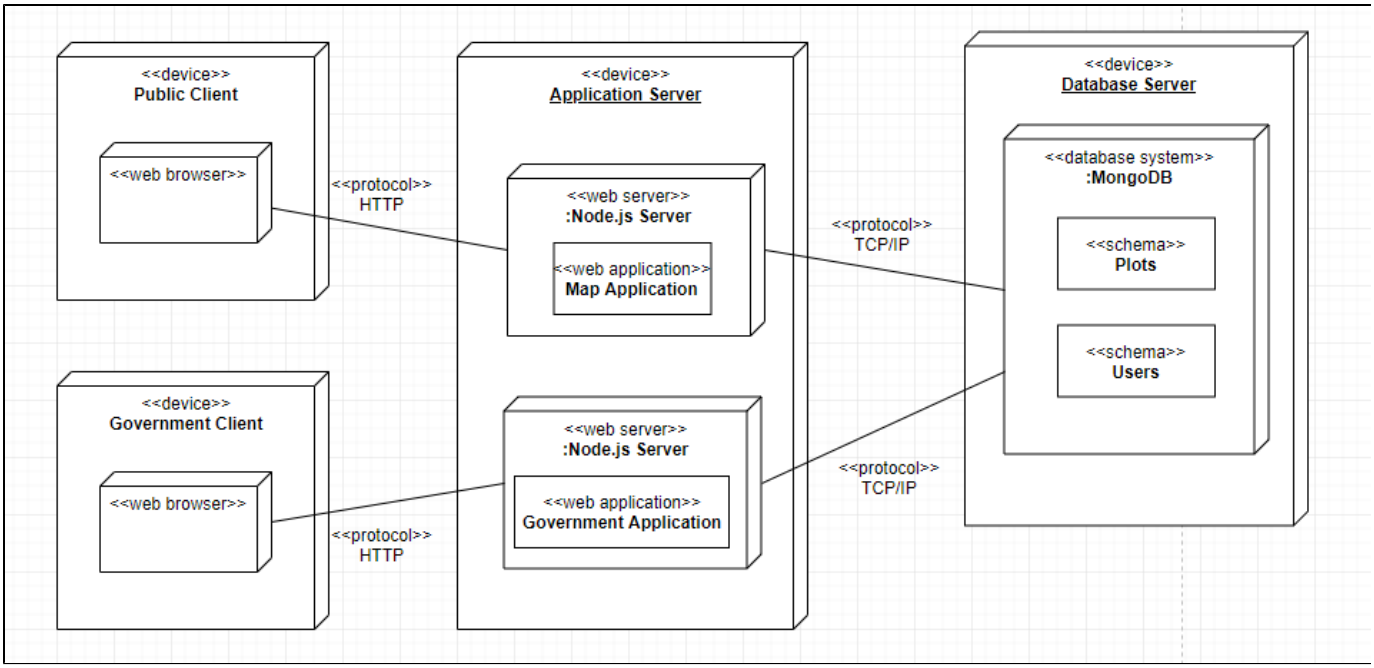


The system front-end will utilise React JS to handle the rendering logic, and will include the use of Material UI for rapid development of easy to understand UI components.

In the domain layer, the map and government application will act independently.

Both domain components will access the same database server, which holds two main data sources.

Deployment View (Deployment Diagram):

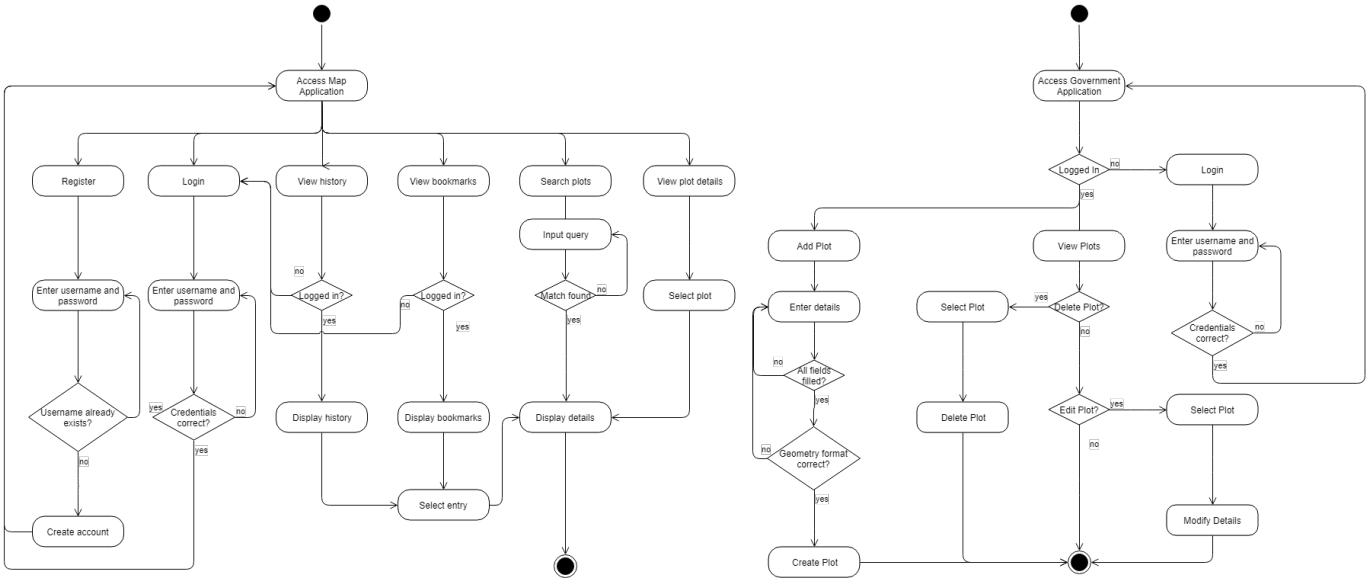


The map and government applications at this current stage is deployed on the same physical device, but on two separate virtual servers.

Public and government users will be accessing their respective applications on separate devices.

The database is currently hosted on a third-party server using mLab. If necessary, it is possible to relocate this to within the application server.

Process View (Activity Diagram):



Public users:

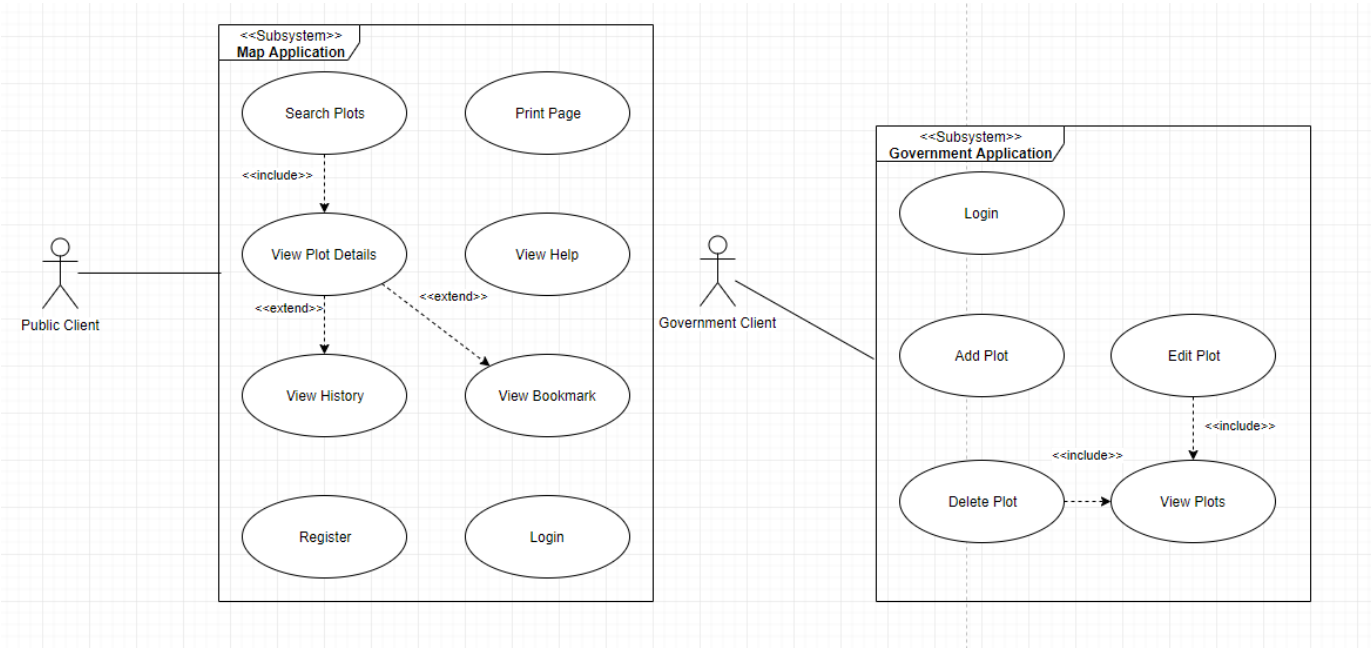
- Have the choice of registering an account and login with said account
- Once logged in, they will be able to make and view bookmarks. Their search history will also be recorded
- Searching and viewing plots do not require an account

Government users:

- Require an account to authenticate themselves for every session

- Can view a list of plots, from which they can choose and delete plots
- Can add a new plot, where all details must be filled, and the geometry field must conform to a correct JSON format

Use Case View (Use Case Diagram):



Alongside the use cases relating to plots, public users should also be able to print the page, and view a help page for assistance with using the system.

Design Justification:

The component diagram clearly displays our architectural design. Below, key design principles will be used to justify our system.

High cohesion: Our system is built upon a layered architecture. The key layers are the UI layer which includes the UserInterface,

the logical layer which includes the MapApp and GovernmentApp components and the data layer which includes our Database Server component.

By breaking down components into distinct layers, we have ensured that each of these components are highly cohesive with respect to their layer.

For example in the UI layer, the React component helps render pages for the end user, its role is contained only within UI related requirements.

Extensibility: Our front-end react component has been built to use any number of API interfaces. Likewise the back-end system is built

to support the addition of a variety of data parsing components. This makes the system more extensible for the addition of new components.

For example if the front-end system needed a new source of data for our map, a developer would just need to create a new sub-component in the

MapApplication component and the Front-end developer who is using React could simply contact that component via a new API endpoint that acts as

an interface between the components.

Maintainability: We have broken up our architectural design into major components and a variety of sub-components. By breaking up the

system into sub-components we have made our system a lot more modular. This means easier maintainability in the future.

For example, if the system needs an upgrade, for example an upgrade to user search history, a future developer would simply go to a sub-component

named ":SearchHistory:". Only this component would have to be manipulated, making maintainability an easier job. Likewise, our layered architecture helps

with this as well because developers who are good with back-end work can be assigned to maintain the data layer and UI experts to maintain the UI layer etc.

Lower Coupling: To improve the coupling of our system we have ensured that logical components only extract data from a relevant data source. We have also

decided to break our system up into a government application component and map application component so that the functionality of government users and public

users are decoupled.

Critical Attributes/Special Requirements:

- **Require a graphical map which can be drawn on:**
  A critical attribute that was specific to our project was that our system was required to include a large interactive map. Picking the React framework was a key
  architectural decision that helped us make this interactive map usable. The React component allowed our application to be a single page application. This is
  important for our map application, because a lot of interaction that occurs on our map should be updated live, without having to keep reloading the page/map.
- **Required to store JSON objects in the database:**
  The database system that we selected as part of our architectural design was mongoDB. The mongoDB server is written in javascript, the language to which
  JSON objects are native. Therefore, the insertion and extraction of JSON objects would be quite easy.

Defining Interfaces between our components:

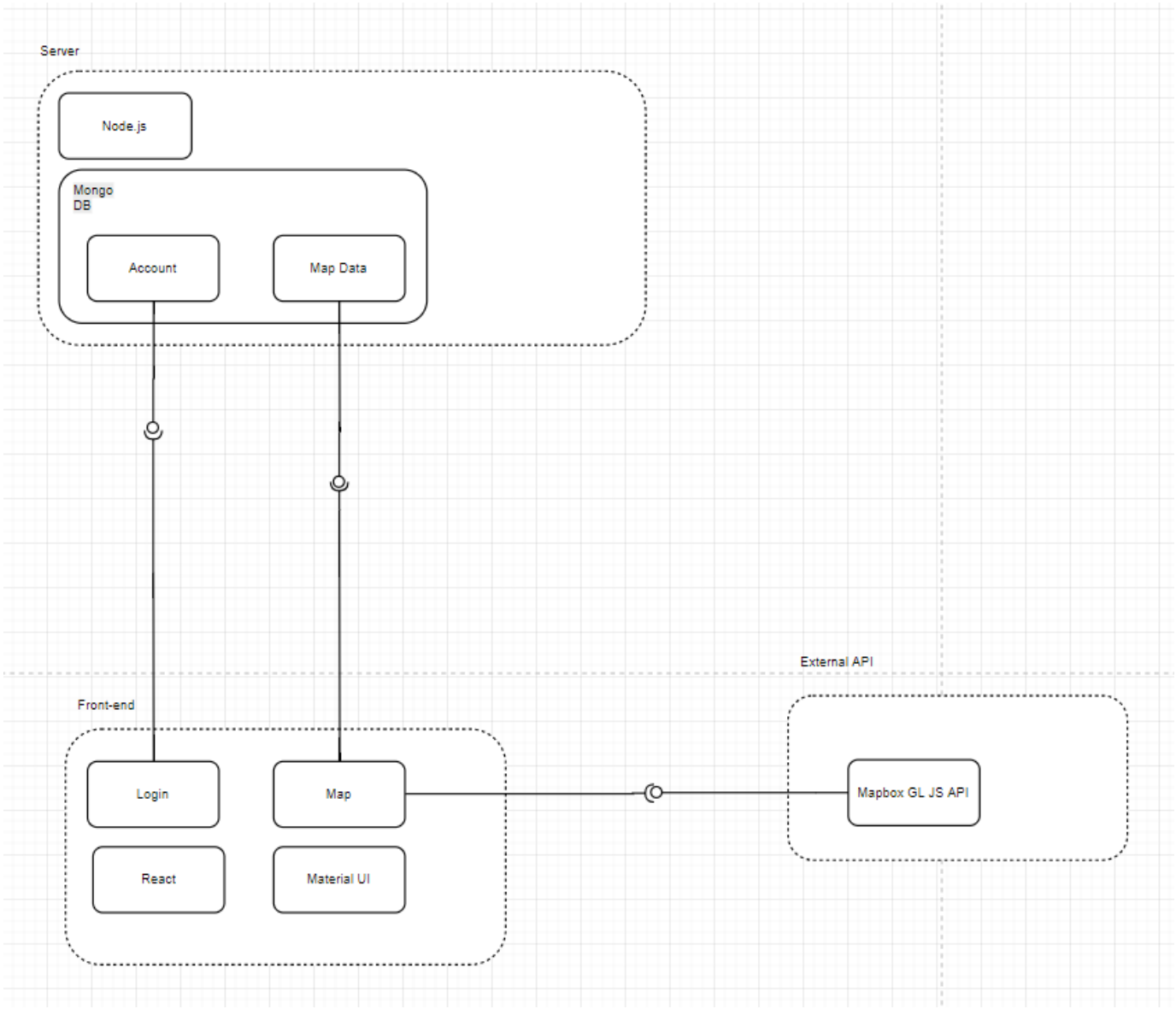
The API endpoints clearly defined the interfaces between the front-end and back-end. Detailed documentation is provided here: API Documentation.

Architectural Design Concept and Reuse Plan [DRAFTS]

Architectural Design Concept & Reuse Plan [Version 1]

Architectural Design Concept & Reuse Plan [Version 2]

Architectural Design Concept & Reuse Plan [Version 1]



Here is a draw.io link to the diagram: https://drive.google.com/file/d/14DZaZPz0Slk0sKQOgNSRX5FQFd_E_yQm/view?usp=sharing edited 20/08/18

Development Stack

Given our team's previous experience and proficiency, we determined that we will be using the MERN stack - MongoDB, Express, React, and Node.js.

By doing so, the team members with previous experience with different components of the frontend and backend are capable of mentoring and providing assistance to those less familiar.

React will also meet the provided architectural requirements concerning the web browsers that the application should run on.

