**Software Requirements Specification (SRS)**

**Project:** Australian Dietary Supplements (ADS) – web application  
**Version:** 1.1  
**Date:** 14 September 2025

**1. Introduction**

**1.1 Purpose**

This document outlines the Software Requirements Specification (SRS) for the Australian Dietary Supplements (ADS) project. The purpose is to define the scope, objectives, requirements and constraints of the web application. This SRS will guide the development team and serve as a reference for stakeholders throughout the project lifecycle.

**1.2 Scope**

The ADS application will provide three key functions:

|  |  |  |
| --- | --- | --- |
| **Reference** | **High-level** | **Detailed** |
| 1 | Price comparison | Aggregating dietary/medicinal product data (e.g., vitamins, supplements, over-the-counter medication, etc.) from major Australian retailers to compare prices, ratings and product information. |
| 2 | AI chatbot assistance | Offering an interactive chatbot to answer user queries about products, categories and general wellbeing information. |
| 3 | Wellbeing recommendations | Suggesting general lifestyle and nutrition tips based on selected or entered symptoms (non-medical advice). |

The system will use a MongoDB database to store scraped product data; an API (Node.js/Express) to serve the application; and a front-end interface styled using Materialize CSS.

**1.3 Definitions, acronyms and abbreviations**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| ADS | Australian Dietary Supplements (project name) |
| ETL | Extract, transform and load process |
| UI | User interface |
| API | Application programming interface |
| MVP | Minimum viable product |

**2. Overall description**

**2.1 Product perspective**

The ADS application is a new, standalone web application. It consists of:

* Scraping modules: scripts to extract product and price data from selected retailers.
* Database (MongoDB via Docker): a MongoDB server deployed as a Docker container, with data persisted in Docker volumes and accessed by the web application over an internal Docker network.
* Web API (Express/Node.js): interfaces between database and front-end.
* Front-end (Materialize CSS): user interface for search, browsing, chatbot and recommendations.

**2.2 Product functions**

* Provide searchable list of products with best available price and product details.
* Display historical price trends and comparisons.
* Support symptom-based recommendations.
* Facilitate natural language interaction through chatbot.

**2.3 User characteristics**

* General users: consumers seeking to compare supplement prices or find wellbeing suggestions.
* Admin users: team members maintaining database integrity and scraper reliability.

**2.4 Constraints**

* Timeframe: ~6 weeks development cycle (course project).
* Ethics: only non-medical advice; strong disclaimer that information is general and not a substitute for professional consultation.
* Technology: limited to MongoDB, Node.js/Express, and Materialize/React front-end and Python; mostly within coursework scope.
* Application requires Docker Desktop to run both the web application container and the MongoDB container.
* Persistent storage depends on Docker volumes; deleting the volume will result in data loss unless backups are performed.

**2.5 Assumptions and dependencies**

* Internet is required only for scraping retailer websites; the MongoDB database itself runs locally inside Docker and does not depend on external cloud services.
* MongoDB container is assumed to remain running and accessible to the application container.
* Retailer websites will remain accessible and allow scraping.
* Reliable internet connectivity is assumed for all users.
* Trello board will manage backlog items (<https://trello.com/b/hRBjFHW9/sit725-group-project-australian-diatery-supplements-website>).

**3. Specific requirements**

**3.1 Functional requirements**

|  |  |
| --- | --- |
| **Reference** | **Detail** |
| FR1 | System shall store product details including name, description, rating, current price and history. |
| FR2 | System shall scrape product data daily from at least two Australian retailers. |
| FR3 | System shall provide a search function with filters (brand, category, price, etc.). |
| FR4 | System shall provide a comparison view across retailers. |
| FR5 | System shall provide a symptom selection form and return general wellbeing recommendations. |
| FR6 | Chatbot shall handle product queries, comparisons and wellbeing suggestions. |
| FR7 | All chatbot responses shall include disclaimers clarifying non-medical advice. |
| FR8 | System shall deploy a MongoDB server as a Docker container, with persistent data stored in a Docker volume and accessible by the web application through an internal Docker network. |

**3.2 Non-functional requirements**

|  |  |
| --- | --- |
| **Reference** | **Detail** |
| NFR1 | Performance: API should respond within 300ms for typical queries. |
| NFR2 | Reliability: scraper jobs should succeed ≥ 90% of attempts. |
| NFR3 | Usability: UI must be responsive on desktop and mobile devices. |
| NFR4 | Security: only team/admin accounts can trigger ingestion endpoints. |
| NFR5 | Portability: system shall run as multiple Docker containers (web application and MongoDB), orchestrated using docker-compose. |
| NFR6 | Persistence: MongoDB container data shall be retained using Docker volumes, ensuring durability across restarts. |

**3.3 Use cases**

|  |  |
| --- | --- |
| **Reference** | **Detail** |
| UC1 | User searches for “Vitamin C 1000mg” à system returns list of matching products with prices. |
| UC2 | User selects product à system shows price history chart and retailer links. |
| UC3 | User inputs symptom “tiredness” à system displays wellbeing tips and relevant nutrient categories. |
| UC4 | User opens chatbot and asks “Compare Blackmores and Swisse Vitamin D” à system fetches and displays results. |

**3.4 Data requirements**

* Products stored in MongoDB with retailer references.
* Prices logged with timestamp for historical charts.
* Recommendations stored in a separate collection with symptom-to-nutrient mapping.
* MongoDB container stores data in a Docker volume mounted to /data/db to ensure persistence.
* Backup and restore operations are carried out using mongodump and mongorestore commands inside the MongoDB container.

**3.5 Testing**

The testing process ensures that the system meets both functional and non-functional requirements as specified in above. Testing will be carried out through a combination of manual testing and automated testing approaches to validate correctness, reliability, performance and usability.

**3.5.1 Automated testing**

* Unit tests: to check individual functions, data validation and database schema rules.
* Integration tests: to validate API endpoints, data flow between modules and correct interaction with external services.
* End-to-end tests: to simulate real user flows such as searching, comparing products and interacting with the chatbot.

Additionally, we are adding a new test case to check the performance of the model.

* Performance tests: to measure response times, scalability and system reliability under load.

**3.5.2 Manual testing**

Manual testing will be performed during early development and user acceptance stages.

* Verifying that all features (search, product comparison, chatbot, data scraping) behave as expected.
* Do error handling and input validation.
* Checking usability, responsiveness, accessibility, and visual design.
* Confirming that disclaimers and security restrictions are clearly visible.

**3.5.3 Test cases (examples)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reference** | **Description** | **Precondition** | **Steps** | **Expected result** |
| **TC1** | Search with filters | Products exist with multiple brands and prices | * 1. Go to search page   2. Enter “Vitamin C”   3. Set brand = Swisse, max price = $20   4. Click search | Only Swisse products priced ≤ $20 are displayed |
| **TC2** | Product comparison view | Product available from at least 2 retailers | 1. Select a product 2. Open comparison view 3. Review price list | Prices sorted ascending; cheapest highlighted. |
| **TC3** | Data scraping validation | Scraper has completed a run | 1. Inspect latest dataset 2. Check schema (Name, Price, Retailer, Stock Keeping Number) 3. Verify no duplicate {Stock Keeping Number, Retailer} entries | All records valid; no duplicates |
| **TC4** | Chatbot response with disclaimer | Chatbot service running | 1. Open chatbot 2. Ask: “Compare Swisse vs Blackmores Vitamin C” | Response contains both brands and a disclaimer (“not medical advice”) |
| **TC5** | Security on ingestion API | Admin and non-admin users exist | 1. POST /admin/ingest token 2. Check with admin token | Unauthorised request = 403 Forbidden; Access denied.  Authorised request = 200/202 success - The request was successfully processed |
| **TC6** | Performance check | Server running with populated DB | 1. Send 1000 or more requests to search details. 2. Measure the response time. | Response time < 300 ms |
| **TC7** | Application response | Application running | * 1. Navigate through search → product → comparison → chatbot | Layout adapts to user specification with all above TC’s verified. |
| **TC8** | Database container health | MongoDB container running | * 1. Stop the MongoDB container | Web application /ready endpoint returns 503 Service Unavailable |
| **TC9** | Persistence check | MongoDB container has existing data | * 1. Remove and recreate MongoDB container with the same volume | Data is still accessible after recreation |
| **TC10** | Network isolation | MongoDB container running | 1. Attempt to connect directly to MongoDB from outside Docker without port mapping | Connection is refused, confirming isolation |

**3.6 UI/UX requirements**

The interface shall prioritise clarity, speed and accessibility. A teammate will deliver Figma mock-ups that serve as the single source of truth for visual design and interaction.

**3.6.1 Design**

* The ADS application will follow a cohesive, minimal and health-oriented design. The interface will be styled using Materialize CSS principles with a consistent colour scheme (green/blue accents for health and wellbeing, white background for clarity). Typography will prioritise readability. Accessibility standards (contrast, labels, alt-text) will be observed.

**3.6.2 Core screens and components**

* Homepage: navigation header (Home, Search and Recommendation), a prominent search bar and quick links to major features.
* Search results page: a filter panel (brand, category, price, rating) and a product listing grid with price, rating and comparison options.
* Product detail page: detailed product information, price history chart and retailer comparison table.
* Recommendation page: a symptom input form with generated wellbeing suggestions and disclaimers.
* Chatbot window: a dedicated interaction panel where users can type queries. Each response will contain a disclaimer.

**3.6.3 Interaction**

* Navigation: a top navbar for quick access to all sections.
* Search: dynamic filters and instant results displayed in real time.
* Responsiveness: the application will adapt seamlessly between desktop, tablet and mobile.
* Accessibility: keyboard navigation and screen-reader support.
* Chatbot window: a dedicated interaction panel where users can type queries. Each response will contain a disclaimer.

**4. Supporting information**

**4.1 References**

* Chemist Warehouse, Priceline and other Australian retail pharmacy websites.
* Deakin University SIT725 coursework guidelines.
* Team’s Trello board for backlog and sprint planning (<https://trello.com/b/hRBjFHW9/sit725-group-project-australian-diatery-supplements-website>).
  1. **Contributions**

Version 1.0 is a vision; a work in progress; and will continuously be updated throughout the project.

With regard to contributions towards Version 1.0:

* Jacki Ngau and Minh Khiem Pham: 1; 2; and 3.1 – 3.4
* Christo Raju: 3.5 – 3.5.3
* Gia Khanh Ngo: 3.6 – 3.6.3