BANK BROKER

Web Application connecting to various banks

for transaction data retrieval

PROJECT REPORT

Marian Sheehy

20109018

March 2026

DECLARATION

I declare that the work which follows is my own, and that any quotations from any sources (e.g. books, journals, the internet) are clearly identified as such by the use of ‘single quotation marks’, for shorter excerpt and identified italics for longer quotations. All quotations and paraphrases are accompanied by (date, author) in the text and a fuller citation is the bibliography. I have not submitted the work represented in this report in any other course of study leading to an academic award.

Student: M Sheehy Date: 30/03/2026

TABLE OF CONTENTS

Declaration ..... **ii**

Table of Figures **4**

1. Introduction **5**

1.1 Background *5*

1.2 Objectives & Scope *5*

2. Research **6**

2.1 IAM *6*

2.2 Okta *6*

2.3 Plaid sandbox *7*

2.4 Backend bridge *7*

3. Modelling **8**

3.1 Frontend *8*

3.2 Backend data *9*

4. Implementation **10**

4.1 Technology Stack *10*

4.2 Frontend Implementation *11*

4.3 Backend Implementation *17*

4.4 Integration with plaid sandbox *19*

5. Reflection **25**

5.1 General *25*

5.2 AI Reflection *25*

References **26**

Appendices **28**

TABLE OF FIGURES

Figure 1: High Level Diagram of Front End 8
Figure 2: Simple Diagram of Data Storage 9
Figure 3: Home Page 11
Figure 4: Sign Up Page 12
Figure 5: Log In Page 12
Figure 6: Log In Password Error 12
Figure 7: Dashboard Page 13
Figure 8: Plaid Link UI Pop-up 13
Figure 9: Institution Selection UI 14
Figure 10: Test Version of Bank Login 14
Figure 11: List of Banks Visited with Timestamp 15
Figure 12: List of All Transactions Sync'd 15
Figure 13: Reports Page with CSV Export 16
Figure 14: My Account - User Page 16
Figure 15: MongoDB String from Cluster 17
Figure 16: MongoDB String Reflected in .env file 18
Figure 17: Details of Studio3T Error 18
Figure 18: MongoDB Compass Displaying Collections 19
Figure 19: API Keys from Plaid Sandbox 20
Figure 20: Plaid Keys reflected in .env file 20
Figure 21: plaid-client.ts 21
Figure 22: api-routes.ts 21
Figure 23: Plaid Transactions model 22
Figure 24: Plaid Data Pushed to Server 22
Figure 25: Example of transaction id from Plaid 23
Figure 26: Example of Cursor from Plaid 23
Figure 27: Plaid Model demonstrating auto sync checks 23
Figure 28: Example of Plaid Transaction showing finance category and png ico .24

1. INTRODUCTION

Bank Broker is a proof-of-concept web application for advanced bank reporting capabilities. Current banking applications are dated in their user friendliness. They are limited in their reporting selection and tend to have very restricted access to historic data. This application will be a user hub with links to all main banks, where the user can access all the data in one place, perform more exhaustive data analysis and create more meaningful financial reporting.

GitHub Repository can be found here: https://github.com/MarianSheehy/bankBroker

1.1 BACKGROUND

My background is in Finance, having used many different software packages over the years. Much of the main financial software providers now allow connection to bank feeds. They sync data in real-time so that finance professionals can access bank transactions through their software. The alternative options to bank feeds include manual input or csv imports (which most software packages allow). The time-saving benefit to bank feeds is that the users don't have to log in to the banks own websites to access the transactions. This can be a huge time-saver for entities with multiple bank accounts.

While there are many applications with bank feeds within full finance packages, this concept allows for simply linking to banks and creating reports. This could be a beneficial tool for personal bank users; for people who require bank transaction reports for loan applications etc. They could access from one login instead of chasing several institutions with varying response times.

1.2 OBJECTIVES & SCOPE

A web application, allowing users to connect to all of their financial institutions, to sync their transactions into one place; manipulate, categorise and download any combination of the transactions they require. The application is clean and simple but effective.

The scope of the project is such that we are not permitted to use sensitive data. Therefore this project uses the sandbox version of Plaid. It relies on test data, which restricts how well it can demonstrate real-world usage.

The project utilises Plaid's personal finance categories which allows personal users to login and categorise their transactions. An objective for additional application features,

post project, will be to link Plaid for use by business users. Plaid is currently introducing capabilities for its business customers, so that when entities sync their transactions, it will allow auto-posting to their correct cost centres, using an automatic business category feature.

One more objective for this application is to create a mobile version.

2. RESEARCH

2.1 IAM

Identity and Access Management (IAM) in banking software involves determining and implementing policies to ensure the correct access levels are applied to the correct user/s and that the reasons behind these policies are rational. It needs to be the right balance between sufficient security, user-friendliness and efficiency. IAM has evolved over the years; from username and password being the main method of authenticating, to the current era of identity driven security, with device identity and biometrics.

I explored existing software used in IAM specifically relating to the banking sector. Okta is a popular IAM solution, and Plaid Sandbox is a platform where you can simulate secure authentication flows for users connecting their bank accounts via Plaid.

Integrating Plaid Sandbox with Okta is not a direct, built-in integration. Instead, you integrate your application with Okta (for login/auth) and that application then calls the Plaid Sandbox APIs for banking data.

2.2 OKTA

Okta is a cloud-based identity and access management platform that helps organisations securely manage user authentication, authorisation, and access to applications and services from a central location. It provides features such as Single Sign-On (SSO), Multi-Factor Authentication (MFA), user lifecycle management, and directory integration so that users can access multiple systems with one set of credentials while administrators retain fine-grained control over who can access what. Okta's Identity Cloud is delivered as a scalable software-as-a-service solution and includes extensive integration support for thousands of cloud and on-premises

applications, making it suitable for both workforce and customer identity scenarios. In the context of this project, Okta is used to verify that a user is who they claim to be and to issue secure tokens that the Bank Broker backend can trust before it retrieves any banking data from external providers

2.3 PLAID SANDBOX

Plaid is a financial data network that connects consumer bank accounts to third-party applications such as personal finance tools, payment apps, and digital banks through a unified API. Instead of each application building and maintaining separate integrations with thousands of financial institutions, developers integrate once with Plaid to access standardised account, balance, and transaction data, as well as related services like identity and income verification. When a user links an account, Plaid provides a secure interface for bank login or OAuth, obtains consent, and then returns only the authorised data to the application, without exposing the user's credentials. This makes Plaid a key enabler for modern fintech solutions that require reliable, structured banking data while maintaining a strong focus on security and user control over shared information.

The Plaid Sandbox is a free and fully-featured environment for application development and testing. All Plaid functionality of both the Plaid API and Plaid Link is supported in the Sandbox environment. The Sandbox provides rich test data, as well as the ability to generate your own test data.

2.4 BACKEND BRIDGE

Okta and Plaid play complementary but distinct roles in enabling secure access to banking data. Okta acts as the identity and access management layer, authenticating users and issuing tokens that the Bank Broker backend trusts before allowing any operations. Plaid, by contrast, is responsible for connecting to financial institutions and providing normalised account, balance, and transaction data once a properly authenticated and authorised user has granted consent. Together, they form a secure flow where Okta confirms who the user is, and Plaid supplies the banking information that Bank Broker uses for reporting and analysis.

3. MODELLING

3.1 FRONTEND

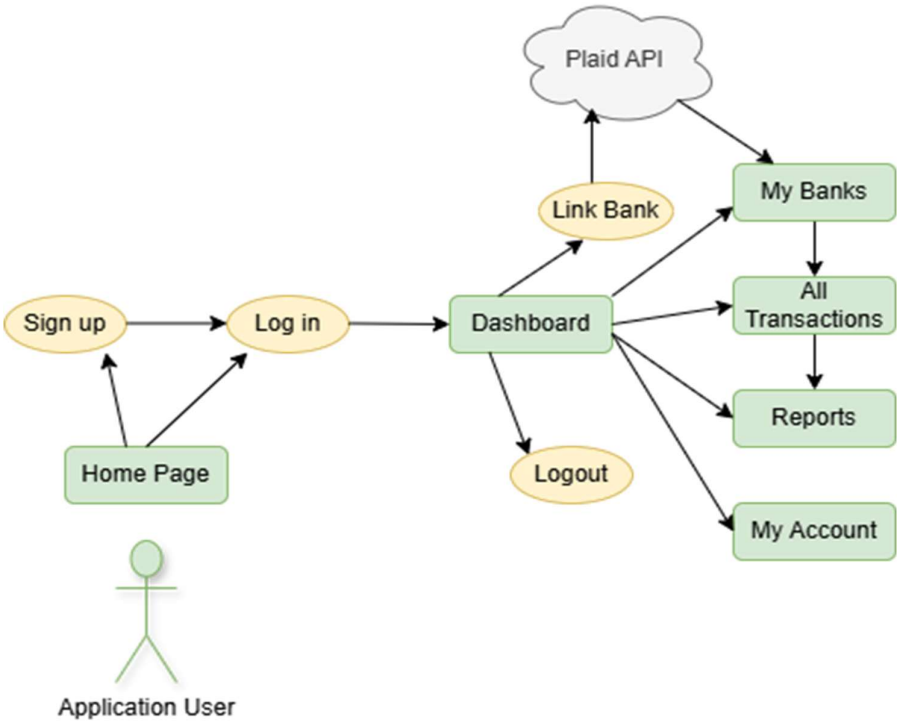


FIGURE 1: HIGH LEVEL DIAGRAM OF FRONT END

When the User opens the webpage they meet the Home Page where they choose to Log in or Sign up. After logging in they are brough to the Dashboard page. There are page options at top menu but main focus is for the user to click on 'Link Bank Account'. When they do this, a Plaid popup appears and the user chooses the institution, account type etc. The sync'd information then appears in 3 places. The 'My Banks' page lists a summary of which institution was visited, with a date and time stamp. There is a button next to the entry which brings the user to the 'All Transactions' page, where the full list of data is stored. The user can visit the 'Reports' page to manipulate the data. It can be filtered based on institution, date, amount and category. The report, as displayed, can be exported to a CSV file. All transactions, and page visits are stored in the application under each user for future visits. The benefit of this is that, over time, transactions can build up over longer periods of time than some of the institutions own records will allow. i.e. BOI only allows viewing of transactions up to 3 months old. Bank Broker can sync it and store it for longer. Users can also access the 'My Account' page where they can see their full name and email address, and can change their password.

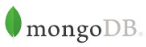
3.2 BACKEND DATA

FIGURE 2: SIMPLE DIAGRAM OF DATA STORAGE

Above is a simple data flow showing how the information is stored in MongoDB. users are created on Sign up. plaiditems are created when a successful Plaid Link is made. plaidtransactions are created when there is available data from the institution and it has sync'd.

4. IMPLEMENTATION

4.1 TECHNOLOGY STACK

Backend	Node.js (TypeScript) with Hapi.js for routing, plugins, and auth	
Views	Handlebars templates + partials with custom helpers (ifEquals, toFixed2)	
Database	MongoDB with Mongoose models for users, Plaid items, and transactions	
Auth & Security	@hapi/cookie session auth for web, hapi-auth-jwt2 for API JWTs, bcryptjs for password hashing, custom password rules on signup/change	
Plaid Integration	Plaid Link on the dashboard plus Node.js Plaid API calls for link token creation, token exchange, and transaction sync	
Frontend UI	Bulma CSS for layout, navbars, tables, cards, and responsive styling	
Reporting & Export	Filterable transaction reports and a CSV export route returning text/csv downloads	

4.2 FRONTEND IMPLEMENTATION

The frontend consists of several main screens, including a login page, a dashboard showing linked accounts, and a reporting view for exploring historic transaction data. After a successful Plaid login, the user is redirected back to the application with an authentication token that is stored securely (for example, in memory or HTTP-only cookies) and attached to subsequent API requests. The dashboard screen calls backend endpoints to load account summaries and recent transactions, while the reporting screen lets users choose date ranges, accounts, and optional filters before requesting a generated report. Components were structured to keep layout, data fetching, and presentation logic separated, making it easier to maintain and extend the interface.

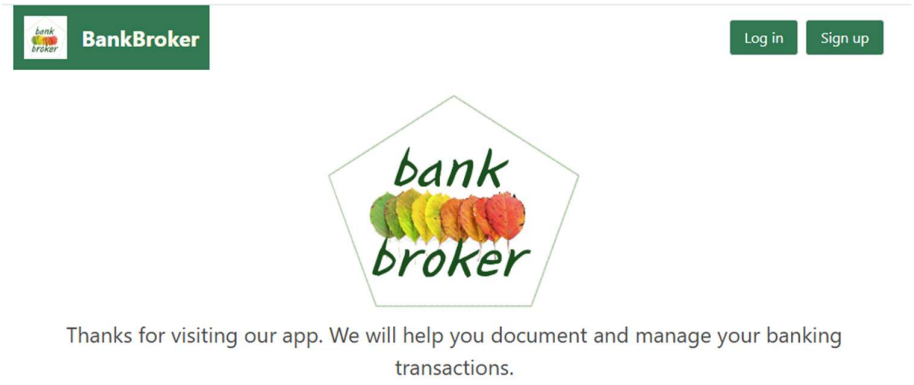


FIGURE 3: HOME PAGE

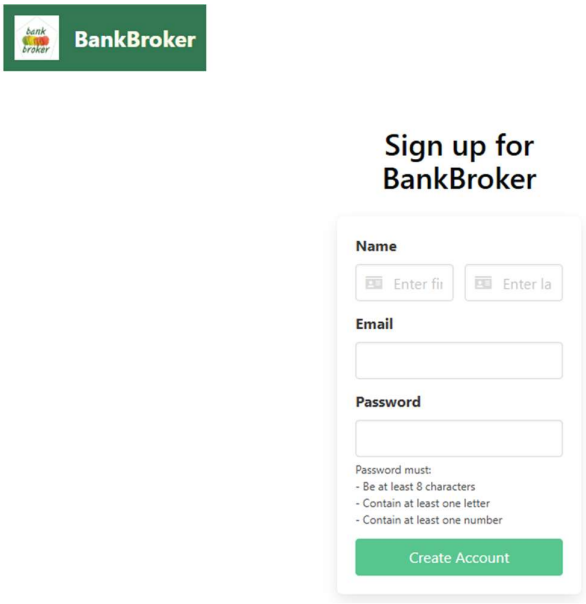


FIGURE 4: SIGN UP PAGE

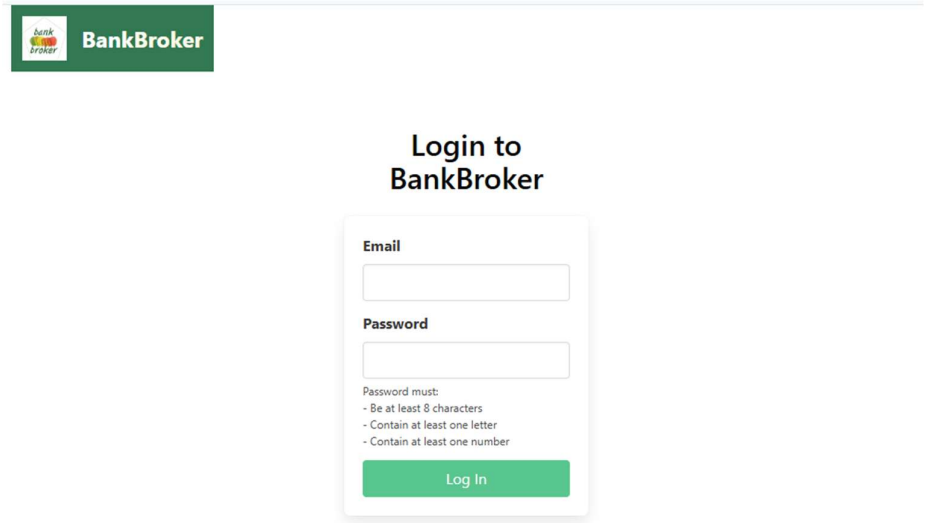


FIGURE 5: LOG IN PAGE

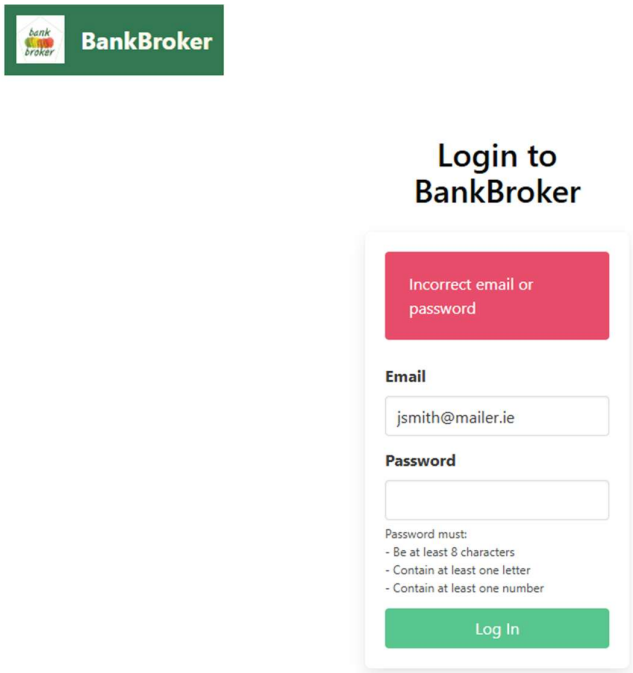


FIGURE 6: LOG IN PASSWORD ERROR

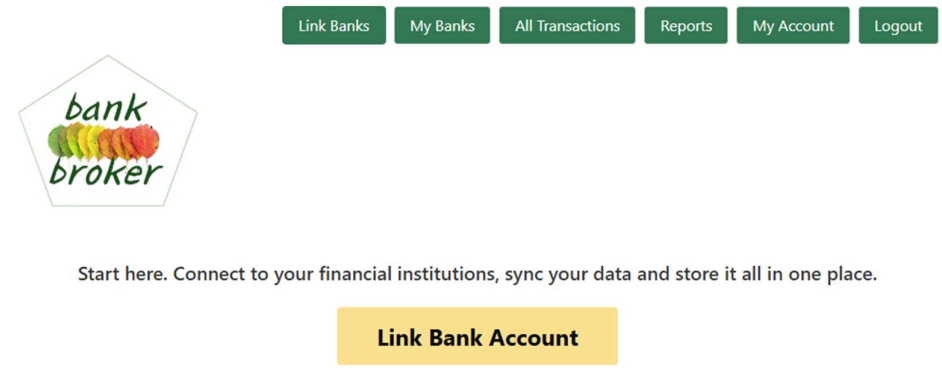


FIGURE 7: DASHBOARD PAGE

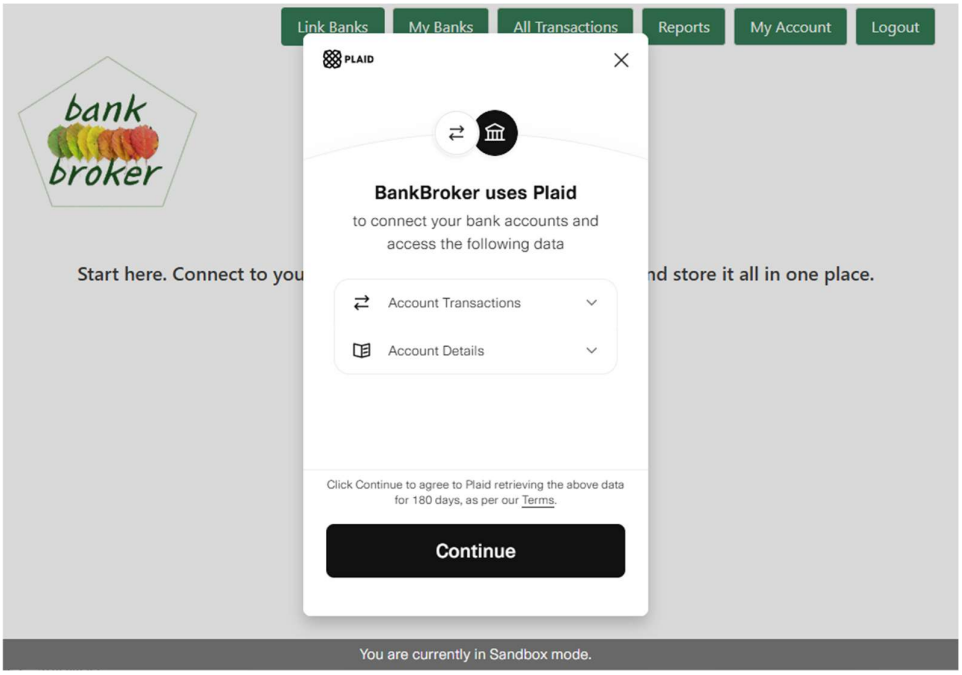


FIGURE 8: PLAID LINK UI POP-UP

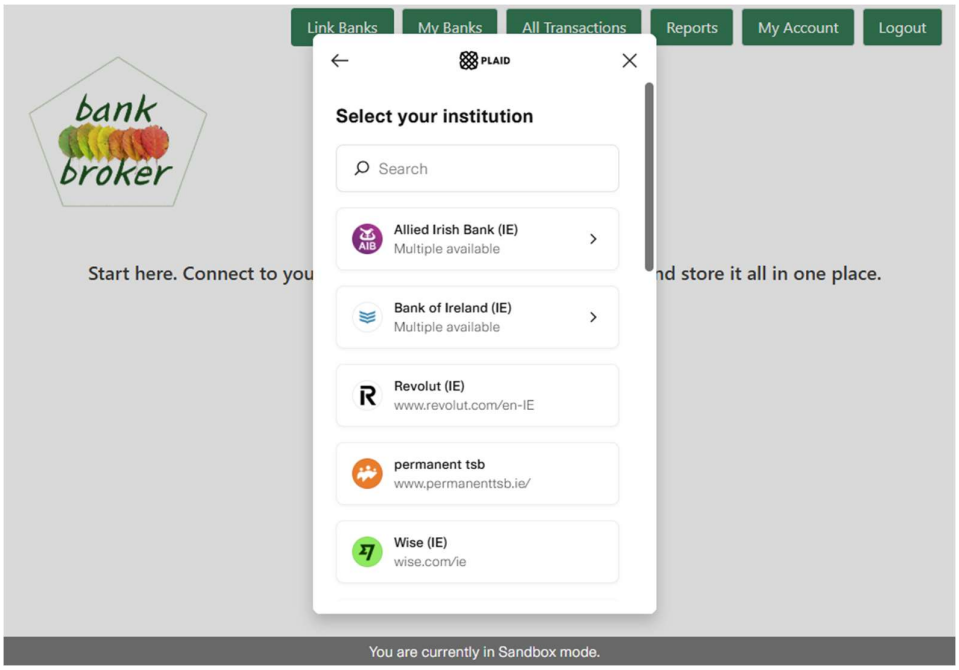


FIGURE 9: INSTITUTION SELECTION UI

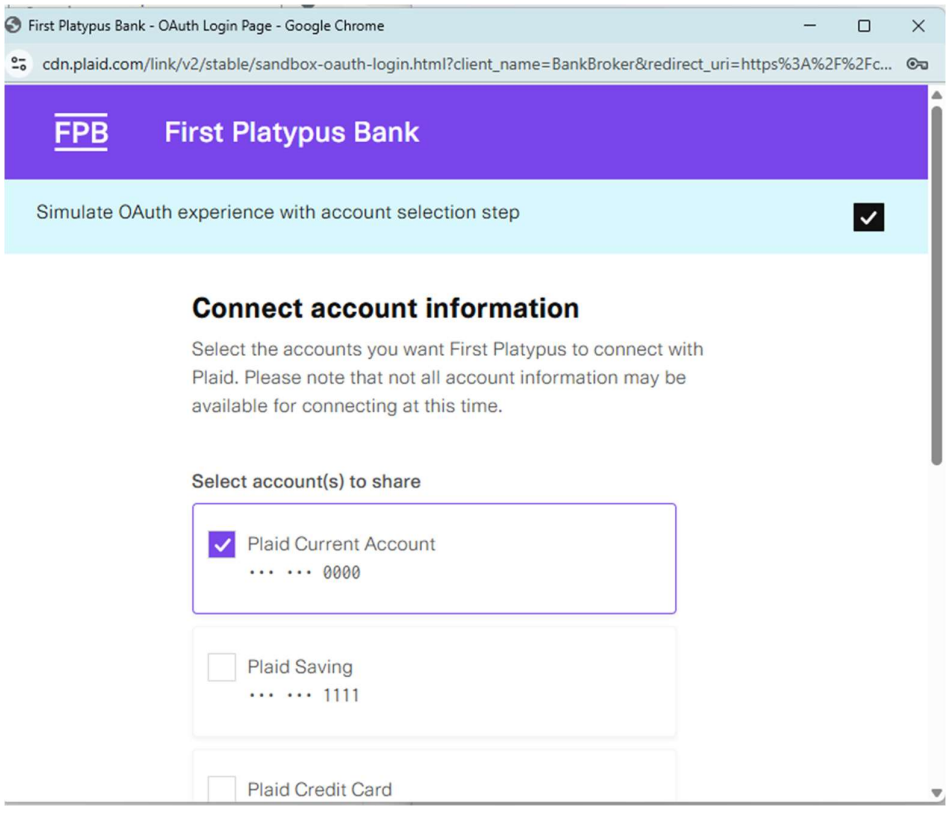


FIGURE 10: TEST VERSION OF BANK LOGIN

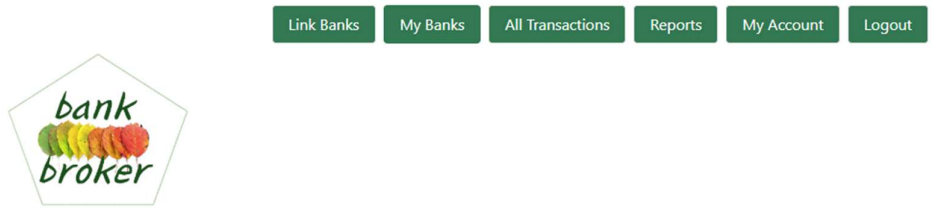


FIGURE 11: LIST OF BANKS VISITED WITH TIMESTAMP