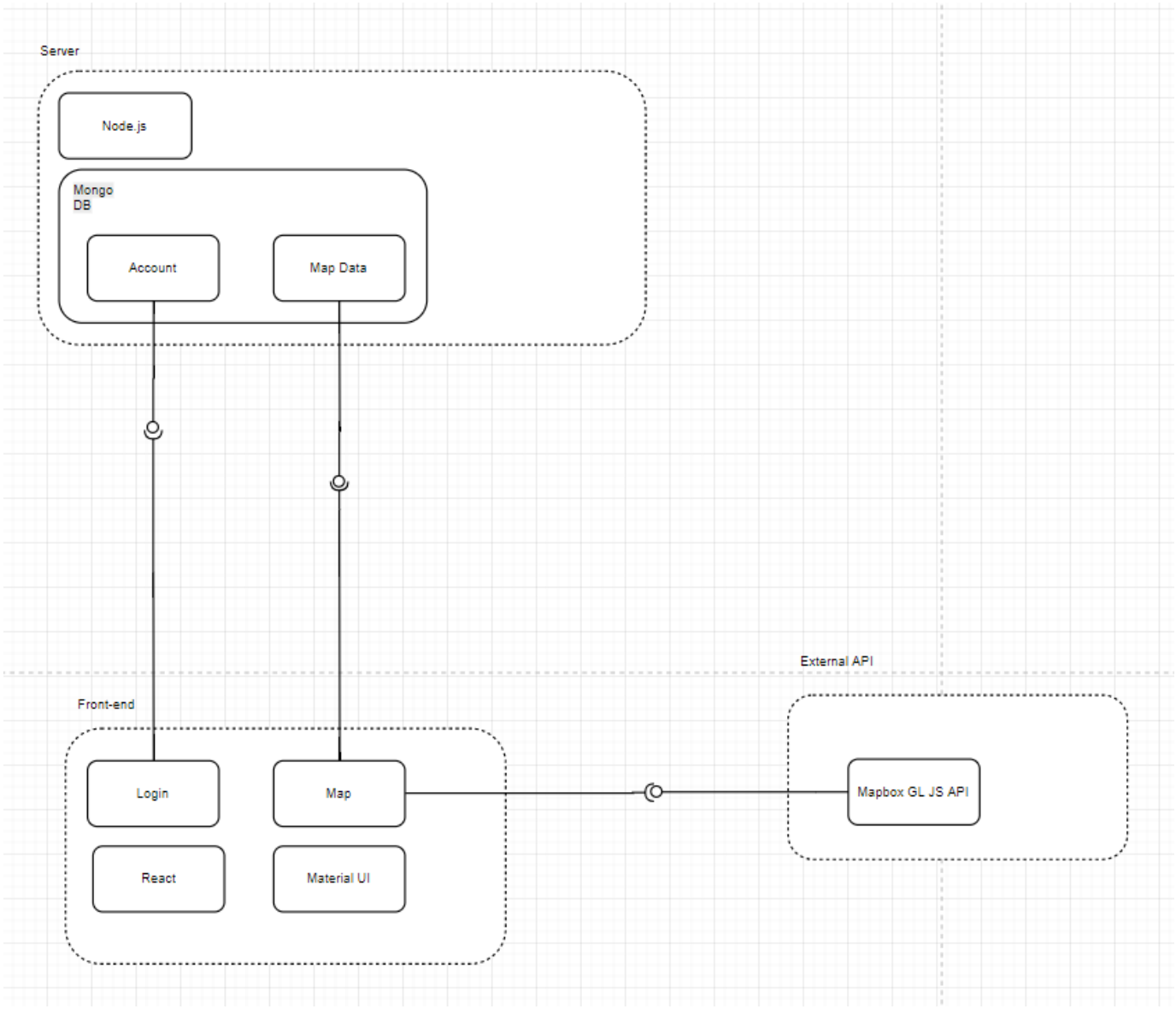
Frameworks and Libraries

In addition to the aforementioned software stack, we will be using uber/react-map-gl, which is a wrapper for the MapboxGL JS library.

We will also be using reduxjs/react-redux, which are the official React bindings for Redux, a predictable state container, and Material-UI, for React components that implement Google's Material Design.

The axios/axios package will be used to manage HTTP requests to the server.

Architectural Design Concept & Reuse Plan [Version 2]



Here is a draw.io link to the diagram: https://drive.google.com/file/d/14DZaZPz0Slk0sKQOgNSRX5FQFd_E_yQm/view?usp=sharing edited 20/08/18

Architecture and Development Stack

This architecture serves a back-end node.js server that is powered by express that enables communication to our non-relational monogoDB database. The back-end server provides

a series of endpoints which the front-end system can use to read and write data to the database. The front-end is encoded using react with a user interface that is powered by material UI and

boot-strap for a better user interface. The front-end map component is heavily reliant on the external Mapbox GL JS API to provide tools for manipulating the display map.

Interfaces between these components: The API endpoints between the front-end and back-end have detailed documentation here: API Documentation.

Design notebook:

Given our team's previous experience and proficiency, we determined that we will be using the MERN stack - MongoDB, Express, React, and Node.js.

By doing so, the team members with previous experience with different components of the frontend and backend are capable of mentoring and

providing assistance to those less familiar.

React will also meet the provided architectural requirements concerning the web browsers that the application should run on. Also using a non-relational database means much easier

manipulation of the data, as well as more flexibility for how data is stored. This stack almost exclusively uses java script and therefore all of the components are built using

java script which results in easier communication throughout the system.

Frameworks and Libraries

In addition to the aforementioned software stack, we will be using the MapboxGL JS library.

We will also be using reduxjs/react-redux, which are the official React bindings for Redux, a predictable state container, and Material-UI, for React components that implement Google's Material Design.

The erikras/redux-form package was used to complement the creation of forms.

The axios/axios package will be used to manage HTTP requests to the server.

typicode/husky and prettier/prettier were also used to format the code before commits, in order to maintain consistency with code quality on the online repository.

Re-use Plan

Mapbox GL JS: The mapbox GL JS component is the most important re-use component. This component does a lot of heavy lifting in terms of rendering and shading sections of a map or overlaying table and data on the map.

It provides a variety of endpoints that makes manipulation of the map easy.

Material UI: The other component that is also a re-use component is matierial UI. This component makes use of a great UI library that provides

a variety of css objects that are native to the react environment and can easily be embedded. This saves some heavy lifting in terms of styling.

MongoDB: Relational database that will be re-used with some development efforts in order to set-up communication with our express based server.

Node.js + Express.js: Server infrastructure that enables server development effort, not much re-use - code heavily developed here.

Map: Embedded in react but uses the MapBox Api (re-use component).

React: Front-end framework that will require heavy development, with not much re-use available.

References to Learning Resources

Map Box: https://www.mapbox.com/mapbox-gl-js/api/#map

Geo JSON: http://geojson.org/

Materil UI: https://v0.material-ui.com/#/

React Redux: https://redux.js.org/

Redux-form: https://redux-form.com/7.4.2/docs/gettingstarted.md/

Prettier: https://prettier.io/docs/en/precommit.html

Mocha and Chai Testing framework tutorial (back end): https://scotch.io/tutorials/test-a-node-restful-api-with-mocha-and-chai

Enzyme and Jest Testing frameworks (front end): https://testdriven.io/tdd-with-react-jest-and-enzyme-part-one#.WtSg_n4B-t4.reddit

Video for setting up a full MERN application: https://www.youtube.com/watch?v=PBTYxXADG_k&list=PLillGF-RfqbbiTGgA77tGO426V3hRF9iE

Bootstrap: https://getbootstrap.com/

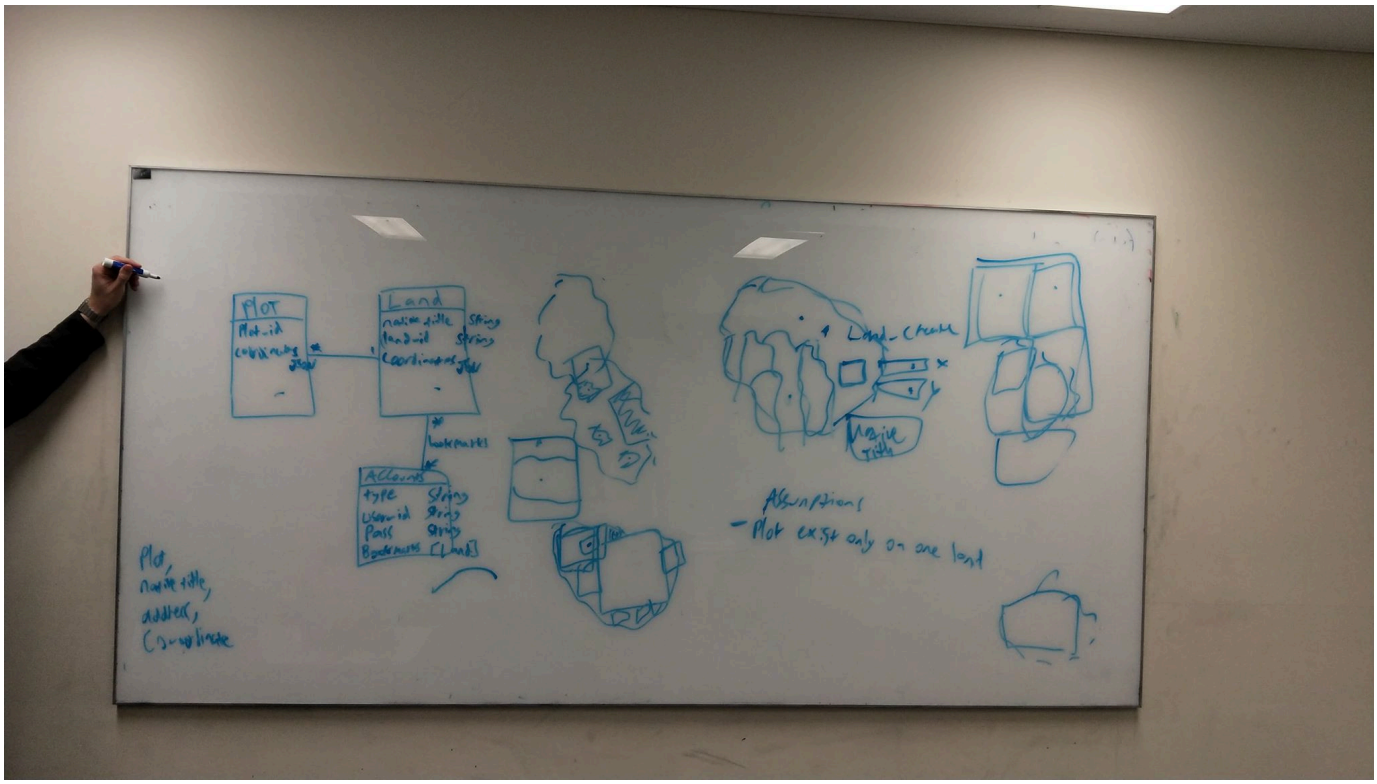
JIRA Ticket 1: **ABBI1418S-79** - Authenticate to see issue details

JIRA Ticker 2: **ABBI1418S-74** - Authenticate to see issue details

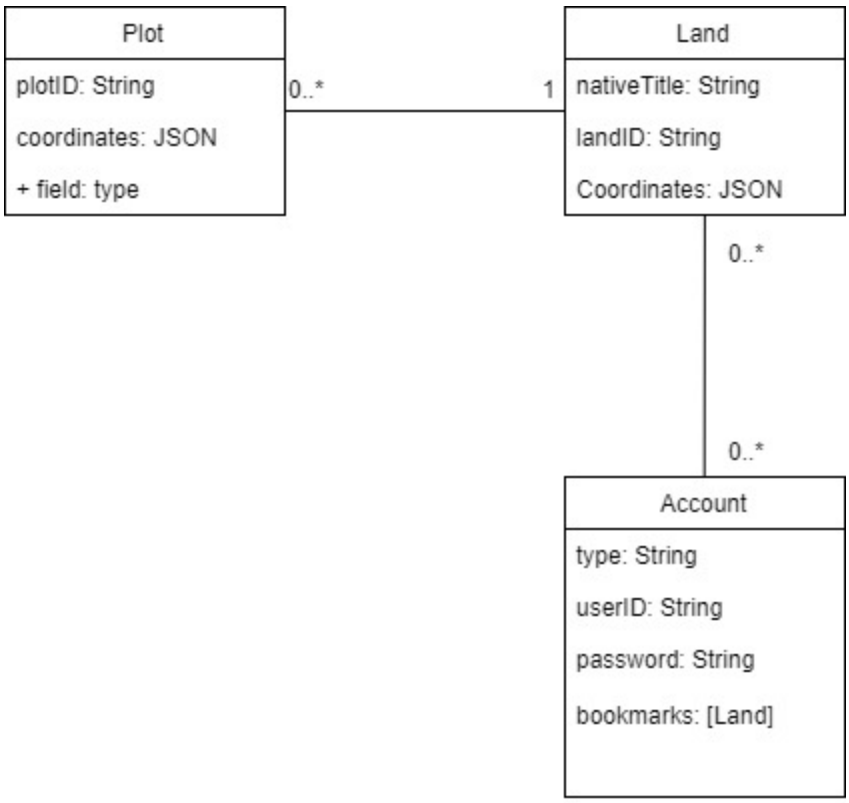
Database Model [DRAFTS]

Database Model (Version 1)

Here is the initial database model that we drew. The detailed diagram is displayed below:



Database Model:



here is a link: https://drive.google.com/file/d/1D0fZXxsWuejnBLnWiARKAx0VFEQSoqfL/view?usp=sharing

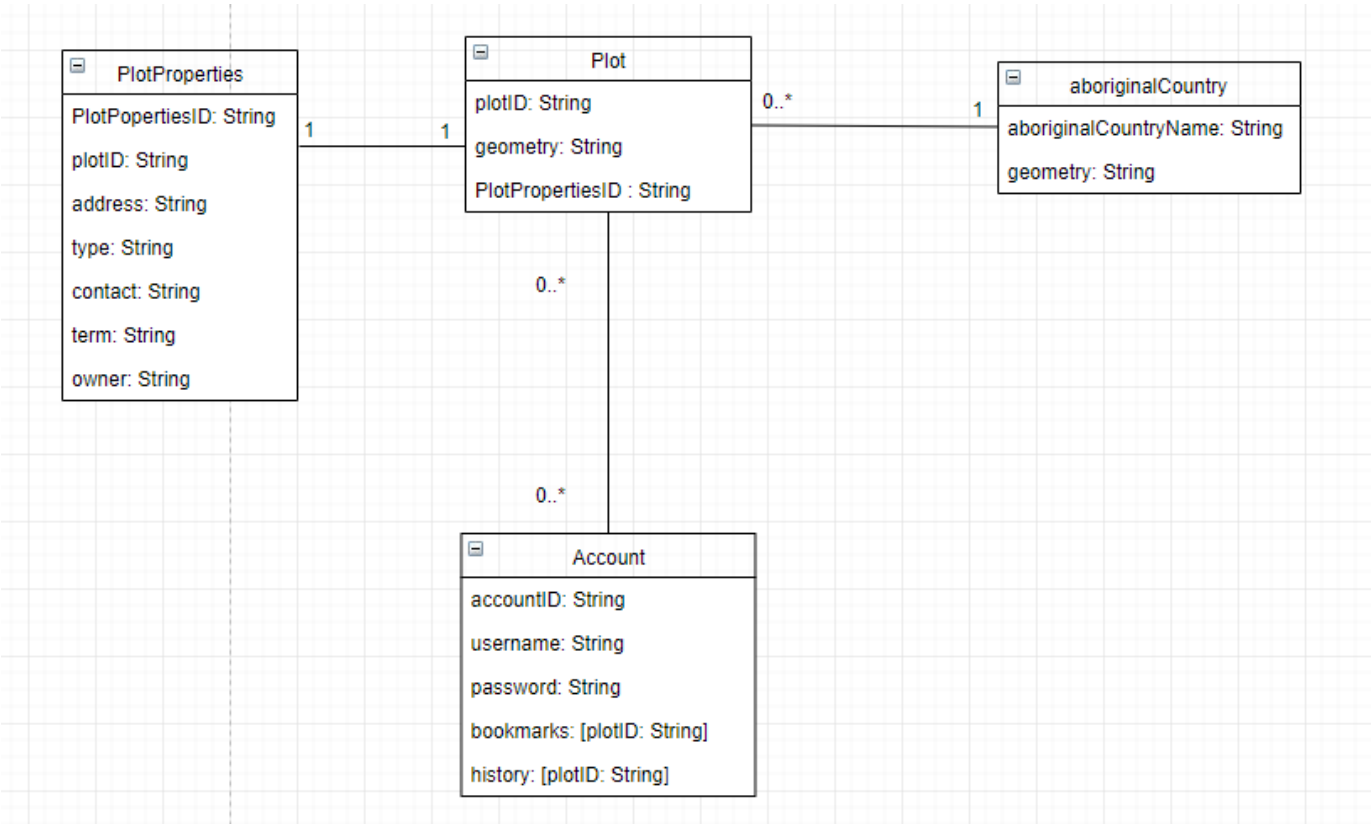
Justifying the Model:

- Every piece of land can be comprised of 0 to multiples plots. Assumption: A plot can only exist on one piece of land not on multiple pieces of land. (No overlaps of land?)
- Each plot, or piece of land will have a JSON object containing the set of co-ordinates that make up the (boundaries of) land or plot area.
- Every account contains and maintains bookmarks of land that the account has bookmarked.
- Achieves searching for plots, due to plot ids, searching for land due to land id. Can search land by native title as each piece of land has a native title.
- Assume each piece of land has one native title.

Database Model (Version 2)

New database model to reflect the storing of geojson and client requirement of aboriginalCountry land areas.

Database Model:



Link to the diagram: https://drive.google.com/file/d/1D0fZXxsWuejnBLnWiARKAx0VFEQSoqfL/view?usp=sharing

Justifying the Model:

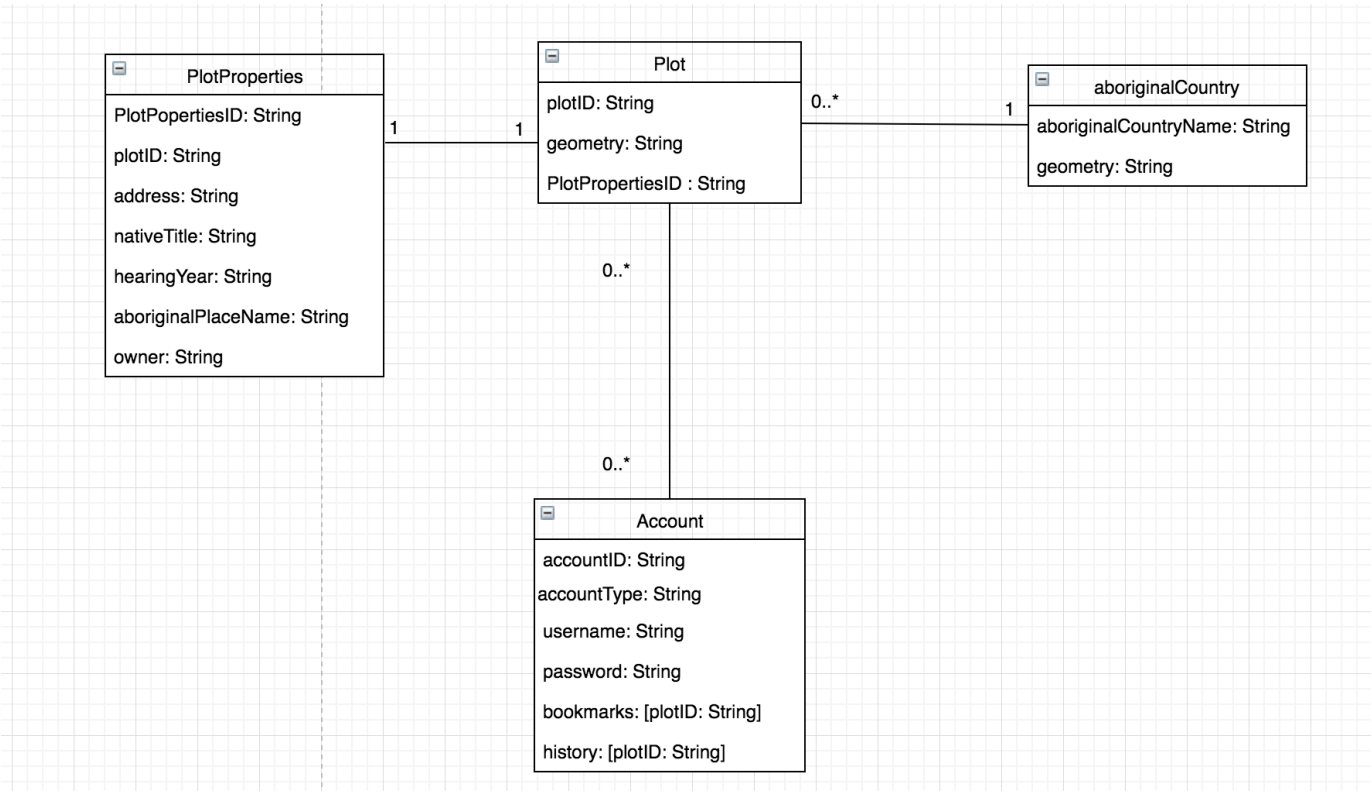
- Land changed to DetArea to accommodate for new nomenclature
- Account now associated with plots instead, as determination areas are currently not deemed as the main objects of concern to users
- PlotProperties separated into another table for clearer database structure and improved search under properties

Relevant JIRA task: **ABBI1418S-48** - Authenticate to see issue details

Database Model (Version 3)

New database model to reflect the client requirement of plotProperties.

Database Model:



Link to the diagram: https://drive.google.com/file/d/1D0fZXxsWuejnBLnWiARKAx0VFEQSoqfL/view?usp=sharing

Justifying the Model:

- DetArea changed to AboriginalCountry to accommodate for new nomenclature
- Account now associated with plots instead, as determination areas are currently not deemed as the main objects of concern to users
- PlotProperties separated into another table for clearer database structure and improved search under properties

Relevant JIRA task: **ABBI1418S-48** - Authenticate to see issue details

Security & Authentication

We have decided to implement an authentication token handling system. This process affects both the front-end and the back-end. Below is a description of how the system works on both the front-end and the back-end.

Front-end Handling

- When a user logs in, they are provided an authentication token.
- The front-end can cache this token in the browser's local storage.
- Users can use the authentication token for any requests they make to the back-end.
- When a user logs out, their token will be removed from local storage.
- When their session has continued for too long, their authentication token will expire and won't be valid.

Back-end Handling

- When a user attempts to log-in they send a request to log in
- Back-end checks if the username and password of the user is valid.
- If the user and password is valid, back end will generate an authentication token and send it in a response to the front-end.
- Back-end generates this token by using the current date and the users identification details.
- Any tokens sent to the back-end are checked for validity: i.e if the token has expired or not.

Development Documentation

- API Documentation

API Documentation

Add a new plot - GOVCLIENT [DRAFTS]

Add a new plot - GOVCLIENT [Version 1]

Description

Adding a new plot and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/addPlot
```

Parameters

Parameter		Description
required	plot	A JSON Object following the GeoJSON specification which includes type, properties, and geometry.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/addPlot
Host: https://[REDACTED]/api
Body:
{
    "plot":
    {
        "type": "Feature",
        "properties:" {
            "plotType": "plot",
   "title": "Example Title",
   "id": "HTC23432",
   "name": "Example Name",
   "owner" : "Jack Smith",
   "address": "23423 Apple St",
   "timeLimit": "2 years",
   "phoneNumber": "+6132433"
        },
        "geometry": {
            "type": "MultiPolygon", // can also be Polygon
            "coordinates": [
                [
                    [
                        [1234, 2345],
                        [2345, 3456],
                        ...
                    ]
                ],
                ...
            ]
        }
    }
}
```

Example Responses

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request" }
```

Add a new plot - GOVCLIENT [Version 2]

Version 2: Geometry object converted to string. Updated to match server.

Description

Adding a new plot and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/addPlot
```

Parameters

Parameter		Description
required	properties	A JSON Object containing the required information about the plot
required	geometry	A string representing the JSON Object that contains the geometry information of the plot, following the GeoJSON specification.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/addPlot
Host: https://[REDACTED]/api
Body:
{
    "properties:" {
  "plotID": "FHRT90035",
  "type": "Free Hold",
  "owner": "Charles Park",
  "term": "2 years",
  "contact": "+61420570622",
  "address": "1002/421, Docklands River Melbourne, Victoria"
    },
    "geometry": "{
        "type": "MultiPolygon", // can also be Polygon
        "coordinates": [
            [
                [
                    [1234, 2345],
                    [2345, 3456],
                    ...
                ]
            ],
            ...
        ]
    }"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Add a new plot - GOVCLIENT [Version 3]

Version 3: Plot properties updated to reflect client requests

Description

Adding a new plot and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/addPlot
```

Parameters

Parameter		Description
required	properties	A JSON Object containing the required information about the plot
required	geometry	A string representing the JSON Object that contains the geometry information of the plot, following the GeoJSON specification.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/addPlot
Host: https://[REDACTED]/api
Body:
{
    "properties:" {
  "plotID": "FHRT90035",
  "nativeTitle": "Free Hold",
  "owner": "Wudjari People",
  "aboriginalPlaceName": "Wudjari",
  "hearingYear": "2014",
  "address": "1002/421, Docklands River Melbourne, Victoria"
    },
    "geometry": "{
        "type": "MultiPolygon", // can also be Polygon
        "coordinates": [
            [
                [
                    [1234, 2345],
                    [2345, 3456],
                    ...
                ]
            ],
            ...
        ]
    }"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Add a new plot - GOVCLIENT [Version 4]

Version 4: Updated to include authorization token

Description

Adding a new plot and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/addPlot
```

Parameters

Parameter		Description
required	properties	A JSON Object containing the required information about the plot
required	geometry	A string representing the JSON object following the GeoJSON specifcation that contains the geometry information of the aboriginal country.
required	id	A string representing the id of the user in the db
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/addPlot
Host: https://[REDACTED]/api
Header:
{

authorization:"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b3
3774128cd8c874763a399026c6226cafe"
}
Body:
{
 "id" : "gwrhegf4wrf4t5et45324rfwed",
    "properties:" {
  "plotID": "FHRT90035",
  "nativeTitle": "Free Hold",
  "owner": "Wudjari People",
  "aboriginalPlaceName": "Wudjari",
  "hearingYear": "2014",
  "address": "1002/421, Docklands River Melbourne, Victoria"
    },
    "geometry": "{
        "type": "MultiPolygon", // can also be Polygon
        "coordinates": [
            [
                [
                    [1234, 2345],
                    [2345, 3456],
                    ...
                ]
            ],
            ...
        ]
    }"
}
```

Example Responses

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Adding a new aboriginal country - GOVCLIENT [DRAFTS]

Adding a new aboriginal country - GOVCLIENT [Version 4]

Version 4: Updated to change determination area to aboriginal country as per client request

Description

Adding a new aboriginal country and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/addAboriginalCountry
```

Parameters

Parameter		Description
required	aboriginalCountryName	A string representing the name of the aboriginal country name.
required	geometry	A string representing the JSON object following the GeoJSON specifcation that contains the geometry information of the aboriginal country.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/addAboriginalCountry
Host: https://[REDACTED]/api
Body:
{
 "aboriginalCountryName" : "some name",
 "geometry": "{
  "type": "MultiPolygon", // can also be Polygon
  "coordinates": [
   [
    [
     [1234, 2345],
     [2345, 3456],
     ...
    ]
   ],
   ...
  ]
 }"
}
```

Example Responses

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request" }
```

Adding a new aboriginal country - GOVCLIENT [Version 5]

Version 5: Updated to use authorization token

Description

Adding a new aboriginal country and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/addAboriginalCountry
```

Parameters

Parameter	Description

required	aboriginalCountryName	A string representing the name of the aboriginal country name.
required	geometry	A string representing the JSON object following the GeoJSON specifcation that contains the geometry information of the aboriginal country.
required	id	A string representing the id of the user in the db
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/addAboriginalCountry
Host: https://[REDACTED]/api
Header:
{

authorization:"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b3
3774128cd8c874763a399026c6226cafe"
}
Body:
{
    "id" : "bfleeqgfu3oqr3qu5t894ufu390",
 "aboriginalCountryName" : "some name",
 "geometry": "{
  "type": "MultiPolygon", // can also be Polygon
  "coordinates": [
   [
    [
     [1234, 2345],
     [2345, 3456],
     ...
    ]
   ],
   ...
  ]
 }"
}
```

Example Responses

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Adding a new determination area - GOVCLIENT [Version 1]

Description

Adding a new determination area and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/addDetArea
```

Parameters

Parameter		Description
required	boundaries	A JSON Object containing an array of the coordinates used to form the boundaries of the determination area.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/addDetArea
Host: https://[REDACTED]/api
Body:
{
 "boundaries": "{[(1234, 2345),
     (2345, 3456),
     ...]}"
}
```

Example Responses

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request" }
```

Adding a new determination area - GOVCLIENT [Version 2]

Version 2: adapted to follow GeoJSON Specification: http://geojson.org/

Description

Adding a new determination area and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/addDetArea
```

Parameters

Parameter		Description
required	detArea	A JSON Object following the GeoJSON specification which includes type, properties, and geometry.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/addDetArea
Host: https://[REDACTED]/api
Body:
{
 "detArea":
 {
  "detAreaName" : "some name",
  "geometry": "{
   "type": "MultiPolygon", // can also be Polygon
   "coordinates": [
    [
     [
      [1234, 2345],
      [2345, 3456],
      ...
     ]
    ],
    ...
   ]
  }"
 }
}
```

Example Responses

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request" }
```

Adding a new determination area - GOVCLIENT [Version 3]

Version 3: Updated to match server implementation.

Description

Adding a new determination area and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/addDetArea
```

Parameters

Parameter		Description
required	detAreaName	A string representing the name of the determination area.
required	geometry	A string representing the JSON object following the GeoJSON specifcation that contains the geometry information of the determination area.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/addDetArea
Host: https://[REDACTED]/api
Body:
{
 "detAreaName" : "some name",
 "geometry": "{
  "type": "MultiPolygon", // can also be Polygon
  "coordinates": [
   [
    [
     [1234, 2345],
     [2345, 3456],
     ...
    ]
   ],
   ...
  ]
 }"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Add to user bookmarks - CLIENT [DRAFTS]

- Add to user bookmarks - CLIENT [Version 1]

Add to user bookmarks - CLIENT [Version 1]

Description

Add a plot to the bookmarks for a currently logged in account.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/addBookmark
```

Parameters

Parameter		Description
required	id	A string representing the id of the user in the db
required	plotID	A string representing the ID of the plot to be added to the bookmarks
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/addBookmark
Host: https://[REDACTED]/api
Header:
{

authorization:"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b3
3774128cd8c874763a399026c6226cafe"
}
Body: {
 id: "5ba5d9e1e942371d58da684c",
 plotID: "AB123" }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Add to user history - CLIENT [DRAFTS]

- Add to user history - CLIENT [Version 1]

Add to user history - CLIENT [Version 1]

Description

Add a plot to the history for a currently logged in account.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/addHistory
```

Parameters

Parameter		Description
required	id	A string representing the id of the user in the db
required	plotID	A string representing the ID of the plot to be added to the history
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/addHistory
Host: https://[REDACTED]/api
Header:
{

authorization:"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b3
3774128cd8c874763a399026c6226cafe"
}
Body: {
 id: "5ba5d9e1e942371d58da684c",
 plotID: "AB123" }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

API Documentation Template [Version 1]

Description

Description describing functionality of API endpoint.

This method only accepts PUT/POST/GET/DELETE requests.

Example:

Adding a new land title and its details for the application.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapGovData/addLandTitle
```

Parameters

Example:

Parameter		Description
required	nativeTitle	A string identifying the name
required	boundaries	A JSON Object containing an array of the coordinates used to form the boundaries of the land title.
... continue as shown...		

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/addLandTitle
Host: https://[REDACTED]/api
Body:
{
 "nativeTitle": "Native Title Example Name",
 "boundaries": {[(1234, 2345),
     (2345, 3456),
     ...]}
}
```

Example Responses

Examples:

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Delete an existing aboriginal country - GOVCLIENT [DRAFTS]

Delete an existing aboriginal country - GOVCLIENT [Version 2]

Updated from Version 1: Updated determination area to aboriginal country as per client request

Description

Delete an existing aboriginal country and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/deleteAboriginalCountry
```

Parameters

Parameter	Description

required	id	A string representing the ID of the aboriginal country in the server.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/deleteAboriginalCountry
Host: https://[REDACTED]/api
Body:
{
 "id" : "5b8ff988b77d8148fc043a46"
}
```

Example Responses

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request" }
```

Delete an existing aboriginal country - GOVCLIENT [Version 3]

Description

Version 3: updated to include authorization token

Delete an existing aboriginal country and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/deleteAboriginalCountry
```

Parameters

Parameter	Description

required	id	A string representing the ID of the aboriginal country in the server.
required	userID	A string representing the id of the user in the db
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/deleteAboriginalCountry
Host: https://[REDACTED]/api
Header:
{

authorization:"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b3
3774128cd8c874763a399026c6226cafe"
}
Body:
{
 "id" : "5b8ff988b77d8148fc043a46",
 "userID" : "fwrty498roh3290r32e"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Delete an existing determination area - GOVCLIENT [Version 1]

Description

Delete an existing determination area and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/deleteDetArea
```

Parameters

Parameter		Description
required	id	A string representing the ID of the determination area in the server.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/deleteDetArea
Host: https://[REDACTED]/api
Body:
{
 "id" : "5b8ff988b77d8148fc043a46"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Delete an existing plot - GOVCLIENT [DRAFTS]

Delete an existing plot - GOVCLIENT [Version 1]

Description

Delete an existing plot and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/deletePlot
```

Parameters

Parameter		Description
required	plotID	A string representing the ID of the plot.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/deletePlot
Host: https://[REDACTED]/api
Body:
{
 "plotID": "AB1"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Delete an existing plot - GOVCLIENT [Version 2]

Description

Version 2: Updated to include authorization token

Delete an existing plot and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/deletePlot
```

Parameters

Parameter		Description
required	plotID	A string representing the ID of the plot.
required	id	A string representing the id of the user in the db
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/deletePlot
Host: https://[REDACTED]/api
Header:
{

authorization:"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b3
3774128cd8c874763a399026c6226cafe"
}
Body:
{
 "plotID": "AB1",
 "id" : "feuiwy8239yr2380h48th408r"
}
```

Example Responses

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Delete from user bookmarks - CLIENT [DRAFTS]

- Delete from user bookmarks - CLIENT [Version 1]

Delete from user bookmarks - CLIENT [Version 1]

Description

Delete a plot from the bookmarks for a currently logged in account.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/deleteBookmark
```

Parameters

Parameter		Description
required	id	A string representing the id of the user in the db
required	plotID	A string representing the ID of the plot to be deleted to the history
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/deleteBookmark
Host: https://[REDACTED]/api
Header:
{
    authorization:
"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b33774128cd8c874
763a399026c6226cafe"
}
Body: {
 id: "5ba5d9e1e942371d58da684c",
 plotID: "AB123" }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Delete from user history - CLIENT [DRAFTS]

- Delete from user history - CLIENT [Version 1]

Delete from user history - CLIENT [Version 1]

Description

Delete a plot from the history for a currently logged in account.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/deleteHistory
```

Parameters

Parameter		Description
required	id	A string representing the id of the user in the db
required	plotID	A string representing the ID of the plot to be deleted to the history
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/deleteHistory
Host: https://[REDACTED]/api
Header:
{
    authorization:
"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b33774128cd8c874
763a399026c6226cafe"
}
Body: {
 id: "5ba5d9e1e942371d58da684c",
 plotID: "AB123" }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Edit an existing aboriginal country - GOVCLIENT [DRAFTS]

- Edit an existing aboriginal country - GOVCLIENT [Version 2]
- Edit an existing aboriginal country - GOVCLIENT [Version 3]
- Edit an existing determination area - GOVCLIENT [Version 1]

Edit an existing aboriginal country - GOVCLIENT [Version 2]

Version 2: Updated to change determination area to aboriginal country as per client request

Description

Edit an existing aboriginal country and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/editAboriginalCountry
```

Parameters

Parameter		Description
required	id	A string representing the ID of the aboriginal country in the server.
required	aboriginalCountryName	A string representing the name of the aboriginal country.
required	geometry	A string representing the JSON object of the geometry information of the aboriginal country, following the GeoJSON specification.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/editAboriginalCountry
Host: https://[REDACTED]/api
Body:
{
 "id" : "5b8ff988b77d8148fc043a46",
 "aboriginalCountryName" : "some name",
 "geometry": "{
  "type": "MultiPolygon", // can also be Polygon
  "coordinates": [
   [
    [
     [1234, 2345],
     [2345, 3456],
     ...
    ]
   ],
   ...
  ]
 }"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Edit an existing aboriginal country - GOVCLIENT [Version 3]

Version 3: Updated to include authorization token

Description

Edit an existing aboriginal country and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/editAboriginalCountry
```

Parameters

Parameter		Description
required	id	A string representing the ID of the aboriginal country in the server.
required	aboriginalCountryName	A string representing the name of the aboriginal country.
required	geometry	A string representing the JSON object of the geometry information of the aboriginal country, following the GeoJSON specification.
required	userID	A string representing the id of the user in the db
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/editAboriginalCountry
Host: https://[REDACTED]/api
Header:
{

authorization:"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b3
3774128cd8c874763a399026c6226cafe"
}
Body:
{
 "id" : "5b8ff988b77d8148fc043a46",
 "userID" : "fehuify49wry8934y38ue2",
 "aboriginalCountryName" : "some name",
 "geometry": "{
  "type": "MultiPolygon", // can also be Polygon
  "coordinates": [
   [
    [
     [1234, 2345],
     [2345, 3456],
     ...
    ]
   ],
   ...
  ]
 }"
}
```

Example Responses

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Edit an existing determination area - GOVCLIENT [Version 1]

Description

Edit an existing determination area and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/editDetArea
```

Parameters

Parameter		Description
required	id	A string representing the ID of the determination area in the server.
required	detAreaName	A string representing the name of the determination area.
required	geometry	A string representing the JSON object of the geometry information of the determination area, following the GeoJSON specification.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/editDetArea
Host: https://[REDACTED]/api
Body:
{
 "id" : "5b8ff988b77d8148fc043a46",
 "detAreaName" : "some name",
 "geometry": "{
  "type": "MultiPolygon", // can also be Polygon
  "coordinates": [
   [
    [
     [1234, 2345],
     [2345, 3456],
     ...
    ]
   ],
   ...
  ]
 }"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Edit an existing plot - GOVCLIENT [DRAFTS]

Edit an existing plot - GOVCLIENT [Version 1]

Description

Editing an existing plot and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

`https://[REDACTED]/api/mapGovData/editPlot`

Parameters

Parameter	Description

required	plotID	A string representing the ID of the plot.
required	properties	A JSON Object containing the required information about the plot.
required	geometry	A string representing the JSON Object that contains the geometry information of the plot, following the GeoJSON specification.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/editPlot
Host: https://[REDACTED]/api
Body:
{
 "plotID": "AB1",
    "properties": {
  "_id": "5b8fcc132a793d513c61f947",
  "plotID": "FHRT90035",
  "type": "Free Hold",
  "owner": "Charles Park",
  "term": "2 years",
  "contact": "+61420570622",
  "address": "1002/421, Docklands River Melbourne, Victoria"
    },
    "geometry": "{
        "type": "MultiPolygon", // can also be Polygon
        "coordinates": [
            [
                [
                    [1234, 2345],
                    [2345, 3456],
                    ...
                ]
            ],
            ...
        ]
    }"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Edit an existing plot - GOVCLIENT [Version 2]

Description

Editing an existing plot and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/editPlot
```

Parameters

Parameter		Description
required	oldPlotID	A string representing the current ID of the plot.
required	newPlotID	A string representing the new ID of the plot.
required	properties	A JSON Object containing the required information about the plot.
required	geometry	A string representing the JSON Object that contains the geometry information of the plot, following the GeoJSON specification.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/editPlot
Host: https://[REDACTED]/api
Body:
{
 "oldPlotID": "FHRT90035",
    "newPlotID": "AB1",
    "properties": {
  "_id": "5b8fcc132a793d513c61f947",
  "plotID": "FHRT90035",
  "type": "Free Hold",
  "owner": "Charles Park",
  "term": "2 years",
  "contact": "+61420570622",
  "address": "1002/421, Docklands River Melbourne, Victoria"
    },
    "geometry": "{
        "type": "MultiPolygon", // can also be Polygon
        "coordinates": [
            [
                [
                    [1234, 2345],
                    [2345, 3456],
                    ...
                ]
            ],
            ...
        ]
    }"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Edit an existing plot - GOVCLIENT [Version 3]

Version 3: Plot properties updated to reflect client requests

Description

Editing an existing plot and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/editPlot
```

Parameters

Parameter		Description
required	oldPlotID	A string representing the current ID of the plot.
required	newPlotID	A string representing the new ID of the plot.
required	properties	A JSON Object containing the required information about the plot.
required	geometry	A string representing the JSON Object that contains the geometry information of the plot, following the GeoJSON specification.

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/editPlot
Host: https://[REDACTED]/api
Body:
{
 "oldPlotID": "FHRT90035",
    "newPlotID": "AB1",
    "properties": {
  "_id": "5b8fcc132a793d513c61f947",
  "plotID": "FHRT90035",
        "nativeTitle": "Free Hold",
        "owner": "Wudjari People",
        "aboriginalPlaceName": "Wudjari",
        "hearingYear": "2014",
        "address": "1002/421, Docklands River Melbourne, Victoria"
    },
    "geometry": "{
        "type": "MultiPolygon", // can also be Polygon
        "coordinates": [
            [
                [
                    [1234, 2345],
                    [2345, 3456],
                    ...
                ]
            ],
            ...
        ]
    }"
}
```

Example Responses

Status Code

200

0K

Status Code

400

```
{ "error": "Bad Request" }
```

Edit an existing plot - GOVCLIENT [Version 4]

Version 4: Updated to include authorization token

Description

Editing an existing plot and its details for the application.

This method only accepts POST requests.

Note: Once accounts have been implemented, should be adapted to include authentication checks.

Resource URL

```
https://[REDACTED]/api/mapGovData/editPlot
```

Parameters

Parameter		Description
required	oldPlotID	A string representing the current ID of the plot.
required	newPlotID	A string representing the new ID of the plot.
required	properties	A JSON Object containing the required information about the plot.
required	geometry	A string representing the JSON Object that contains the geometry information of the plot, following the GeoJSON specification.
required	id	A string representing the id of the user in the db
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapGovData/editPlot
Host: https://[REDACTED]/api
Header:
{

authorization:"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b3
3774128cd8c874763a399026c6226cafe"
}
Body:
{
 "id" : "fh3iry389rr3r83u9oi3",
 "oldPlotID": "FHRT90035",
    "newPlotID": "AB1",
    "properties": {
  "_id": "5b8fcc132a793d513c61f947",
  "plotID": "FHRT90035",
        "nativeTitle": "Free Hold",
        "owner": "Wudjari People",
        "aboriginalPlaceName": "Wudjari",
        "hearingYear": "2014",
        "address": "1002/421, Docklands River Melbourne, Victoria"
    },
    "geometry": "{
        "type": "MultiPolygon", // can also be Polygon
        "coordinates": [
            [
                [
                    [1234, 2345],
                    [2345, 3456],
                    ...
                ]
            ],
            ...
        ]
    }"
}
```

Example Responses

Status Code

200

OK

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Login to the system - CLIENT/GOVCLIENT [DRAFTS]

- Login to the system - CLIENT/GOVCLIENT [Version 2]
- Login to the system - CLIENT [Version 1]

Login to the system - CLIENT/GOVCLIENT [Version 2]

Changes: Includes attribute to show which portal user is logging in from

Description

Logs in to system given a valid username and password. Returns authorization token and user id.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/login
```

Parameters

Parameter		Description
required	username	A string representing the username of the user
required	password	A string representing the password of the user
required	portal	A string representing the client portal - can either be "normal" or "government"

Response Fields

Response	Description
id	A string representing the id of the user in the db
token	A string representing the authorization token of the session

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/login
Host: https://[REDACTED]/api
Body: {
 username: "ejjymejjy",
 password: "jumblyjoo",
 portal: "normal"
 }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

```
{
 token: "tokeny232353dwefefwefewfb",
 id: "f1437rf8w3ydmumbojumbo"
}
```

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

500

```
{ "error": "Database error."}
```

Login to the system - CLIENT [Version 1]

Description

Logs in to system given a valid username and password. Returns authorization token and user id.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/login
```

Parameters

Parameter		Description
required	username	A string representing the username of the user
required	password	A string representing the password of the user

Response Fields

Response	Description
id	A string representing the id of the user in the db
token	A string representing the authorization token of the session

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/login
Host: https://[REDACTED]/api
Body: {
 username: "ejjymejjy",
 password: "jumblyjoo"
 }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

```
{
 token: "tokeny232353dwefefwefewfb",
 id: "f1437rf8w3ydmumbojumbo"
}
```

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

500

```
{ "error": "Database error."}
```

Performing a search for a plot- CLIENT [DRAFTS]

Performing a search for a plot [Version 1]

Description

Performing a search for a plot using the search bar present on the client application.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/searchPlot
```

Parameters

Example:

Parameter		Description
required	searchTerm	A string representing the input into the search bar
required	searchType	A string representing the selected search type (.e.g., "plotID", "address")

Response Fields

Response	Description
plot	A JSON object containing all relevant client-side information on the plot.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapData/searchPlot
Host: https://[REDACTED]/api
Body:
{
 "searchTerm": "FHRT90035",
 "searchType": "plotID"
}
```

Example Responses

Examples:

Status Code

200

```
{
 "plot":
  { "plotID" : "FHRT90035",
  "properties": {
   "plotID": "FHRT90035",
   "type": "Free Hold",
   "owner": "Charles Park",
   "term": "2 years",
   "contact": "+61420570622",
   "address": "1002/421, Docklands River Melbourne, Victoria",
   ...
  },
  "geometry": "{
   "type": "MultiPolygon",
   "coordinates": [
    [
     [
      [1234, 2345],
      [2345, 3456],
      ...
     ]
    ],
    ...
   ]
  }"
 }
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Performing a search for a plot [Version 2]

Version 2: Plot properties updated to reflect client requests

Description

Performing a search for a plot using the search bar present on the client application.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/searchPlot
```

Parameters

Example:

Parameter		Description
required	searchTerm	A string representing the input into the search bar
required	searchType	A string representing the selected search type (.e.g., "plotID", "address")

Response Fields

Response	Description
plot	A JSON object containing all relevant client-side information on the plot.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapData/searchPlot
Host: https://[REDACTED]/api
Body:
{
 "searchTerm": "FHRT90035",
 "searchType": "plotID"
}
```

Example Responses

Examples:

Status Code

200

```
{
 "plot":
  { "plotID" : "FHRT90035",
  "properties": {
   "plotID": "FHRT90035",
         "nativeTitle": "Free Hold",
         "owner": "Wudjari People",
         "aboriginalPlaceName": "Wudjari",
         "hearingYear": "2014",
         "address": "1002/421, Docklands River Melbourne, Victoria",
   ...
  },
  "geometry": "{
   "type": "MultiPolygon",
   "coordinates": [
    [
     [
      [1234, 2345],
      [2345, 3456],
      ...
     ]
    ],
    ...
   ]
  }"
 }
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Register a user - CLIENT/GOVCLIENT [DRAFTS]

- Register a user - CLIENT/GOVCLIENT [Version 1]

Register a user - CLIENT/GOVCLIENT [Version 1]

Description

Registers a user in the system given a valid username and password.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/addUser
```

Parameters

Parameter		Description
required	username	A string representing the username of the user
required	password	A string representing the password of the user
required	accountType	A string representing the client type - can either be "normal" or "government"

Response Fields

None

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/login
Host: https://[REDACTED]/api
Body: {
 username: "ejjymejjy",
 password: "jumblyjoo",
 accountType: "normal"
 }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

201

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

422

```
{ "error": "Username is taken."}
```

Status Code

500

```
{ "error": "Database error."}
```

Retrieve all information on single plot - CLIENT [DRAFTS]

Retrieve all information on single plot - CLIENT [Version 1]

Description

Retrieve all information on a single plot.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getPlotInformation
```

Parameters

Parameter		Description
required	plotID	A string identifying the plot ID.

Response Fields

Response	Description
plot	A JSON object containing all relevant client-side information on the plot.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapData/getPlotInformation
Host: https://[REDACTED]/api
Body:
{
 "plotID": "FHRT90035"

}
```

Example Responses

Examples:

Status Code

200

```
{
 "plot":
  {
  "plotID": "FHRT90035",
  "coordinates": [
   [
    [
     [1234, 2345],
     [2345, 3456],
     ...
    ]
   ],
   ...
  ],
  "type": "Free Hold",
  "owner": "Charles Park",
  "Term": "2 years",
  "Contact": "+61420570622",
  "Address": "1002/421, Docklands River Melbourne, Victoria",
  ...
 }
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieve all information on single plot - CLIENT [Version 2]

Description

Version 2: adapted to follow GeoJSON Specification: http://geojson.org/

Retrieve all information on a single plot.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getPlotInformation
```

Parameters

Parameter		Description
required	plotID	A string identifying the plot ID.

Response Fields

Response	Description
plot	A JSON object containing all relevant client-side information on the plot.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapData/getPlotInformation
Host: https://[REDACTED]/api
Body:
{
 "plotID": "FHRT90035"

}
```

Example Responses

Examples:

Status Code

200

```
{
 "plot":
  {
  "properties": {
   "plotID": "FHRT90035",
   "plotType": "Free Hold",
   "owner": "Charles Park",
   "Term": "2 years",
   "Contact": "+61420570622",
   "Address": "1002/421, Docklands River Melbourne, Victoria",
   ...
  }
  "geometry": {
   "type": "MultiPolygon",
   "coordinates": [
    [
     [
      [1234, 2345],
      [2345, 3456],
      ...
     ]
    ],
    ...
   ]
  }
 }
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieve all information on single plot - CLIENT [Version 3]

Description

Version 3: adapted to follow GeoJSON Specification: http://geojson.org/ and cater our db requirements.

