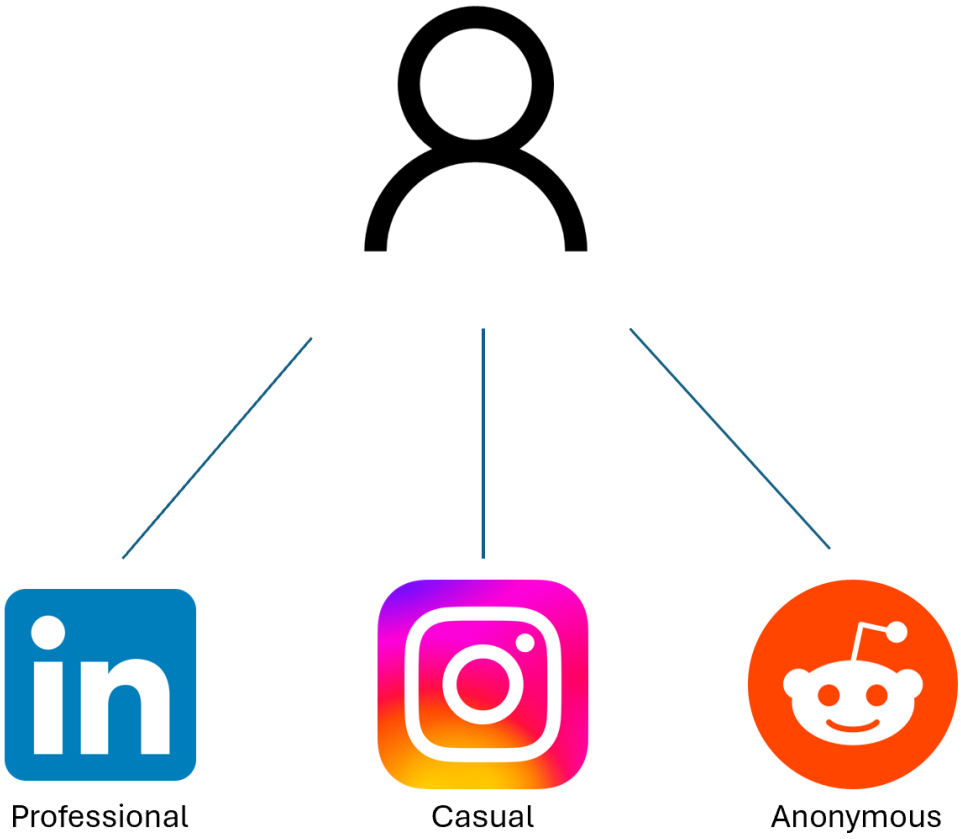
Created by: Bryan Loo Chun Wai

Final Project Report

Table of Contents

Chapter 1 – Introduction (976/1000)

1.1 Project Concept



IdentiCore is a secure, user-friendly platform that helps people manage the different names and profiles they use online. Today, it is common for someone to have one name for LinkedIn, another for Instagram, and maybe even a completely anonymous handle for Reddit or a blog. It can become confusing trying o keep track of them all. In a worse case, ensuring they stay separate when needed.

That is where IdentiCore comes in. It gives users a way to securely store, tag, and manage all their different aliases in one place. Each alias can be tagged by context, such as professional or casual use. Aliases also feature visibility toggling, allowing them to be marked public or private depending on who is requesting access. The system is built around user control, privacy, and flexibility.

1.2 Motivation

Most of us present different versions of ourselves online depending on the situation. A freelancer might use his real name on a portfolio site, a brand name on social media, a gamer tag on Discord. Students might use formal names in school related spaces but nicknames elsewhere. Additionally content creators often have public personas that are totally separate from their personal lives.

The problem is, there is no easy way to manage all these identities. One may might accidentally expose a private alias. Someone might figure out ones anonymous blog through how they write. Or maybe one may just look inconsistent across platforms, which makes him seem less professional.

Existing identity systems are either too corporate or are too basic. For example, think enterprise level single sign on system. What is missing is a lightweight, personal tool that gives the user full control over how their identities are managed and shown.

1.3 Template used

This project follows template 7.1, Identity and Profile Management API from the CM3035 Advanced Web Design Module. The original idea behind the template is to create a secure API that lets users manage names and identities in different contexts while protecting sensitive data.

JSON Web Token (JWT) is a compact, token format used for securely transmitting information between parties as a JSON object. It is digitally signed to verify authenticity and integrity, and often used for authentication and authorization in web applications.

Advanced Encryption Standard (AES) is a symmetric encryption algorithm widely used to secure digital data. It encrypts and decrypts information using the same secret key, operates on fixed bloxk sizes in 128 buts, and supports key lengths of 128, 192, or 256 bits, making it fast, reliable, and highly secure.

IdentiCore takes the idea further by adding three core enhancements:

1) Tagging by platform and context

Anonymous tagging on Reddit or formal tagging on LinkedIn, enabling flexible organization across professional, casual, or anonymous categories.

2) Encryption for sensitive data

Security is central when dealing with user credentials and personal data. IdentiCore applies token based authentication for logins, AES encryption for storing sensitive information such as addresses, and Bcrypt hashing for passwords, ensuring data remains unreadable if the database is compromised

3) Alias visibility controls

Users can set aliases as public or private. Public aliases are retrievable through the API or interface, while private ones are shown only to the owner, preventing accidental exposure and strengthening control over personal data.

1.4 Objectives

This project has 6 main goals, mainly focusing on ensuring the visibility of user accounts and data security.

RESTful API also known as a Representational State Transfer Application programming interface that follows the principles of Representational State Transfer (REST), using standard HTTP methods (GET, POST, PUT, DELETE) to enable communication between clients and servers in a stateless, resource-oriented way.

CRUD stands for Create, Read, Update and Delete operations that are the four fundamental functions used to manage data in a database or storage system.

Firstly, to create a secure RESTful API where users can register, log in, and mange their data. By using this method we are able to efficiently carry out operations such as registration and login, as well as to test each API endpoint to ensure the features work.

Secondly, to build CRUD operations. Create, read, update and delete for aliases. Management of user aliases is essential in IdentiCore for users to manipulate user records such as to change contexts and visibility preferences.

Thirdly, allow users to tag aliases by contexts such as work, casual and anonymous. Platforms such as Instagram, GitHub, TikTok can be added by the user to help identify their alias. Making it visually straightforward for the user to identify what each account is used for.

Fourthly, to add visibility toggles so users can decide which aliases are public or private. On top of allowing users to have the access to change contexts, now users are also able to change how their data are being displayed to others. In other words if the alias is meant to be public others will be able to see it. And if the alias is private the data will not be visible to others.

Lastly, to encrypt personal data like usernames or real names to keep things private. IdentiCore uses AES encryption for storing addresses, and bcrypt hashing for passwords. If database is revealed to hackers, data such as real addresses stay unreadable without the key.

Postman collection will be used to demonstrate the CRUD alias management and user account management. The website will also be using a RESTful API to use routes to send and retrieve data.

1.5 Impacts of this approach

This tool can provide strong benefits for people juggling multiple digital identities. The main target groups include:

Freelancers – separating work for different clients across platforms.

Influencers and creators – keeping public personas separate from private lives, protecting against exposure and harassment.

Students – managing different names for academic work, side projects, and online communities.

Privacy-conscious users – Individuals who prefer not to have everything tied to one name.

Chapter 2 - Literature review (1716/2500)

2.1 Introduction to digital identity management

In today's digital landscape, people no longer represent themselves with a single unified identity. Instead, they present different facets depending on the platform and audience. Research from Marwick and Boyd 2011 introduced the concept of, context collapse.

Creators face difficulties presenting a consistent self-image when faced with multiple audiences converge. This creates the need for tools to manage identities based on context. This means categorizing user aliases based on what they are being used for. For example, in IdentiCore three contexts are used, professional, casual and anonymous.

Digital identity management is no longer limited to corporate or government systems. It has evolved into a personal challenge for freelancers, content creators, and privacy conscious users, Studies such as Marwick and Boyd (2011) emphasize on the concept of context collapse, where users must present different version of themselves to divers audiences across platforms. This is directly relevant to IdentiCore, which seeks to give individuals control over how they present themselves in different contexts while maintaining privacy and separation between identities.

2.2 Fragmented Identity and alias use

Alias usage is common in online forums, gaming platforms, and even professional networks. According to Goffman's theory of self presentation 1959, people act differently depending on the stage they are on.

The digital space intensifies this behavior. A Reddit user, for instance, may have a throwaway account for sensitive topics, a pseudonym for community interaction, and a professional handle on LinkedIn. These identities must be managed carefully.

These different platforms serve various purposes and intentions. For instance, LinkedIn is used for posting job achievements as well as to seek for career opportunities.

Goffman's self presentation theory (1959) explains how people adapt their front and backstage personas. In the digital space, these personas can span multiple platforms. For example, LinkedIn for professional identity, Instagram for personal branding, and Reddit for anonymous interaction. However, existing tools fail to provide seamless management of these fragmented identities. IdentiCore aims to bridge this gap by offering context tagging and visible controls, inspired Goffman's theory. This is to help users keep their professional and personal lives distinct.

2.3 Risks of Stylometric Leakage

Stylometric refers to identifying users based on writing style. Several studies (Narayann et al. 2012 and Afroz et al 2013) have demonstrated that users attempting to remain anonymous can be deanonymized by analyzing their language patterns.

While these studies focused on deanonymization, they also show a need for tools that help people maintain clear separations between aliases to reduce linkage risks.

Research by Narayan et al.(2012) and Afroz et al (2013) demonstrates that even when users adopt multiple aliases, writing style can inadvertently reveal their identities through stylometric analysis. While these studies focus on deanonymization, they underline a key risk. The risk that users face is the exposure of identity to maintain clear separations between aliases to reduce linkage risks. IdeniCore addresses this challenge by offering AES encrypted storage of sensitive data and strict separation of alias contexts.

2.4 Alias Detection Tools and Gaps

Research such as "Detecting Multiple Aliases in Social Media" (Solorio et al 2013) focused on finding users operating multiple accounts.

However, little attention has been paid to help users manage aliases ethically and securely. Most work centers on detection, and not prevention or control.

Most existing research studies focus on detecting multiple aliases from law enforcement or marketing, rather than empowering individuals to manage their own alias ethically and securely. This represents a critical gap that IdentiCore attempts to fill by offering a user-centric solution, not for surveillance, but for personal identity management with privacy focused design.

2.5 Web Security Practices in Identity Systems

Good API design for identity management follows certain principles, such as strong authentication. JWT and OAuth are some examples of strong authentication methods. Another principle is to encrypt data at rest and granular access control, such as OWASP in 2023.

These principles are crucial for storing sensitive information like email addresses, username, and bios. A relevant point here is that unlike enterprise IAM systems, personal identity APIs like IdentiCore offers simplicity and transparency.

2.6 Related Project and Tools

The RUN Proxy Project (Adeyemi et al. 2016) was developed to enable anonymous browsing within a university context. It used role based access controls and allowed internal usage monitoring.

Similarly, Saudi Arabia's centralized biometric e-ID (AJBM 2010) emphasized unified identity infrastructure. However, these were state level or institutional projects and not designed for individual users.

While projects like the RUN Proxy Project offers anonymized browsing and institutional identity management, they lack user level alias customization. Likewise, Saudi Arabia's biometric e-ID system is centralized and not applicable for personal mutli-identity needs. IdentiCore differentiates itself by being lightweight, modular, and built with a focus on individual users rather than organizations, thus offering flexibility and visibility control which are absent in these larger systems.

2.7 State management in modern web systems

A 2023 NTNU Master's thesis explored how client and server side state management can affect code clarity and maintainability in modern web applications. Though not directly focused on identity, the findings are relevant for designing a maintainable and scalable backend architecture.

2.8 Theoretical insights shaping Identicore's features

The challenge of managing multiple digital identities has long been recognized in the literature, particularly in relation to the phenomenon of context collapse. Young (2013), in Managing Online Identity In The Age Of Context Collapse: Audience Segregation on Facebook and Beyond, examines how overlapping audiences on social networks force individuals into self censorship or maintaining multiple profiles. Young's findings highlighted how users depend on privacy settings and selective posting to manage identity tensions, which inspired my design decision to implement visibility toggles (public/private) and context tags (professional, casual, anonymous). Although Young's study is focused primarily on Facebook, its insights were generalized to cross platform identity management.

In addition, Laing (2017) in Layered Online Identities in Digital Branding explores how writers and professionals curate layered identities to communicate effectively with different audiences. Professional, peer and personal identities are some examples that can be used for this case. The study highlights that distinguishing between layers of identity improves clarity and audience targeting, thereby strengthening digital branding. This concept of layering inspires the IdentiCore dashboard's grouping of aliases by context and the differentiation of platform tags such as GitHub, LinkedIn, Instagram. While Laing's work centers on writers, its relevance is expanded to suit freelancers, students, and content creators who similarly need to balance different digital personas.

However, online anonymity is not always as secure as it may appear. Johansson et al. (2015), in Deanonymization Risks in Multi-Alias Environments: Linking Identities through Timeprints, demonstrates how metadata such as posting times and behavioral patterns can be used to deanonymize users, even when aliases are not directly connected. Their findings underscore the fragility of online pseudonymity and reinforce the need for strong privacy measures. In IdentiCore, this research justified the implementation of Advanced Encryption Standard (AES) encryption for sensitive fields and a data minimization strategy, whereby only essential information is stored to reduce the risk of linkage attacks.

The theoretical foundation of identity management models is provided by Allen et al. (2020), in Models of Digital Identity management: Centralised, Federated, and self-Soverign Approaches. This work compares three dominant frameworks and stresses that robust identity management systems must integrate cryptography, user consent, and transparent data control. IdentiCore translated these high level recommendations into concrete design choices, including AES encryption, JSON Web Token (JWT) authentication, and restricted Cross-Origin Resource Sharing (CORS) policies, ensuring that only trusted frontends could interact with the backend. Although Allen et al.'s study is abstract and model driven, IdentiCore operationalized its recommendations within a practical application.

Another strand of relevant literature addresses the role of identity cues in sustaining digital engagement. Chou and Lu (2021), in their BIT Conference study on Twitch: Digital Participation and Content Creation Intention, demonstrates that clear identity signals and reminders encourage users to remain active in content creation and digital participation. The study's findings that nudges, feedback, and analytics encourage sustained engagement shaped the design of IdentiCore's alias statistics dashboard, activity counts, and optional reminders via alerts or banners. Although the original research was situated in the Twitch context, the same behavioral principle were adapted to identity management, ensuring that users are gently encouraged to keep their alias records up to date.

Finally, a Doctoral Research Report (2025), in Bridging Digital and Online Identity: Integrating Security Protocols with Persona Management Tools provides a timely synthesis of identity research. It distinguishes between digital identity, which relates to credentials and authentication, and online identity, which refers to the multiple personas individuals present in different contexts. The report

argues that modern identity systems must integrate both domains to be effective. IdentiCore embodies this recommendation by balancing backend security mechanisms (AES encryption, JWT authentication, Bcrypt hashing) with front-end tools for organizing online personas (dashboard grouping, alias filters, practical user tips). This dual focus highlights how security and usability can work together to deliver a privacy first and accessible identity management solution.

In summary, these six works illustrate the breadth of academic perspectives on identity management. The sociological implications of context collapse (Young, 2013), together with the technical frameworks of digital identity models (Allen et al., 2020), shaped the conceptual foundation of IdentiCore and guided the development of its practical features, ensuring the system is both theoretically grounded and functionally robust.

2.9 Summary of gaps

Across the literature, three key gaps emerged. Firstly, there is a lack of tools designed for individual identity control, as most existing solutions focus on enterprise-level management. Secondly, much of the research concentrates on detecting aliases rather than enabling user control or customization. Finally, there are very few projects that integrate usability, tagging, and visibility control within a single platform.

IdentiCore aims to fill these gaps by offering a lightweight, personal, and context-aware alias manager that supports real world identity patterns across platforms.

Based on the reviewed literature, it is evident that most identity management systems prioritize organizational needs, leaving individual users underserved. There is also a lack of tools that combine encryption, tagging, and visibility control for multi-identity management. IdentiCore directly addresses these gaps by integrating state of the art security practices like JWT authentication and AES encryption, with user friendly alias categorization, making it both secure and practical for personal use.

Chapter 3 – Design (2531/2000)

3.1 Project Structure Overview

The design of IdentiCore follows the principles of a three-tier web architecture. Which separates the system into distinct but interconnected layers. The presentation tier provides the interface through which users interact with the system, whether through a front-end client or testing tools such as Postman. The application tier contains the business logic, handling processes such as authentication, API routing, and request validation. Finally, the data tier manages the secure storage and retrieval of information, including encrypted alias records and user details. This layered approach ensures modularity, scalability, and a clear separation of concerns, forming the structural foundation for IdentiCore's implementation.

IdentiCore is designed as a RESTful web service with a secure backend that manages user authentication and alias storage. The application consists of four main layers:

Firstly, authentication layer. It handles user registration, login, and session token management using JWT.