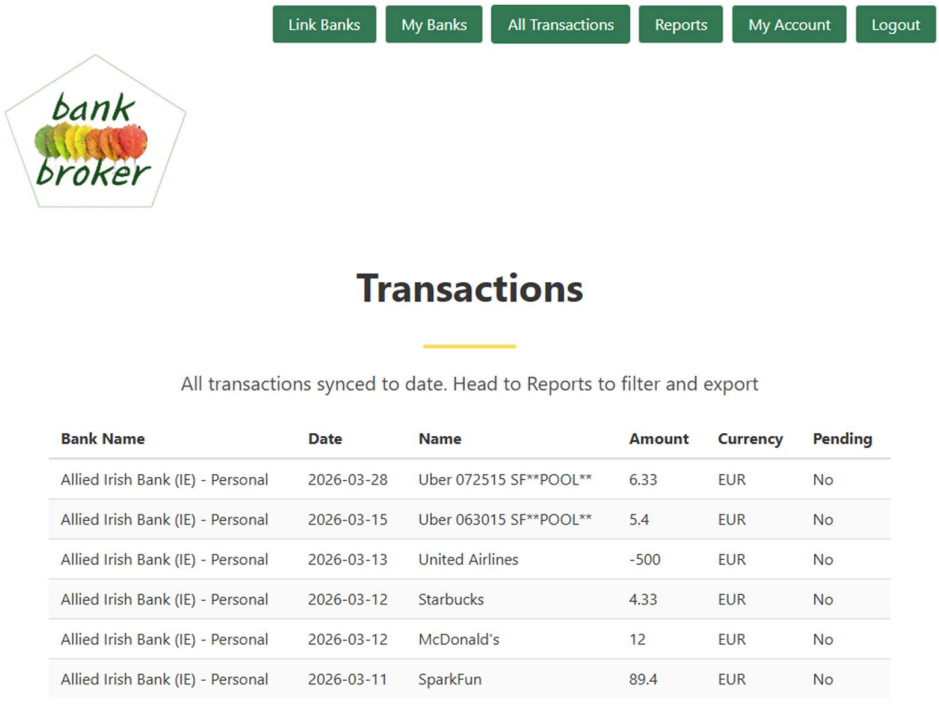


FIGURE 12: LIST OF ALL TRANSACTIONS SYNC'D

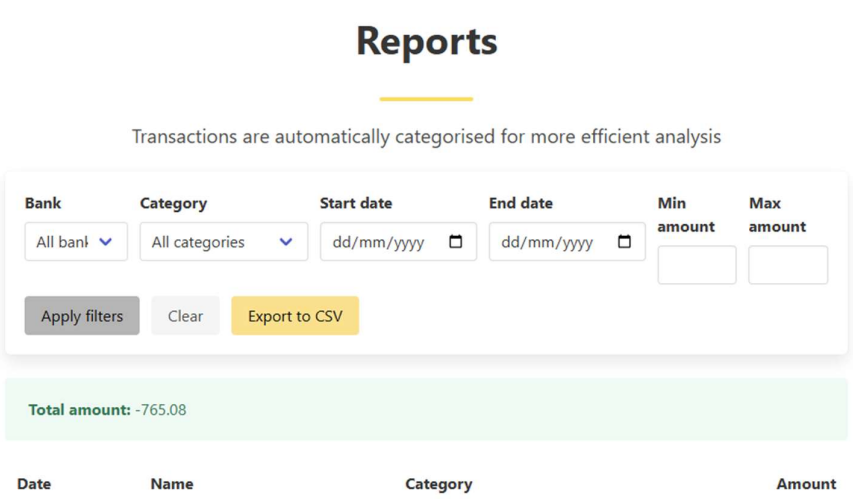


FIGURE 13: REPORTS PAGE WITH CSV EXPORT

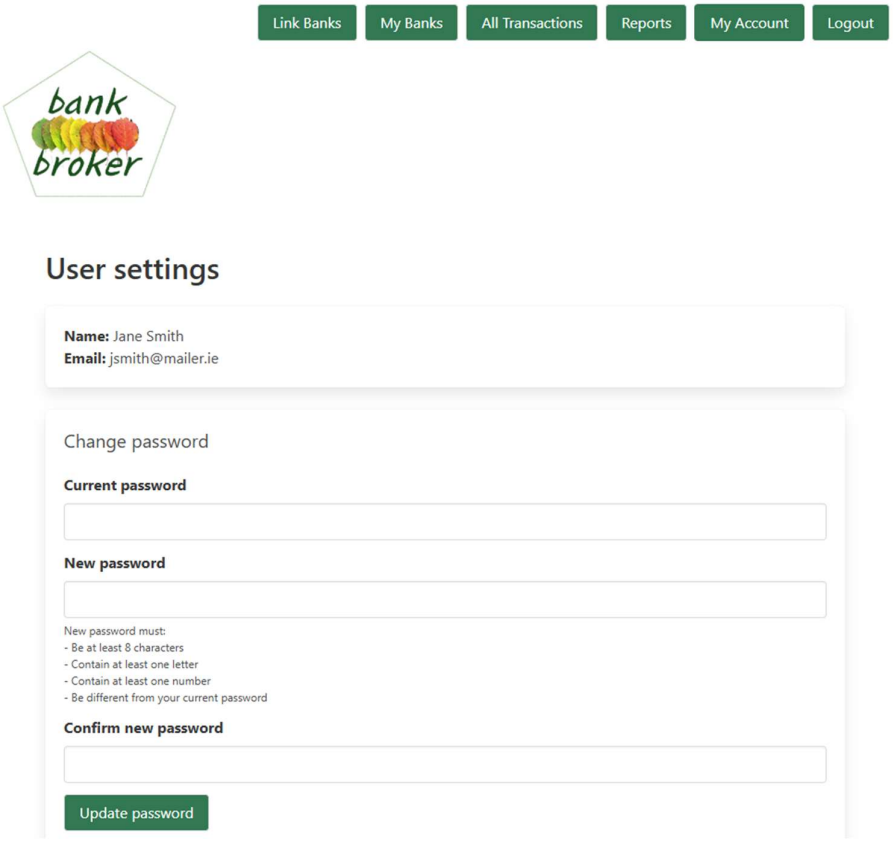


FIGURE 14: MY ACCOUNT - USER PAGE

4.3 BACKEND IMPLEMENTATION

The backend acts as the bridge between the frontend and the external Plaid Sandbox APIs. The backend uses Plaid client libraries or HTTPS calls to request account and transaction information, then aggregates this data into the format required by the frontend. Additional filtering is implemented on the server to reduce the amount of processing required in the browser.