Retrieve all information on a single plot.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getPlotInformation
```

Parameters

Parameter		Description
required	plotID	A string identifying the plot ID.

Response Fields

Response	Description
plot	A JSON object containing all relevant client-side information on the plot.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapData/getPlotInformation
Host: https://[REDACTED]/api
Body:
{
 "plotID": "FHRT90035"

}
```

Example Responses

Examples:

Status Code

200

```
{
 "plot":
  { "plotID" : "FHRT90035",
  "properties": {
   "plotID": "FHRT90035",
   "type": "Free Hold",
   "owner": "Charles Park",
   "term": "2 years",
   "contact": "+61420570622",
   "address": "1002/421, Docklands River Melbourne, Victoria",
   ...
  },
  "geometry": "{
   "type": "MultiPolygon",
   "coordinates": [
    [
     [
      [1234, 2345],
      [2345, 3456],
      ...
     ]
    ],
    ...
   ]
  }"
 }
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieve all information on single plot - CLIENT [Version 4]

Description

Version 4: **Plot properties updated to reflect client requests**

Retrieve all information on a single plot.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getPlotInformation
```

Parameters

Parameter		Description
required	plotID	A string identifying the plot ID.

Response Fields

Response	Description
plot	A JSON object containing all relevant client-side information on the plot.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
POST /mapData/getPlotInformation
Host: https://[REDACTED]/api
Body:
{
 "plotID": "FHRT90035"

}
```

Example Responses

Examples:

Status Code

200

```
{
 "plot":
  { "plotID" : "FHRT90035",
  "properties": {
   "plotID": "FHRT90035",
         "nativeTitle": "Free Hold",
         "owner": "Wudjari People",
         "aboriginalPlaceName": "Wudjari",
         "hearingYear": "2014",
         "address": "1002/421, Docklands River Melbourne, Victoria",
   ...
  },
  "geometry": "{
   "type": "MultiPolygon",
   "coordinates": [
    [
     [
      [1234, 2345],
      [2345, 3456],
      ...
     ]
    ],
    ...
   ]
  }"
 }
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieve bookmarks for an account - CLIENT [DRAFTS]

- Retrieve bookmarks for an account - CLIENT [Version 1]
- Retrieve bookmarks for an account - CLIENT [Version 2]

Retrieve bookmarks for an account - CLIENT [Version 1]

Description

Retrieving information on all plots in the bookmarks of an account, including plot IDs, plot properties and geometry.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/getUserBookmarks
```

Parameters

Parameter		Description
required	id	A string representing the id of the user in the db
required	token	A string representing the authorization token of the session

Response Fields

Response	Description
plots	An array containing the plot JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/getUserBookmarks
Host: https://[REDACTED]/api
Header:
{
    authorization: {token:
"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b33774128cd8c874
763a399026c6226cafe"}
}
Body: { id: "5ba5d9e1e942371d58da684c" }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

```
{
 "plots": [{
   {  "plotID" : "FHRT90035",

   "properties": {
            "plotID": "FHRT90035",
            "type": "Free Hold",
            "owner": "Charles Park",
            "term": "2 years",
            "contact": "+61420570622",
            "address": "1002/421, Docklands River Melbourne, Victoria",
        },
        "geometry": "{
            "type": "MultiPolygon",
            "coordinates": [
                [
                    [
                        [1234, 2345],
                        [2345, 3456],
                        ...
                    ]
                ],
                ...
            ]
        }"
 }]
}
```

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Retrieve bookmarks for an account - CLIENT [Version 2]

Version 2: Plot properties updated to reflect client requests

Description

Retrieving information on all plots in the bookmarks of an account, including plot IDs, plot properties and geometry.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/getUserBookmarks
```

Parameters

Parameter		Description
required	id	A string representing the id of the user in the db
required	token	A string representing the authorization token of the session

Response Fields

Response	Description
plots	An array containing the plot JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/getUserBookmarks
Host: https://[REDACTED]/api
Header:
{
    authorization: {token:
"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b33774128cd8c874
763a399026c6226cafe"}
}
Body: { id: "5ba5d9e1e942371d58da684c" }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

```
{
 "plots": [{
   {  "plotID" : "FHRT90035",

   "properties": {
             "plotID": "FHRT90035",
          "nativeTitle": "Free Hold",
          "owner": "Wudjari People",
          "aboriginalPlaceName": "Wudjari",
          "hearingYear": "2014",
          "address": "1002/421, Docklands River Melbourne, Victoria"
        },
        "geometry": "{
            "type": "MultiPolygon",
            "coordinates": [
                [
                    [
                        [1234, 2345],
                        [2345, 3456],
                        ...
                    ]
                ],
                ...
            ]
        }"
 }]
}
```

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Retrieve existing aboriginal countries - CLIENT [DRAFTS]

- Retrieve existing aboriginal countries - CLIENT [Version 4]
- Retrieve existing determination areas - CLIENT [Version 1]

- Retrieve existing determination areas - CLIENT [Version 2]
- Retrieve existing determination areas - CLIENT [Version 3]

Retrieve existing aboriginal countries - CLIENT [Version 4]

Description

Version 4: Updated to change determination area to aboriginal country as per client request

Retrieve existing aboriginal countries and their details for the front end client application.

This method only accepts GET requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getAboriginalCountries
```

Parameters

None.

Response Fields

Response	Description
aboriginalCountries	An array containing the aboriginal country JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /mapData/getAboriginalCountries
Host: https://[REDACTED]/api
```

Example Responses

Examples:

Status Code

200

```
{
 "aboriginalCountries": [
 {
  "aboriginalCountryName" : "some name",
  "geometry": "{
   "type": "MultiPolygon", // can also be Polygon, where MultipPolygon is
an array of Polygon
   "coordinates": [
    [
     [
      [1234, 2345],
      [2345, 3456],
      ...
     ]
    ],
    ...
   ]
  }"
 }
 ...
 ]
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieve existing determination areas - CLIENT [Version 1]

Description

Retrieve existing determination areas and their details for the front end client application.

This method only accepts GET requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getDetAreas
```

Parameters

None.

Response Fields

Response	Description
detArea	An array containing the determination area JSON objects and their information.

error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /mapData/getDetAreas
Host: https://[REDACTED]/api
```

Example Responses

Examples:

Status Code

200

```
{
 "detAreas": [
 {
  "landID": "234a3ffieu3",
  "boundaries": {[(1234, 2345),
      (2345, 3456),
      ...]}
 },
 ...
 ]
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieve existing determination areas - CLIENT [Version 2]

Description

Version 2: adapted to follow GeoJSON Specification: http://geojson.org/

Retrieve existing determination areas and their details for the front end client application.

This method only accepts GET requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getDetAreas
```

Parameters

None.

Response Fields

Response	Description
detArea	An array containing the determination area JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /mapData/getDetAreas
Host: https://[REDACTED]/api
```

Example Responses

Examples:

Status Code

200

```
{
 "detAreas": [
 {
  "type": "Feature",
  "properties:" {
   "detAreaID": "dsfsdfsd",
   "plotType": "detArea"
  }
  "geometry": {
   "type": "MultiPolygon", // can also be Polygon, where MultipPolygon is
an array of Polygon
   "coordinates": [
    [
     [
      [1234, 2345],
      [2345, 3456],
      ...
     ]
    ],
    ...
   ]
  }
 }
 ...
 ]
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieve existing determination areas - CLIENT [Version 3]

Description

Version 3: Added strings example response for properties and geometry.

Retrieve existing determination areas and their details for the front end client application.

This method only accepts GET requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getDetAreas
```

Parameters

None.

Response Fields

Response	Description
detArea	An array containing the determination area JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /mapData/getDetAreas
Host: https://[REDACTED]/api
```

Example Responses

Examples:

Status Code

200

```
{
 "detAreas": [
 {
  "detAreaName" : "some name",
  "geometry": "{
   "type": "MultiPolygon", // can also be Polygon, where MultipPolygon is
an array of Polygon
   "coordinates": [
    [
     [
      [1234, 2345],
      [2345, 3456],
      ...
     ]
    ],
    ...
   ]
  }"
 }
 ...
 ]
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieve search history for an account - CLIENT [DRAFTS]

Retrieve search history for an account - CLIENT [Version 1]

Description

Retrieving information on all plots in the search history of an account, including plot IDs, plot properties and geometry.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/getUserHistory
```

Parameters

Parameter		Description
required	id	A string representing the id of the user in the db
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
plots	An array containing the plot JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/getUserHistory
Host: https://[REDACTED]/api
Header:
{
    authorization:
"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b33774128cd8c874
763a399026c6226cafe"
}
Body: { id: "5ba5d9e1e942371d58da684c" }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

```
{
 "plots": [{
   {  "plotID" : "FHRT90035",

   "properties": {
            "plotID": "FHRT90035",
            "type": "Free Hold",
            "owner": "Charles Park",
            "term": "2 years",
            "contact": "+61420570622",
            "address": "1002/421, Docklands River Melbourne, Victoria",
        },
        "geometry": "{
            "type": "MultiPolygon",
            "coordinates": [
                [
                    [
                        [1234, 2345],
                        [2345, 3456],
                        ...
                    ]
                ],
                ...
            ]
        }"
 }]
}
```

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Retrieve search history for an account - CLIENT [Version 2]

Version 2: Plot properties updated to reflect client requests

Description

Retrieving information on all plots in the search history of an account, including plot IDs, plot properties and geometry.

This method only accepts POST requests.

Resource URL

Example:

```
https://[REDACTED]/api/userData/getUserHistory
```

Parameters

Parameter		Description
required	id	A string representing the id of the user in the db
required	authorization	A string representing the authorization token of the session

Response Fields

Response	Description
plots	An array containing the plot JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /userData/getUserHistory
Host: https://[REDACTED]/api
Header:
{
    authorization:
"b34c043b17b013d8dee6e3f34a3d711da9f81ea9ef478383d2a581653b33774128cd8c874
763a399026c6226cafe"
}
Body: { id: "5ba5d9e1e942371d58da684c" }
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

```
{
 "plots": [{
   {  "plotID" : "FHRT90035",

   "properties": {
             "plotID": "FHRT90035",
          "nativeTitle": "Free Hold",
          "owner": "Wudjari People",
           "aboriginalPlaceName": "Wudjari",
          "hearingYear": "2014",
          "address": "1002/421, Docklands River Melbourne, Victoria"
        },
        "geometry": "{
            "type": "MultiPolygon",
            "coordinates": [
                [
                    [
                        [1234, 2345],
                        [2345, 3456],
                        ...
                    ]
                ],
                ...
            ]
        }"
 }]
}
```

Status Code

400

```
{ "error": "Bad Request"}
```

Status Code

401

```
{ "error": "No authentication token provided or invalid details"}
```

Status Code

402

```
{ "error": "Invalid or expired authentication token provided"}
```

Status Code

500

```
{ "error": "Database error"}
```

Retrieving basic information on all plots - CLIENT [DRAFTS]

- Retrieving basic information on all plots - CLIENT [Version 1]
- Retrieving basic information on all plots - CLIENT [Version 2]

Retrieving basic information on all plots - CLIENT [Version 1]

Description

Retrieving information on all plots, including its plot ID, type and its boundaries.

This method only accepts GET requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getPlots
```

Parameters

None.

Response Fields

Response	Description
plots	An array containing the plot JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /mapData/getPlots
Host: https://[REDACTED]/api
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

```
{
    "plots": [
    {
  "plotID": "2897sawseu3",
        "detAreaID": "234a3ffieu3",
  "type": "Free Hold",
        "boundaries": {[(1234, 2345),
                        (2345, 3456),
                        ...]}
    },
    ...
    ]
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieving basic information on all plots - CLIENT [Version 2]

Description

Version 2: adapted to follow GeoJSON Specification: http://geojson.org/

Retrieving information on all plots, including its plot ID, type and its boundaries.

This method only accepts GET requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getPlots
```

Parameters

None.

Response Fields

Response	Description
plots	An array containing the plot JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /mapData/getPlots
Host: https://[REDACTED]/api
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

```
{
 "plots": [
 {
  "type": "Feature",
  "properties:" {
   "plotID": "sfsdfasdfasdf",
   "detAreaID": "dsfsdfsd",
   "plotType": "plot"
  }
  "geometry": {
   "type": "MultiPolygon", // can also be Polygon, where MultipPolygon is
an array of Polygon
   "coordinates": [
    [
     [
      [1234, 2345],
      [2345, 3456],
      ...
     ]
    ],
    ...
   ]
  }
 }
 ...
 ]
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieving basic information on all plots - CLIENT [Version 3]

Description

Version 3: Added strings example response for properties and geometry.

Retrieving information on all plots, including its plot ID, type and its boundaries.

This method only accepts GET requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getPlots
```

Parameters

None.

Response Fields

Response	Description
plots	An array containing the plot JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /mapData/getPlots
Host: https://[REDACTED]/api
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

```
{
 "plots": [{
   {  "plotID" : "FHRT90035",

   "properties": {
            "plotID": "FHRT90035",
            "type": "Free Hold",
            "owner": "Charles Park",
            "term": "2 years",
            "contact": "+61420570622",
            "address": "1002/421, Docklands River Melbourne, Victoria",
        },
        "geometry": "{
            "type": "MultiPolygon",
            "coordinates": [
                [
                    [
                        [1234, 2345],
                        [2345, 3456],
                        ...
                    ]
                ],
                ...
            ]
        }"
 }]
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Retrieving basic information on all plots - CLIENT [Version 4]

Description

Version 4: **Plot properties updated to reflect client requests**.

Retrieving information on all plots, including its plot ID, type and its boundaries.

This method only accepts GET requests.

Resource URL

Example:

```
https://[REDACTED]/api/mapData/getPlots
```

Parameters

None.

Response Fields

Response	Description
plots	An array containing the plot JSON objects and their information.
error	If the request was unsuccessful, a message detailing why is returned.

Example JSON Request

Example (values and shape are NOT accurate, do not use as a baseline):

```
GET /mapData/getPlots
Host: https://[REDACTED]/api
```

Example Responses

Examples:(labels and values and tentative)

Status Code

200

```
{
 "plots": [{
   {  "plotID" : "FHRT90035",

   "properties": {
             "plotID": "FHRT90035",
          "nativeTitle": "Free Hold",
          "owner": "Wudjari People",
          "aboriginalPlaceName": "Wudjari",
          "hearingYear": "2014",
          "address": "1002/421, Docklands River Melbourne, Victoria"
        },
        "geometry": "{
            "type": "MultiPolygon",
            "coordinates": [
                [
                    [
                        [1234, 2345],
                        [2345, 3456],
                        ...
                    ]
                ],
                ...
            ]
        }"
 }]
}
```

Status Code

400

```
{ "error": "Bad Request" }
```

Quality Assurance and Verification/Validation Documentation

Bit Bucket Usage and Branch Naming Convention [Version 2]

Updated to include tagging policy

Bit Bucket Usage

- Each team member will only work on their own created branch.
- Each branch will represent a task, as represented by the Sprint Plan or the JIRA board.
- Pushing directly to master is prohibited.
- Team members will request to merge their branch to master AFTER pulling/rebasing from master into their branch and testing.
- Merge requests to master should be additionally tested and approved by another team member.
- Team members should regularly pull/rebase from master.

Branch Naming Convention

Branches will have 3 identifiers, where sub-identifiers specify the nature of the task. The format will be:

```
identifier/sub-identifier/relevant-task-description
```

Identifiers:

1. backend: any work related to the server-side of the application
2. frontend-client: any work related to the main map React client
3. frontend-employee: any work related to the government employee React client

Sub-identifiers:

1. feature: for new functionality
2. defect: for bugs
3. test: for creating tests
4. issue: for work that fits in none of the other sub-identifiers, e.g. refactoring.

An example branch name could be:

```
frontend-client/feature/add-plots
```

Tagging Policy

A runnable build at the completion of each sprint will be appropriately tagged as follows:

```
Sprint-X-Runnable-Build
```

where X is the number identifier of the sprint.

For example, the appropriate tag for Sprint 1 would be:

```
Sprint-1-Runnable-Build
```

Confluence Workflow

Our confluence page represents and documents our entire project and so it is important to ensure that all team members adhere to the following workflow:

Creating a new page:

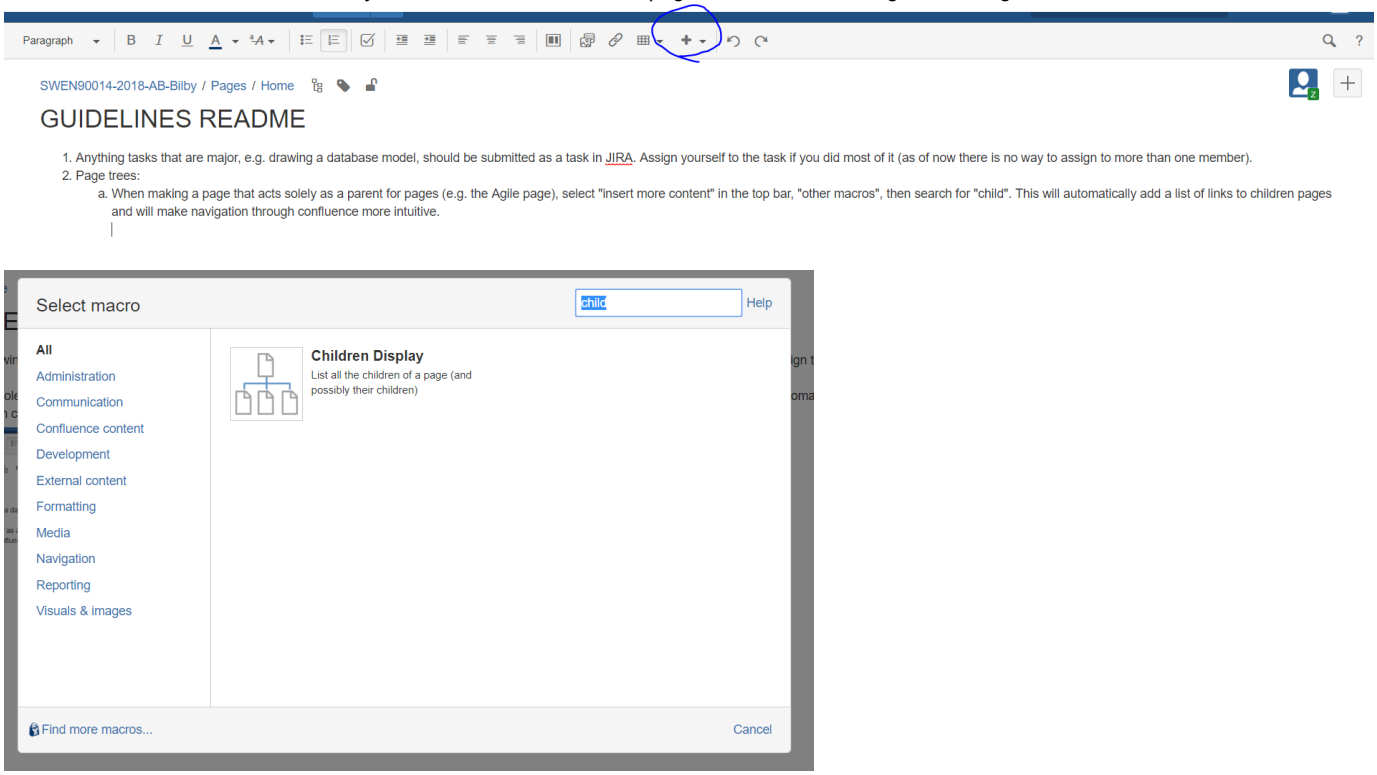
- When creating an important page, it should be submitted as a task in JIRA.
- The person who works on the page should be assigned to the task.
- The confluence page should link to the task in JIRA.

Versioning:

- Assume there is a page called "Design Concept [Version 1]"
- If there is a new version of this document to be created, then there should be a new page created, called "Design Concept [Version 2]"
- There can be any amount of versions and these version can all be stored under a directory called "Design Concept".
- Each new version should have a short justification at the top of the document indicating new updates.

Page trees:

When making a page that acts solely as a parent for pages (e.g. the Agile page), select "insert more content" in the top bar, "other macros", then search for "child". This will automatically add a list of links to children pages and will make navigation through confluence more intuitive.



Editing:

If the changes are not important, untick "notify watchers" at the bottom right so others are not spammed with emails.

JIRA Workflow

Creating tasks

- After the sprint planning meeting, all tasks for the coming sprint are added to that sprint in JIRA.
- Each task is given a cost estimate.
- If there is a relevant page in confluence, the task should be linked to that page. If there is no page in confluence related to that task, consider making one.
- Tasks created begin in the "To Do" column in JIRA and are not allocated to anyone.

Working on tasks

- Once you choose a task to work on, you assign it to yourself and move it to the "In Progress" column. It is important that you work on one task at a time, so there should be a maximum of one task per team member in the "In Progress" column.
- If the task is a coding task, the task is only complete when the required testing has passed and the pull request has been reviewed and approved by another member of the development team.

- Once a task is complete, it is moved to the "Done" column.

Reviewing Artefacts

The following table outlines key artefacts and the details of how each of these artefacts are reviewed.

Artefacts needing Review	Style of Review	Details of Review
Bitbucket Code	**Inspection and** **Simulation.**	**Circumstance of Review:** A merge request has been made by a developer. The purpose of this review is to ensure, that code that is added into our product is both functional and of good quality. When nominating a reviewer, the developer should be familiar with the code. **Inspection:** To ensure that code quality is maintained, it's important that the submitted code is reviewed by at least one other developer. The first stage in review is simply the visual inspection of code. This helps ensure that code is well written. **Simulation:** The second stage of review is the process of simulation. This involves running the application and ensuring that any new functionalities enabled by the additional code is working. It also involves running the code against the pre-existing test-suite to check for bugs. **Our goals are to ensure:** • Ensure that Bitbucket workflow is followed: Bit Bucket Usage and Branch Naming Convention [Version 2] • Code Completeness • Code Accuracy • Code Readability • Code Testability

JIRA Tasks	**Audit**	**Circumstance of Review:** A new JIRA task has been made. Whenever a task is made in JIRA, the task is required to be reviewed by the scrum master to ensure that the task is important and relevant to the current sprint. The review involves an auditing process, where the scrum master evaluates the current sprint progress and determines if its a development task or documentation task. If it's a development task, then the scrum master reviews if the task is relevant to the current sprint backlog. If it's a documentation task, then the scrum master evaluates whether that confluence page is relevant to the project. **Our goals are to ensure:** • Ensure that JIRA workflow is followed: J IRA Workflow • Ensure task can be completed by the assignee • Ensure task is relevant to the sprint
Confluence Documents	**Audit**	**Circumstance of Review:** A new confluence document has been made. Whenever a new confluence document has been completed by a team member, it should be audited by another team member. This review process involves determining the nature of the document and whether the contents of the document are well-written. It also involves determining if the document is actually informative and relevant to the project or team. If so, the document is approved and officially published. **Our goals are to ensure:** • Ensure that Confluence workflow is followed: Confluence Workflow • Adequacy of user documentation • Ensure documents are well-written and informative

Product	**Walk-through**	**Circumstance of Review:** A sprint has come to an end. The purpose of this review is to determine whether the product is meeting client requirements. This review is conducted at the end of each sprint, during a meeting with our client . The style of the review involves a walk through of the current problem, detailing any new functionality that has been integrated into the product. and is is conducted as a walk through. This is essentially a user acceptance test. **Our goals are to to ensure:** • Compliance with product requirement • Satisfaction of the client • Document the outcome and

Software Quality Assurance Plan

Purpose

- Identify the SQA responsibilities of the team
- List and define processes to be done as a part of SQA

Responsibilities

Role	Name	SQA Responsibility
QA Lead	Ziwei	Create and manage software quality assurance plan
Scrum Master	Marko	Motivate team to follow the software quality assurance plan
Backend Test Lead	Michael	Develop testing suites that ensure the software runs as expected

Processes

Pull Request Review

Pull requests are to be reviewed by at least one other member before a merge is allowed. Any member can be a reviewer. Further details of the process can be found here: BitBucket and Branch Naming Convention [Version 1]

Risk Assessment

During sprint planning/sprint retrospective, existing risks are to be reassessed, and new risks are to be identified (if any). The risks assessment process can be viewed here: Risk Identification Process

After assessing the risks, the team is to decide on the risk category of the project at the current stage as follows:

- Low: no risks have been identified, or potential/existing low risk problems are to be addressed
- Medium: problems with a high probability of impacting the project exist
- High: Serious problems exist without foreseeable solutions, and probability of impact on user acceptance and project is high

Verification of Requirements

Determine whether or not the product at a given phase fulfill the requirements established before the start of that phase. A clear goal must be established during sprint planning for this to happen. The following criteria can be assessed:

- Correctness
- Consistency
- Completeness
- Accuracy
- Readability
- Testability

Validation of Requirements

Determining whether the product will meet clients' requirements.

- Plan acceptance testing which includes:
    - Compliance with requirements
    - Adequacy of user documentation
    - Performance at boundaries and stress conditions (e.g. high amounts of data transfer)
- Plan the documentation of test tasks and results
- Execute acceptance test and document the results

Testing

Integration Testing

Backend [DRAFTS]

Backend Integration Test [Version 1]

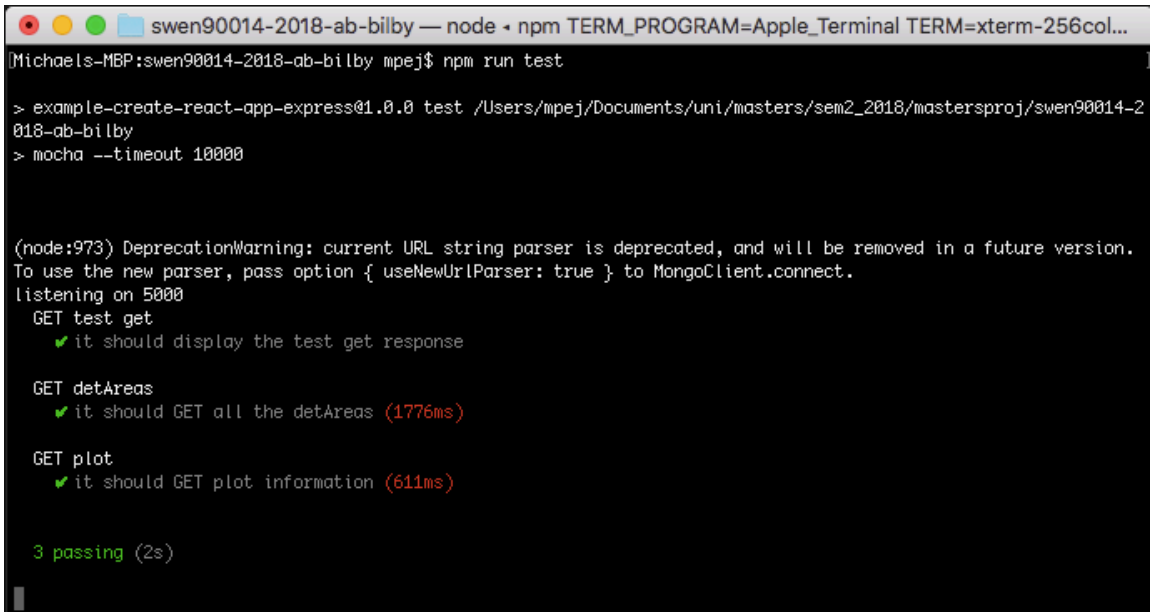
The test suite for the backend integration tests is built using Mocha as the testing framework and Chai for the assertion library includes the following tests

Test	Action	Expectation
getTest response	send a GET request to the test page	display test response
get detAreas	send a GET request for all detAreas	all detAreas in db are returned in an array
get plot	send a GET request for a single plot	plot information for a single plot (including the plot properties) are returned

To run the test suite, make sure you are in the root directory in you command line and run "npm run test"

Successful tests are marked with a green tick. Unsuccessful tests are marked with a red tick.

Below is an example of a successful run of the test suite.



Backend Integration Test [Version 2]

Updates from Version 1: Testing on testdb with Mockgoose for mocking the db, added tests for all endpoints and extended the tests already in place, made commands for both linux and windows to run test suite

The test suite for the backend integration tests is built using Mocha as the testing framework, Chai for the assertion library and Mockgoose as a wrapper for Mongoose to mock the db.

The verification process of this test suite involved creating a pull request and having another member of the backend team review the code and each of the tests before merging the changes to the master.

The test suite includes the following tests

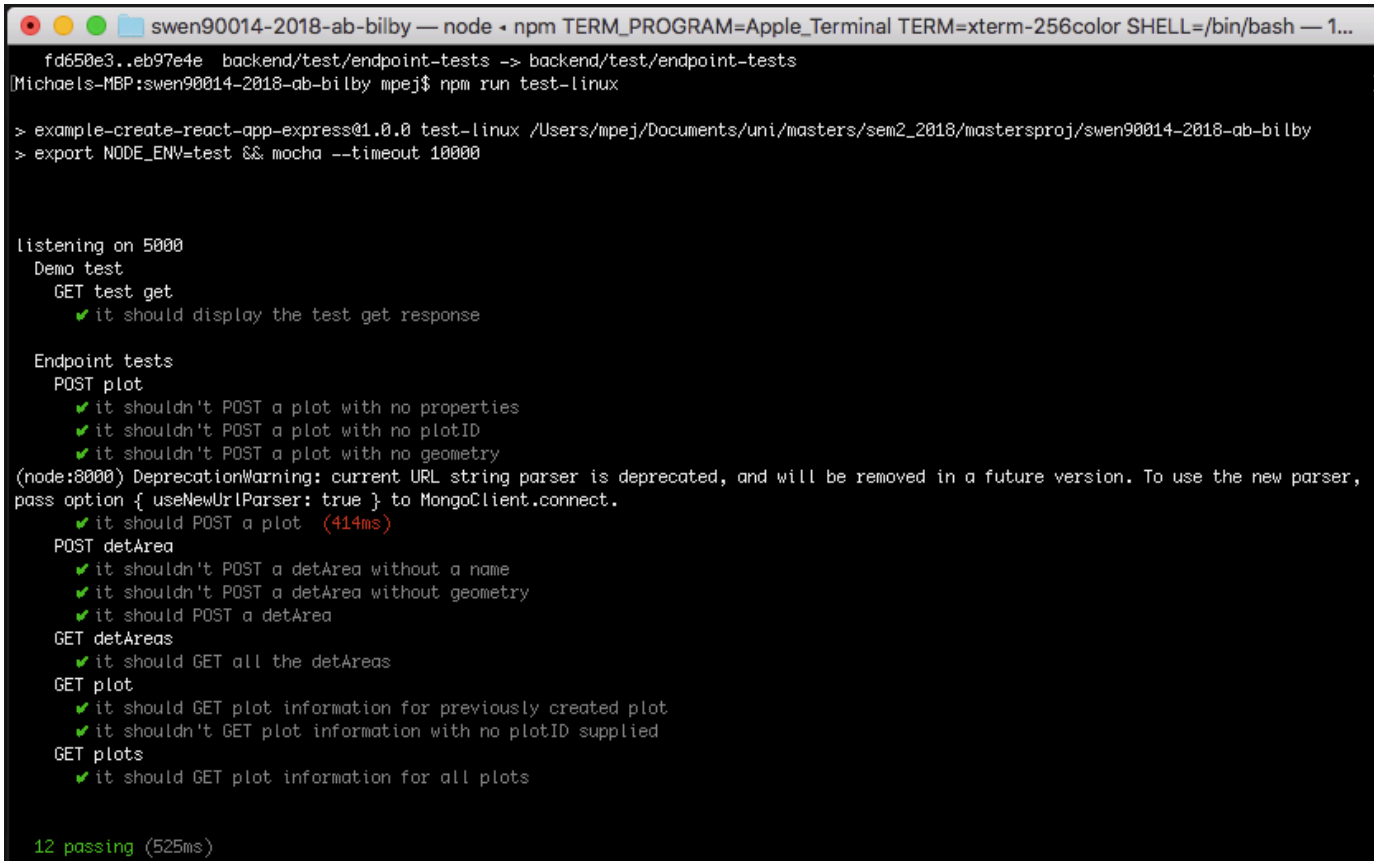
API Endpoint being tested	Action	Expectation
/api/	send a GET request to the test page	display test response
/api/mapGovData/addPlot	send a POST request to add a plot to the db with all the required fields apart from the properties field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a plot to the db with all the required fields apart from the plotID field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a plot to the db with all the required fields apart from the geometry field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a plot to the db with all the required fields	It should successfully add the plot to the db, returning response 201
/api/mapGovData/addDetArea	send a POST request to add a detArea to the db with all the required fields apart from the name field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a detArea to the db with all the required fields apart from the geometry field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a detArea to the db with all the required fields	It should successfully add the detArea to the db, returning response 201
/api/mapData/getDetAreas	send a GET request to retrieve all of the detAreas in an array, including the detArea successfully added in a previous test	It should successfully get the detAreas from the db, returning response 200

/api/mapData/getPlotInformation/	send a GET request to retrieve all of the plot information for the plot successfully created in a previous test	It should successfully get the plot information from the db, returning response 200
	send a GET request to retrieve all of the plot information for a plot with no plotID provided	It should fail to get the plot information from the db, returning response 400
/api/mapData/getPlots	send a GET request to retrieve all of the plot information for every plot in an array, including the plot successfully added in a previous test	It should successfully get the plot information from the db, returning response 200

To run the test suite, make sure you are in the root directory in you command line and run "npm run test-linux" or "npm run test-windows" depending on the operating system of the machine you are running it on.

Successful tests are marked with a green tick. Unsuccessful tests are marked with a red tick.

Below is an example of a successful run of the test suite.



Backend Integration Test [Version 3]

Version 3: Added tests for addUser, login endpoints and extended the tests already in place by adding auth check tests

The test suite for the backend integration tests is built using Mocha as the testing framework, Chai for the assertion library and Mockgoose as a wrapper for Mongoose to mock the db.

The verification process of this test suite involved creating a pull request and having another member of the backend team review the code and each of the tests before merging the changes to the master.

The test suite includes the following tests

API Endpoint being tested	Action	Expectation
/api/	send a GET request to the test page	display test response
/api/userData/addUser	send a POST request to add a normal user to the system with all the required fields	It should successfully add a normal user to the system, returning response 201.

	send a POST request to add a government user to the system with all the required fields	It should successfully add a government user to the system, returning response 201.
	send a POST request to add the same user as the previous test to the system with all the required fields	It should fail to add a user that is already in the system, returning response 422 and displaying Username is taken error message.
/api/userData/login	send a POST request to login a user that does not exist in the system with all the required fields	It shouldn't login a user that does not exist in the system, returning response 401 and displaying error message.
	send a POST request to login a user that does exist in the system with all the required fields and an incorrect password	It shouldn't login a user with the wrong password, returning response 401 and displaying error message.
	send a POST request to login a normal user that does exist in the system with all the required fields and a correct password	It should login the normal user to the system, returning response 200 as well as the id of the user in the db and an auth token for the session.
	send a POST request to login a government user that does exist in the system with all the required fields and a correct password	It should login the government user to the system, returning response 200 as well as the id of the user in the db and an auth token for the session.
/api/mapGovData/addPlot	send a POST request to add a plot to the db with all the required fields apart from the properties field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a plot to the db with all the required fields apart from the plotID field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a plot to the db with all the required fields apart from the geometry field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a plot to the db with all the required fields apart from the user id field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a plot to the db with an invalid user id	It should fail, returning response 402 and display the invalid or expired authentication error message.
	send a POST request to add a plot to the db with an invalid auth token	It should fail, returning response 402 and display the invalid or expired authentication error message.
	send a POST request to add a plot to the db with no auth token	It should fail, returning response 401 and display the no authentication token provided or invalid details error message.
	send a POST request to add a plot to the db with all the required fields	It should successfully add the plot to the db, returning response 201
/api/mapGovData/addAboriginalCountry	send a POST request to add a aboriginalCountry to the db with all the required fields apart from the name field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a aboriginalCountry to the db with all the required fields apart from the geometry field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a aboriginalCountry to the db with all the required fields apart from the user id field	It should fail, returning response 400 and display the bad request error message.
	send a POST request to add a aboriginalCountry to the db with an invalid user id	It should fail, returning response 402 and display the invalid or expired authentication error message.

	send a POST request to add a aboriginalCountry to the db with an invalid auth token	It should fail, returning response 402 and display the invalid or expired authentication error message.
	send a POST request to add a aboriginalCountry to the db with no auth token	It should fail, returning response 401 and display the no authentication token provided or invalid details error message.
	send a POST request to add a aboriginalCountry to the db with all the required fields	It should successfully add the aboriginalCountry to the db, returning response 201
/api/mapData/getAboriginalCountries	send a GET request to retrieve all of the aboriginalCountry in an array, including the aboriginalCountry successfully added in a previous test	It should successfully get the aboriginalCountries from the db, returning response 200
/api/mapGovData/editAboriginalCountry	send a POST request to edit an aboriginalCountry with all the required fields and the aboriginalCountryName updated	It should successfully update the aboriginalCountry with the new information, returning response 200
	send a GET request to retrieve all of the aboriginalCountries in an array, including the aboriginalCountry successfully edited in the previous test	It should successfully get the aboriginalCountries from the db, with the aboriginalCountry from the previous test updated to have a new aboriginalCountryName returning response 200
	send a POST request to edit an aboriginalCountry with all the required fields and the geometry updated	It should successfully update the aboriginalCountry with the new information, returning response 200
	send a GET request to retrieve all of the aboriginalCountries in an array, including the aboriginalCountry successfully edited in the previous test	It should successfully get the aboriginalCountries from the db, with the aboriginalCountry from the previous test updated to have a new geometry returning response 200
	send a POST request to edit an aboriginalCountry with an invalid aboriginalCountry id	It should fail to edit the aboriginalCountry, returning response 500 and the database error message
	send a POST request to edit an aboriginalCountry without an aboriginalCountry id	It should fail to edit the aboriginalCountry, returning response 400 and the bad request message
	send a POST request to edit an aboriginalCountry without an aboriginalCountryName	It should fail to edit the aboriginalCountry, returning response 400 and the bad request message
	send a POST request to edit an aboriginalCountry without a geometry	It should fail to edit the aboriginalCountry, returning response 400 and the bad request message
	send a POST request to edit an aboriginalCountry without a user id	It should fail to edit the aboriginalCountry, returning response 400 and the bad request message
	send a POST request to edit an aboriginalCountry with an invalid user id	It should fail to edit the aboriginalCountry, returning response 402 and the invalid or expired authentication message
	send a POST request to edit an aboriginalCountry without an auth token	It should fail to edit the aboriginalCountry, returning response 401 and the no authentication or invalid message
	send a POST request to edit an aboriginalCountry with an invalid auth token	It should fail to edit the aboriginalCountry, returning response 402 and the invalid or expired authentication message
/api/mapGovData/deleteAboriginalCountry	send a POST request to delete an aboriginal Country without a user id	It should fail to delete the aboriginalCountry, returning response 400 and the bad request message

	send a POST request to delete an aboriginal Country with an invalid user id	It should fail to delete the aboriginalCountry, returning response 402 and the invalid or expired authentication message
	send a POST request to delete an aboriginal Country without an auth token	It should fail to delete the aboriginalCountry, returning response 401 and the no authentication or invalid message
	send a POST request to delete an aboriginal Country with an invalid auth token	It should fail to delete the aboriginalCountry, returning response 402 and the invalid or expired authentication message
	send a POST request to delete an aboriginalCountry with all the required fields and authentication	It should successfully delete the given aboriginalCountry from the db, returning response 200
	send a GET request to retrieve all of the abo riginalCountries in an array	It should return an empty array due to the only aboriginalCountry being deleted
/api/mapData/getPlotInformation/	send a POST request to retrieve all of the plot information for the plot successfully created in a previous test	It should successfully get the plot information from the db, returning response 200
	send a POST request to retrieve all of the plot information for a plot with no plotID provided	It should fail to get the plot information from the db, returning response 400
/api/mapData/getPlots	send a GET request to retrieve all of the plot information for every plot in an array, including the plot successfully added in a previous test	It should successfully get the plot information from the db, returning response 200
/api/mapGovData/editPlot	send a POST request to edit a plot without a user id	It should fail to edit the plot, returning response 400 and the bad request message
	send a POST request to edit a plot with an invalid user id	It should fail to edit the plot, returning response 402 and the invalid or expired authentication message
	send a POST request to edit a plot without an auth token	It should fail to edit the plot, returning response 401 and the no authentication or invalid message
	send a POST request to edit a plot with an invalid auth token	It should fail to edit the plot, returning response 402 and the invalid or expired authentication message
	send a POST request to edit a plot with all the required fields and authentication	It should successfully edit the given plot in the db, returning response 200
	send a POST request to retrieve all of the plot information for the plot successfully edited in the previous test	It should successfully get the new plot information for that plot from the db, returning response 200
/api/userData/addBookmark	send a POST request to add a bookmark without a user id	It should fail to add the bookmark, returning response 400 and the bad request message
	send a POST request to add a bookmark wit h an invalid user id	It should fail to add the bookmark, returning response 402 and the invalid or expired authentication message
	send a POST request to add a bookmark wit hout an auth token	It should fail to add the bookmark, returning response 401 and the no authentication or invalid message
	send a POST request to add a bookmark wit h an invalid auth token	It should fail to add the bookmark, returning response 402 and the invalid or expired authentication message
	send a POST request to add a bookmark with all the required fields and authentication	It should successfully add the given bookmark in the db, returning response 200

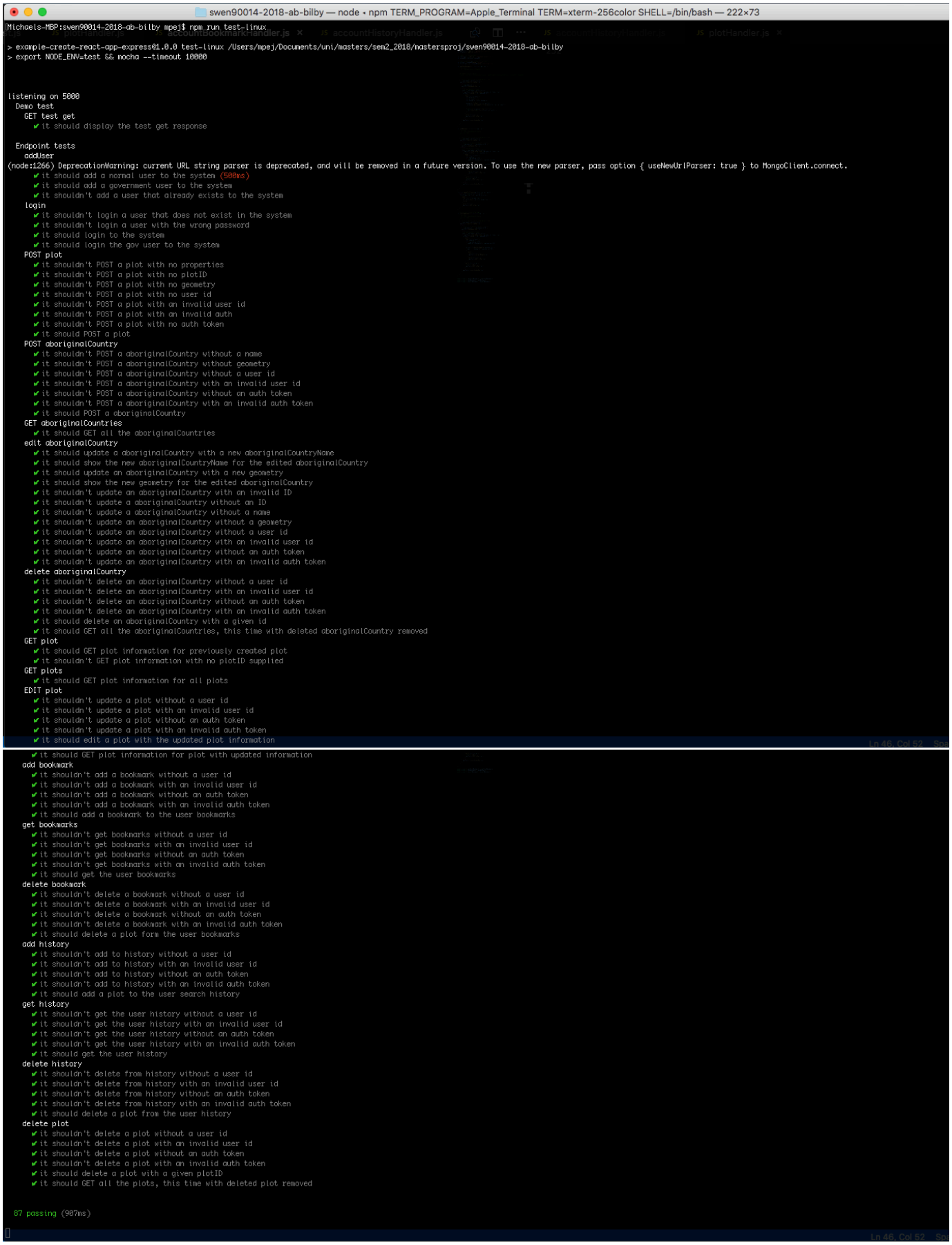
/api/userData/getUserBookmarks	send a POST request to get all bookmarks for the given user without a user id	It should fail to get the bookmarks, returning response 400 and the bad request message
	send a POST request to get all bookmarks for the given user with an invalid user id	It should fail to get the bookmarks, returning response 402 and the invalid or expired authentication message
	send a POST request to get all bookmarks for the given user without an auth token	It should fail to get the bookmarks, returning response 401 and the no authentication or invalid message
	send a POST request to get all bookmarks for the given user with an invalid auth token	It should fail to get the bookmarks, returning response 402 and the invalid or expired authentication message
	send a POST request to get all bookmarks for the given user with all the required fields and authentication	It should successfully get the bookmarks for the given user from the db, returning response 200
/api/userData/deleteBookmark	send a POST request to delete a bookmark for the given user without a user id	It should fail to delete the bookmark, returning response 400 and the bad request message
	send a POST request to delete a bookmark f or the given user with an invalid user id	It should fail to delete the bookmark, returning response 402 and the invalid or expired authentication message
	send a POST request to delete a bookmark f or the given user without an auth token	It should fail to delete the bookmark, returning response 401 and the no authentication or invalid message
	send a POST request to delete a bookmark f or the given user with an invalid auth token	It should fail to delete the bookmark, returning response 402 and the invalid or expired authentication message
	send a POST request to delete a bookmark f or the given user with all the required fields and authentication	It should successfully delete the bookmark fo r the given user from the db, returning response 200
/api/userData/addHistory	send a POST request to add to history for the given user without a user id	It should fail to add to history, returning response 400 and the bad request message
	send a POST request to add to history for the given user with an invalid user id	It should fail to add to history, returning response 402 and the invalid or expired authentication message
	send a POST request to add to history for the given user without an auth token	It should fail to add to history, returning response 401 and the no authentication or invalid message
	send a POST request to add to history for the given user with an invalid auth token	It should fail to add to history, returning response 402 and the invalid or expired authentication message
	send a POST request to add to history for the given user with all the required fields and authentication	It should successfully add to history for the given user from the db, returning response 200
/api/userData/getUserHistory	send a POST request to get the history for the given user without a user id	It should fail to get the history, returning response 400 and the bad request message
	send a POST request to get the history for the given user with an invalid user id	It should fail to get the history, returning response 402 and the invalid or expired authentication message
	send a POST request to get the history for the given user without an auth token	It should fail to get the history, returning response 401 and the no authentication or invalid message
	send a POST request to get the history for the given user with an invalid auth token	It should fail to get the history, returning response 402 and the invalid or expired authentication message

	send a POST request to get the history for the given user with all the required fields and authentication	It should successfully get the history for the given user from the db, returning response 200
/api/userData/deleteHistory	send a POST request to delete from history for the given user without a user id	It should fail to delete from history, returning response 400 and the bad request message
	send a POST request to delete from history for the given user with an invalid user id	It should fail to delete from history, returning response 402 and the invalid or expired authentication message
	send a POST request to delete from history for the given user without an auth token	It should fail to delete from history, returning response 401 and the no authentication or invalid message
	send a POST request to delete from history for the given user with an invalid auth token	It should fail to delete from history, returning response 402 and the invalid or expired authentication message
	send a POST request to delete from history for the given user with all the required fields and authentication	It should successfully delete from history for the given user from the db, returning response 200
/api/mapGovData/deletePlot	send a POST request to delete a plot without a user id	It should fail to delete a plot, returning response 400 and the bad request message
	send a POST request to delete a plot with an invalid user id	It should fail to delete a plot, returning response 402 and the invalid or expired authentication message
	send a POST request to delete a plot without an auth token	It should fail to delete a plot, returning response 401 and the no authentication or invalid message
	send a POST request to delete a plot with an invalid auth token	It should fail to delete a plot, returning response 402 and the invalid or expired authentication message
	send a POST request to delete a plot with all the required fields and authentication	It should successfully delete a plot for the given user from the db, returning response 200
	send a GET request to retrieve all of the plots	It should successfully retrieve all of the plots, which should be an empty array as the only plot was deleted in the previous test

To run the test suite, make sure you are in the root directory in you command line and run "npm run test-linux" or "npm run test-windows" depending on the operating system of the machine you are running it on.

Successful tests are marked with a green tick. Unsuccessful tests are marked with a red tick. When tests are unsuccessful an error message is shown below the list of tests detailing the nature of the failure.

Below is an example of a successful run of the test suite.


swen90014-2018-ab-bilby — node • npm TERM_PROGRAM=Apple_Terminal TERM=xterm-256color SHELL=/bin/bash — 222×73
Michaels-MBP:swen90014-2018-ab-bilby mpej$ npm run test-linux
> example-create-react-app-express01.0.0 test-linux /Users/mpej/Documents/uni/masters/sem2_2018/mastersproj/swen90014-2018-ab-bilby
> export NODE_ENV=test && mocha --timeout 10000
listening on 5000
Demo test
GET test get
✔ it should display the test get response
Endpoint tests
addUser
(node:1266) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
✔ it should add a normal user to the system (500ms)
✔ it should add a government user to the system
✔ it shouldn't add a user that already exists to the system
login
✔ it shouldn't login a user that does not exist in the system
✔ it shouldn't login a user with the wrong password
✔ it should login to the system
✔ it should login the gov user to the system
POST plot
✔ it shouldn't POST a plot with no properties
✔ it shouldn't POST a plot with no plotID
✔ it shouldn't POST a plot with no geometry
✔ it shouldn't POST a plot with no user id
✔ it shouldn't POST a plot with an invalid user id
✔ it shouldn't POST a plot with an invalid auth
✔ it shouldn't POST a plot with no auth token
✔ it should POST a plot
POST aboriginalCountry
✔ it shouldn't POST a aboriginalCountry without a name
✔ it shouldn't POST a aboriginalCountry without geometry
✔ it shouldn't POST a aboriginalCountry without a user id
✔ it shouldn't POST a aboriginalCountry with an invalid user id
✔ it shouldn't POST a aboriginalCountry without an auth token
✔ it shouldn't POST a aboriginalCountry with an invalid auth token
✔ it should POST a aboriginalCountry
GET aboriginalCountries
✔ it should GET all the aboriginalCountries
edit aboriginalCountry
✔ it should update a aboriginalCountry with a new aboriginalCountryName
✔ it should show the new aboriginalCountryName for the edited aboriginalCountry
✔ it should update an aboriginalCountry with a new geometry
✔ it should show the new geometry for the edited aboriginalCountry
✔ it shouldn't update an aboriginalCountry with an invalid ID
✔ it shouldn't update a aboriginalCountry without an ID
✔ it shouldn't update a aboriginalCountry without a name
✔ it shouldn't update an aboriginalCountry without a geometry
✔ it shouldn't update an aboriginalCountry without a user id
✔ it shouldn't update an aboriginalCountry with an invalid user id
✔ it shouldn't update an aboriginalCountry without an auth token
✔ it shouldn't update an aboriginalCountry with an invalid auth token
delete aboriginalCountry
✔ it shouldn't delete an aboriginalCountry without a user id
✔ it shouldn't delete an aboriginalCountry with an invalid user id
✔ it shouldn't delete an aboriginalCountry without an auth token
✔ it shouldn't delete an aboriginalCountry with an invalid auth token
✔ it should delete an aboriginalCountry with a given id
✔ it should GET all the aboriginalCountries, this time with deleted aboriginalCountry removed
GET plot
✔ it should GET plot information for previously created plot
✔ it shouldn't GET plot information with no plotID supplied
GET plots
✔ it should GET plot information for all plots
EDIT plot
✔ it shouldn't update a plot without a user id
✔ it shouldn't update a plot with an invalid user id
✔ it shouldn't update a plot without an auth token
✔ it shouldn't update a plot with an invalid auth token
✔ it should edit a plot with the updated plot information
✔ it should GET plot information for plot with updated information
add bookmark
✔ it shouldn't add a bookmark without a user id
✔ it shouldn't add a bookmark with an invalid user id
✔ it shouldn't add a bookmark without an auth token
✔ it shouldn't add a bookmark with an invalid auth token
✔ it should add a bookmark to the user bookmarks
get bookmarks
✔ it shouldn't get bookmarks without a user id
✔ it shouldn't get bookmarks with an invalid user id
✔ it shouldn't get bookmarks without an auth token
✔ it shouldn't get bookmarks with an invalid auth token
✔ it should get the user bookmarks
delete bookmark
✔ it shouldn't delete a bookmark without a user id
✔ it shouldn't delete a bookmark with an invalid user id
✔ it shouldn't delete a bookmark without an auth token
✔ it shouldn't delete a bookmark with an invalid auth token
✔ it should delete a plot form the user bookmarks
add history
✔ it shouldn't add to history without a user id
✔ it shouldn't add to history with an invalid user id
✔ it shouldn't add to history without an auth token
✔ it shouldn't add to history with an invalid auth token
✔ it should add a plot to the user search history
get history
✔ it shouldn't get the user history without a user id
✔ it shouldn't get the user history with an invalid user id
✔ it shouldn't get the user history without an auth token
✔ it shouldn't get the user history with an invalid auth token
✔ it should get the user history
delete history
✔ it shouldn't delete from history without a user id
✔ it shouldn't delete from history with an invalid user id
✔ it shouldn't delete from history without an auth token
✔ it shouldn't delete from history with an invalid auth token
✔ it should delete a plot from the user history
delete plot
✔ it shouldn't delete a plot without a user id
✔ it shouldn't delete a plot with an invalid user id
✔ it shouldn't delete a plot without an auth token
✔ it shouldn't delete a plot with an invalid auth token
✔ it should delete a plot with a given plotID
✔ it should GET all the plots, this time with deleted plot removed
87 passing (907ms)

Frontend [DRAFTS]

Frontend Government Client Integration Test [Version 1]

The test suite for the frontend government client integration tests is built using Jest, a testing framework for ReactJS projects, and Enzyme, a testing utility.

The verification process for this test suite is identical to the process of verifying changes to the codebase - a pull request is made, and another member of the frontend team will review the code and each of the tests before merging the changes into master.

The test suite currently includes the following tests.

Component being tested	Action	Expectation
AddDetArea	shallow rendering using Enzyme	independent component renders without failure
AddPlot	shallow rendering using Enzyme	independent component renders without failure
App	shallow rendering using Enzyme	independent component renders without failure
DetAreaManager	shallow rendering using Enzyme	independent component renders without failure
EditDetArea	shallow rendering using Enzyme	independent component renders without failure
EditPlot	shallow rendering using Enzyme	independent component renders without failure
Home	shallow rendering using Enzyme	independent component renders without failure
PlotManager	shallow rendering using Enzyme	independent component renders without failure
Redux reducer being tested	**Action**	**Expectation**
ActiveDetAreaReducer	initialise without action	returns specified initial state
ActivePlotReducer	initialise without action	returns specified initial state
MapDataReducer	initialise without action	returns specified initial state

To run the test suite, navigate to the /govclient directory and run:

```
npm run test
```

Below is an example of a successful run of the test suite.

Frontend Government Client Testing Suite [Version 2]

Updated to include additional test cases

The test suite for the frontend government client integration tests is built using Jest, a testing framework for ReactJS projects, and Enzyme, a testing utility.

The verification process for this test suite is identical to the process of verifying changes to the codebase - a pull request is made, and another member of the frontend team will review the code and each of the tests before merging the changes into master.

The test suite currently includes the following tests.

Component being tested	Action	Expectation
AddAboriginalCountry	shallow rendering using Enzyme	independent component renders without failure
AddPlot	shallow rendering using Enzyme	independent component renders without failure
App	shallow rendering using Enzyme	independent component renders without failure
EditAboriginalCountry	shallow rendering using Enzyme	independent component renders without failure
EditPlot	shallow rendering using Enzyme	independent component renders without failure
Home	shallow rendering using Enzyme	independent component renders without failure
Login	shallow rendering using Enzyme	independent component renders without failure
PlotManager	shallow rendering using Enzyme	independent component renders without failure
Register	shallow rendering using Enzyme	independent component renders without failure
ViewAboriginalCountries	shallow rendering using Enzyme	independent component renders without failure
Redux container being tested	**Action**	**Expectation**
AboriginalCountryList	shallow rendering using Enzyme and mockStore	independent component renders without failure
AddAboriginalCountryForm	shallow rendering using Enzyme and mockStore	independent component renders without failure

AddPlotForm	shallow rendering using Enzyme and mockStore	independent component renders without failure
EditAboriginalCountryForm	shallow rendering using Enzyme and mockStore	independent component renders without failure
EditPlotForm	shallow rendering using Enzyme and mockStore	independent component renders without failure
LoginForm	shallow rendering using Enzyme and mockStore	independent component renders without failure
PlotList	shallow rendering using Enzyme and mockStore	independent component renders without failure
RegisterForm	shallow rendering using Enzyme and mockStore	independent component renders without failure
Redux reducer being tested	**Action**	**Expectation**
ActiveAboriginalCountryReducer	initialise without action	returns specified initial state
ActiveAboriginalCountryReducer	dispatch loadAboriginalCountry	returns expected updated state
ActivePlotReducer	initialise without action	returns specified initial state
ActivePlotReducer	dispatch loadPlot	returns expected updated state
MapDataReducer	initialise without action	returns specified initial state
MapDataReducer	dispatch getAboriginalCountries	returns expected updated state
MapDataReducer	dispatch getPlots	returns expected updated state
MapDataReducer	dispatch deleteAboriginalCountry	returns expected updated state
MapDataReducer	dispatch deletePlot	returns expected updated state
Redux action being tested	**Action**	**Expectation**
addAboriginalCountry	dispatch dummy action with correct type	dummy action is added to the actions within the redux store
addPlot	dispatch dummy action with correct type	dummy action is added to the actions within the redux store
getAboriginalCountries	dispatch dummy action with correct type	dummy action is added to the actions within the redux store
getPlots	dispatch dummy action with correct type	dummy action is added to the actions within the redux store
editAboriginalCountry	dispatch dummy action with correct type	dummy action is added to the actions within the redux store
editPlot	dispatch dummy action with correct type	dummy action is added to the actions within the redux store
loadAboriginalCountry	dispatch dummy action with correct type	dummy action is added to the actions within the redux store
loadPlot	dispatch dummy action with correct type	dummy action is added to the actions within the redux store
deleteAboriginalCountry	dispatch dummy action with correct type	dummy action is added to the actions within the redux store
deletePlot	dispatch dummy action with correct type	dummy action is added to the actions within the redux store
login	dispatch dummy action with correct type	dummy action is added to the actions within the redux store

register	dispatch dummy action with correct type	dummy action is added to the actions within the redux store

To run the test suite, navigate to the /govclient directory and run:

```
npm run test
```

or from the root directory run:

```
npm run govclient-test
```

Below is an example of a successful run of the test suite as of 12/10/18.

Testing Approach

The manual testing performed as part of the creation and code review process for pull requests will involve a system test to ensure existing functionality is not impacted, and the added functionality works as intended.

Part of the development process will include an automated testing suite that will be available for both front-end and back-end code bases. This process would include maintaining, as well adding, artefacts and functionality to the testing suite. Any changes to the master branch must pass the relevant testing suite fully.

The front-end automated testing suite will include both unit testing and integration testing.

The back-end automated testing suite will include integration testing.

Execution

Both testing suites will be executable at the command-line as specified in the documents found under these pages:

Frontend Government Client Testing Suite [Version 2]

Backend Integration Test [Version 3]

Test Failure Process

If in the case that a test failure has been discovered, the tester will make an attempt to identify what the potential cause for the issue is, and then notify the team and/or the individual responsible for the relevant portion of the code. If a change is required, a corresponding issue branch will be created for the appropriate changes to be made, which will go through the pull review process.

Risk Management

- Risk Identification Process

- Risk List
- Risk List [Version 2]

Risk Identification Process

Purpose

- To define the process of risk identification
- Provide examples of risk for reference

Possible Risk Sources

- Product size: risks due to large size or high complexity of the project
- Business impact: risks due to constraints imposed by management or marketplace
- Customer characteristics: risks due to customer sophistication or poor communication with customer (e.g. delayed communication, continuous change of requirements)
- Process definition: risks due to poor definition of software production process, or poor adherence to the process
- Development environment: risks due to low availability of tools, or low quality tools
- Technology to be built: risks due to complexity of system and lack of maturity in use in the real world
- Staff size and experience: risks due to small team size and inexperience of team members

Risk Estimation

- Each risk is to be assigned an estimated probability of occurrence (low, medium, high)
- An estimated impact on the project (on a scale of 1-5, where 5 is catastrophic).

Risk Control

- At any point in time during the project when risks are identified, there needs to be an appropriate way to mitigate the risk.
- Any risks that have already been defined are required to be monitored.
- There also needs to be appropriate management of each risk.

Terms	Definition
Mitigation	Ways to reduce the likelihood and/or impact of risk.
Monitoring	Ways to track the change in a risk's likelihood and impact.
Management	Actions to execute when a risk occurs.

Risk List

ID	Description	Probability	Impact	Mitigation	Monitor	Manage
1	Team fails to deliver expected product by the end of sprint 3	medium	5	Setup clear goals/requirements for each sprint	Track each member's process regularly with stand-ups	Refer to user stories and attempt to deliver high priority features
2	Team falls behind due to lack of experience in tools	high	3	Have members teach each other the usage of tools	Regularly check if members are having problems	Provide assistance immediately
3	Client does not approve components deployed	medium	5	Confirm the suitability of components with client	Regularly update client with current components	Convince client that the component is more suitable than alternatives, or change it based on severity of requirement

4	Team falls behind due to few members	low	3	Alert lecturer/client that the project may be too large for the group, and reject new requirements	Watch out for any new requirements	Catch up by working longer
5	Team loses member/s due to unforeseen circumstances	low	5	Confirm each member's schedule in advance	Track attendance of members	Alert lecturer and request reduction in scope
6	The amount of plot data is too large to load quickly all at once	high	3	Increase code efficiency to reduce loading duration	Monitor loading speeds	Use alternative algorithms to increase performance (e.g. load data in chunks)
7	Product requirements need to change from what was previously specified	medium	3	Confirm product requirements of concern with the client	Track changes using the variation request log	Use the variation request log and confirm that changes have been approved
8	Code quality is poor	medium	2	Use existing code styling guidelines to enforce consistency	Review pull requests as specified by existing processes	Deny or approve code changes depending on whether they meet code quality standards Refactor code to meet the standards
9	Cannot meet client in a timely manner	medium	2	Initiate contact and plan meetings early	Monitor email correspondence	Communicate with supervisor and client to see if any arrangements can be made

Risk List [Version 2]

Changes: Added columns for tracking occurrence of risks

ID	Description	Probability	Impact	Mitigation	Monitor	Manage	Has this occurred?	Result
1	Team fails to deliver expected product by the end of sprint 3	medium	5	Setup clear goals/requirements for each sprint	Track each member's process regularly with stand-ups	Refer to user stories and attempt to deliver high priority features	Not yet.	
2	Team falls behind due to lack of experience in tools	high	3	Have members teach each other the usage of tools	Regularly check if members are having problems	Provide assistance immediately	Yes	Team members cooperated to learn the tools
3	Client does not approve components deployed	medium	5	Confirm the suitability of components with client	Regularly update client with current components	Convince client that the component is more suitable than alternatives, or change it based on severity of requirement	Not yet.	
4	Team falls behind due to few members	low	3	Alert lecturer/client that the project may be too large for the group, and reject new requirements	Watch out for any new requirements	Catch up by working longer	Not yet.	
5	Team loses member/s due to unforeseen circumstances	low	5	Confirm each member's schedule in advance	Track attendance of members	Alert lecturer and request reduction in scope	Not yet.	

6	The amount of plot data is too large to load quickly all at once	high	3	Increase code efficiency to reduce loading duration	Monitor loading speeds	Use alternative algorithms to increase performance (e.g. load data in chunks)	Not yet.	
7	Product requirements need to change from what was previously specified	medium	3	Confirm product requirements of concern with the client	Track changes using the variation request log	Use the variation request log and confirm that changes have been approved	Yes.	Variation request log was updated and confirmed with the client
8	Code quality is poor	medium	2	Use existing code styling guidelines to enforce consistency	Review pull requests as specified by existing processes	Deny or approve code changes depending on whether they meet code quality standards Refactor code to meet the standards	Yes.	Reviewed and refactored codebase.
9	Cannot meet client in a timely manner	medium	2	Initiate contact and plan meetings early	Monitor email correspondence	Communicate with supervisor and client to see if any arrangements can be made	Not yet.	
10	Acceptance tests do not align with product	low	4	Prepare and update the existing acceptance tests ahead of time, and practice delivering.	Check the acceptance tests and monitor whether the product meets them.	Redesign the acceptance test session.	Not yet.	

Acceptance Session Video