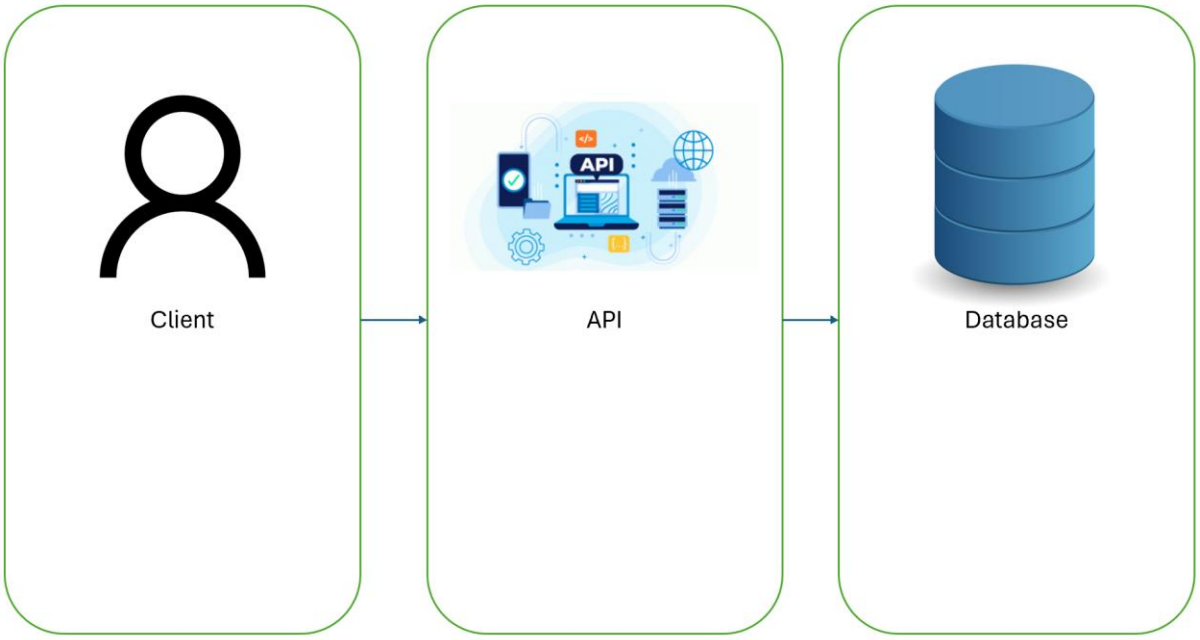
Secondly, data management layer. Manages alias creation, updating, filtering, and deletion in the MongoDB database.

Thirdly, API routing layer. It routes HTTP requests to appropriate controller functions using Express.js.

Lastly, the testing interface layer provides a way to simulate front-end interactions through Postman, allowing developers to verify that authentication, data handling, and API routes perform as intended.

All communications are secured via HTTP, and access to protected endpoints is authorized using bearer tokens.

3.2 System Architecture



The system architecture follows a common 3 layer model which consists of firstly the Client (Postman or Web user interface), then Express.js and REST API, and lastly MongoDB Database.

There are 4 main components to this architecture.

Firstly, authentication module. It handles registration and login with hashed passwords and JWT generation

Secondly, Alias manager module. It perform CRUD operations on alias entries, tagging them with context and visibility.

Thirdly, Tag filter module. It enables retrieval of aliases based on context (for example work or casual).

Lastly, encryption module. Encrypts email addresses and other sensitive fields before storage.

Each component communicates through RESTful routes and uses middleware to validate tokens, sanitize inputs, as well as to enforce visibility rules.

3.3 Features Implemented

1) User Registration or Login
   - Passwords hashed with bcrypt and JWT issued for session management
   - JWT tokens are generated upon successful login and returned to the client.
   - Tokens are validated on every protected request.

2) Alias management
   - Aliases include fields, platform, username, bio, context such as work or casual or anonymous, visibility and encrypted email that are AES encrypted.
   - Users can create, read, update and delete alias entries.

3) Context tagging and filtering
   - Enables dynamic queries such as “show all professional aliases”.
   - Users can retrieve aliases based on context, allowing clear segmentation of digital identities.

4) Visibility controls
   - Each alias is marked public or private. Private aliases are only returned to authenticated users.
   - Public aliases are accessible through general API calls without authentication.

5) Secure Data Storage
   - Sensitive fields encrypted using AES level or field encryption
   - All passwords are stored using bcrypt hashing.

- Email addresses are encrypted using AES before database insertion.
- JWT ensures stateless session management and secure access.

3.4 Design Decisions

Tool	Justification
HTML	Strong structural foundation + accessible from any modern web browser + manage aliases using a visual interface
CSS	Styling + user-friendly + visually consistent
MongoDB	Flexible document schema + dynamic alias metadata
ExpressJS	Minimalistic and fast
JWT	Stateless and secure
Bcrypt	Ensures passwords are irreversibly hashed before storage
AES encryption	Protects sensitive fields like addresses even if database access is compromised
Tag based filtering	Context aware identity management
Postman	Testing + reduce development load

Detailed explanation

The technology choices and architectural design were made with a focus on simplicity, security and future extensibility in mind.

HTML, also known as Hypertext Markup Language was chosen for this project to form a structural foundation for the frontend. A static HTML interface that allows users to register, log in, and manage aliases through a visual interface and can also be used for Postman.

CSS, also known as Cascading stylesheet is chosen for customized styling and layout of the frontend. A custom CSS stylesheet will be used to enhance the user experience with a responsive design, clean layout, and intuitive visual feedback.

MongoDB is chosen for its flexible document schema, ideal for dynamic alias metadata. In other words, its flexible schema supports dynamic alias fields and quick iteration during development.

ExpressJS provides minimalistic and fast route handling with middleware for security. It has a lightweight framework that is ideal for API development and also suitable for middleware integration.

JWT authentication ensures that each API call is stateless and secure. In other words, it enables a secure, stateless authentication, reducing session overhead.

Bcrypt is a password hashing function that is based on the Blowfish cipher and incorporates a salt to protect against rainbow table attacks, and also over time the iteration count can be increased to make it slower hence making it resistant to brute force search attacks even with an increase in computational power.

AES encryption, also known as advanced encryption standard is a widely used symmetric encryption algorithm that protects data by transforming it into an unreadable format using secret key that is used to decrypt the data back to its original form.

Tag based filtering aligns directly with user needs for context aware identity management

IdentiCore is designed based on clear separation of concerns. The backend focuses on secure data management and API logic, while the frontend provides a clean interface for users to interact with the system.

3.5 Technology Stack

Component	Technology	Purpose
Frontend	HTML + CSS	Interaction with API
Backend framework	NodeJs with Express	API routing and middleware
Database	MongoDB + Mongoose	Data storage with schema modeling
Authentication	Bcrypt + JWT	User login security and session handling
Encryption	AES-256	Secure sensitive fields
Testing	Postman	Manual endpoint testing and validation
Package management	NPM (Node package manager)	Managing backend dependencies

Explanations on some technologies not previously mentioned in the design:

AES-256, a Node.js crypto module is a widely recognized symmetric encryption algorithm used to secure sensitive data at rest. This project will use these technologies by encrypting fields like addresses to protect user information in case of data breaches. Ensuring confidentiality even if attackers gain access to stored data.

NPM, also known as Node Package Manager is used to manage project dependencies in the Node.js environment, it allows for easy installation and maintenance of libraries such as Express, Mongoose, Bcrypt, and JSON Web Token, ensuring consistent versioning and reducing setup complexity during development.

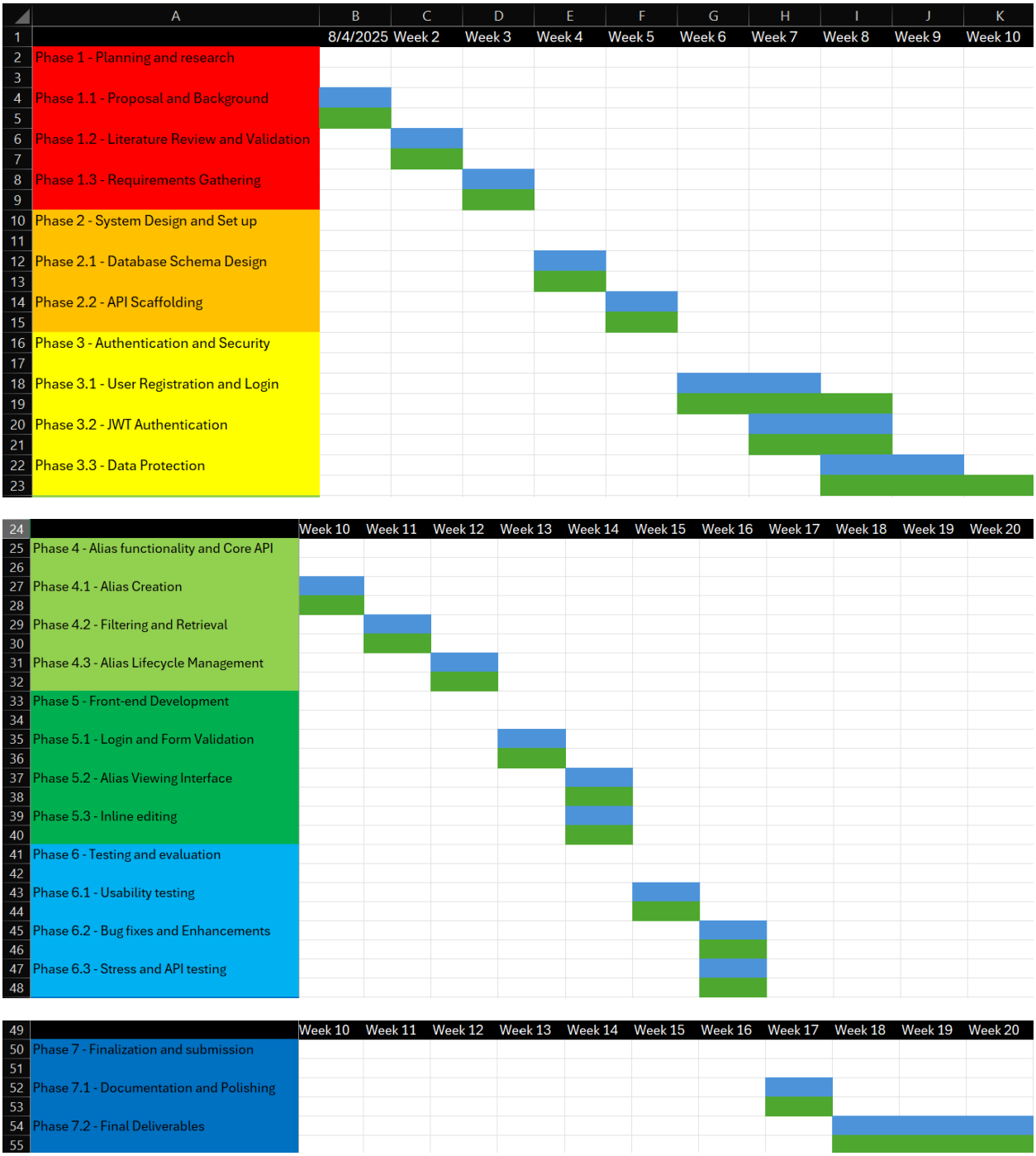
Hence, by integrating AES-256 encryption with the Node.js crypto module and managing dependencies through NPM, the project achieves both strong data security and streamlined development.

3.6 Work plan and milestones

Phase breakdown

Phase 1	Planning and research
1.1 Proposal and Background	Initial proposal drafting, approval, and background research
1.2 Literature Review and Validation	Reviewing related work, validating feasibility, and refining the project idea
1.3 Requirements Gathering	Identifying system requirements, outlining features, and setting clear objectives
Phase 2	System Design and setup
2.1 Database Schema Design	Designing entity relationships and schema for alias and user data
2.2 API Scaffolding	Establishing project structure, initial API routes, and preparing middleware setup
Phase 3	Authentication and security
3.1 User registration and Login	Implementing registration and login endpoints, including Bcrypt hashing
3.2 JWT Authentication	Adding token generation, parsing and validation for secure sessions
3.3 Data Protection	Integrating encryption, security middleware, and enforcing access controls
Phase 4	Alias functionality and core API
4.1 Alias creation	Inserting new aliases with platform, context, and visibility options
4.2 Filtering and Retrieval	Enabling search by context and applying visibility flags
4.3 Alias Lifecycle Management	Supporting updates, deletions, and database consistency
Phase 5	Front-end development
5.1 Login and Form validation	Building browser based user interface for login and alias creation with client side validation
5.2 Alias Viewing Interface	Designing a tabular user interface with real time filtering of aliases
5.3 Inline editing	Adding update and delete controls directly within the frontend display
Phase 6	Testing and evaluation
6.1 Usability testing	Engaging peers for testing workflows, bug discovery, and feedback
6.2 Bug fixes and Enhancements	Addressing token expiry issues, filed sanitization
6.3 Stress and API testing	Conducting manual tests to evaluate performance under load
Phase 7	Finalization and submission
7.1 Documentation and Polishing	Preparing final documentation, conducting accessibility checks, and refining presentation
7.2 Final deliverables	Recording video demo, packaging source code, and submitting the completed project

Gantt chart



Explanation:

The Gantt chart shows the project progressing through seven phases, from planning and research to finalization and submission, with blue representing the planned schedule and green the actual progress. Overall, tasks generally followed the intended timeline, with some activities such as

requirements gathering, authentication, and data protection extending slightly beyond their planned weeks. System design, alias functionality, and front-end development were mostly completed on schedule, while testing and evaluation showed minor overlaps. Final documentation and deliverables extended into week 20, but overall the actual progress remained closely aligned with the plan, demonstrating steady execution with only small delays

3.7 Contingency plans

To ensure that the project remains functional and deliverable under various constraints, the following contingency plans have been identified.

Case 1: Encryption implementation fails

Risk description: AES encryption may fail to integrate or encrypt or decrypt properly

Follow-up actions: Hashing or data masking methods for sensitive fields like address. This code will be directly integrated into the project to ensure that the data are not displayed as plaintext that can be easily be used by hackers or any individual that might gain access.

Case 2: Time constraints

Risk description: Insufficient time to develop a complete frontend before the deadline

Follow-up actions: Defer frontend development, conduct full testing via Postman and document UI as future extensibility. This ensures that the backend API endpoints work, and the development of the project proceeds as planned with the objectives of the project met.

Case 3: Frontend Bus or Incompletion

Risk description: The HTML or CSS frontend might be unstable or incomplete

Follow-up actions: Fall back to using Postman collection as the demonstration interface. As well as to isolate UI bugs for after submission fixes. This ensures that the app is usable and can be presented directly in the Postman collection.

3.8 Evaluation criteria

Criteria 1: Secure user registration and login

Purpose: Ensure only valid users can access the system

Explanation: Test with valid and invalid inputs. This verifies the JWT token is issued and passwords are hashed.

Criteria 2: Alias entry management

Purpose: Confirm users can create, view, update, and delete aliases

Explanation: Perform complete CRUD operations using Postman. This verifies database updates and proper API responses.

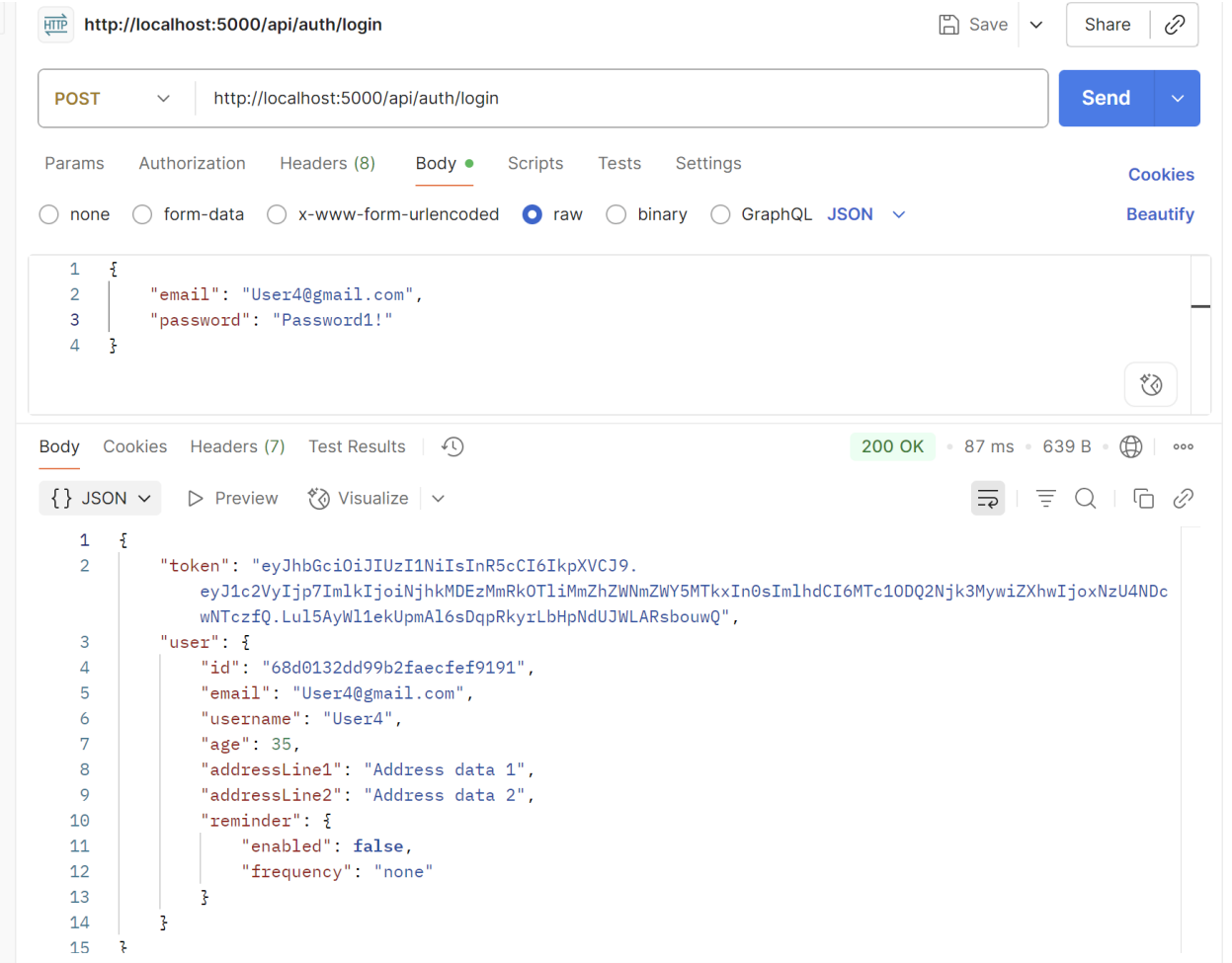
Criteria 3: Context-based filtering

Purpose: Enable users to filter aliases by context

Explanation: Use “GET /context/:tag” and confirm correct subset of aliases is returned. When using the GET method it returns the data from the target tag from the URL address. This ensures that the function works properly and can then be developed into the app that is used by the user to retrieve these data in the app and not just the Postman collection.

3.9 Postman collection

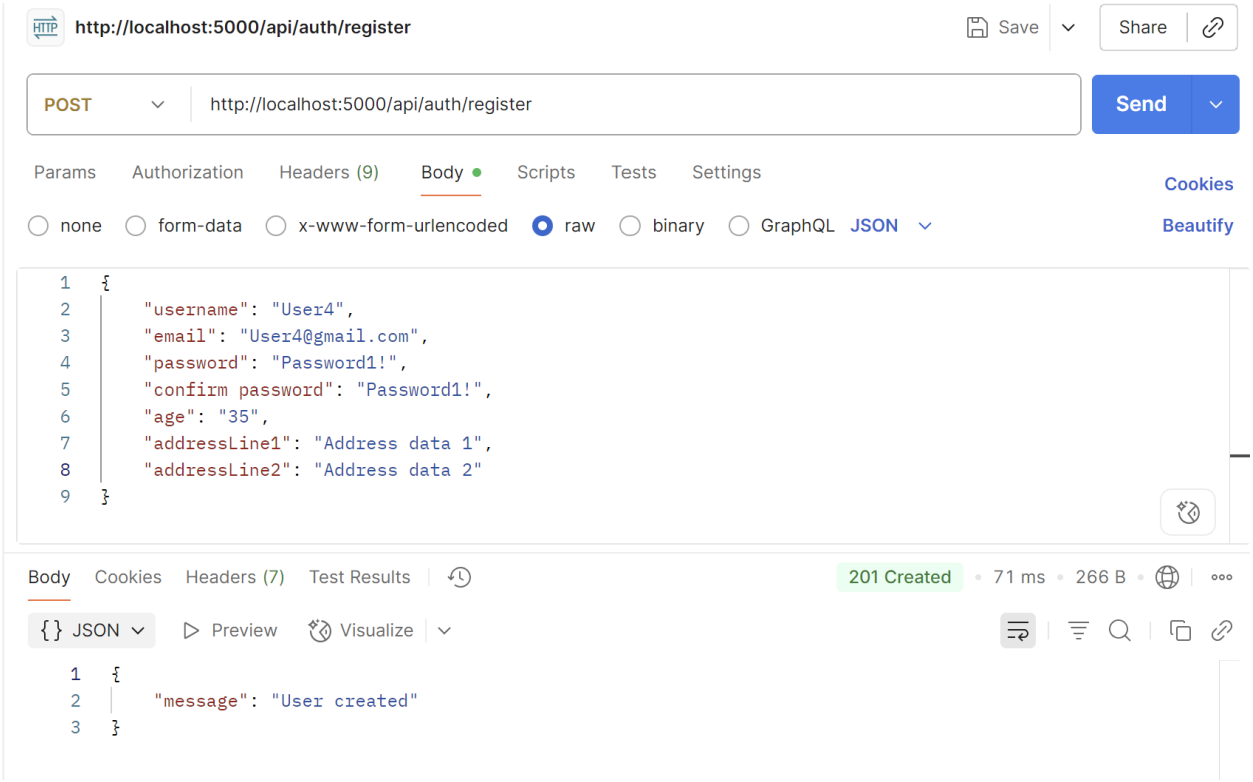
1) Login API endpoint test



Purpose: Authenticate existing user and return JWT token

Expected behavior: Requires correct credentials, returns a token used in subsequent protected requests. This can be seen in the screenshot, where the email and password data are passed in and the token is returned in the body in the Postman collection.

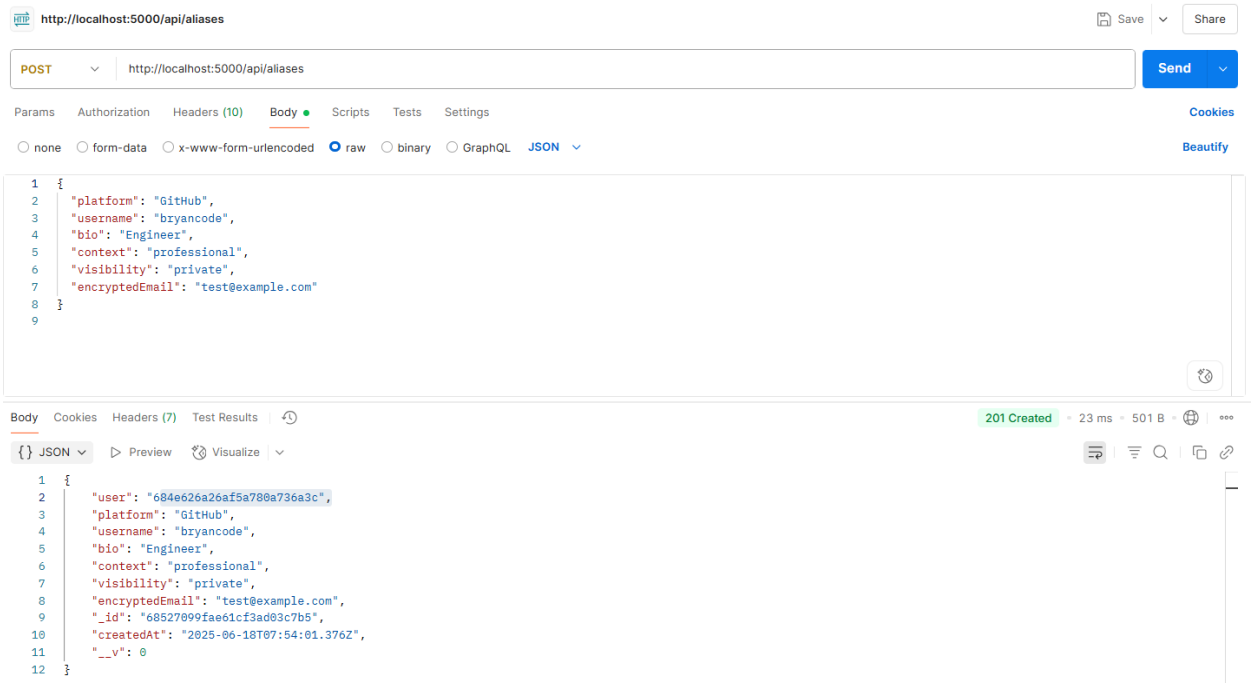
2) Registration API endpoint test



Purpose: Register a new user

Expected behavior: Accepts email and password inserted into input fields. Returning success message and stores user with hashed password. Referring to the screenshot above, it can be seen that the email and password data are passed into the fields and a success message returned in the body. This shows that the user has been successfully created.

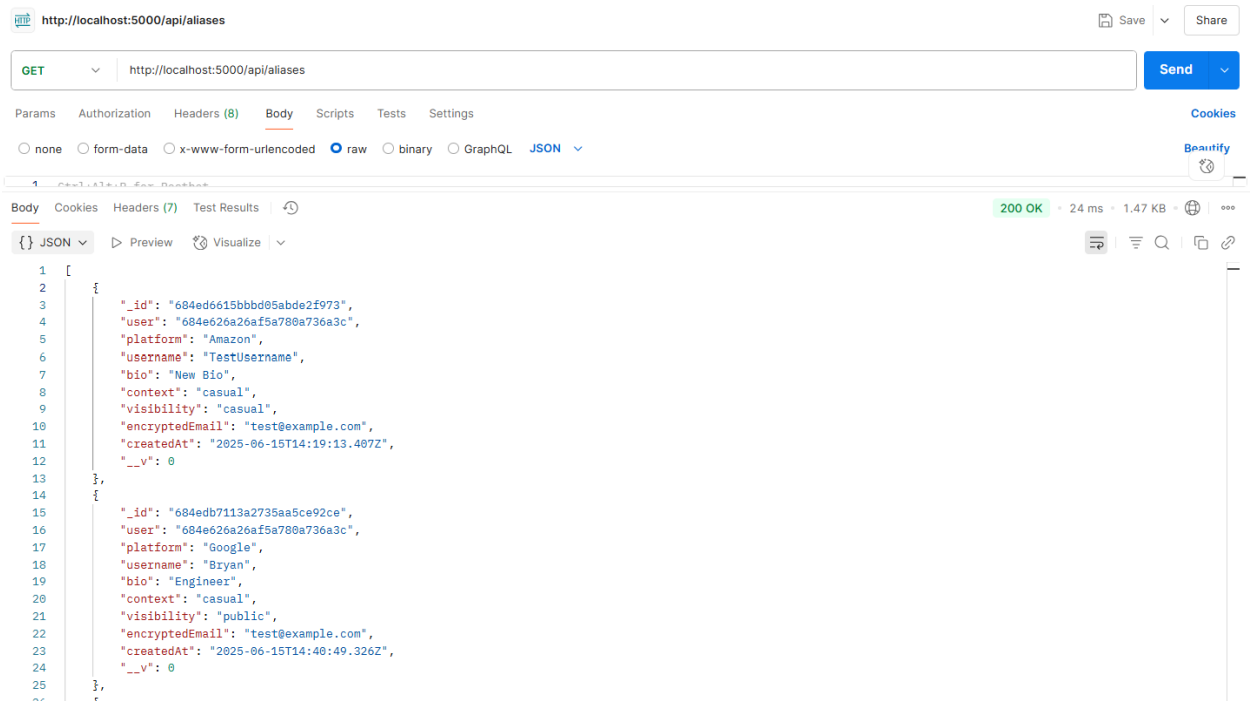
3) Create new alias API endpoint test



Purpose: Create a new alias entry

Expected behavior: Requires JWT token in headers. Alias data includes platform, context, bio visibility, and email. With reference to the screenshot above, it can be seen that the platform, username, bio, context, visibility as well as encrypted email as the fields for creating a new alias. And are displayed below it can be seen that the new alias being created with the user token as the header.

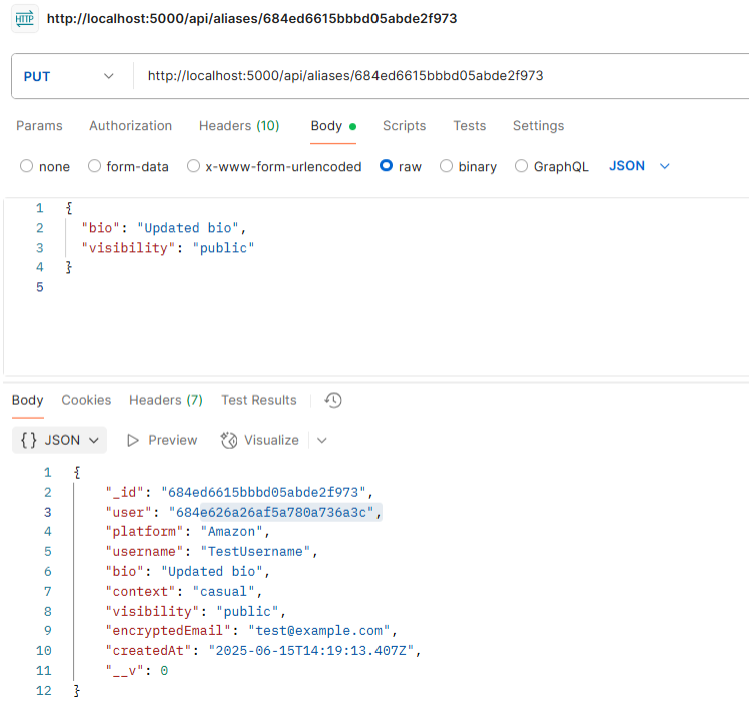
4) Retrieve all aliases for current user API endpoint test



Purpose: Retrieve all aliases for the authenticated user

Expected behavior: Returns a list of aliases owned by the user. Private aliases only show if the JWT token used is valid. With reference to the screenshot above, the aliases are retrieved and displayed in the body of the Postman collection. The system stores key fields, including user, platform, username, bio, context, visibility, and an encrypted email.

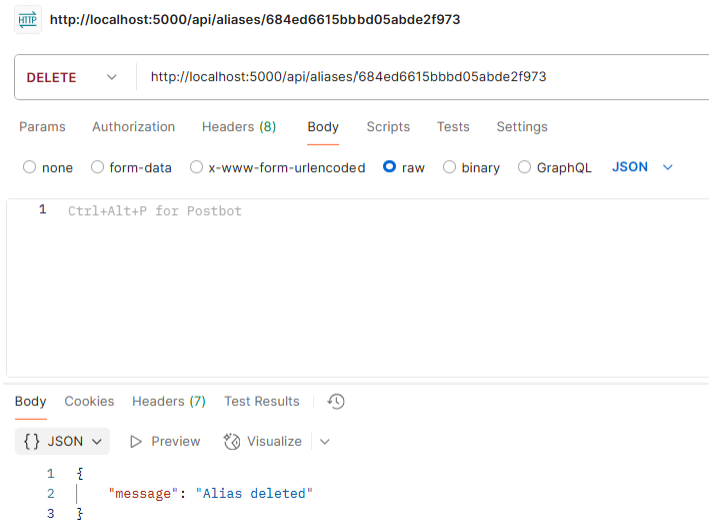
5) Update alias API endpoint testing



Purpose: Update a specific alias by ID

Expected behavior: Modifies fields such as context, platform, or visibility. To successfully update these fields, the JWT token is required. With reference to the screenshot, an attempt was made to update the bio and visibility of the alias. After successful update of the alias, the details of the updated alias is returned in the body of the Postman collection as presented.

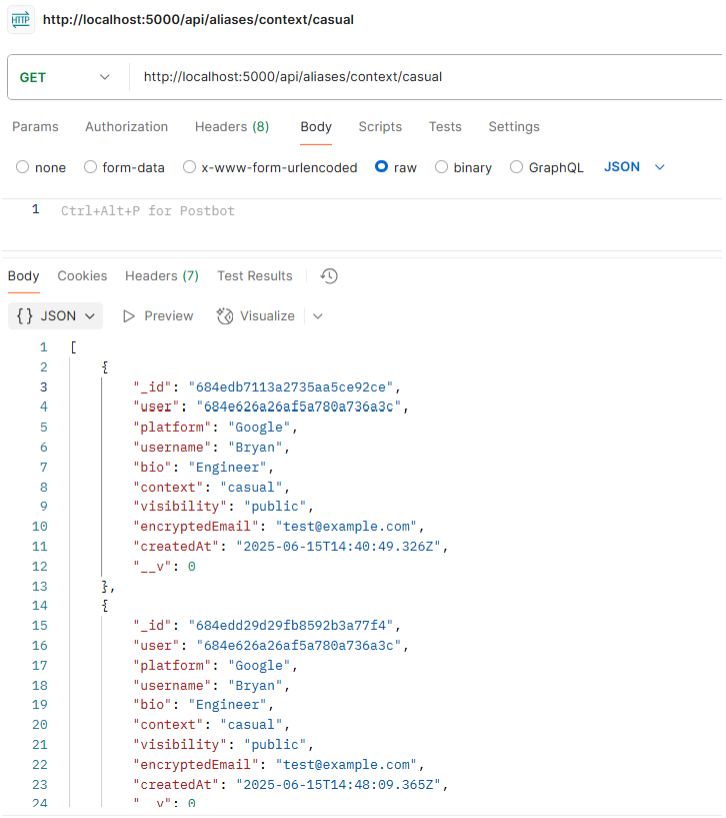
6) Delete existing alias record API endpoint test



Purpose: Delete an alias by ID

Expected behavior: Deletes a specific alias entry. While using the JWT token, then the alias can be deleted. This ensures that only the owner can delete their own data. With reference to the screenshot an attempt was made to delete an alias and the user token provided in the URL. After successfully deleting the alias, the success message is returned in the body to signal that the operation is completed.

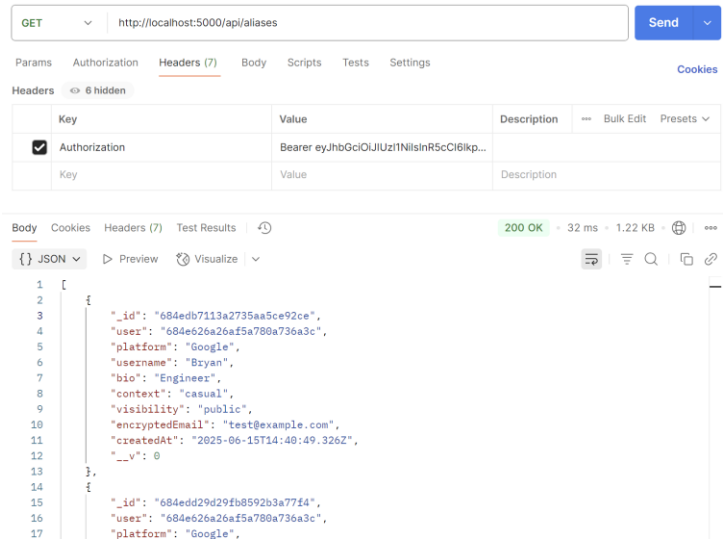
7) Retrieve alias by context API endpoint test



Purpose: Filter aliases by context

Expected behavior: It returns only aliases matching the specified context. With reference to the screenshot above, the casual context was inserted in the GET method URL. After user authentication, the returned aliases are reflected with the casual context tagging. According to the screenshot it can be seen that the list of aliases returned in the body in JSON format.

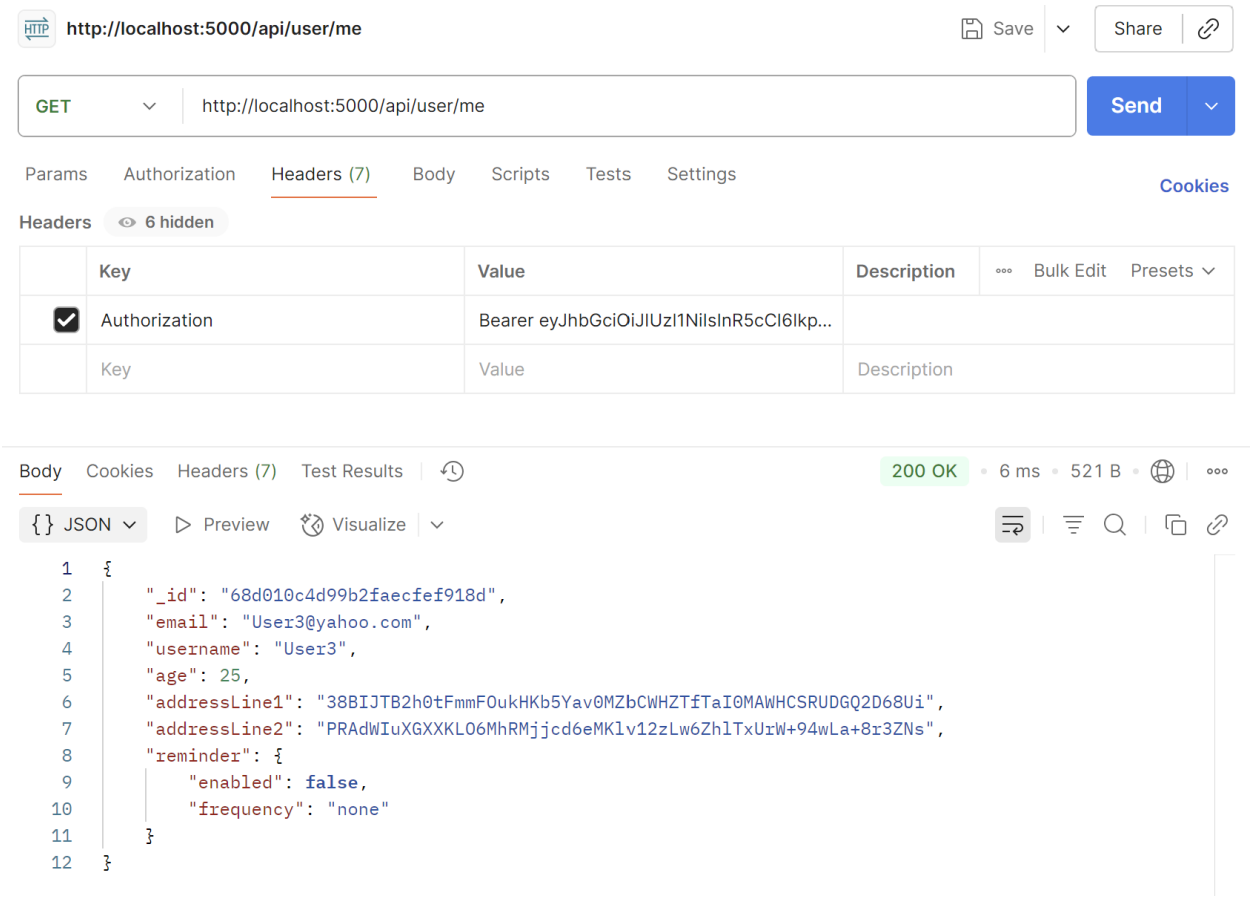
8) Using JWT token in Postman



Purpose: To authenticate and use JWT token for user specific tasks

Expected behavior: To ensure that the token is valid to carry out tasks such as retrieving aliases and performing CRUD operations that only the owner of the alias has the ability to do. With reference to the screenshot the JWT token was inserted as the authorization header, and used the token to retrieve the aliases using the token.

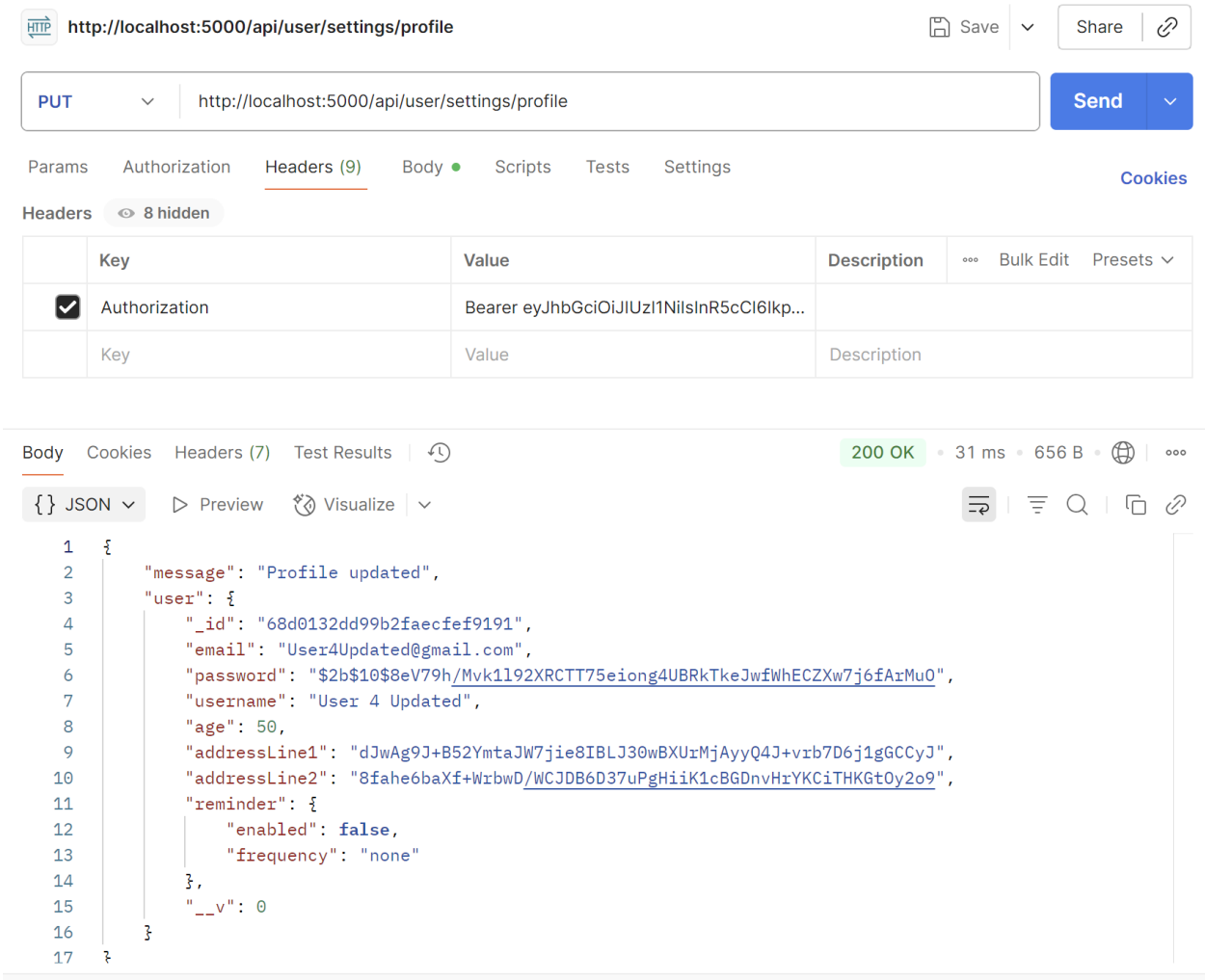
9) Retrieve User Credentials



Purpose: To view all user credentials

Expected behavior: returns JSON body with the user's credentials but only of the JWT bearer token is present in the header using authentication.

10) Update User Profile



Before changes:

```
_id: ObjectId('68d0132dd99b2faecfef9191')
email : "User4@gmail.com"
password : "$2b$10$8eV79h/Mvk1l92XRCTT75eiong4UBRkTkeJwfWhECZXw7j6fArMuO"
username : "User4"
age : 35
addressLine1 : "dJwAg9J+B52YmtaJW7jie8IBLJ30wBXUrMjAyyQ4J+vrb7D6j1gGCCyJ"
addressLine2 : "8fahe6baXf+WrbwD/WCJDB6D37uPgHiiK1cBGDnvHrYKCiTHKGtOy2o9"
▸ reminder : Object
__v : 0
```

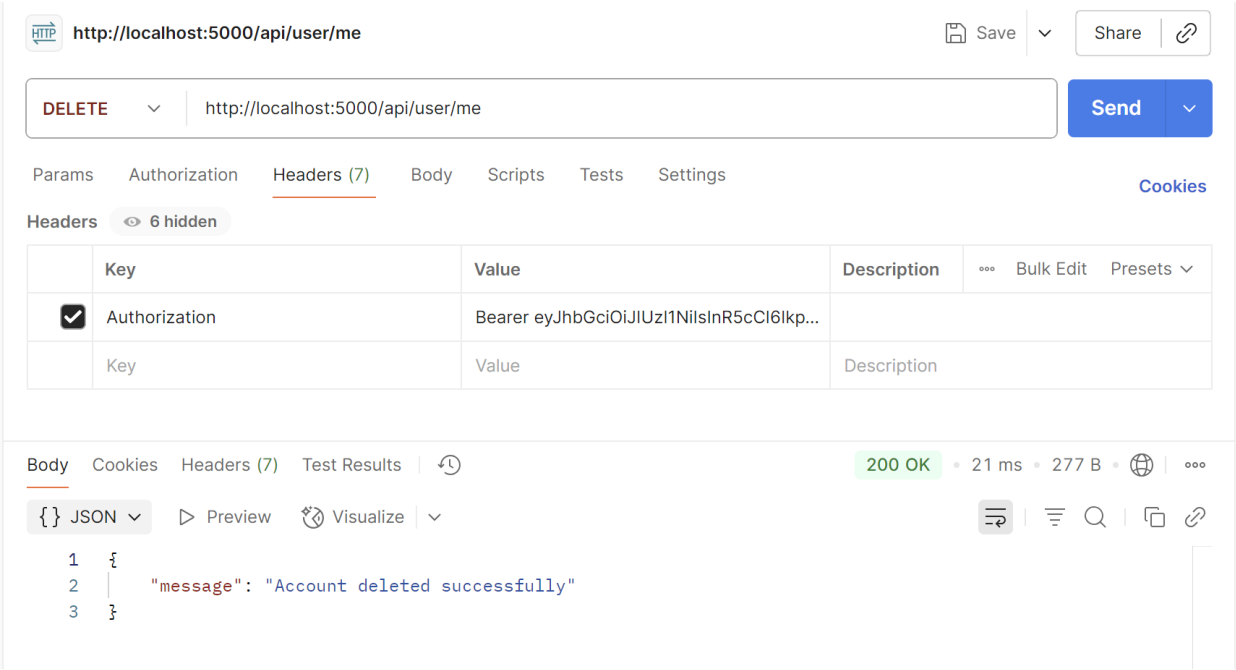
After updating:


```
_id: ObjectId('68d0132dd99b2faecfef9191')
email : "User4Updated@gmail.com"
password : "$2b$10$8eV79h/Mvk1l92XRCTT75eiong4UBRkTkeJwfWhECZXw7j6fArMuO"
username : "User 4 Updated"
age : 50
addressLine1 : "dJwAg9J+B52YmtaJW7jie8IBLJ30wBXUrMjAyyQ4J+vrb7D6j1gGCCyJ"
addressLine2 : "8fahe6baXf+WrbwD/WCJDB6D37uPgHiiK1cBGDnvHrYKCiTHKGtOy2o9"
▸ reminder : Object
__v : 0
```


Purpose: To update user credentials

Expected behavior: Updates user credentials for username, email and age. The before and after process can be seen in the screenshots

11) Delete User Account



Purpose: To authenticate and use JWT token for user specific tasks

Expected behavior: To allow full control of account management, by allow account delete option. This promotes trust with the user by giving them full control

Chapter 4 – Implementation (2029/ 2500)

Objective 1: Create a RESTful API

```
// user.js
// declare constants and requirements
const express = require('express');
const auth = require('../middleware/authMiddleware');
const User = require('../models/User');

const router = express.Router();

// get current user's profile
router.get('/me', auth, async (req, res) => {
 try {
  const user = await User.findById(req.user.id).lean();
  if (!user) return res.status(404).json({ error: 'User not found' });

  res.json({
   _id: user._id,
   email: user.email,
   username: user.username,
   age: user.age,
   addressLine1: user.addressLine1,
   addressLine2: user.addressLine2,
   createdAt: user.createdAt,
   reminder: user.reminder
  });
 } catch (e) {
  res.status(500).json({ error: e.message });
 }
});

// update reminder settings
router.put('/settings/reminder', auth, async (req, res) => {
 try {
  const { enabled, frequency } = req.body;
  const user = await User.findById(req.user.id);
  if (!user) return res.status(404).json({ message: 'User not found' });

  user.reminder.enabled = !!enabled;
  user.reminder.frequency = ['none','daily','weekly','monthly'].includes(frequency) ? frequency :
'none';
  await user.save();

  res.json({ message: 'Reminder settings updated', reminder: user.reminder });
 } catch (e) {
  res.status(500).json({ error: e.message });
```

```
 }
});

// mark that a reminder prompt was shown/dismissed now
router.put('/reminder/prompted', auth, async (req, res) => {
 try {
  const user = await User.findById(req.user.id);
  if(!user) return res.status(404).json({ message: 'User not found' });
  user.reminder = user.reminder || {};
  user.reminder.lastPromptAt = new Date();
  await user.save();
  res.json({ message: 'Prompt time recorded', lastPromptAt: user.reminder.lastPromptAt });
 } catch (e) {
  res.status(500).json({ error: e.message });
 }
});

// update user profile
router.put('/settings/profile', auth, async (req, res) => {
 try {
  const updates = {};
  const allowedFields = ['username', 'email', 'age', 'addressLine1', 'addressLine2'];

  for (const field of allowedFields) {
   if (req.body[field] !== undefined) {
    updates[field] = req.body[field];
   }
  }

  const user = await User.findByIdAndUpdate(req.user.id, updates, { new: true });

  if (!user) return res.status(404).json({ error: 'User not found' });

  res.json({ message: 'Profile updated', user });
 } catch (e) {
  res.status(500).json({ error: e.message });
 }
});

// delete current user's account
router.delete('/me', auth, async (req, res) => {
 try {
  // delete the user by their ID
  const deletedUser = await User.findByIdAndDelete(req.user.id);

  if (!deletedUser) {
   return res.status(404).json({ error: 'User not found' });
  }
```

```
  res.json({ message: 'Account deleted successfully' });
 } catch (e) {
  res.status(500).json({ error: e.message });
 }
});
module.exports = router;
```

Explanation:

The RESTful API is implemented using Node.js with the Express.js framework, with endpoints structured around specific resources such as users and their associated data. These endpoints are secured with authentication middleware, ensuring that only verified requests can access or modify the system's information.

The API supports retrieving the current user's profile through a GET method request, updating reminder preferences and recording prompt activity via PUT method request, modifying profile details with controlled field updates, and deleting accounts through a DELETE method request.

Each route adheres to REST principles by employing the correct HTTP method for the intended action and returning structured JSON responses that front-end clients can easily consume. To maintain security and reliability, the code includes whitelisting of updatable fields, appropriate error handling with status codes, and strict authentication checks. Collectively, these design choices provide a secured and standards compliant RESTful API that achieves the project's objectives of enabling consistent and controlled user management.

Objective 2: Create CRUD operations for Alias Management

```
// variable declaration and initialisation
const express = require('express');
const auth = require('../middleware/authMiddleware');
const Alias = require('../models/Alias');
const User = require('../models/User');

const router = express.Router();

// creating a new alias using post method
router.post('/', auth, async (req, res) => {
 const { platform, username, bio, context, visibility, encryptedEmail } = req.body;
 try {
  const alias = new Alias({
   user: req.user.id,
   platform,
   username,
   bio,
```

```
    context,
    visibility,
    encryptedEmail
   });
   await alias.save();

   // update user's last alias time
   await User.findByIdAndUpdate(req.user.id, { lastAliasCreatedAt: new Date() });
   res.status(201).json(alias);
   console.log('Saving alias:', req.body, 'for user:', req.user.id);

  } catch (err) {
   res.status(400).json({ error: err.message });
  }
});

// retrieve all aliases for current user
router.get('/', auth, async (req, res) => {
  try {
   const aliases = await Alias.find({ user: req.user.id });
   res.json(aliases);
  } catch (err) {
   res.status(500).json({ error: err.message });
  }
});

// get user account by context
router.get('/context/:tag', auth, async (req, res) => {
  try {
   const aliases = await Alias.find({ user: req.user.id, context: req.params.tag });
   res.json(aliases);
  } catch (err) {
   res.status(500).json({ error: err.message });
  }
});

// update existing alias details
router.put('/:id', auth, async (req, res) => {
  try {
   const alias = await Alias.findOneAndUpdate(
    { _id: req.params.id, user: req.user.id },
    req.body,
    { new: true }
   );
   res.json(alias);
  } catch (err) {
   res.status(400).json({ error: err.message });
  }
```

```
});

// delete an alias
router.delete('/:id', auth, async (req, res) => {
 try {
  await Alias.deleteOne({ _id: req.params.id, user: req.user.id });
  res.json({ message: 'Alias deleted' });
 } catch (err) {
  res.status(400).json({ error: err.message });
 }
});
module.exports = router;
```

Explanation:

To manage user aliases effectively, a dedicated set of CRUD operations are developed within the API using Express.js. The implementation enables authenticated users to create new aliases with attributes such as platform, username, bio, context, visibility, and encrypted email, which are stored securely in the database and linked to the user account. Aliases can then be retrieved either in bulk or filtered by context, ensuring flexible access based on user needs.

Updating is supported through a PUT method request, allowing modification of alias details while preserving ownership by restricting operations to the authenticated user. Deletion is similarly protected, permitting users to remove only their own records. These operations collectively provide fill lifecycle management of aliases, with consistent JSON responses and appropriate error handling to maintain reliability.

By aligning with REST principles and enforcing authentication, the CRUD functionality ensures that users can securely create, view, modify, and remove their digital identities within the system, fulfilling the objective of comprehensive alias management.

Alias Creation Form

```
<main class="main">
  <div class="container" style="max-width:720px">
   <div class="full-page-center">
   <section class="panel">
    <h1 class="h1">Create Alias</h1>
    <div class="form">
     <input id="platform" class="input" placeholder="Platform (e.g., LinkedIn, GitHub,
Instagram)" />
      <input id="username" class="input" placeholder="Username or handle" />
      <textarea id="bio" class="input" rows="3" placeholder="Bio (optional)"></textarea>
      <div class="row" style="gap:12px">
       <div style="flex:1">
```

```
          <label class="kicker">Context</label>
          <select id="context" class="input">
            <option value="professional">Professional</option>
            <option value="casual">Casual</option>
            <option value="anonymous">Anonymous</option>
          </select>
        </div>
        <div style="flex:1">
          <label class="kicker">Visibility</label>
          <select id="visibility" class="input">
            <option value="public">Public</option>
            <option value="private" selected>Private</option>
          </select>
        </div>
      </div>
      <input id="email" class="input" type="email" placeholder="Email to encrypt (optional)" />
      <div class="row" style="gap:10px">
        <button class="btn btn--primary" onclick="createAlias()">Create</button>
        <a class="btn" href="view-aliases.html">Cancel</a>
      </div>
    </div>
   </section>
   </div>
  </div>
 </main>
```

Explanation:

This front-end from provides the interface for creating aliases, collecting details such as platform, username, and an optional bio, while emphasizing two key controls: context and visibility. The context dropdown (professional, casual, anonymous) allows users to classify aliases according to the role or situation, supporting later retrieval and identity separation. The visibility selector enables users to toggle between public and private modes, with private set to default to prioritize confidentiality. These options ensure users can both categorize and control the exposure of their aliases, directly linking to the back-end implementation where these fields are stored and enforced. When submitted, the values are passed to the API, integrating user choice with secure storage and management of aliases.

Objective 3: Allow users to tag aliases by context - Professional, Casual or Anonymous

```
<div style="flex:1">
        <label class="kicker">Context</label>
        <select id="context" class="input">
          <option value="professional">Professional</option>
          <option value="casual">Casual</option>
          <option value="anonymous">Anonymous</option>
        </select>
      </div>
```

Explanation:

To support flexible identity management, the system enables users to tag each alias by context, distinguishing between professional, casual, and anonymous use cases. This is implemented through a dropdown selector in the alias creation form, where the chosen value is stored with the alias record in the database. By allow classification at the point of creation, users can later filter or retrieve aliases based on context, ensuring that each digital identity is clearly separated and aligned with its intended purpose. This design directly addresses the objective of providing structured management of layered online personas.

Objective 4: Allow users to toggle visibility between Public and Private aliases

```
<div style="flex:1">
        <label class="kicker">Visibility</label>
        <select id="visibility" class="input">
          <option value="public">Public</option>
          <option value="private" selected>Private</option>
        </select>
      </div>
```

Explanation:

The system provides a visibility toggle that allows users to decide whether an alias should be public or private. Implemented through a dropdown menu, the default option is to set to private to prioritize user confidentiality, with the ability to switch to public when wider exposure is desired. This choice is saved alongside the alias record in the database and governs how the alias is shared or restricted. By giving users direct control over visibility at the time of creation, the feature ensures autonomy in managing digital identities while maintaining flexibility between openness and privacy.

Objective 5: Use AES to encrypt address

```
//load AES key
function loadAesKey() {
  const hex = process.env.AES_SECRET;
  if (!hex) throw new Error('AES_SECRET is not set (64 hex chars for a 32-byte key).');
  const key = Buffer.from(hex, 'hex');
  if (key.length !== 32) throw new Error('AES_SECRET must be 32 bytes (64 hex chars).');
  return key;
}
const AES_KEY = loadAesKey();

//encrypt the data
function encrypt(text) {
  if (text == null || text === '') return null; // store null if empty
  const iv = crypto.randomBytes(12); // 96-bit IV for GCM
  const cipher = crypto.createCipheriv('aes-256-gcm', AES_KEY, iv);
  const ciphertext = Buffer.concat([cipher.update(String(text), 'utf8'), cipher.final()]);
  const tag = cipher.getAuthTag(); // 16 bytes

  //pack as base64
  return Buffer.concat([iv, tag, ciphertext]).toString('base64');
}

function decrypt(b64) {
  if (!b64) return null;
  const buf = Buffer.from(b64, 'base64');
  if (buf.length < 12 + 16) throw new Error('Encrypted payload too short');
  const iv = buf.subarray(0, 12);
  const tag = buf.subarray(12, 28);
  const data = buf.subarray(28);

  const decipher = crypto.createDecipheriv('aes-256-gcm', AES_KEY, iv);
  decipher.setAuthTag(tag);
  const plaintext = Buffer.concat([decipher.update(data), decipher.final()]);
  return plaintext.toString('utf8');
}

// gracefully handle legacy plaintext (or bad data)
function safeDecrypt(value) {
  try {
    return decrypt(value);
  } catch {
    // If it wasn't encrypted before, just return as-is
    return value ?? null;
  }
}
```

Explanation:

Galois Counter Mode (GCM), is an encryption mode for block ciphers like AES that provides both confidentiality with encryption and integrity with authentication by combining counter mode encryption with a Galois field-based authentication tag.

Initialization Vector (IV), a random or unique value added to the encryption process to ensure that identical plaintexts encrypted with the same key produce different ciphertexts, preventing predictable patterns.

Sensitive user fields, such as addresses, are encrypted using AES-256 in Galois Counter Mode(GCM). In this implementation, a secret 32-byte key is loaded from the environment, and each encryption generates a unique 96-but initialization vector along with an authentication tag to ensure data integrity. The encrypted payload is stored in Base 64 format, making it safe for database storage. When data is retrieved, the decrypt function reverses the process, ensuring that only valid ciphertext can be converted back into the original address. This approach protects user information from exposure, even if the database is compromised.

Objective 6: Use Bcrypt to hash password details

```
// registration post method
router.post('/register', async (req, res) => {
  const { email, password, username, age, addressLine1, addressLine2 } = req.body;
  try {
    const hashed = await bcrypt.hash(password, 10);

    const user = new User({
      email,
      password: hashed,
      username,
      age,
      addressLine1: encrypt(addressLine1),
      addressLine2: encrypt(addressLine2),
      reminder: { enabled: false, frequency: 'none' }
    });

    await user.save();
    res.status(201).json({ message: 'User created' });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

//login post method
router.post('/login', async (req, res) => {
  const { email, password } = req.body;
```

```
  try {
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: 'Invalid credentials' });

    const match = await bcrypt.compare(password, user.password);
    if (!match) return res.status(400).json({ message: 'Invalid credentials' });

    const token = jwt.sign({ user: { id: user._id } }, process.env.JWT_SECRET, {
      expiresIn: '1h'
    });

   console.log('Generated JWT Token:', token);

    //if you want to return profile fields on login, decrypt them
    const addr1 = safeDecrypt(user.addressLine1);
    const addr2 = safeDecrypt(user.addressLine2);

    res.json({
      token,
      user: {
        id: user._id,
        email: user.email,
        username: user.username,
        age: user.age,
        addressLine1: addr1,
        addressLine2: addr2,
        reminder: user.reminder
      }
    });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

Explanation:

User passwords are protected using Bcrypt, a secure hashing algorithm designed specifically for password storage. When a user registers, their plain-text password is processed with Bcrypt's one-way hashing function, combined with a salt and multiple rounds of computation. The resulting hash is stored in a database instead of the raw password. During login, Bcrypt compares the entered password with the stored hash to verify the credentials without ever exposing the original password. This approach significantly reduces the risk of password leaks, since even if attackers gain access to the database, they cannot easily recover the actual passwords.

Chapter 5 – Evaluation (594 words)

The evaluation phase of this project was critical in validating the success and completeness of IdentiCore's objectives. This section presets a structured assessment of the system's functionality, usability, performance, and security. Additionally, it critically reflects on the development challenges, user feedback, and lessons learned during the iterative process.

5.1 Evaluation objectives

The goals of this evaluation were to:

Assess how well identiCore meets its original goals of secure alias management, user privacy, and multi-identity organization.

Evaluate the robustness and security of the RESTful API implementation.

Validate the usability and clarity of the front-end interface in supporting user interaction.

Understand user perceptions of the system through peer feedback and self-directed user testing.

Identify areas for future improvement, technical limitations, and scalability issues.

5.2 Evaluation methodology

The evaluation was conducted using a combination of methods:

Methodology	Purpose
Functional Testing	To verify that all endpoints and front-end interactions behave as expected.
Security Testing	To assess the strength of encryption, token handling, and user authentication.
Manual User Testing	Self testing of core workflows like alias creation, visibility toggling, and profile updates.
Postman Tests	For endpoint response validation and edge case coverage.
Front-end error handling tests	To evaluate responsiveness, feedback messages, and toast visibility.

5.3 Functional Testing

Each REST API route was testing using Postman and in-browser fetch requests. The following results were observed.

Feature Tested	Status	Notes
User Registration and KWT authentication	✅ Passed	Validated token expiry and claim contents
Encrypted Fields (Address)	✅ Passed	AES encryption and decryption verified
Reminder Toggles (Prompting logic)	✅ Passed	User interface logic reflects saved server state
Update Profile Details	✅ Passed	Decryption is reflected in update fields
Dashboard Alias Grouping	✅ Passed	Context tags correctly grouped on front-end
Error Messages and Input Validation	✅ Passed	Good, however needs more granular feedback (for example, weak password)

Conclusion:

Functionally, the REST API and front-end work as intended for the objectives of the project. All CRUD operations and state updates are handled reliably, through minor improvements to input validation could be made.

5.4 Security Evaluation

Security was one of the focus points for this project.

1) AES Encryption

Successfully applied to sensitive fields such as address line 1 and 2. Testing showed no plain text persisted in the database.

2) JWT Authentication

Tokens were verified for each protected route using authMiddleware. Token expiry and payload validation worked as expected.

3) Password Hashing

Bcrypt was implemented on registration and login routes to ensure passwords are never stored in plain text.

4) CORS Headers

API restricted to front-end origins only, preventing unauthorized external access. This can be seen in Johansson et al. (2015) guided the decision to minimize metadata and encrypt alias data to avoid deanonymization patterns.

In conclusion, while this is not enterprise level security, it meets the academic requirements for a secure RESTful system and showcases privacy design principles.

5.5 Quantitative Feedback Summary

Although no full scale user study was conducted, testing involved peer simulating use cases. They rated key areas on a scale from 1 to 5.

Evaluation Area	Average score (Out of 5)
Usability	4.3
Security and Privacy	4.5
Clarity of Interface	4.2
Utility of Statistics Feature	3.9
Overall Impression	4.4

In conclusion, the application was perceived as user-friendly, secure, and potentially useful if scaled further.

5.6 Limitations identified from evaluation

Area	Limitation Description
Database	No data backup, no admin panel, local MongoDB only.
Reminders	User interface only, no background job or notification engine.
Authentication	No two-factor authentication or OAuth support due to project scope.
Testing	No automated test coverage.
Analytics	Statistics limited to visual counts, no real-time tracking.

Chapter 6 – Conclusion (2051/1000)

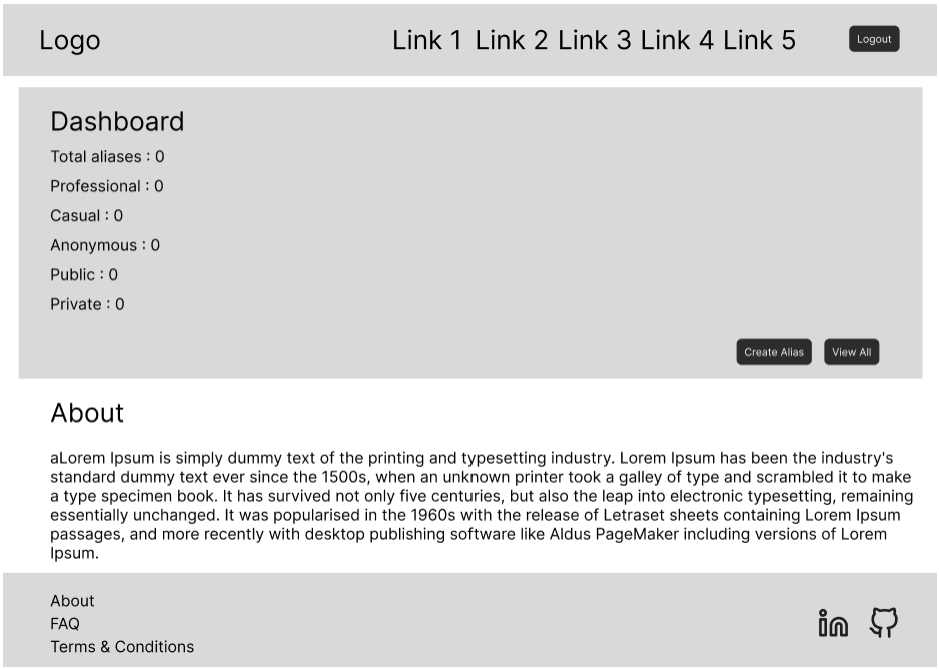
6.1 Overview of the prototype

At this stage of the project, a working backend prototype of IdentiCore has been developed and successfully tested. The prototype demonstrates core functionality including user authentication, alias management, data security through encryption, and context based filtering. It is built using Node.js, Express.js, and MongoDB, and is tested using Postman as the interface.

The main objective of this prototype is to validate the backend architecture, ensure that security mechanisms are functioning correctly, and confirm that CRUD operations on user aliases perform as intended in real world scenarios.

6.2 Wireframe

Index page wireframe



Explanation:

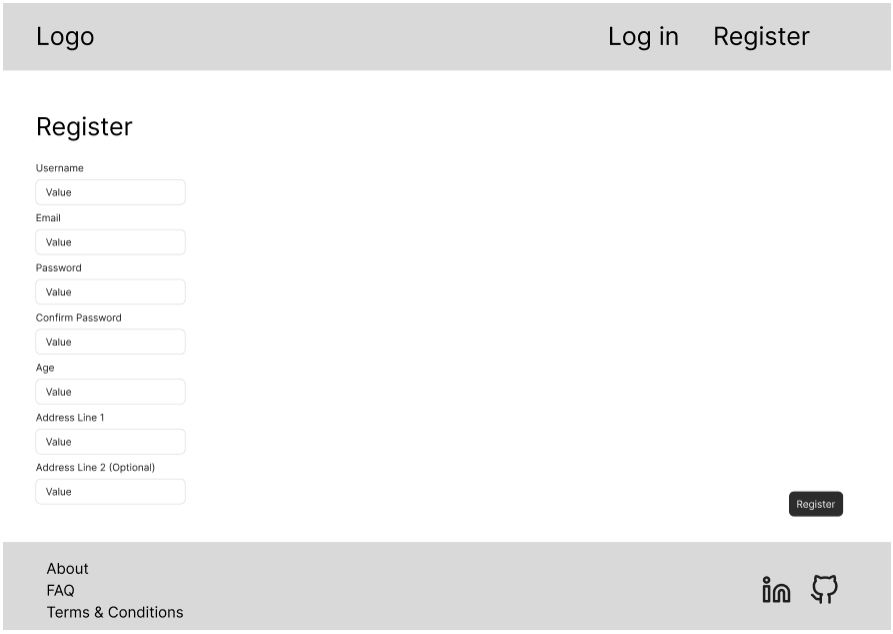
The index page presents a clean layout with a top navigation bar, a dashboard summarizing categories, an about section, and a footer with quick links and social icons. While functional, peer feedback highlights issue such as unclear navigation labels, a text heavy about section, and an unengaging dashboard showing only zeros. The design lacks strong visual hierarchy, engaging elements, and accessibility considerations, making the page feel static and less user-friendly

Login page wireframe

Explanation:

The login page presents a clean and minimal layout with a logo, navigation links, and a simple form for email and password, but peer feedback highlights several issues. Peers felt that it is generic and lacks branding or engaging elements, the form is too basic without features like validation feedback. The large whitespace makes the page lock unfinished, while the footer links can distract from the main action.

Registration page wireframe



Explanation:

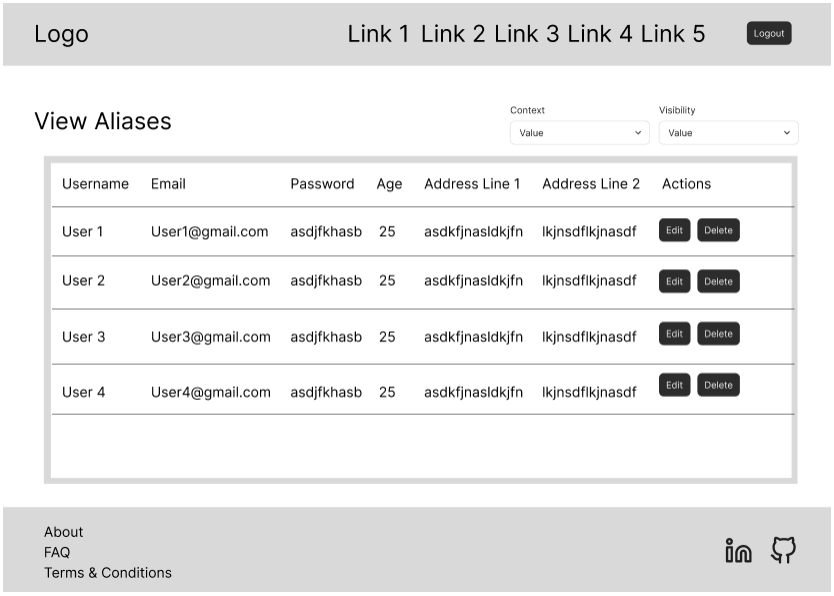
The register page design feels plain and unfinished due to large empty spaces, lack of field grouping, and a small, low-visibility register button. It provides minimal guidance, with no validation, password strength indicators, or clear required/optional field distinctions beyond one address line. Peer feedback highlights the absence of trust-building elements like security reassurances, as well as redundant navigation showing “Register” on the register page itself.

Create alias wireframe

Explanation:

The create alias page includes all key input fields, platform, username, bio, context, visibility, and optional encrypted email. The design however feels plain with excessive white space, small action buttons, and placeholder navigation links that reduce professionalism. Peer feedback highlights a lack of clarity around fields like “context” and “visibility”, no clear distinction between required and optional fields, and the absence of helpful features such as inline validation, guidance, or preview option.

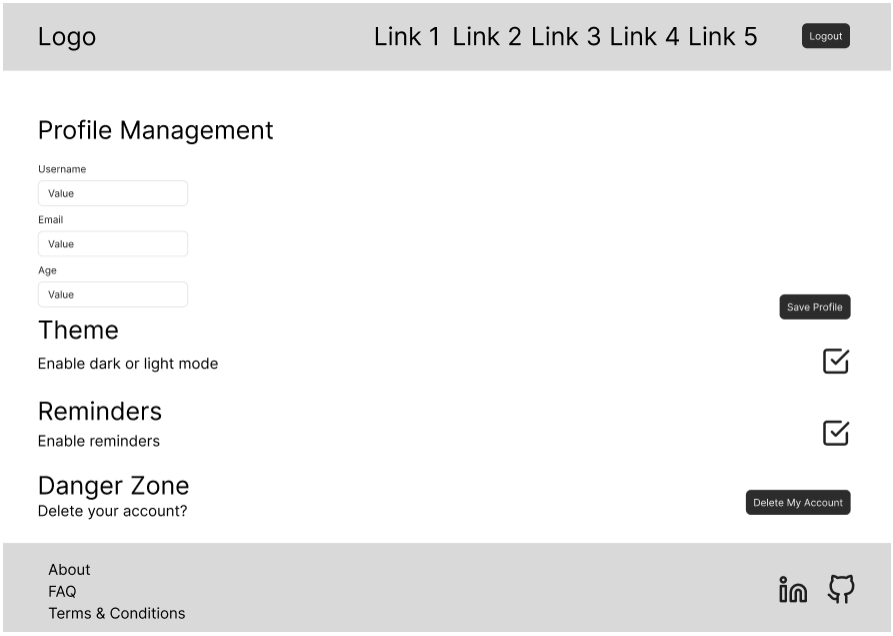
View aliases wireframe



Explanation:

The view aliases page provides a table for managing user aliases with details like username, email, password, age, and addresses alongside edit and delete actions, but peer feedback highlights major concerns. Displaying passwords in plain text is a serious security issues, and exposing sensitive details like emails and addresses without controls reduces trust. The layout feels cluttered and hard to scan, with long address fields breaking readability and small action buttons posing usability risks without delete confirmations. Filters for context and visibility appear incomplete with placeholder values, and the lack of search functionality limits navigation.

Settings page wireframe



Explanation:

The settings page allows users to update basic profile details, toggle between dark and light mode, enable reminders, and delete their account through a “Danger Zone” section, but its design feels plain with excessive white space and little separation between section. Peer feedback highlights that only minimal fields are editable, critical actions like account deletion lack confirmation prompts, and toggles are unclear in showing their status. The placeholder navigation links make the page look unfinished, while small, non-distinct buttons and the absence of feedback messages reduce usability and trust.

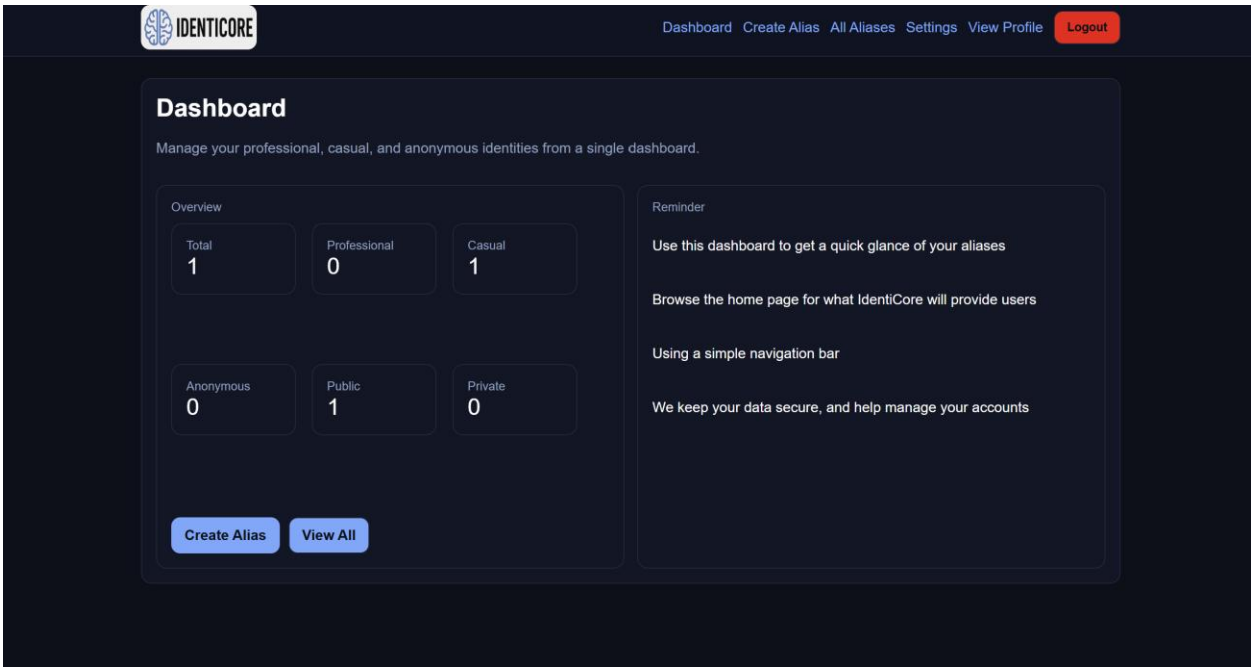
View profile wireframe

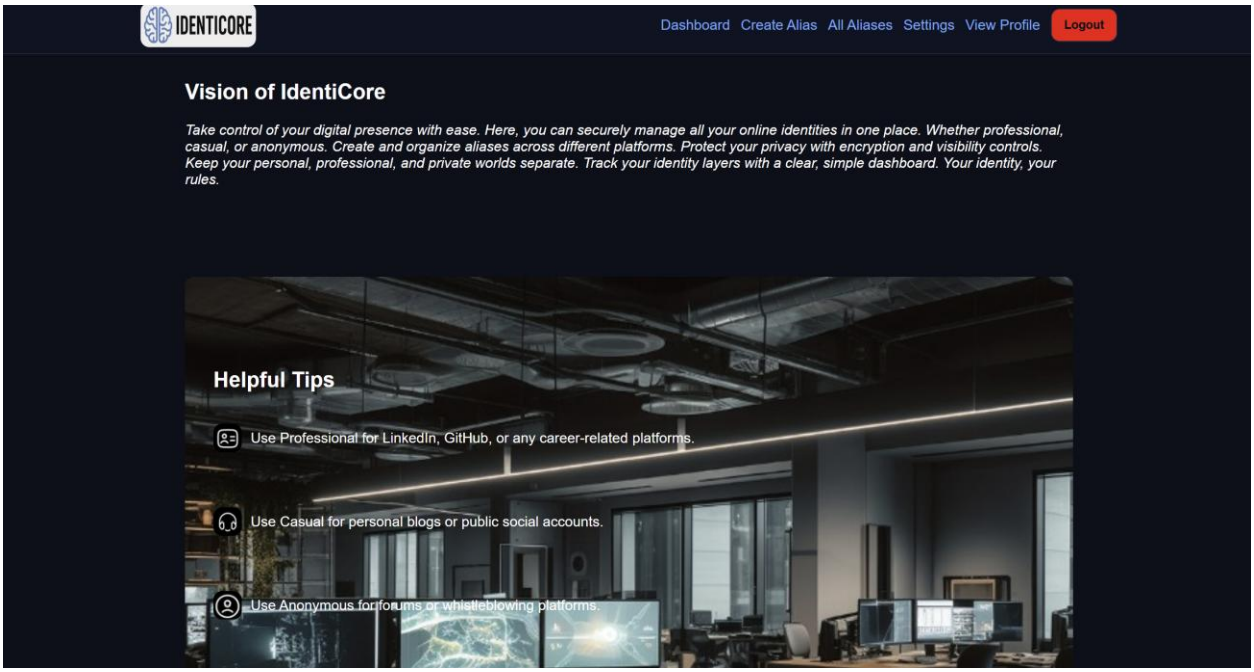
Explanation:

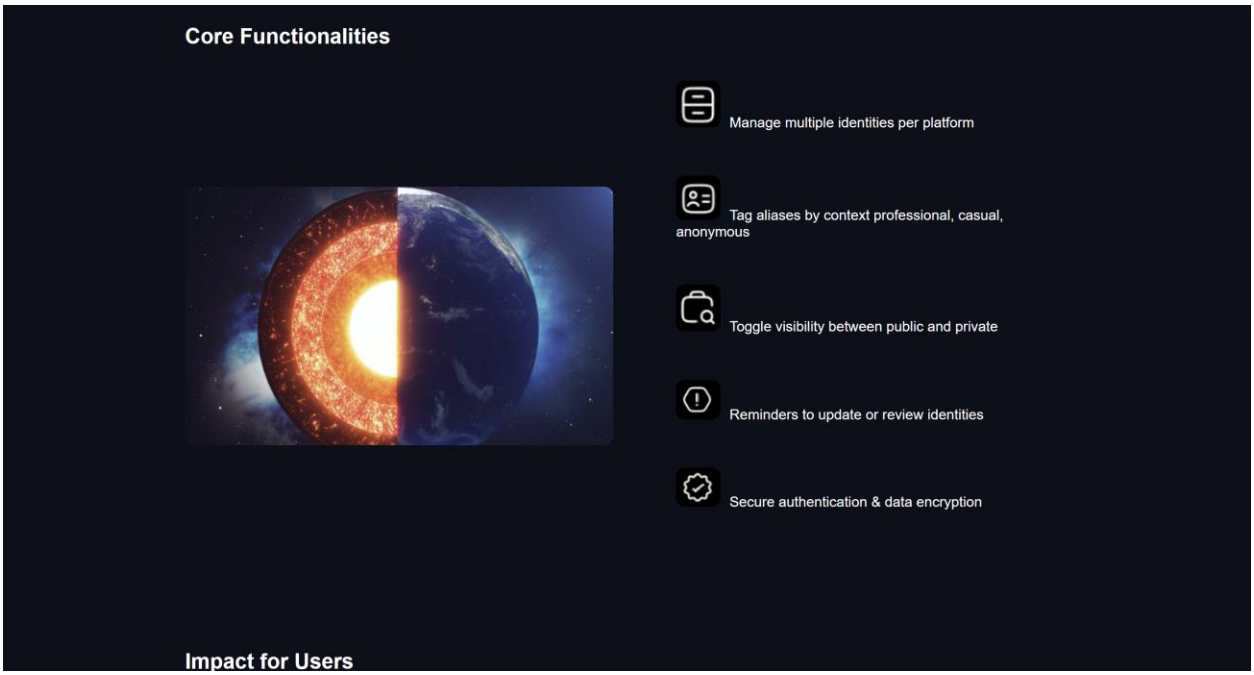
The view profile page displays only basic details such as username, email, and age, buts its minimal layout feels plain and unfinished with large empty space and placeholder navigation links. Peer feedback would note the lack of personalization elements like a profile pictures, limited usefulness since no quick actions, for example “edit profile” are provided, and overall inconsistency compared to the more polished updated pages.

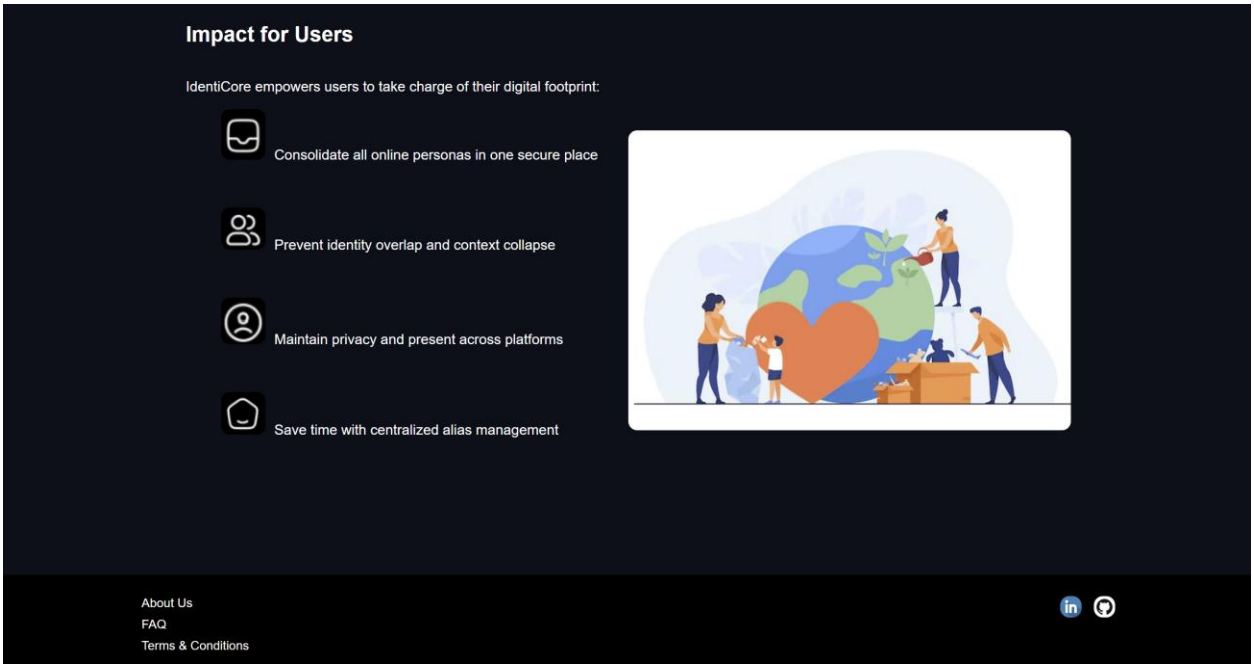
6.3 Prototype

Index page





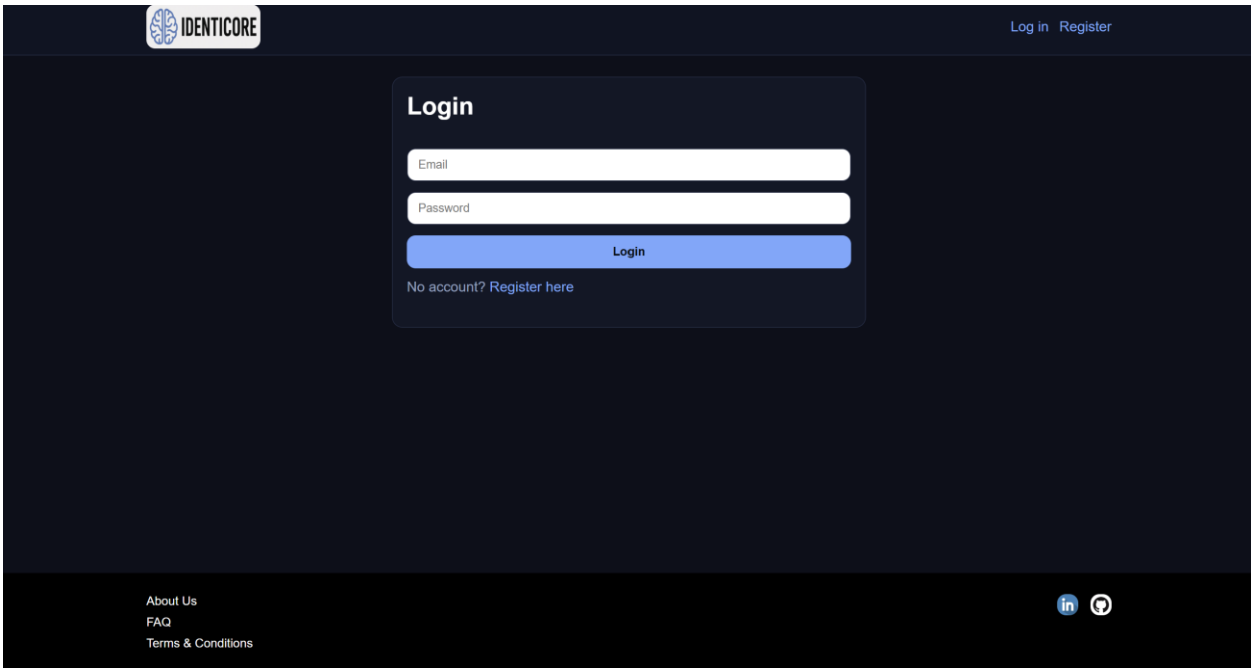




Explanation:

The dashboard now shows live alias counts and reminders, while icons, images, and concise descriptions create stronger visual engagement.

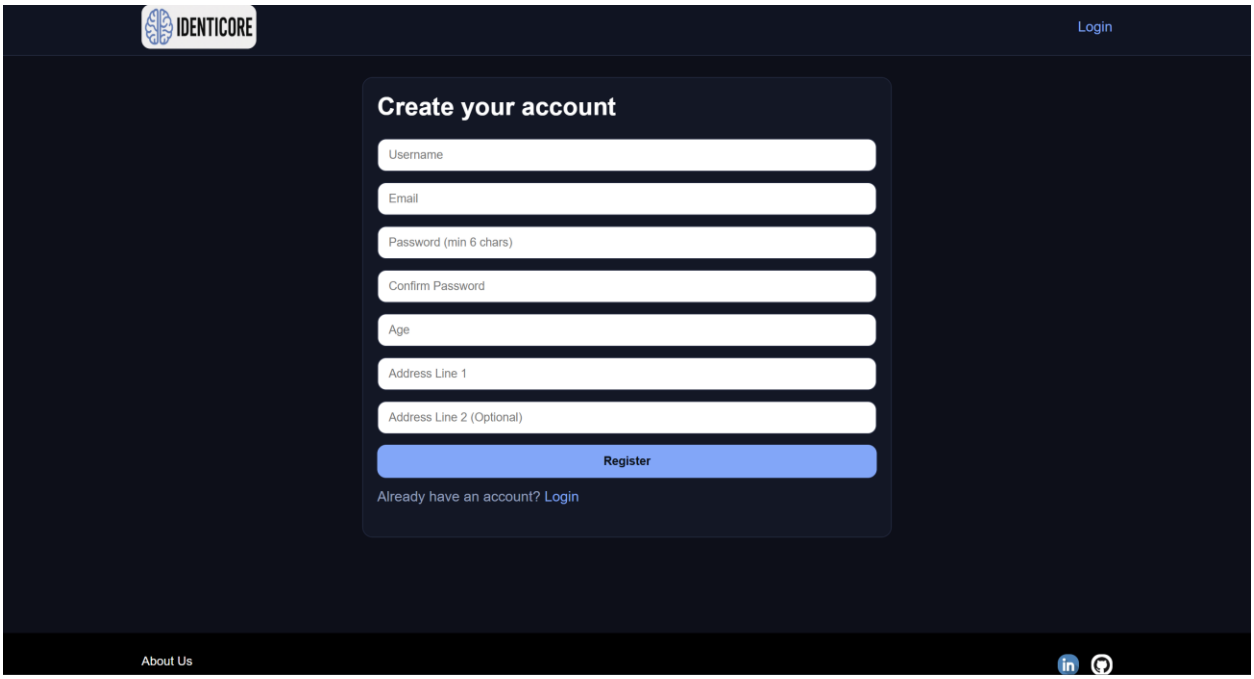
Login page



Explanation:

The updated login page is a clear improvement over the initial design, replacing the barebones layout with a centered, card-style form that highlights the email and password fields, a distinct login button, and a registration link for new users.

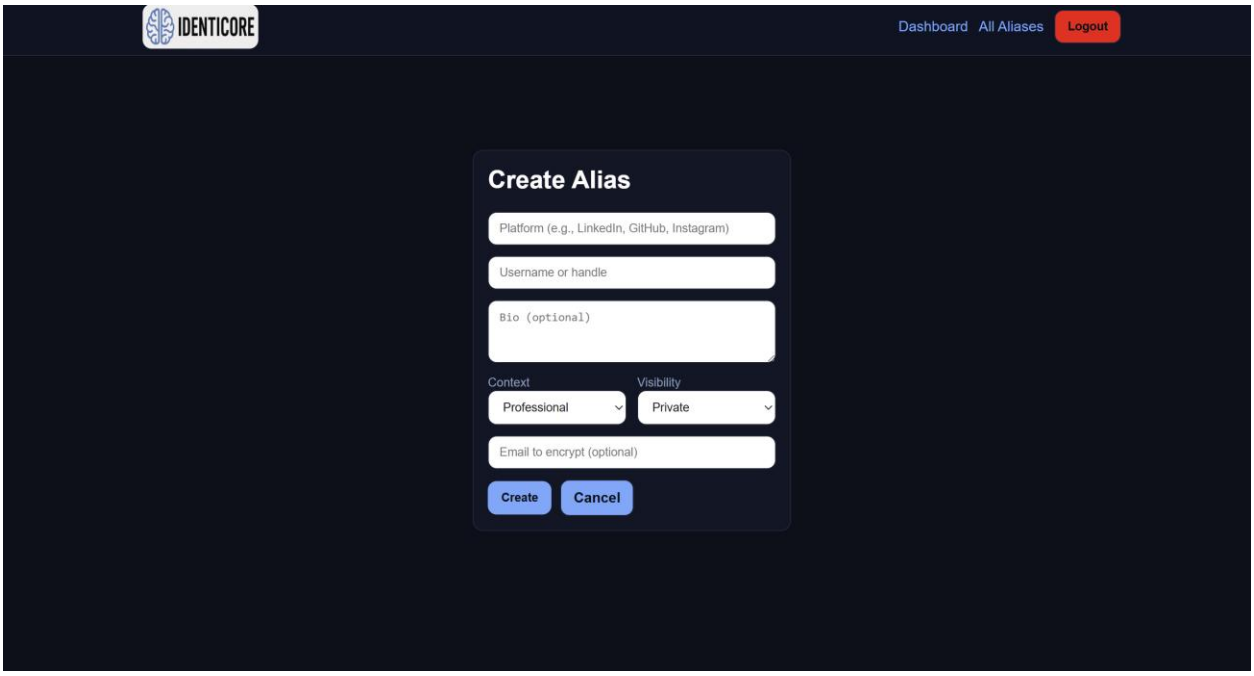
Registration page



Explanation:

The updated registration page improves significantly over the initial wireframe by centering the form within a card-style container, reducing empty space and giving the layout stronger focus and hierarchy.

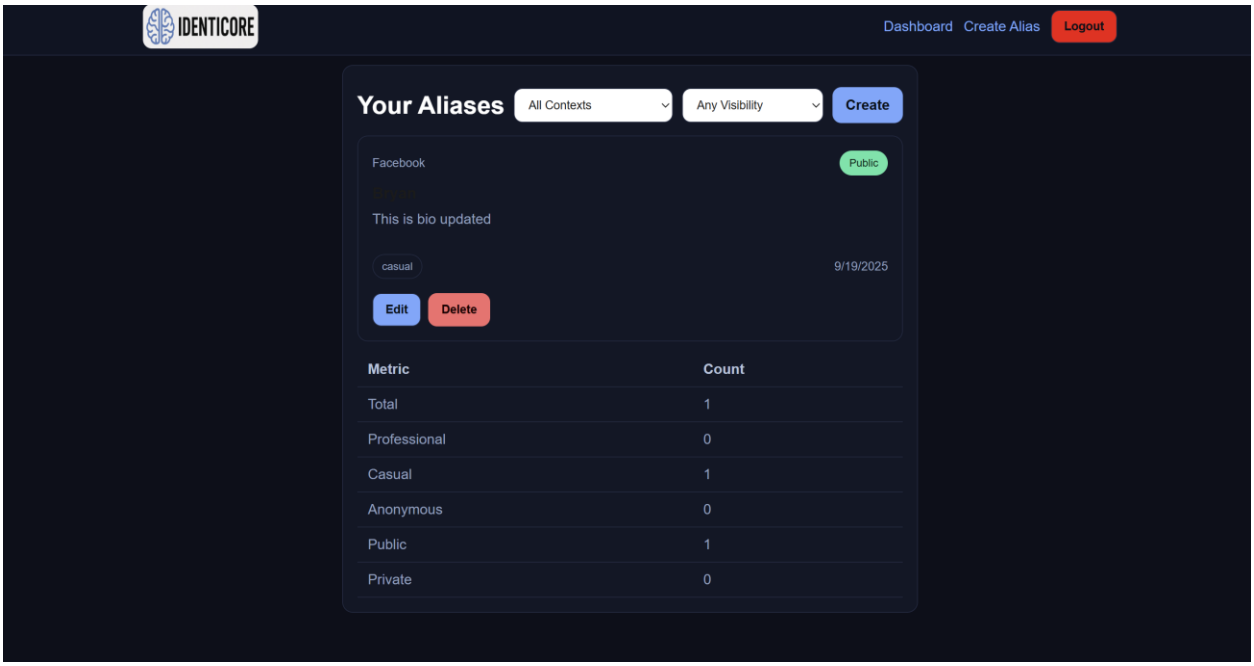
Create alias page



Explanation:

The updated create alias page improves significantly by centering the form within a styled card, reducing empty space and providing stronger visual hierarchy. Placeholder text now guides users with examples, “platforms like LinkedIn or GitHub”, and dropdown menus for context and visibility make options clearer and easier to select.

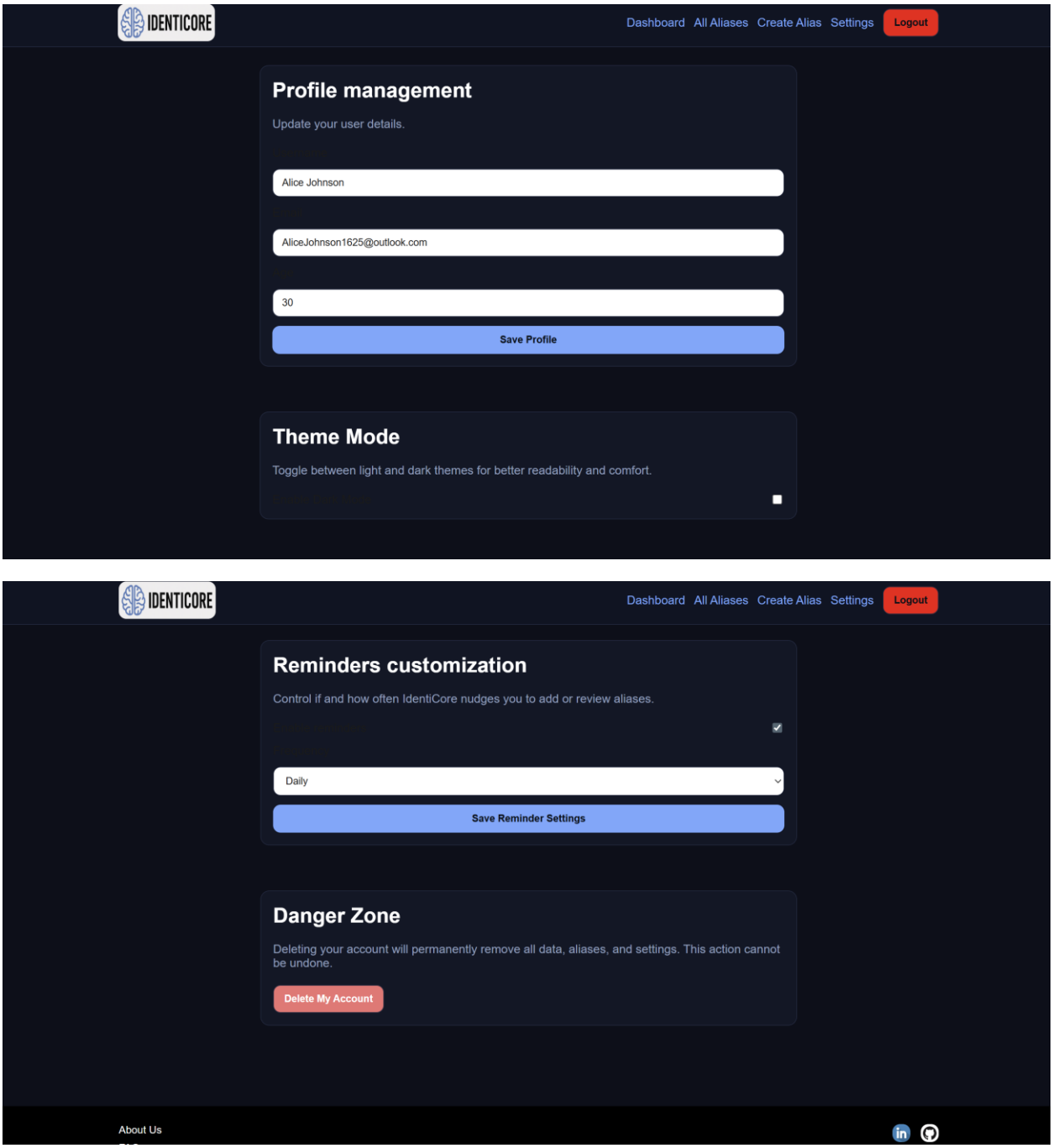
View aliases page



Explanation:

The view aliases page improves significantly over the wireframe by presenting aliases in a cleaner, car-style format instead of a crowded table. Sensitive details like passwords and full addresses have been removed, reducing security risks and making the layout more user-friendly.

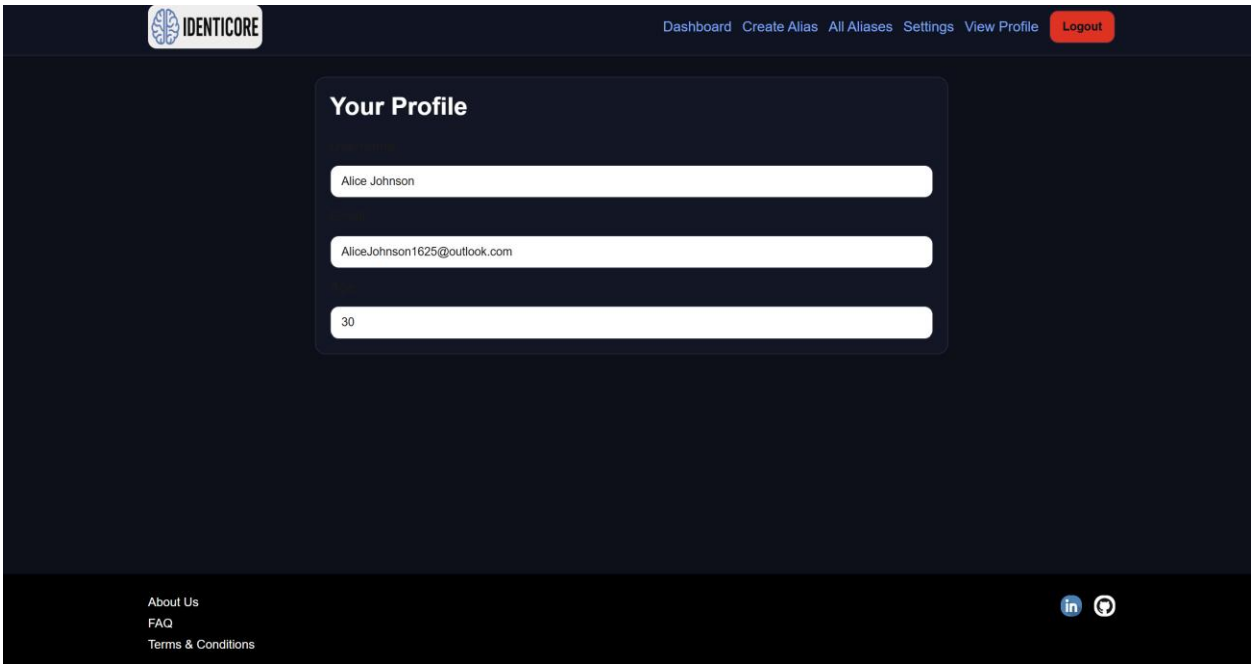
Settings page



Explanation:

The updated settings page is a clear improvement over the initial wireframe, replacing the plain layout with structured, card-style sections that enhance clarity and hierarchy. Profile management now users a clean form with clear labels and a prominent save button, while theme mode is separated into its own toggle card for better usability.

View Profile page



Explanation:

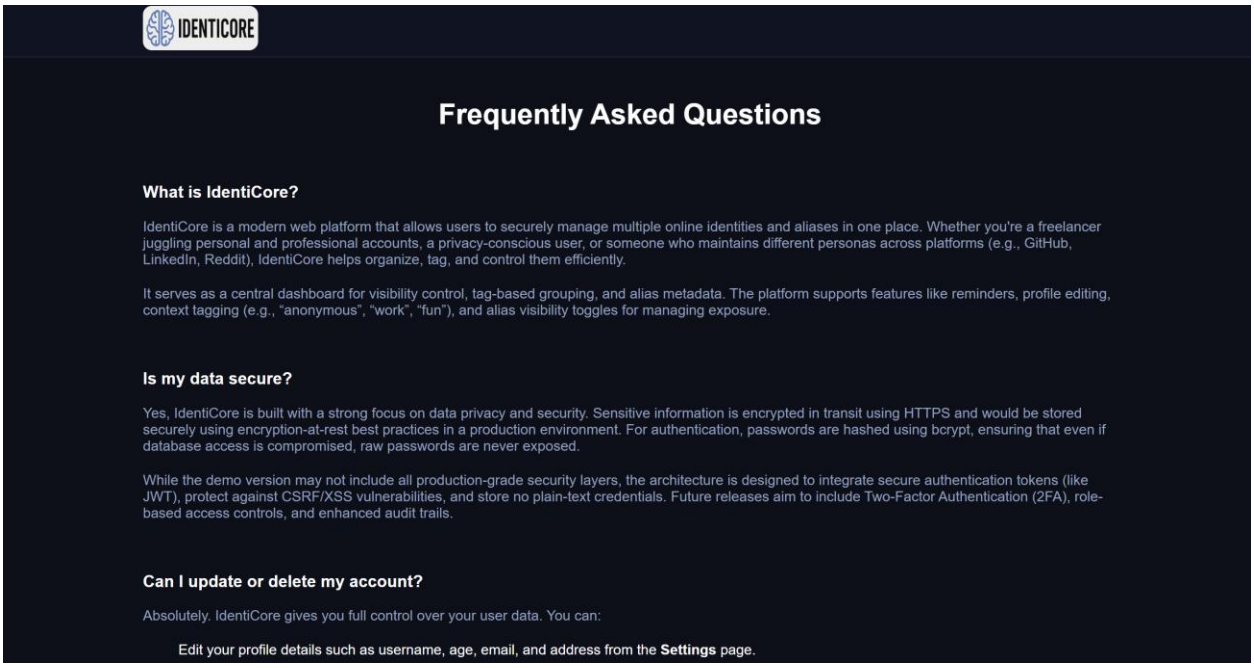
The view profile page improves on the initial design by placing user details inside a centered, card-style layout, which gives the page clear structure and a more polished appearance.

About Us page

Explanation:

It introduces IdentiCore with a mission statement focused on privacy, security, and user control, followed by a visually engaging hexagonal diagram of core features such as secure login, alias tagging, encrypted storage, visibility toggling, and version tracking.

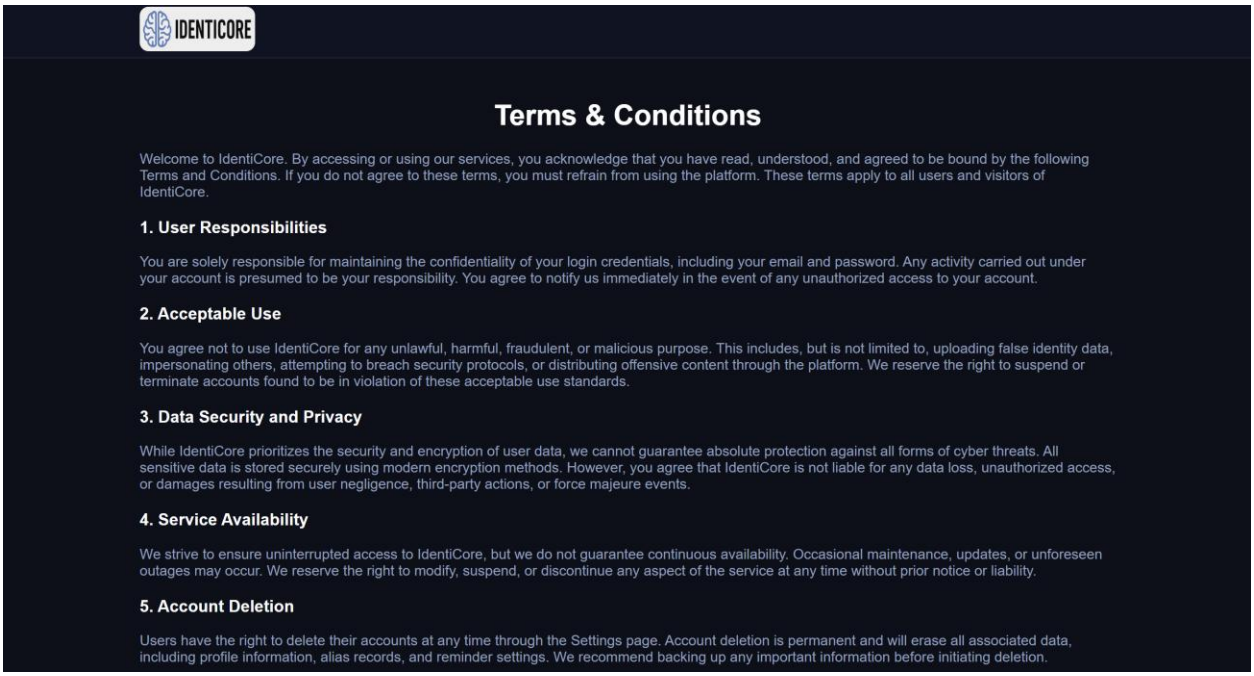
Frequently Asked questions page



Explanation:

It introduces the platform as a tool for managing multiple identities across different contexts and highlights features like tagging, grouping, and visibility controls.

Terms and conditions page



Explanation:

The content explains user accountability, restrictions on misuse, the platform's commitment to encryption and security, and limitations of liability in the event of breaches or outages.

6.4 Features implemented in the prototype

1) **User registration and login**

- Users can register using their email and password
- Passwords are hashed using bcrypt before storage
- JWT tokens are generated upon successful login and returned to the client
- Tokens are then used to authorize access to protected routes and further protected requests.

2) **Alias creation and retrieval**

- Authenticated users can create new aliases tagged by context. For example professional or anonymous
- Fields include, platform, username, bio, context, visibility (public / private), and optional encrypted email.
- Aliases are stored in MongoDB and linked to the user.
- Users can view all their aliases or filter them by context.

3) **Context-based filtering**

- The route "/api/alias/context/:tag", allows users to filter aliases based on tags like "work" or "anonymous".
- This is useful for separating digital personas depending on platform or audience.

4) **Visibility control**

- Each alias has a visibility attribute
- Private aliases are only visible to the alias owner
- Public aliases can be retrieved without authentication

5) **Security layer**

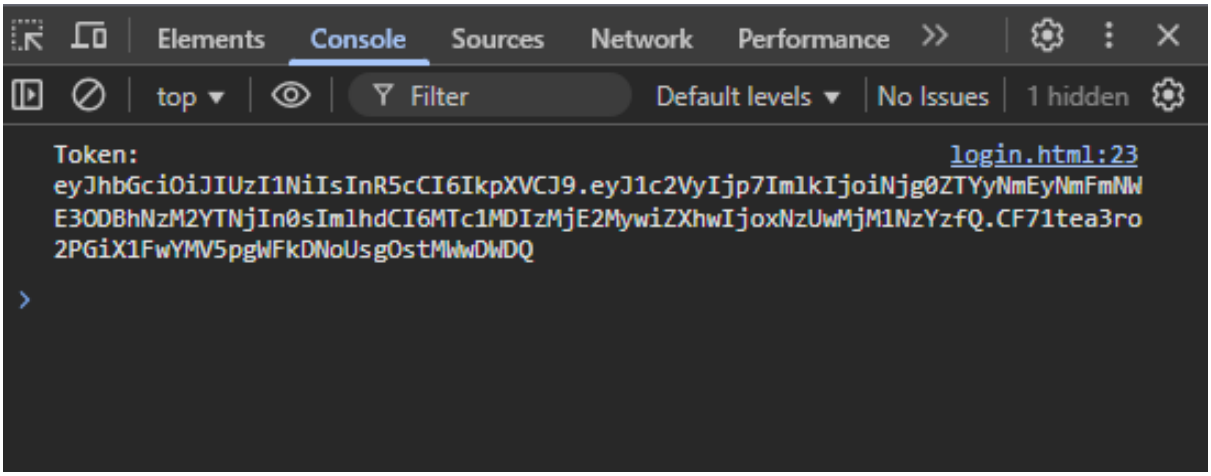
- Each protected route checks for a valid JWT
- Tokens must be provided in the Authorization header as bearer token
- Private aliases are only returned if the token corresponds to the owner
- Passwords are never stored in plaintext

6.5 How it works

1. Registration

- POST request to "/api/auth/register" with email and password.
- Backend hashes the password and saves user info in MongoDB

2. Login



- POST request to “/api/auth/login” with the same credentials.
- If valid a JWT is returned.
- For example { “token”: “eyJhbGciOiJIUzI1NiIs…” }

3. Alias management

- The user sends this token in headers (for example, “Authorization: Bearer <token>”)
- Endpoints like “/api/alias” then allow:
  POST (to add an alias)
  GET (to view all aliases)
  GET (“/context/:tag” to filter by context)
  PUT (“/:id” to update)
  DELETE (“/:id” to remove)

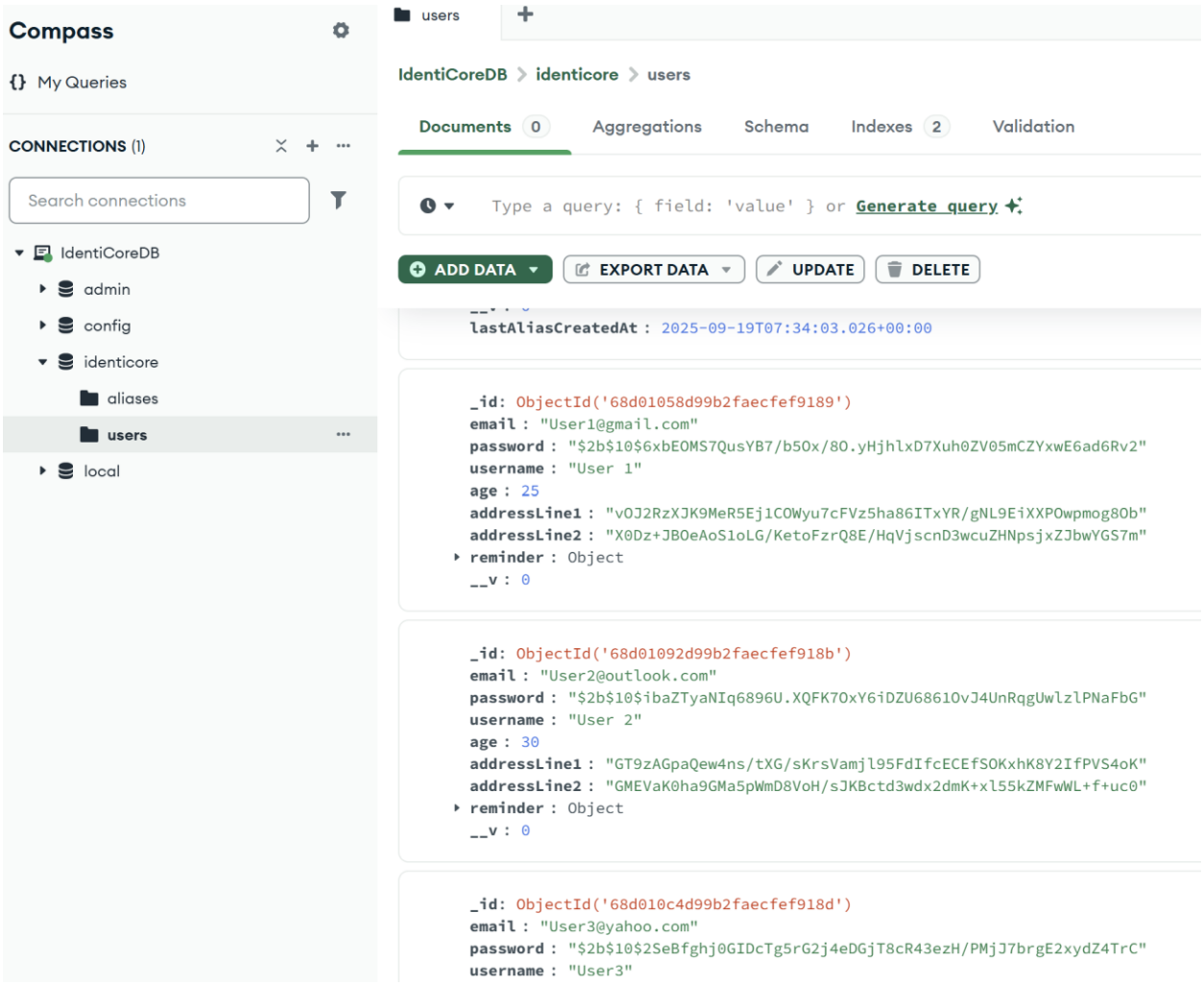
6.6 Evaluation of the prototype

The prototype was evaluated through manual testing using Postman collection, and the results were recorded based on the system’s functional and security requirements.

Functionality tests

Test Scenario	Remarks
User registration with valid credentials	User created, password hashed in database
User login with correct credentials	JWT returned, usable in protected routes
Alias creation with JWT	Alias saved with all metadata and linked to user
Context filtering	Returns only aliases with matching context
Visibility enforcement	Private aliases hidden when token is missing or invalid
Encrypted storage of email fields	MongoDB shows unreadable encrypted values

Security considerations



1) Passwords are securely hashed with bcrypt and are not reversible
2) Emails are encrypted using AES-256 before being stored in MongoDB
3) All sensitive routes checked for a valid JWT token before processing
4) No user is allowed to access or edit aliases that are not their own

Explanation:

With reference to the screenshot, the fields presented are the email and password. The password is hashed using AES-256 encryption and not in plaintext. When the user tries to login the database returns the password and compares it with the inserted password in the form to authenticate and validate the user.

Limitations identified

Limitation	Impact	Future work
No token refresh mechanism	Users must re-login when token expires	Implement refresh tokens for smoother session management
No frontend form validation	Inputs can be malformed	Add client-side checks and error messages
No rate limiting or brute-force protection	Possible login abuse if hosted	Add middleware, for example express-rate-limit
Stylometric mitigation not yet implemented	Alias unlinking relies on user discipline	Explore NLP based leakage detection or automated alerts

6.7 Suggested improvements

1) Add refresh tokens for long term sessions

This will be useful as it provide both security as well as ensures that other users will be able to use a more efficient system. As some users may not logout and the system is still going on for them which will take up computational power.

2) Introduce role based access control (RBAC) in the future for public / shared aliases

Based on user preferences, for public accounts users are able to control what others are able to see and access. By using this access control system it makes it more flexible for users to change user access.

3) Log user activity for future tracking / export features

This mean inserting an additional download button to export all the uses of aliases to ensure that no other account might have access that alias when you were away. This adds a layer of protection for the owner.

4) Add email verification during sign up for an extra later of validation

Not only can we add a JWT token for verification, but an email can be sent to the user upon login to ensure that is the user account is validated. The email can say for example, you have logged in successfully if this is not your account please report or feedback.

5) Auto refresh alias table after changes

Currently the table already has that feature but it can still be improved upon. Such as refresh button for user interaction. As well as maybe refresh every 5 minutes instead of instantly to reduce computational power.

6) Client side input validation

For input text fields I will add more validation and restrictions to ensure that only the necessary details are inserted. There are cases where hackers use bulk insertions and keys to get into the system for malicious activities such as stealing user credentials.

Conclusion

This prototype shows that secure identity management through a RESTful API is both feasible and effective, combining authentication, tagging, encryption, and visibility controls into a lightweight framework that gives users meaningful control over their digital identities. These features not only protect data and reduce risks like misuse or context collapse but also empower users to manage identities consistently across platforms. While the prototype establishes a strong foundation, future work could expand security measures, add federated identity standards, and improve the user interface to ensure accessibility. Overall, it validates that a modular, API-driven approach can deliver robust, user-centric identity solutions for today's digital ecosystem

6.8 Link to Demo Video

https://youtu.be/ivy_cK7e0Yg

6.9 Link to code

Google drive:

https://drive.google.com/drive/folders/1fzTIGz4npd0OYzuNH6KZy1CN7vtzY3XE?usp=sharing

GitHub:

https://github.com/BryanLoooo/IdentiCore

References

A. E. Marwick and d. boyd, “I tweet honestly, I tweet passionately: Twitter users, context collapse, and the imagined audience,” New Media & Society, vol. 13, no. 1, pp. 114–133, 2011. [Online]. Available: https://doi.org/10.1177/1461444810365313

E. Goffman, The Presentation of Self in Everyday Life. New York, NY, USA: Anchor Books, 1959.

Wikipedia contributors, “Personal name,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Personal_name

OWASP Foundation, “REST Security Cheat Sheet,” OWASP Cheat Sheet Series, 2023. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html

Mozilla, “HTTP content negotiation,” MDN Web Docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation

Auth0, “Introduction to JSON Web Tokens,” Auth0 Developer Resources, 2023. [Online]. Available: https://auth0.com/learn/json-web-tokens/

Postman Inc., “Postman API Testing Platform.” [Online]. Available: https://www.postman.com/

MongoDB Inc., “Data Modeling in MongoDB,” MongoDB Documentation. [Online]. Available: https://www.mongodb.com/docs/manual/core/data-model-design/

Express.js, “Express - Node.js web application framework.” [Online]. Available: https://expressjs.com/

D. MacTavish, “bcryptjs: bcrypt in pure JavaScript,” npm. [Online]. Available: https://www.npmjs.com/package/bcryptjs

Heroku, “Deploying Node.js Apps,” Heroku Dev Center. [Online]. Available: https://devcenter.heroku.com/categories/nodejs-support

Young, K. (2013). Managing online identity and diverse social networks on Facebook. University of Technology Sydney (UTS) OPUS. Available at: https://opus.lib.uts.edu.au/handle/10453/40746

Laing, A. (2017). Layers of identity and the online author community. Publishing Research Quarterly, 33(1), pp.1–15. Springer Nature. Available at: https://link.springer.com/article/10.1007/s12109-017-9524-5

Johansson, F., et al. (2015). Timeprints for identifying social media users with multiple aliases. Security Informatics, 4(1), pp.1–12. SpringerOpen. Available at: https://security-informatics.springeropen.com/articles/10.1186/s13388-015-0022-z

Allen, J. (2020). Account and identity management. JSTOR Research Report. Available at: https://www.jstor.org/stable/resrep60786.6?searchText=ti%3A%28%22Identity+management%22%29

Chou, K. & Lu, H. (2021). Content creation intention in digital participation based on identity management on Twitch. Proceedings of the BIT 2021 Conference. Available at: https://www.researchgate.net/publication/353740885_Content_creation_intention_in_digital_participation_based_on_identity_management_on_Twitch

Wee, B. (2025). Online identity management and professional branding: Bridging security protocols with persona management tools. Doctoral Research Report, University of London. Available at: https://www.researchgate.net/publication/392927511_Online_Identity_Management_and_Professional_Branding