The MongoDB backend implementation stores all linked banking data in a structured, way so the Hapi server can render dashboards, transactions, and reports efficiently. Using Mongoose models like PlaidItem and PlaidTransaction, the app maps each user's Items and their Plaid transactions into collections, capturing fields such as userId, itemId, account and transaction IDs, amounts, dates, and the raw Plaid payload, while also adding app-specific fields like userCategory, for custom categorisation. Controllers query these models with filters for user, account, date range, amount range, and category, then perform simple aggregations (for example, computing totals) before passing objects into Handlebars views, which keeps the UI responsive and makes it easy to extend the schema (for bank names, Plaid personal finance category codes, or report metadata) without changing the overall request–response flow.

Steps Taken to Integrate MongoDB

- Login to MongoDB – using organisation setup in previous lab – added new project called bankBroker
- Created a cluster with AWS as provider – selected to allow access from anywhere and created user/password
- Connected to driver, obtained connection string and saved this in .env file

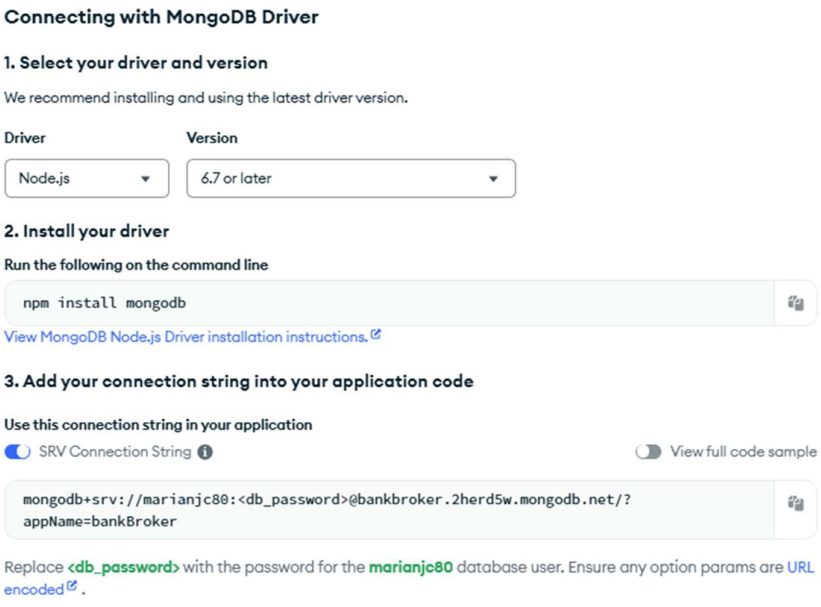


FIGURE 15: MONGODB STRING FROM CLUSTER

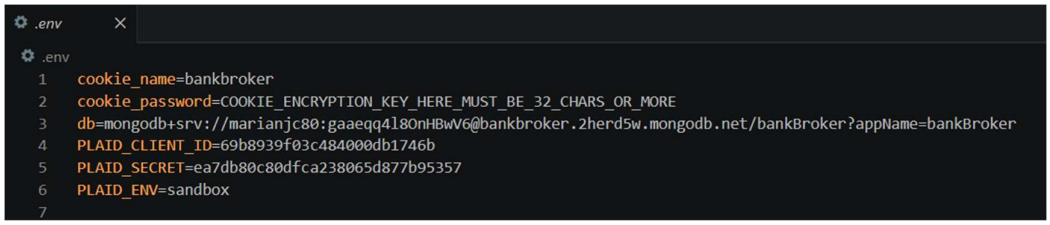


FIGURE 16: MONGODB STRING REFLECTED IN .ENV FILE

- Connected to Studio3T but I could not access the database collections as I was receiving this error

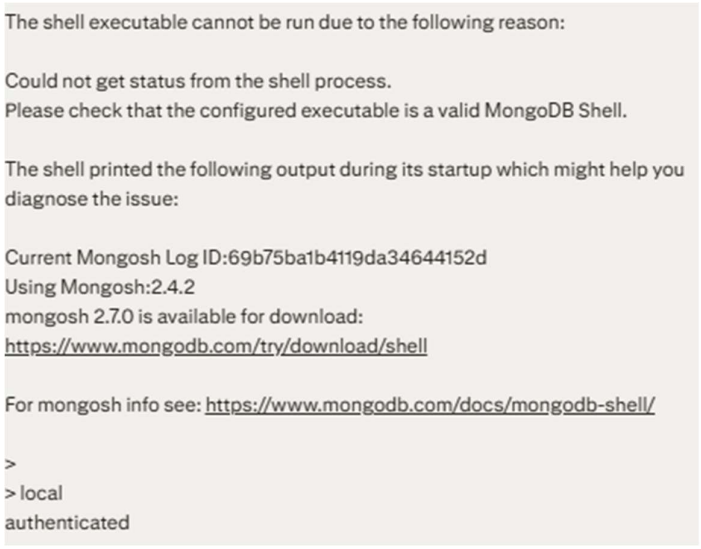


FIGURE 17: DETAILS OF STUDIO3T ERROR

- I asked Perplexity what the error meant – the response was:

'Studio 3T is bundling an older mongosh 2.4.2 by default, while your system has the newer mongosh 2.7.0 (which successfully connected to Atlas). This version mismatch causes Studio 3T's IntelliShell to fail when it tries to launch and communicate with the shell process.'

- I tried a quick fix and then instead switched to using MongoDB Compass with instant success. The connection string used was bankbroker.2herd5w.mongodb.net – same as used in .env file.

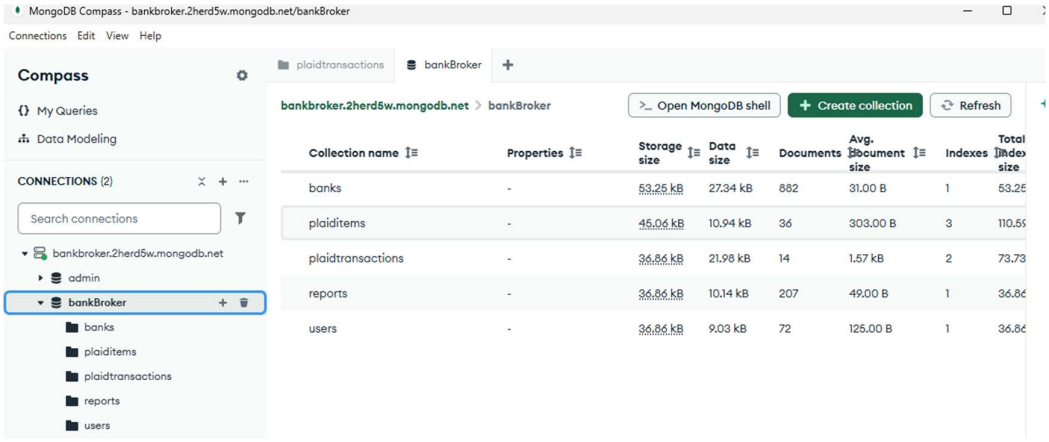


FIGURE 18: MONGODB COMPASS DISPLAYING COLLECTIONS

4.4 INTEGRATION WITH PLAID SANDBOX

Plaid Sandbox is integrated by registering a Plaid application, obtaining API keys, and configuring them as secure environment variables in the backend. The frontend uses Plaid Link (or a similar widget) to allow users to simulate connecting a bank account; when the user completes this flow, a temporary public token is returned to the frontend and then sent to the backend. The backend exchanges this public token for a long-lived access token using the Plaid API, stores it securely (for example, in memory or a development datastore), and then uses it to call endpoints for retrieving accounts and transactions. These results are processed and returned to the frontend, where they are displayed in the dashboard and used to populate the reporting views.

Steps Taken to Integrate Plaid

- Signed up for a free account at plaid.com/docs/quickstart
- Once logged in, accessed ‘API Keys’ from user dashboard
- Added the keys into .env file
- Installed the package – npm install plaid
- Added a plaid client module @ src/api/plaid-client.ts
- Added plaid API routes @ src/api/plaid-api.ts
- Added the routes @ src/api-routes.ts

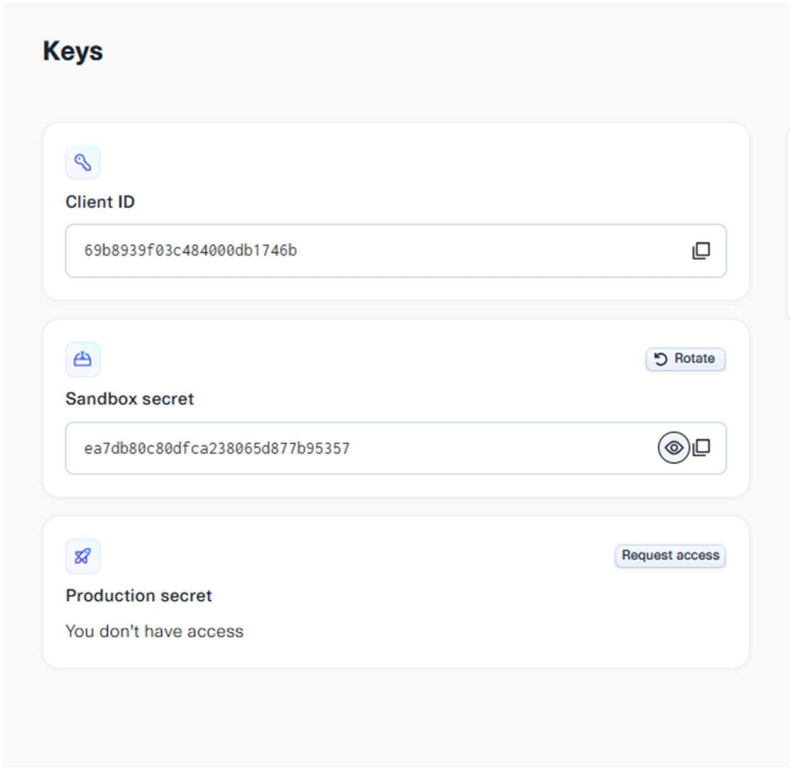


FIGURE 19: API KEYS FROM PLAID SANDBOX

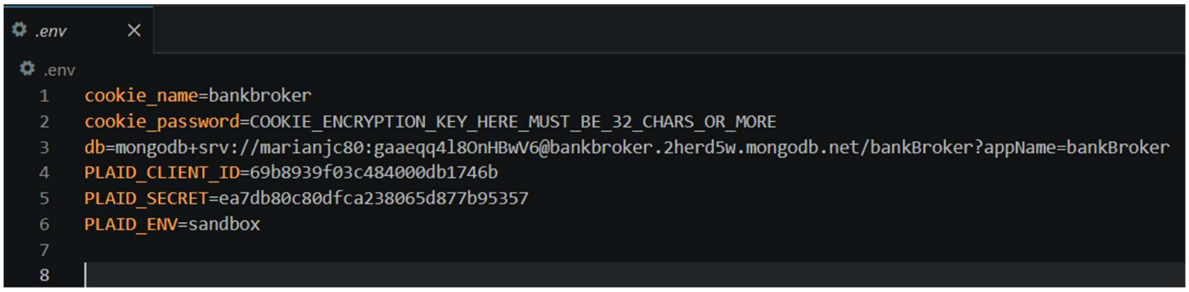


FIGURE 20: PLAID KEYS REFLECTED IN .ENV FILE

plaid-client.ts

src > api > plaid-client.ts > ...

```ts
import { Configuration, PlaidApi, PlaidEnvironments } from "plaid";
import dotenv from "dotenv";

dotenv.config();

const configuration = new Configuration({
  basePath: PlaidEnvironments[process.env.PLAID_ENV || "sandbox"],
  baseOptions: {
    headers: {
      "PLAID-CLIENT-ID": process.env.PLAID_CLIENT_ID,
      "PLAID-SECRET": process.env.PLAID_SECRET,
    },
  },
});

export const plaidClient = new PlaidApi(
  new Configuration({
    basePath: PlaidEnvironments.sandbox,
    baseOptions: {
      headers: {
        "PLAID-CLIENT-ID": process.env.PLAID_CLIENT_ID!,
        "PLAID-SECRET": process.env.PLAID_SECRET!,
      },
    },
  })
);
```

FIGURE 21: PLAID-CLIENT.TS

api-routes.ts

src > api-routes.ts > ...

```ts
import { userApi } from "./api/user-api.js";
import { bankApi } from "./api/bank-api.js";
import { plaidApi } from "./api/plaid-api.js";

export const apiRoutes = [
  { method: "GET", path: "/api/users", config: userApi.find },
  { method: "POST", path: "/api/users", config: userApi.create },
  { method: "DELETE", path: "/api/users", config: userApi.deleteAll },
  { method: "GET", path: "/api/users/{id}", config: userApi.findOne },
  { method: "POST", path: "/api/users/authenticate", config: userApi.authenticate },

  { method: "GET", path: "/api/banks", config: bankApi.find },
  { method: "DELETE", path: "/api/banks", config: bankApi.deleteAll },
  { method: "DELETE", path: "/api/banks/{id}", config: bankApi.deleteOne },

  { method: "POST", path: "/api/plaid/create-link-token", config: plaidApi.createLinkToken },
  { method: "POST", path: "/api/plaid/exchange-token", config: plaidApi.exchangeToken },
  { method: "POST", path: "/api/plaid/accounts", config: plaidApi.getAccounts },
  { method: "POST", path: "/api/plaid/transactions", config: plaidApi.getTransactions },
  { method: "POST", path: "/api/plaid/transactions/sync", config: plaidApi.syncTransactions },
  { method: "GET", path: "/api/plaid/transactions/local", config: plaidApi.getLocalTransactions },
];
```

FIGURE 22: API-ROUTES.TS

Plaid Transactions

/transactions/get – this endpoint will retrieve transactions but won't allow for any amendments to those transactions done on the bank end. This would be cumbersome for the application, as it would have to reconcile the transactions against the data previously retrieved.

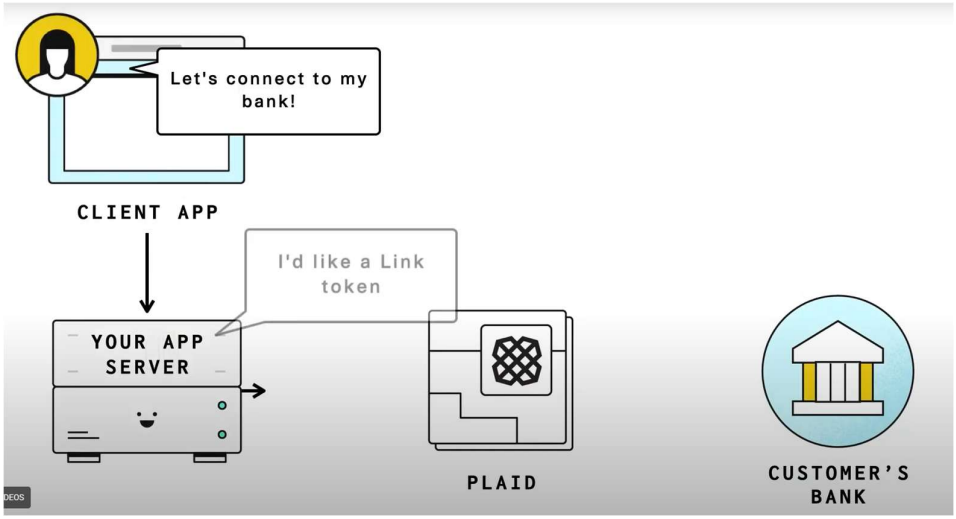


FIGURE 23: PLAID TRANSACTIONS MODEL

/transactions/sync – this endpoint is recommended as it keeps the transaction dataset incrementally up to date instead of performing a full reload every time. The API appears more complicated but it is much more time-saving in the long run.

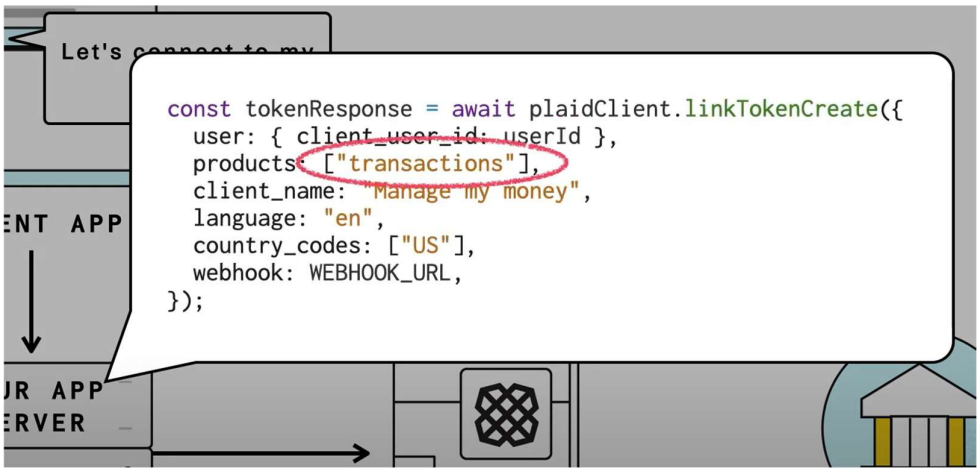


FIGURE 24: PLAID DATA PUSHED TO SERVER

Transactions downloaded from the bank are given a 'transaction_id' along with the usual details (description, amount etc). It is important that the transaction_id is stored in the database. The next time you sync the data with the bank, the same transaction_id will be linked to the same transaction, but any modifications will be overwritten.

"transaction_id": "c3hQy0lH33tZc0XxEZPl4GOwYvm63SFcA59bu",

FIGURE 25: EXAMPLE OF TRANSACTION ID FROM PLAID

At the end of the 'sync call' you are given a cursor (which is like a bookmark). Next time you call sync you include the cursor value. Plaid will only send a list of changes that have happened since the previous cursor. Plaid checks for new transaction data a few times per day, and when updates are available it sends a webhook notification. The app then uses the latest sync cursor to fetch the changes.

"next_cursor": "dIHjGhbtMWI9cQFBLiSXZpnghvbC9lLU32TydCvMc88rPsz2xhXIdfAPMyyhtz0ELGz9Dlc95MwrB2wUzMlac9P=",

FIGURE 26: EXAMPLE OF CURSOR FROM PLAID

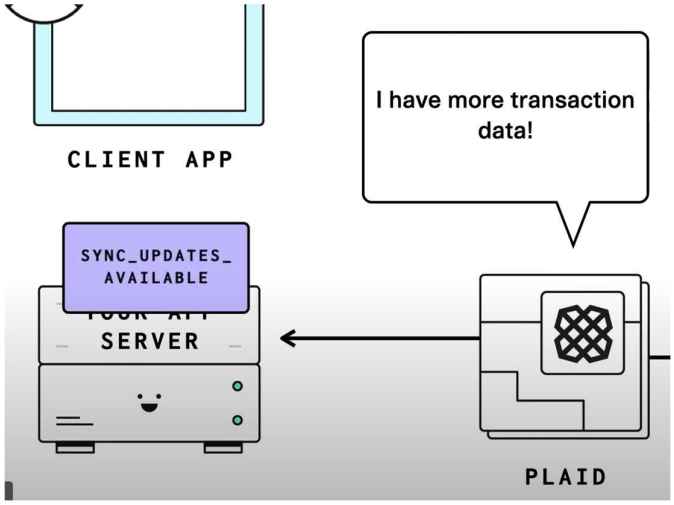


FIGURE 27: PLAID MODEL DEMONSTRATING AUTO SYNC CHECKS

Saving the Sync'd Data

Plaid does not write to the database. The server has to call /transactions/sync, to read the results and save them. When the user finishes Plaid Link, the public_token is exchanged for an access_token. This is then saved in the database along with the Plaid item_id and an empty cursor. As instructed by Plaid, the first sync is done with no cursor value, and the cursor is what lets later syncs fetch only changes.

- Mongo collection created for Plaid items - src/models/mongo/plaid-item-store.ts – this stores itemId, accessToken and cursor for each record
- Mongo collection created for Plaid transactions – src/models/mongo.plaid-transaction-store.ts – this stores all the transaction data with transaction_id as the unique key

- Plaid client created and exported at src/api/plaid-client.ts

```
export const plaidClient = new PlaidApi(
  new Configuration({
    basePath: PlaidEnvironments.sandbox,
    baseOptions: {
      headers: {
        "PLAID-CLIENT-ID": process.env.PLAID_CLIENT_ID!,
        "PLAID-SECRET": process.env.PLAID_SECRET!,
      },
    },
  })
);
```

- plaid-api.ts updated for API helper functions
  - exchangePublicToken(publicToken)
  - syncTransactions(itemId)
  - saveItem(...)
  - saveTransactions(...)

Plaid Transaction Categories

Plaid provides a CSV taxonomy file for general use. It includes an extensive list of personal finance categories, of which transactions are placed into. Application designers can use this file as a static master list of every possible personal finance category Plaid supports. Firstly, I linked this file to my sync'd transaction data with a dropdown selection option, so the user could assign a category to a transaction.

I then noticed from the transaction data in Mongo, that Plaid assigns a personal_finance_category to every transaction. It also assigned a png icon accordingly. I implemented changes to allow this information to feed into the Reports page of the application.

```
_id: ObjectId('69c95d89071c2fe8d0c6bd15')
userId : "69c95cdc5c0dab95b4429c1f"
itemId : "lKeevj6BnnI6K3Av8Pqetg64xKv6lruLMyzE4"
transactionId : "x7AA4yvknnIrLdpeZ4qRULk1alBRR6iyekJPa"
__v : 0
accountId : "rdppz1vNnnikdBNaVz9nUr8P6yqlPmcdlR8Dq"
amount : 12
createdAt : 2026-03-29T17:12:41.114+00:00
date : "2026-03-12"
isoCurrencyCode : "EUR"
merchantName : null
name : "McDonald's"
pending : false
personalFinanceCategory : Object
  primary : "FOOD_AND_DRINK"
  detailed : "FOOD_AND_DRINK_FAST_FOOD"
  confidence_level : "LOW"
  version : "v2"
  _id : ObjectId('69c95d895c0dab95b4429c3b')
personalFinanceCategoryIconUrl : "https://plaid-category-icons.plaid.com/PFC_FOOD_AND_DRINK.png"
raw : Object
updatedAt : 2026-03-29T17:12:41.114+00:00
userCategory : null
```

FIGURE 28: EXAMPLE OF PLAID TRANSACTION SHOWING FINANCE CATEGORY AND PNG ICO

5. Reflection

5.1 General

My personal reflection on this project is a lesson that you almost never end up doing what you want or expect to do. At the beginning of this course, I envisioned a revolutionary final project; an app that would change the world but now at the eleventh hour, I am just happy that I didn't give up! It may seem to those with the years of knowledge in this field, that the effort could have been more but really what the final project comes down to it stamina (and work/life/study balance). This application is simple, but it works, and in the short amount of time I dedicated to it (in comparison to what I would have liked to dedicate), I have learned a lot. No regrets! I now know about bcrypt hashing and am looking forward to building this application into something more professional.

5.2 AI Reflection

I used the AI tool Perplexity in this project, to implement the code for Plaid integration. Overall I found it excellent in helping me. It reminded me of the web development labs in the HDip. It would explain why something was going in such a page and then give the code. I was able to implement it step by step while the app was running. I could see the changes immediately. It was also helpful in pinpointing the cause of errors, which were few. I mentioned this in Section 4.3, that it was helpful to shed more light on an error from Studio 3T.

My takeaway from using this tool is that people should use this in coding where feasible, for work and study, on the basis that they have enough knowledge to ascertain if it is giving the correct response or not. As mentioned above, I have enough knowledge to know where to place the code, why it is amending a controller or a route etc.

To conclude on this, I do think that it is still important for students to learn the 'old way', step by step and that AI can be a later tool for speeding things up. To use a finance analogy, it would be like saying you don't need to learn how to calculate because Excel will do it for you. AI will be invaluable for many professions but we don't want to risk dumbing ourselves down.

REFERENCES

Bulma.io. (2026). Color helpers. [online] Available at: https://bulma.io/documentation/helpers/color-helpers/ [Accessed 28 Mar. 2026].

bulma.io. (n.d.). Concepts. [online] Available at: https://bulma.io/documentation/customize/concepts/. [Accessed 28 Mar. 2026].

code.visualstudio.com. (n.d.). Build Node.js Apps with Visual Studio Code. [online] Available at: https://code.visualstudio.com/docs/nodejs/nodejs-tutorial. [Accessed 22 Mar. 2026].

GitHub. (2025). quickstart/frontend at master · plaid/quickstart. [online] Available at: https://github.com/plaid/quickstart/tree/master/frontend [Accessed 22 Mar. 2026].

Perplexity AI. (2026a). Perplexity. [online] Available at: https://www.perplexity.ai/search/how-can-i-get-plaid-sandbox-ba-acjmSfN9Q5qYySaZet9oSA [Accessed 22 Mar. 2026].

Perplexity AI. (2026c). Perplexity. [online] Available at: https://www.perplexity.ai/search/assertionerror-err-assertion-m-QoMN_PLmTUS2BicLGkuDAw [Accessed 16 Mar. 2026].

Plaid (2025). Plaid Effects 2025 | Business accounts & categorization update. [online] YouTube. Available at: https://www.youtube.com/watch?v=yjwx2MmzzA0 [Accessed 28 Mar. 2026].

plaid (2021). GitHub - plaid/plaid-node: Node bindings for Plaid. [online] GitHub. Available at: https://github.com/plaid/plaid-node [Accessed 30 Mar. 2026].

plaid (2025). tutorial-resources/transactions/finished/server at main · plaid/tutorial-resources. [online] GitHub. Available at: https://github.com/plaid/tutorial-resources/tree/main/transactions/finished/server [Accessed 22 Mar. 2026].

Plaid.com. (2026). Transactions. [online] Available at: http://plaid.com/docs/transactions [Accessed 01 Mar. 2026].

Sheehy, M. (n.d.). https://github.com/MarianSheehy/BirdWatch.

Tutors.dev. (2026). Available at: https://tutors.dev/lab/wit-hdip-comp-sci-2024-full-stack-1/unit-3-deployment/topic-12-deployment/unit-0/book-0-mongo-atlas/01 [Accessed 15 Mar. 2026].

W3schools (2019). Java Tutorial. [online] W3schools.com. Available at: https://www.w3schools.com/java/default.asp. [Accessed 10 Feb. 2026].

www.youtube.com. (n.d.). Plaid Transactions Full Tutorial -- Plaid Academy. [online] Available at: https://www.youtube.com/watch?v=Pin0-ceDKcI [Accessed 22 Mar. 2026].

APPENDICES

Step 1: Get Plaid API Keys

1. Sign up for a free account at plaid.com/docs/quickstart `plaid`
2. From the Plaid Dashboard, grab your `client_id` and `secret` (start with the **Sandbox** environment for testing)
3. Add them to your `.env` file:

```text
PLAID_CLIENT_ID=your_client_id_here
PLAID_SECRET=your_sandbox_secret_here
PLAID_ENV=sandbox
```

Step 2: Install the Plaid Node Package

```bash
npm install plaid
```

Step 3: Create a Plaid Client Module

Create `src/api/plaid-client.ts`:

```ts
import { Configuration, PlaidApi, PlaidEnvironments } from "plaid";
import dotenv from "dotenv";

dotenv.config();

const configuration = new Configuration({
  basePath: PlaidEnvironments[process.env.PLAID_ENV || "sandbox"],
  baseOptions: {
    headers: {
      "PLAID-CLIENT-ID": process.env.PLAID_CLIENT_ID,
      "PLAID-SECRET": process.env.PLAID_SECRET,
    },
  },
});

export const plaidClient = new PlaidApi(configuration);
```

Step 4: Add Plaid API Routes

Create `src/api/plaid-api.ts`:

```ts
import Boom from "@hapi/boom";
import type { Request, ResponseToolkit } from "@hapi/hapi";
import { plaidClient } from "./plaid-client.js";
import { CountryCode, Products } from "plaid";

export const plaidApi = {
  // Step 1: Create a link_token for the frontend
  createLinkToken: {
    auth: { strategy: "jwt" },
    handler: async function (request: Request, h: ResponseToolkit) {
      try {
        const user = request.auth.credentials as any;
        const response = await plaidClient.linkTokenCreate({
          user: { client_user_id: user._id },
          client_name: "BankBroker",
          products: [Products.Auth, Products.Transactions],
          language: "en",
          country_codes: [CountryCode.Ie], // Ireland
        });
        return h.response({ link_token: response.data.link_token }).code(200);
      } catch (err) {
        return Boom.serverUnavailable("Failed to create link token");
      }
```

```
      }
    },
    tags: ["api"],
    description: "Create a Plaid Link token",
  },

  // Step 2: Exchange public_token for access_token
  exchangeToken: {
    auth: { strategy: "jwt" },
    handler: async function (request: Request, h: ResponseToolkit) {
      try {
        const { public_token } = request.payload as { public_token: string };
        const response = await plaidClient.itemPublicTokenExchange({ public_token
});
        const accessToken = response.data.access_token;
        const itemId = response.data.item_id;

        // TODO: Store accessToken in your database linked to the user
        console.log("Plaid access token received for item:", itemId);

        return h.response({ success: true, item_id: itemId }).code(200);
      } catch (err) {
        return Boom.serverUnavailable("Failed to exchange token");
      }
    },
    tags: ["api"],
    description: "Exchange Plaid public token for access token",
  },
```

```
  // Step 3: Fetch accounts using stored access_token
  getAccounts: {
    auth: { strategy: "jwt" },
    handler: async function (request: Request, h: ResponseToolkit) {
      try {
        const { access_token } = request.payload as { access_token: string };
        const response = await plaidClient.accountsGet({ access_token });
        return h.response(response.data.accounts).code(200);
      } catch (err) {
        return Boom.serverUnavailable("Failed to fetch accounts");
      }
    },
    tags: ["api"],
    description: "Get linked bank accounts",
  },
```

```
  // Step 4: Fetch transactions
  getTransactions: {
    auth: { strategy: "jwt" },
    handler: async function (request: Request, h: ResponseToolkit) {
      try {
        const { access_token, start_date, end_date } = request.payload as {
          access_token: string;
          start_date: string;
          end_date: string;
        };
        const response = await plaidClient.transactionsGet({
          access_token,
          start_date,
          end_date,
        });
        return h.response(response.data.transactions).code(200);
      } catch (err) {
        return Boom.serverUnavailable("Failed to fetch transactions");
      }
    },
    tags: ["api"],
    description: "Get transactions for a linked account",
  },
};
```

Step 5: Register the Routes

Add these to your `src/api-routes.ts`:

```ts
import { plaidApi } from "./api/plaid-api.js";

// Add to the existing apiRoutes array:
{ method: "POST", path: "/api/plaid/create-link-token", config:
plaidApi.createLinkToken },
{ method: "POST", path: "/api/plaid/exchange-token", config: plaidApi.exchangeToken
},
{ method: "POST", path: "/api/plaid/accounts", config: plaidApi.getAccounts },
{ method: "POST", path: "/api/plaid/transactions", config: plaidApi.getTransactions
},
```

How the Flow Works

The overall pattern according to the Plaid docs is: `plaid`

1. Your server calls `/api/plaid/create-link-token` → gets a temporary `link_token`
2. The frontend uses that `link_token` to open Plaid Link (a secure bank login widget)
3. User logs into their bank through the widget → Plaid returns a `public_token`
4. Your server exchanges the `public_token` for a permanent `access_token` via `/api/plaid/exchange-token`
5. Use the `access_token` to fetch accounts, transactions, balances, etc.

Testing in Sandbox

In sandbox mode, use these test credentials when Plaid Link opens: `plaid`

- Username: `user_good`
- Password: `pass_good`

This gives you mock financial data to develop against before going live.

plaid-controller.ts

```
/**
 * Web controller for the Plaid Link page.
 * Renders the Handlebars view that contains the Plaid Link widget.
 */

import type { Request, ResponseToolkit } from "@hapi/hapi";

export const plaidController = {
  index: {
    handler: async function (request: Request, h: ResponseToolkit) {
      const loggedInUser = request.auth.credentials;
      return h.view("plaid-view", {
        title: "Link Your Bank Account",
        user: loggedInUser,
      });
    },
  },
};
```