**Wessel Boshoff**

Credit submission

Contents

[Intro 3](#_Toc207640471)

[SoftwareFundamentalsToolkit 3](#_Toc207640472)

[AlgorithmDesign 3](#_Toc207640473)

[ChatBot 3](#_Toc207640474)

[RecursiveFunctions 4](#_Toc207640475)

[Sort 4](#_Toc207640476)

[DataStructures 5](#_Toc207640477)

[OnlineRetailDatabase 5](#_Toc207640478)

[Html5JsCss3Demo 6](#_Toc207640479)

[Site 6](#_Toc207640480)

[NetworkingSimulator 7](#_Toc207640481)

[CSharpFundamentals 8](#_Toc207640482)

[AdoptionPortal 8](#_Toc207640483)

[OOPProject 9](#_Toc207640484)

[WebAppPortal 10](#_Toc207640485)

[Api 10](#_Toc207640486)

[Site 12](#_Toc207640487)

# Intro

I’m submitting this portfolio to show the knowledge and practical skills I’ve built up in software development. The work included here highlights my experience in designing, building, and connecting different kinds of systems — from APIs and web applications to socket programming, WPF applications, databases, algorithms, and aspects of information security.

The document is meant as a guide for the reviewer, explaining the purpose and layout of each part of the portfolio. Each section includes a short description, the main features, the technologies I used, and how it links to the relevant UNISA modules. Taken together, these projects reflect both the theory I’ve learned and the hands-on problem-solving I’ve done, which is why I believe they are a solid basis for this credit application.

**Repo is located at:** [**https://github.com/Wessel-Boshoff/MyDevPortal.git**](https://github.com/Wessel-Boshoff/MyDevPortal.git)

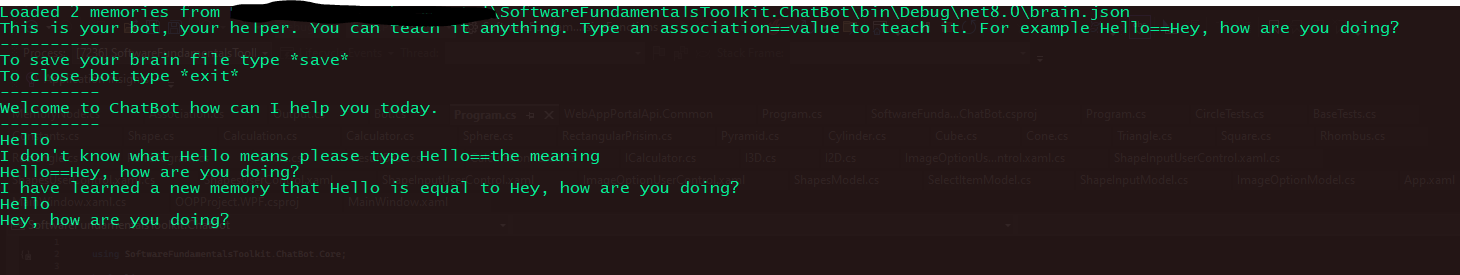
# SoftwareFundamentalsToolkit

## AlgorithmDesign

### ChatBot

In this project, I built a console-based chatbot in C#. The design makes use of object-oriented principles, events, delegates, and custom models (such as MemoryNode and Association) to simulate basic learning and memory. The chatbot can retain information by saving its knowledge in a JSON file, which demonstrates practical use of file I/O, serialization, and state management.

To manage and strengthen the stored associations, I applied searching, sorting, and string processing techniques. The program also follows good design practices by keeping the core logic (the Bot) separate from the output layer (Output), ensuring clear separation of concerns.

This project highlights my ability to apply algorithms, data structures, OOP concepts, and event-driven programming in a practical setting.

**Key features:**

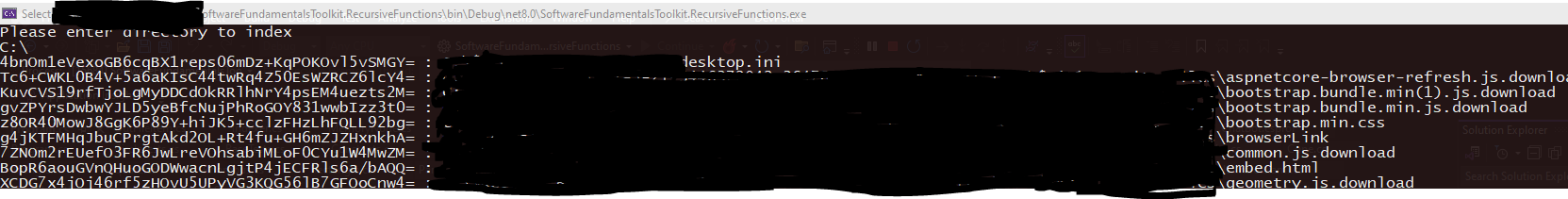
* Basic programming (loops, conditions, string handling).
* Object-oriented coding (classes, events, methods).
* Working with lists and simple algorithms (search, sort, ranking).
* File saving and loading (JSON).
* A simple AI-style memory system.

**Relevant modules:**

* COS1511 – Introduction to Programming I
* COS1512 – Introduction to Programming II
* COS2611 – Data Structures and Algorithms
* COS3711 – Artificial Intelligence (basic knowledge systems)

This was super fun to do. And this concept can be infinitely expanded!!!

### RecursiveFunctions

In this project, I built a program that scans a folder and all of its subfolders, identifying every file it contains. For each file, the program generates a SHA256 hash, effectively creating a unique digital fingerprint. The results are placed in a thread-safe queue, allowing different parts of the program to run safely in parallel without interfering with one another. I also implemented custom events and delegates to display progress messages in real time whenever files are discovered. The tool can be applied to practical scenarios such as indexing large directories, detecting duplicate files, or verifying file integrity.

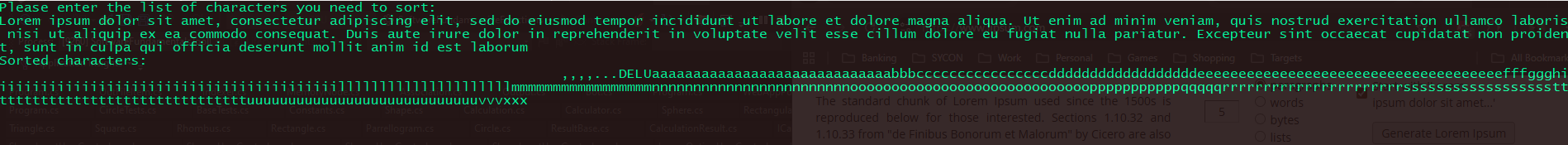
**Key features:**

* File system operations – reading folders and files.
* Recursion – exploring subfolders automatically.
* Events and delegates – triggering actions when a file is found.
* SHA256 hashing – generating unique fingerprints for files.
* ConcurrentQueue & Tasks – safe and fast processing using multiple threads.
* Error handling – handling missing files or permission errors.

**Relevant modules:**

* COS1511 – Introduction to Programming I (basic I/O and control structures)
* COS1512 – Introduction to Programming II (OOP, events, delegates)
* COS2611 – Data Structures and Algorithms (recursion, queues)
* COS2661 – Programming: Concurrency (thread-safe collections, async tasks)
* COS3711 – Computer Security (cryptographic hashing with SHA256)

### Sort

In this project, I created a console-based C# application that takes a string of characters entered by the user, converts it into a list, and then sorts the characters using the Bubble Sort algorithm. The program demonstrates core programming concepts such as implementing sorting algorithms, working with control flow through loops and conditionals, managing data structures like arrays and lists, and handling user input and output effectively.

**Key features:**

* C# .NET Console Application
* Collections (List, Array)
* Basic Algorithm Design

**Relevant modules:**

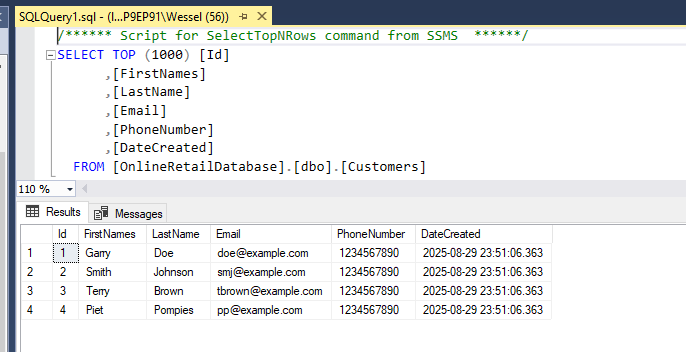
• COS1511 – Introduction to Programming I (basic I/O and control structures)

• COS1512 – Introduction to Programming II (arrays, lists, custom algorithms)

• COS2611 – Data Structures and Algorithms (sorting algorithms, efficiency considerations)

## DataStructures

### OnlineRetailDatabase

This project defines and seeds an Online Retail Database using Microsoft SQL Server. It demonstrates relational database design by creating normalized tables (Customers, Orders, Products, Payments, etc.), applying constraints, relationships, and referential integrity. It further includes stored procedures for CRUD operations, reporting views (e.g., top customers), triggers for automatic stock updates, and merge scripts to populate lookup/reference data such as Payment Statuses and Order Status

**Key features:**

* Microsoft SQL Server (T-SQL)
* Tables, Constraints, and Relationships (PK, FK, Identity, Checks)
* Stored Procedures for CRUD operations
* Views for reporting
* Triggers for business rules (stock management)
* MERGE for seeding and synchronizing data

**Relevant modules:**

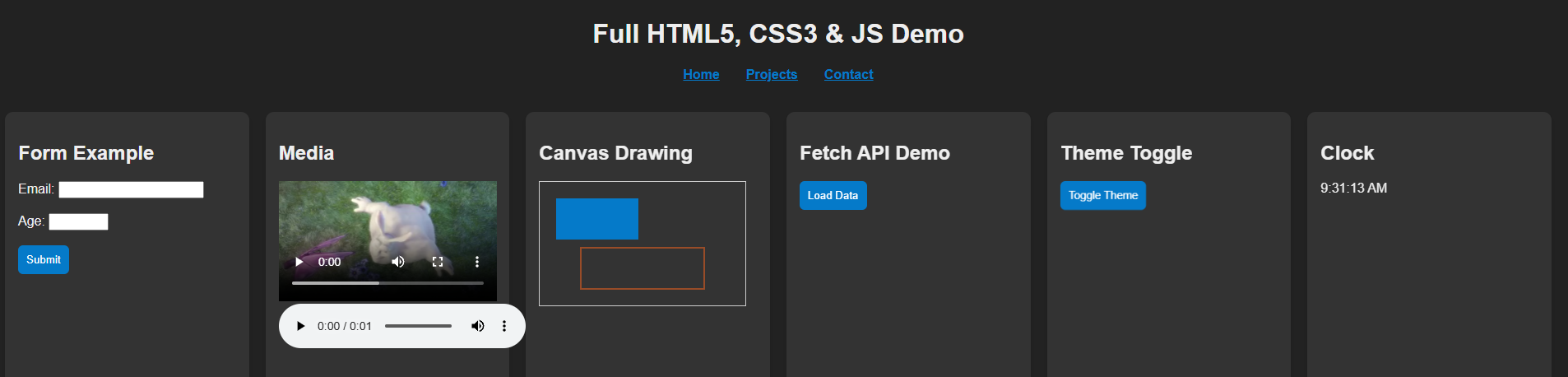
* ICT3715 – Database Systems (relational database design, SQL programming)
* COS1511 – Introduction to Programming I (basic logical thinking, I/O adapted to SQL context)
* COS1512 – Introduction to Programming II (procedural programming in stored procedures)
* COS2614 – Software Engineering (designing structured, maintainable database solutions)
* COS3711 – Computer Security (data integrity constraints, safe transaction handling)

# Html5JsCss3Demo

## Site

This project demonstrates a complete front-end application using HTML5, CSS3, and JavaScript. It includes:

* Responsive layout & theming (CSS variables, media queries, dark/light theme toggle)
* Form handling & validation (email, numeric input, feedback messages)
* Multimedia integration (video and audio embedding)
* Canvas API (basic 2D graphics rendering)
* Asynchronous JavaScript (AJAX/Fetch API) to load and display data from a remote API
* Dynamic DOM manipulation with event listeners and real-time updates (live clock, data list)
* Animations using CSS transitions and keyframes



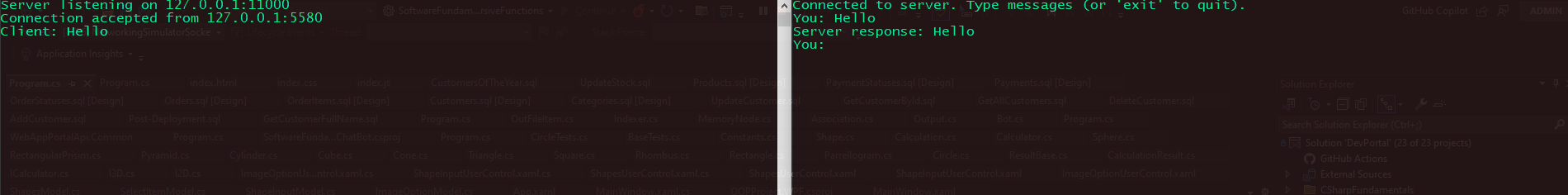
**Key features:**

* HTML5 (semantic tags, forms, media elements, canvas)
* CSS3 (variables, flex/grid layout, transitions, animations, responsive design)
* JavaScript (ES6+) (event listeners, fetch API, async/await, DOM manipulation, localStorage)

**Relevant modules:**

* COS1511 – Introduction to Programming I (basic input/output, event handling)
* COS1512 – