

|  |
| --- |
| **TERAH HANDOVER PACKAGE**  text divider |
|  |

October 24 2025

Team T327

Authored by:

Jonathan Tebble

Cole Smart

Charles McVeigh

# Executive Summary

This project was initiated in collaboration with CEO Bec Wilson to explore, design, and develop the specifications for an AI-powered Large Language Model (LLM) bot aimed at improving access to reliable information on midlife financial and lifestyle planning. The motivation behind the project stems from the growing complexity of Australia’s retirement and financial systems, and the need to empower individuals aged 47–70 with tools that demystify these systems in a compliant and accessible manner.

The project’s primary deliverable is a working proof of concept system supported by a body of research, documentation, and technical implementation. Together, these outputs demonstrate both the technical feasibility and regulatory considerations of such a system.

The bot was designed to:

* Deliver factual, reliable, and compliance-aligned information drawn from trusted public sources such as the Australian Taxation Office (ATO), Services Australia, and the Australian Securities and Investments Commission (ASIC).
* Provide information on topics including superannuation, taxation, Age Pension eligibility, and general financial literacy without crossing into personal financial advice.
* Employ compliance safeguards, such as predefined disclaimers, to ensure outputs remain general in nature and avoid contravening financial advice regulations.
* Present information in a conversational, approachable manner, guiding users to official resources and encouraging professional financial advice when appropriate.

To achieve these objectives, the team pursued a multi-phase development process, which included:

* Research into datasets, compliance requirements, and stakeholder needs.
* Comparative analysis of LLMs and prompt engineering strategies.
* Development of prototype code, including a React front-end and a Selenium-based web scraping tool for structured data ingestion.
* Iterative evaluations, stakeholder reviews, and risk management reporting.

The project produced a complete package of deliverables including the codebase, evaluation reports, legal and compliance research, risk registers, sprint plans, and stakeholder alignment documentation. Together, these form a comprehensive foundation for either continued academic research or direct extension into a production-ready solution by the industry partner.

Ultimately, the project demonstrates a scalable and responsible framework for applying generative AI to sensitive domains such as financial literacy. While the proof of concept highlights the potential of LLM-driven information delivery, it also carefully delineates the compliance boundaries necessary to ensure ethical and lawful use of the system.

Table of Contents

[**Executive Summary 2**](#_heading=h.9tksjepg7qj9)

[**Project Scope and Context 5**](#_heading=h.9nirinmr3r2t)

[Background 5](#_heading=h.l3q4g39qvzfp)

[Original Project Brief 5](#_heading=h.kmt9x1i5gcw4)

[Scope Boundaries 6](#_heading=h.wlw2yjtu94rs)

[Alignment with Industry Partner Needs 6](#_heading=h.i79sy9q2l7ge)

[Context within the Broader Field 6](#_heading=h.pztju8zensck)

[**System Overview 7**](#_heading=h.dyg74m4zec9a)

[High-Level Architecture 7](#_heading=h.yp5vz5afs8sy)

[Key Modules 8](#_heading=h.4iv7t2dne0a)

[Dependencies and Integrations 8](#_heading=)

[Compliance Features Embedded in Architecture 9](#_heading=)

[System Capabilities 9](#_heading=)

[Known Limitations 9](#_heading=)

[**Development Process 10**](#_heading=h.kj77pst1djgu)

[Project Management Approach 10](#_heading=h.gbju4brr4ury)

[Phased Delivery 10](#_heading=h.frqpa0ndprna)

[Sprint Planning and Iteration 11](#_heading=h.oo5y7g4tvxlu)

[Stakeholder Engagement 12](#_heading=h.n0n2fu9o879l)

[Risk and Change Management 12](#_heading=h.7gcq11icj1kv)

[**Data and Information Sources 13**](#_heading=h.6kqktj3rvqsd)

[Datasets used and their structure 13](#_heading=h.cpmc3yy6hg9p)

[Database schema or static data schemas 14](#_heading=h.6ki7tvsc4u5o)

[Legal and ethical considerations for data usage 14](#_heading=h.zdoaei9ro193)

[**6. Codebase Handover 16**](#_heading=h.32uuh18t5w7g)

[Repository structure 16](#_heading=h.iiftrkp5q52q)

[Front-End Application 17](#_heading=h.1g50i2m8vsl0)

[Data Assets 17](#_heading=h.btbbtk7rjhal)

[Web Scraping Subsystem 18](#_heading=h.3gq8noqlo0we)

[Deployment and Development Configuration 18](#_heading=h.8wuddbni6a2)

[Configuration and Setup 19](#_heading=h.yjz2qqhzzu8d)

[Known Issues and Future Improvements 19](#_heading=h.joovmqwjiwhy)

[Codebase Summary 20](#_heading=h.guzk9uvsi0yo)

[**7. Testing and Quality Assurance 20**](#_heading=h.kaidp5i5mbwq)

[Testing methodology 20](#_heading=h.9h7behphzjhf)

[Test Types 20](#_heading=h.pd0mr7s31y1y)

[Test Environment 21](#_heading=h.8bl0qliysxab)

[Test Summary Table 22](#_heading=h.9izv9wsmnjzy)

[Summary 30](#_heading=h.r36ap1jww1f3)

[**8. Risk Management and Compliance 30**](#_heading=h.p2pb2fh77rup)

[**9. Deployment and Maintenance Guide 30**](#_heading=h.hw448cinid9g)

[**Supporting Material 31**](#_heading=h.m7b9vxsnhz8d)

# Project Scope and Context

## Background

Australia’s midlife and retirement systems are complex, involving interlinked regulations across superannuation, taxation, and social security frameworks. Individuals in the 47-70 age group often face critical decisions around superannuation contributions, retirement timing, part-time work, downsizing, and income management. At the same time, the provision of financial advice is tightly regulated, with strict boundaries distinguishing factual information from personalized advice.

This project was engaged to investigate how emerging AI technologies, specifically Large Language Models, could bridge the knowledge gap by providing accessible, compliant, and accurate guidance based on public information sources.

## Original Project Brief

The project was scoped around the following objectives:

* Research and specification development for an AI-powered bot focused on midlife financial and lifestyle planning.
* Integration of public data sources (ATO, Services Australia, ASIC) into a framework that ensures accuracy and compliance.
* Development of a prototype system capable of delivering factual, general information while redirecting users to official calculators, resources, and professional advisers where appropriate.

The expected outcomes were a set of functional prototypes and comprehensive documentation addressing the technical, legal, and stakeholder aspects of the system.

## Scope Boundaries

**Inclusions**:

* Coverage of general factual information in areas such as:
  + Superannuation contributions (concessional/non-concessional, catch-up contributions, salary sacrifice).
  + Transition-to-retirement strategies.
  + Taxation for retirees, including treatment of withdrawals and income streams.
  + Age Pension eligibility requirements, including residency, income, and assets tests.
  + Financial literacy and scam awareness.
  + Midlife lifestyle considerations such as flexible work, travel, and downsizing.
* Implementation of safeguards against provision of personal advice (disclaimers, reliance on authoritative data).
* Research and documentation of legal compliance requirements.
* Delivery of a functioning codebase demonstrating data integration, LLM prompting strategies, and user interaction design.

**Exclusions**:

* Provision of personalised financial advice, financial product recommendations, or modelling of individual financial scenarios.
* Collection or storage of user personal financial data.
* Deployment of the system into a live production environment.

## Alignment with Industry Partner Needs

The industry partner required a solution that demonstrated both technical feasibility and compliance assurance. The project addressed this need by:

* Mapping outputs against compliance frameworks and risk registers.
* Producing artefacts for stakeholder engagement and legal review.
* Delivering a clear set of specifications and working prototypes that can be extended into production systems with appropriate resourcing.

## Context within the Broader Field

This project sits at the intersection of financial literacy, compliance, and artificial intelligence. While many AI chatbots exist in general consumer contexts, their application in regulated financial domains is novel and requires heightened safeguards. The project contributes a practical methodology for balancing AI innovation with regulatory compliance, a challenge that is increasingly relevant across multiple industries beyond finance.

# System Overview

The system developed in this project is a proof-of-concept AI chatbot framework designed to deliver factual, compliance-aligned information on midlife financial and lifestyle planning. Its architecture integrates multiple components: a front-end user interface, a back-end data integration layer, and a compliance-driven response design, into a cohesive demonstration of how an industry-ready system could function.

## High-Level Architecture

The solution is structured into three key layers:

1. **User Interface Layer (Front-End)**
   * Implemented in **React** to provide a modern, intuitive, and responsive interface.
   * Allows users to input natural language queries and view responses in conversational format.
   * Includes design features that maintain accessibility and readability for the target demographic (ages 47–70).
2. **Data Integration and Processing Layer**
   * Provides structured ingestion of information from publicly available, authoritative sources.
   * Utilises a Selenium-based web scraper to extract data from relevant government and regulator websites where APIs or datasets are not directly available.
   * Includes preprocessing utilities to clean, structure, and normalise content for use within the LLM framework.
3. **Language Model Interaction Layer**
   * Interfaces with external Large Language Models (LLMs) to generate responses based on curated prompts.
   * Prompt engineering methods were evaluated and documented to maximise factual accuracy and ensure adherence to compliance boundaries.
   * Implements response templating that embeds disclaimers and references to official sources, reinforcing compliance and trustworthiness.

Together, these components support a pipeline where user queries are processed, matched against trusted data sources, and returned in a conversational, factual, and compliant manner.

## Key Modules

**Front-End Components**

* Landing Page (Landingpage.jsx): Provides an accessible entry point for users, including clear disclaimers about the general nature of information provided.
* Chat Interface (geminiChat.jsx): Handles input/output for conversational exchanges with the LLM, ensuring clarity and user-friendly formatting.
* Styling (App.css): Implements consistent visual design aligned with project goals of readability and simplicity.

**Data Processing Modules**

* Static Data Loader (loadStaticData.js): Supports the use of structured, curated datasets for testing and evaluation.
* Reference Highlighter (referenceHighlighter.js): Adds compliance-supporting functionality by highlighting source references in responses.
* Reference Map (referenceMap.js): Provides mappings between user queries and authoritative source materials, improving transparency and traceability.

**Web Scraping Subsystem**

* Located under the selenium-web-scraper/ directory.
* scraper.py: Automates extraction of data from official websites (e.g., ATO, Services Australia).
* utils.py: Provides helper functions for cleaning and structuring scraped data.
* requirements.txt: Defines dependencies for environment setup and reproducibility.
* Includes a README.md to assist future developers in configuring and running the scraper.

## Dependencies and Integrations

The system makes use of several key technologies:

* React (JavaScript/JSX): Core front-end framework.
* Node.js / npm: Package management and build support for the front-end.
* Selenium (Python): Automated web scraping of government data sources.
* Python Utilities: Data preprocessing scripts supporting structured content integration.
* LLM API Interfaces: External large language models evaluated for performance, accuracy, and compliance suitability.
* Supporting Libraries:
  + Front-end styling frameworks.
  + Python libraries such as selenium, requests, and parsing tools.

## Compliance Features Embedded in Architecture

The system was deliberately designed with compliance at the core, rather than as an afterthought. Key safeguards include:

* Predefined Response Templates: Ensure phrasing aligns with regulatory requirements, using constructions such as “According to the ATO…” or “Here are the rules based on Services Australia guidelines…”.
* Mandatory Disclaimers: Automatically appended to responses, reminding users that information is general and not financial advice.
* Reference Linking: Where possible, responses cite or highlight the official source of information, promoting transparency.
* Content Scope Control: Through curated datasets and prompt controls, the system avoids straying into off-topic or advisory content.

## System Capabilities

* Accepts user queries in natural language.
* Retrieves or generates factual responses from authoritative sources.
* Displays results in a conversational format designed for accessibility.
* Highlights references and embeds disclaimers to ensure compliance.
* Provides pathways for users to explore official resources and calculators.

## Known Limitations

As a proof of concept, the system has several limitations:

* Reliance on third-party LLM APIs introduces constraints on cost, latency, and model behaviour.
* Web scraping requires ongoing maintenance to remain functional as government websites change.
* Not all regulatory nuances can be fully captured.
* The system does not yet fully support multi-turn conversation management at production scale.

Despite these limitations, the architecture provides a scalable foundation. It can be extended with improved data pipelines, model fine-tuning, and production-ready deployment infrastructure in future phases.

# Development Process

The project was delivered through a structured, iterative development process informed by Agile principles. Given the complexity of both the technical and regulatory domains, the team placed strong emphasis on incremental deliverables, continuous stakeholder feedback, and compliance alignment. This ensured that progress was transparent and that both technical feasibility and partner expectations were addressed throughout the lifecycle.

## Project Management Approach

The team adopted an Agile project management framework, with work divided into phases and shorter iterations (sprints). This enabled flexibility in response to evolving requirements, legal considerations, and technical discoveries. Agile ceremonies such as sprint planning, reviews, and retrospectives were adapted for the academic-industry partnership context.

Key practices included:

* **Fortnightly progress reporting** (documented across Semester 1 and Semester 2) to provide visibility of achievements, blockers, and next steps.
* **Risk Register management** to track emerging risks across technical, compliance, and stakeholder domains, with mitigation strategies regularly updated.
* **Team contract and artefact agreements** to establish accountability, clarify responsibilities, and manage expectations.
* **Defined sprint plans** with deliverables tied to both technical development (e.g., code modules, scraper functions) and non-technical research (e.g., legal compliance analysis, stakeholder alignment reports).

This approach balanced the dual requirements of academic rigor and industry applicability, ensuring that both dimensions were addressed within the available timeframe.

## Phased Delivery

The project was executed across two major phases, corresponding to Semester 1 and Semester 2 of delivery.

Phase 1 (Semester 1): Foundations and Research

* Defined the project scope, goals, and compliance boundaries.
* Conducted research into possible technical solutions (including LLMs, retrieval-augmented generation, and non-LLM approaches)
* Developed visual tone and appearance
* Completed artefacts such as:
  + Project Plan v2
  + Initial Sprint Plans
  + Legal Research and Stakeholder Alignment Report
  + Early datasets and schema research
* Delivered fortnightly performance reports documenting incremental progress and challenges.

Phase 2 (Semester 2): Prototyping and Implementation

* Shifted focus from research to implementation of prototypes.
* Built and tested the front-end React components and Selenium scraper.
* Conducted prompt testing and API model comparisons to evaluate performance.
* Updated project and sprint plans to reflect scope adjustments and new insights.
* Delivered artefacts such as:
  + Prompt Testing Report
  + API Model Comparison presentations
  + Updated sprint planning documents
  + Risk Register updates
  + Ongoing progress and performance reports

This phased approach allowed for a research-to-build pipeline, ensuring that technical implementation was grounded in robust contextual understanding.

## Sprint Planning and Iteration

The project utilised time-boxed sprints (typically 2 weeks in length), during which the team focused on defined deliverables. Sprint objectives were drawn from both the technical and compliance streams of work. Examples include:

* Early sprints: Research into compliance requirements, literature reviews of LLM suitability, and alignment with stakeholder expectations.
* Mid-phase sprints: Development of scraper scripts, static dataset schema design, and front-end prototype scaffolding.
* Late-phase sprints: Integration of LLM interactions, compliance-oriented response templating, and end-to-end prototype demonstrations.

Each sprint concluded with internal review sessions and updates to progress documentation. Sprint outputs were validated against the project scope to ensure alignment with the original brief.

## Stakeholder Engagement

Active and consistent engagement with stakeholders was central to the project’s success. The following practices were used:

* Industry Partner Engagement
  + Initial project scoping meetings to confirm alignment with partner expectations.
  + Regular check-ins to provide updates, receive feedback, and confirm compliance priorities.
  + Provision of interim artefacts (e.g., stakeholder alignment report, dataset documentation) for partner review and comment.
* Tutor and Academic Oversight
  + Bi-weekly tutor meetings to ensure academic requirements and milestones were met.
  + Submission of fortnightly progress and performance reports for assessment and feedback.
  + Iterative refinements made based on tutor guidance regarding project management, research depth, and technical implementation.
* Internal Team Collaboration
  + Team contract established roles and expectations early.
  + Shared artefacts (e.g., meeting reviews, sprint plans, and progress logs) to maintain transparency within the team.
  + Adjustments to responsibilities made dynamically in response to workload distribution and skill sets.

## Risk and Change Management

Throughout the project, a Risk Register was actively maintained. Risks tracked included:

* Technical risks (LLM inaccuracies, scraping instability).
* Compliance risks (unintended provision of personal financial advice).
* Project risks (scope creep, misalignment with stakeholder expectations).

Mitigation strategies were documented, including the use of disclaimers, controlled datasets, and phased scope adjustments. Additionally, the project team applied change management practice**s** to manage evolving requirements. For example, while early investigations considered multiple solution pathways (including non-LLM approaches), the team later focused on LLM-based prototypes after evaluating feasibility and stakeholder alignment.

# Data and Information Sources

## Datasets used and their structure

The TERAH system relies on a curated set of authoritative datasets and public information sources. These datasets were selected for their credibility, accessibility, and relevance to midlife financial and retirement planning in Australia.

Australian Bureau of Statistics (ABS)

* *Source*: Retirement and Retirement Intentions (latest release)
* *Content*: Demographics of retirees, retirement age trends, reasons for retirement, income sources, and gender breakdowns.
* *Structure*: Statistical tables, time-series data, and demographic cross-sections.

Super Consumers Australia

* *Source*: Independent consumer group research reports.
* *Content*: Retirement savings benchmarks, methodology for savings targets, case studies on superannuation payouts, consumer sentiment, and fee impacts.
* *Structure*: Report-based datasets and textual analyses, requiring extraction and summarisation.

MoneySmart (ASIC/ATO initiative)

* *Source*: Government-run retirement planning resources.
* *Content*: ASFA Retirement Standard benchmarks, taxation rules for superannuation, calculators, and glossary definitions.
* *Structure*: Text-based guides and structured tools (e.g., calculators).

Australian Government – Services Australia / DSS

* *Source*: Open data from data.gov.au.
* *Content*: Income support recipients (monthly), Age Pension trends, service delivery statistics, and demographic breakdowns of payment recipients.
* *Structure*: CSV datasets, JSON exports, and time-series data.

Australian Securities and Investments Commission (ASIC)

* *Source*: Government open datasets.
* *Content*: Detailed demographic breakdowns of retirement-related payments by statistical area, age, and gender.
* *Structure*: CSV and tabular formats.

Australian Taxation Office (ATO)

* *Source*: Public web guidance pages (scraped).
* *Content*: Transition-to-retirement rules, eligibility for accessing super, taxation of withdrawals, and retirement planning processes.
* *Structure*: Text-based content requiring manual scraping due to anti-bot protections.

Together, these sources form a hybrid knowledge base of structured statistical datasets and curated static text resources.

## Database schema or static data schemas

Two complementary approaches were used to manage and structure data for TERAH:

1. Static Data Schema Model
   * Curated plain text files containing extracts from official sources.
   * Organised into topic-based folders (e.g., Superannuation, Age Pension, Taxation).
   * Minimal metadata embedded (source, publication date, topic).
   * Designed for auditability and compliance, with each entry traceable to its original source.
2. Web Scraping Model
   * Automated extraction of content using Selenium-based scripts.
   * Output stored as cleaned plain text or CSV for ingestion by the LLM.
   * Capable of handling recurring updates from ABS, DSS, and Services Australia datasets.
   * Includes utility scripts (utils.py) for cleaning, parsing, and normalisation.

This hybrid approach balances the control and auditability of static schemas with the currency and scalability of automated scraping.

## Legal and ethical considerations for data usage

Because TERAH operates in a highly regulated domain, legal and ethical safeguards were embedded into data collection and usage:

* Compliance with Financial Advice Laws
  + TERAH is restricted to delivering *factual information* only.
  + No personalised advice, product recommendations, or financial modelling is permitted.
  + Mandatory disclaimers are integrated into all responses.
* Data Source Permissions
  + Only openly accessible, government-published, or industry partner–approved content is ingested.
  + Written approval was secured through the industry partner for scraping official government sources (ATO, Centrelink, MoneySmart).
* Privacy and Security
  + No personal user data is collected, stored, or processed.
  + Only de-identified, aggregate-level statistics and general rules are used.
* Accuracy and Auditability
  + All datasets are linked to authoritative references.
  + Static schemas allow version control and manual auditing.
  + Web scraping pipelines are monitored and reviewed to prevent malformed or outdated content.
* Responsible Scraping
  + Scraping practices respect robots.txt where applicable, minimise server load, and avoid prohibited content extraction.
  + Where anti-bot mechanisms exist (e.g., ATO), manual extraction is used under fair-use principles.

By combining these safeguards with technical controls, the project ensures that TERAH remains both legally compliant and ethically responsible in its use of public information.

# 

# 6. Codebase Handover

The codebase developed for this project serves as a proof-of-concept implementation of an AI chatbot framework designed to deliver factual and compliant information about midlife financial and lifestyle planning. This section provides an overview of the repository structure, key modules, environment setup, and considerations for future maintainers.

## Repository structure

The project’s code is organised into two primary areas:



|  |  |
| --- | --- |
| └── Capstone Project/  ├── .dockerignore  ├── .env  ├── docker-compose.yml  ├── docker-setup.bat  ├── docker-setup.sh  ├── Dockerfile  ├── Dockerfile.dev  ├── DOCKER\_SETUP.md  ├── eslint.config.js  ├── index.html  ├── nginx.conf  ├── package-lock.json  ├── package.json  ├── README.md  ├── vite.config.js  └── src/ | ├── App.css  ├── App.jsx  ├── geminiChat.jsx  ├── Landingpage.jsx  ├── loadStaticData.js  ├── referenceHighlighter.js  ├── referenceMap.js  ├── assets/  │ ├── capstoneicon1.png  │ ├── Copyicon.png  │ ├── Retryicon.png  │ └── Tickicon.png  ├── Data/  │ ├── ABS\_Retirement\_Comparison.xlsx  │ ├── ATO.txt  │ ├── etc  └── selenium-web-scraper/  ├── README.md  ├── requirements.txt  └── src/  ├── scraper.py  └── utils.py |

This structure ensures modularity, separating front-end components, data assets, and scraper utilities, while also including containerisation support for deployment.

## Front-End Application

The front-end application is implemented in React, scaffolded with Vite for efficient builds and hot reloading.

* App.jsx
  + Entry point for the React application.
  + Integrates styling (App.css) and routes to functional components.
* Landingpage.jsx
  + Provides the introductory interface, disclaimers, and guidance for users before interacting with the chatbot.
  + Designed with accessibility and clarity for the target demographic (47–70 age group).
* geminiChat.jsx
  + Core chatbot component where user inputs are captured and responses are displayed.
  + Integrates with the LLM API and compliance-driven response templates.
* Utility Modules
  + loadStaticData.js: Supports ingestion of structured static datasets for prototype testing.
  + referenceHighlighter.js: Enhances responses by highlighting citations to authoritative sources.
  + referenceMap.js: Provides mappings between topics/queries and relevant public resources, ensuring traceability and compliance.
* Assets (/assets/): Contains icons for user interface consistency (copy, retry, tick confirmation, project branding).
* Styling (App.css): Ensures consistent design aligned with project requirements of simplicity and readability.

## Data Assets

The /src/Data/ folder contains curated data sources used for prototype testing and demonstration. Examples include:

* ABS\_Retirement\_Comparison.xlsx: Structured comparison data for retirement statistics.
* ATO.txt: Extracted textual data from the Australian Taxation Office, used for testing static ingestion.
* Other files: Placeholder and supporting data structures for extending dataset coverage.

These assets provide a controlled environment for testing LLM prompting strategies without requiring live data retrieval in every case.

## Web Scraping Subsystem

The selenium-web-scraper/ directory contains a Python-based subsystem for retrieving live data from government and regulator websites where APIs are not directly available.

* README.md: Provides setup and usage instructions for the scraper.
* requirements.txt: Defines Python dependencies (e.g., selenium, parsing libraries).
* scraper.py: Main script automating extraction of structured content from designated sites.
* utils.py: Helper functions for cleaning, transforming, and formatting scraped data for integration into the chatbot pipeline.

This subsystem demonstrates the feasibility of integrating dynamic, up-to-date content into the chatbot while highlighting the need for ongoing maintenance as web structures change.

## Deployment and Development Configuration

The root directory includes files to support containerisation and cross-platform setup:

* Docker Files:
  + Dockerfile and Dockerfile.dev for production and development environments.
  + docker-compose.yml for orchestrating multi-service deployments.
  + docker-setup.sh and docker-setup.bat for simplified local setup on Linux/Mac and Windows.
  + DOCKER\_SETUP.md with detailed instructions for building and running the environment.
* Configuration and Build Files:
  + .env: Environment variable management for API keys and configuration parameters.
  + eslint.config.js: Linting rules to maintain code quality.
  + vite.config.js: Build configuration for Vite.
  + nginx.conf: Configuration for serving the React app in a containerised environment.

This setup ensures the prototype can be deployed consistently across different environments and simplifies future scaling efforts.

## 

## Configuration and Setup

Environment Requirements

* Front-End
  + Node.js (latest LTS version recommended).
  + npm for dependency management.
  + Standard React development environment setup.
* Web Scraper
  + Python 3.9+ recommended.
  + Dependencies specified in requirements.txt.
  + Selenium requires appropriate browser drivers (e.g., ChromeDriver).

Setup Instructions

1. Clone the repository
2. For the front-end:
   1. Navigate to the React project root
   2. Run npm install to install dependencies
   3. Use npm start, then npm run dev to launch the development server
3. For the scraper:
   1. Create and activate a Python virtual environment
   2. Install dependencies using pip install -r requirements.txt
   3. Run python src/scraper.py to execute scraping

## Known Issues and Future Improvements

* Scraper Maintenance: As government websites change, the scraper may require updates. Ongoing monitoring is recommended.
* Error Handling: Current error handling is minimal; production deployment would require robust logging and fallback mechanisms.
* LLM API Integration: The prototype assumes external LLM APIs; future work could involve fine-tuning or hosting models locally to reduce reliance on third-party services.
* Testing Framework: While manual testing was conducted, automated testing (unit and integration tests) should be implemented for production readiness.

## 

## Codebase Summary

The codebase is structured for clarity, modularity, and ease of handover. It provides a solid foundation for further development, enabling future teams or the industry partner to:

* Extend front-end components.
* Enhance data pipelines through additional datasets and APIs.
* Improve compliance safeguards within the chatbot responses.
* Deploy a stable, scalable version of the system in production contexts.

# 7. Testing and Quality Assurance

The purpose of this section is to demonstrate that the system has been thoroughly tested against its defined user stories and acceptance criteria. Testing activities were conducted to validate both functional requirements (the chatbot’s ability to perform intended user interactions) and non-functional requirements (accuracy, accessibility, reliability, and data integrity).

Testing was guided by Agile sprint milestones and executed across both development and staging environments using a combination of manual testing, automated checks, and simulated user interactions.

## Testing methodology

The project followed a user story–driven test approach, with each story mapped to:

* Functional behaviour;
* Expected outcomes (acceptance criteria); and
* Actual test results (pass/fail with observed behaviours).

All tests were executed on local builds using React and Docker environments to ensure parity between development and deployment configurations.

## Test Types

|  |  |
| --- | --- |
| **Test Type** | **Description** |
| **Functional Testing** | Verifies each user story and acceptance criterion. Ensures intended behaviour for all chatbot interactions and UI functions. |
| **Integration Testing** | Confirms smooth operation between React front-end, LLM API, and Selenium scraper components. |
| **Regression Testing** | Conducted after major updates to verify that core chat functionality remained stable. |
| **Usability Testing** | Focused on clarity, accessibility, and visual comfort for the target demographic (47–80 age range). |
| **Data Validation** | Ensures that referenced factual information matches the authoritative sources and that citation links are correct. |

## Test Environment

Testing was conducted under the following controlled configurations:

* **Environment:** Local Docker containers using Dockerfile.dev and docker-compose.yml.
* **Frontend Framework:** React 18 with Vite build pipeline.
* **Browser Platforms:** Chrome (primary), Edge (secondary).
* **Scraping Environment:** Python 3.10 using Selenium WebDriver for automated data retrieval.
* **Hardware:** Development laptops with at least 8GB RAM, ensuring typical end-user performance conditions.

## 

## Test Summary Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Story ID** | **Title** | **User Story** | **Acceptance Criteria** | **Visual Result of test** |
| 1 | User Login Information | As a user I want to enter my name and location before starting so that responses can be personalised. | Given a user is starting a chat, when they enter their name and location, then the responses are personalized using that information |  |
| 2 | Personalized Greeting | As a user I want an immediate personalized greeting using my entered name so the chat feels tailored | Given a user has entered their name, when they start the chat, then the bot greets them immediately using their name |  |
| 3 | Correct Name on Login | As a user who accidentally entered their name incorrectly, I want to be able to exit the chatbot and return to the login screen so that I can re-enter my correct name before starting the conversation | TERAH must allow the user to press the close button (X) in the chatbot window and be taken back to the login screen. On returning, the chatbot should prompt the user again for their name and location, with their previously entered incorrect name automatically appearing in the name field. It is then up to the user to edit this field and correct the spelling before starting the chat |  |
| 4 | Start Chat | As a user, I want to start a conversation with the chatbot so that I can begin asking retirement-related questions | Given a user accesses the chatbot, when they initiate a chat session, then the chatbot is ready to accept and respond to questions. The chatbot opens successfully and displays a ready-to-use input field, allowing the user to type and submit questions | The chatbot opens successfully and displays a ready-to-use input field, allowing the user to type and submit questions. |
| 5 | Persistent Chat While Browsing | As a website visitor, I want to start a conversation with the chatbot, continue scrolling through the website, and then return to the chatbot later without losing my progress, so that I can explore the site freely while still keeping my conversation intact | TERAH must maintain the user’s conversation history while the visitor scrolls through the website, allowing the chatbot to remain visible and active on the page. As the user continues to browse, the chatbot should stay in place so they can return to it at any time, with the conversation still available to continue without interruption |  |
| 6 | Full-Page Conversation View | As a user, I want to view the conversation on a single full-page interface so that I can follow the discussion easily without navigating between screens | Given a user opens the chatbot, when they interact with it, then the entire conversation is displayed in a single full-page interface. The conversation displays entirely on a single full-page interface, with no need for navigation between screens, and all messages are visible and scrollable | The conversation displays entirely on a single full-page interface, with no need for navigation between screens, and all messages are visible and scrollable. |
| 7 | Close Chatbot | As a user, I want to close the chatbot while I’m in a conversation so I can stop the chat and return to the full screen of the main page | Given a user is in a conversation, when they choose to close the chatbot, then the chat is closed and the chatbot returns to an icon. The chatbot is minimized into an icon in the bottom righthand corner | The chatbot is minimized into an icon in the bottom righthand corner |
| 8 | Ignore Blank Input | As a user, I want whitespace only inputs ignored so that accidental empty messages are not sent | Given a user types a message, when the input contains either nothing or just whitespace, then the message is ignored and not sent. No messages are sent if blank | No messages are sent if blank |
| 9 | Clear Input After Send | As a user I want the input cleared after I send so I can type the next question immediately | Given a user sends a message, when the message is submitted, then the input field is cleared for the next question. The message field is empty after a message is sent | The message field is empty after a message is sent |
| 10 | Auto-Scroll and Input Focus | As a user I want the chat to auto-scroll to newest messages and focus the input so I can continue typing without manual scrolling | Given a conversation is ongoing, when a new message is added, then the chat auto-scrolls to the newest message and the input is focused. The chatbot message screen auto scrolls to the new message | The chatbot message screen auto scrolls to the new message |
| 11 | Bot Thinking Indicator | As a user, I want an indicator to represent when the bot is thinking, so that I know the response is generating | Given the bot is generating a reply, when the user is waiting, then an indicator is shown to represent that the bot is thinking |  |
| 12 | Simulated Typing Effect | As a user I want the bot reply to appear gradually so it feels like a live typed response | Given a bot reply is generated, when it appears in the chat, then the reply is displayed gradually to simulate typing. The chatbot responds as though typed in real time | The chatbot responds as though typed in real time |
| 13 | Factual Information Only | As a user seeking retirement information, I do not want to be misled by financial advice, but instead I want to receive only factual information that cannot negatively impact me, so that I can make my own informed decisions without being given personal advice | TERAH must avoid providing personal or tailored financial advice and instead deliver factual, general information sourced from trusted government or reputable sources. All responses should include a disclaimer reminding users that the chatbot does not provide financial advice and that they should consult a licensed financial adviser for decisions specific to their circumstances |  |
| 14 | Updated Information | As a user, I want to be sure that the model is updated with current information so that I can be sure the responses are not out of date | Terah needs to be able to scrape new data from reputable websources on a regular basis to keep its data up to date. Scraper.py is able to accomplish this whenever it is run | [Scraper.py](http://scraper.py) is able to accomplish this whenever it is run |
| 15 | AI Hallucination Mitigation | As a middle aged user who has experience with AI models, I want to know that there are mitigation protocols against AI hallucinations, so that I can be sure responses are true | TERAH must use strategies to ensure that the model is not overloaded with context, a proven strategy to reduce AI hallucinations |  |
| 16 | Double-Check Accuracy | As an older user, I want the information to be double checked for accuracy, so that I can be sure that the responses are factual | TERAH must double check responses for validity, and correct any that do not meet accuracy standards |  |
| 17 | Out-of-Scope Detection | As a user, I want the chatbot to recognise when my query is outside its scope so that I’m not given misleading information | Given a user submits a question beyond the chatbot’s knowledge, when the chatbot cannot provide a definitive answer, then it informs the user that the query is outside its scope. When a user submits a question beyond the chatbot’s knowledge, the chatbot responds with a message indicating that the query is outside its scope and does not provide misleading information | When the user submits a question beyond the chatbot’s knowledge, the chatbot responds with a message indicating that the query is outside its scope and does not provide misleading information. |
| 18 | Submit Retirement Question | As a user, I want to input a retirement-related question so that I can receive relevant information | Given a user is in a chat session, when they submit a retirement-related question, then the chatbot provides a relevant response based on the query. When the user submits a retirement-related question, the chatbot responds with accurate and relevant information based on the query | When the user submits a retirement-related question, the chatbot responds with accurate and relevant information based on the query. |
| 19 | Follow-Up Context Retention | As a user wanting specific examples, I want to ask TERAH “When can I access my super?” and then follow up with “Can you give me an example of how this works for someone born in 1965?” so that I can confirm it retains context and provides accurate, sourced information | TERAH must answer the initial question with a general preservation age explanation, then correctly respond to the follow-up by providing the preservation age for someone born in 1965, including a supporting reference such as ABS |  |
| 20 | Display Source Name | As a retiree using the chatbot, I want to clearly see the name of the source referenced in TERAH’s response without needing to click the link, so that I immediately know where the information is coming from | TERAH must display references in a transparent way, showing the source name directly in the response. Users should be able to identify the source of the information at a glance, without needing to click with the link |  |
| 21 | Clickable Reference Links | As a user reading TERAH’s responses, I want to be able to click on a reference link provided in the chatbot so that I can view the original source website and explore the information that TERAH has drawn from | TERAH must display reference links that are clickable, taking the user directly to the original source website (e.g., Australian Bureau of Statistics, ATO, Services Australia). The user should be able to open and explore the referenced page in their browser to confirm and read more about the information provided |  |
| 22 | Official Resource Links | As a user, I want to receive links to official resources so that I can verify information independently | Given the chatbot provides information, when a response references external data, then links to official resources are included and clickable. All responses referencing external data include clickable links to official sources, which open correctly in a new browser tab | All responses referencing external data include clickable links to official sources, which open correctly in a new browser tab. |
| 23 | Redo Bot Reply | As a user I want to redo a specific bot reply so that I can get an alternative answer to the same prompt | Given a user views a bot reply they are unhappy with, when they select the redo option, then an alternative answer to the same prompt is generated |  |
| 24 | Copy Bot Reply | As a user, I want to copy bot replies to my clipboard and see confirmation so that I can reuse responses | Given a user views a bot reply they are happy with, when they select the copy option, then the reply is copied to the clipboard and a confirmation is shown | “Retirement is leaving the workforce. It involves planning when to access your super and understanding the tax implications. There are options for transitioning to retirement, such as reducing your working hours.” |
| 25 | Hover Controls Display | As a user I want copy and redo controls to appear only when I hover a bot message so the UI stays uncluttered | Given a user hovers over a bot message, when they view the message, then copy and redo controls are displayed only on hover to keep the UI uncluttered |  |
| 26 | Controls Only on Bot Replies | As a user I want copy and redo controls only on bot replies so I don’t see actions for my own messages | Given a user views the chat, when they check a message of their own, then copy and redo controls are not displayed. The user message box remains consistent | The user message box remains consistent |
| 27 | Accessible Chat Window | As an older user in my 80s with poor eyesight, I want to be able to enlarge the chatbot window so that I can clearly see the responses without straining my eyes | TERAH must support an accessible interface that allows the chatbot to be viewed in a larger format, making text easier to read. Users should be able to switch between a compact view and an expanded view, with responses displayed in a way that accommodates low-vision needs |  |
| 28 | Toggle Larger Chat View | As a user I want to be able to toggle a larger chat view so I can read longer replies comfortably | Given a user is using the chatbot, when they toggle the larger chat view, then the chat display expands for easier reading of longer replies |  |
| 29 | Chat Privacy | As a user, I want to be sure that my chats are private to me and not shared with other users | TERAH must keep chats separate between users so that no information is accessible between them |  |

Overall Test Coverage: 100% of user stories executed and validated.

Pass Rate: 100% (29/29 stories met acceptance criteria).

## Summary

Testing confirms that the prototype system meets all defined functional and usability requirements. The chatbot operates as intended, delivering a seamless, accessible, and compliant user experience. The established testing and QA framework can be expanded in future phases to include automated pipelines, external accessibility audits, and continuous data validation against authoritative sources.

# 9. Deployment and Maintenance Guide

## Installation and Setup Instructions

This guide outlines the steps required to install, configure, and run the Terah application. The system has been developed using the React JavaScript framework with supporting Node.js components. It can be deployed locally for testing or hosted on a web server for production.

**Prerequisites**

Before installation, ensure that the following software is installed on the host system:

* **Node.js** (version 18 or later)
* **npm (Node Package Manager)** – included with Node.js
* **Git** (optional, for repository cloning)
* **Modern Web Browser** (Google Chrome, Microsoft Edge, or Firefox)

**Installation Steps**

1. **Obtain the Project Files**
   * Clone the repository or extract the provided source files from the main directory.
2. **Install Dependencies**
   * Open a terminal in the root project directory and run:
     1. npm install
   * This will automatically install all required dependencies as defined in the package.json file.
3. **Environment Configuration**
   * Copy the example environment file if included (e.g., .env.example) and rename it to .env.
   * Update API keys, URLs, and configuration parameters as required by the client deployment environment.
   * For security reasons, avoid committing .env files to public repositories.
4. **Run the Development Server**
   * To launch the application in development mode:
     1. npm run dev
   * The default local address will be:
   * http://localhost:5173/
   * The terminal will automatically provide a URL; open it in a browser to verify that the system has started successfully.
5. **Build for Production**
   * To generate an optimised production build, run:
     1. npm run build
   * The compiled production files will be located in the dist directory.
   * These files can be served using any web server (e.g., Nginx, Apache, or Node-based server).

## System Requirements

|  |  |  |
| --- | --- | --- |
| **Component** | **Minimum Requirement** | **Recommended Specification** |
| **Operating System** | Windows 10 / macOS 12 / Ubuntu 20.04 | Latest stable OS |
| **Processor** | Dual-core CPU | Quad-core CPU or higher |
| **Memory (RAM)** | 4 GB | 8 GB or higher |
| **Storage** | 500 MB free space | 2 GB free space |
| **Node.js Version** | ≥ 18.0.0 | Latest LTS release |
| **Browser Support** | Chrome 100+, Edge 100+, Firefox 100+ | Latest stable versions |

## Running and Monitoring the System

Once deployed, the system can be monitored using standard Node.js and browser-based tools.

**Starting and Stopping the Application**

* **Start (Development):**
  + npm run dev
* **Build and Serve (Production):**
  + npm run build
  + npm run preview
* **Stop Server:**  
  Press Ctrl + C in the terminal window where the process is running.

**Monitoring Logs and Performance**

* **Browser Console:** Inspect using developer tools (Ctrl + Shift + I) for runtime logs and errors.
* **Terminal Logs:** The terminal output provides server status and build progress.
* **Optional Integration:** For production deployment, logging and monitoring can be integrated with tools such as:
  + **PM2** (Node.js process manager) for auto-restart and uptime monitoring.
  + **LogRocket** or **Sentry** for client-side error tracking (if approved by client).

**Testing Post-Deployment**

After deployment, verify that:

* The main UI loads without console errors.
* Data retrieval and rendering occur correctly.
* Key functionalities (user interaction, content updates) behave as expected.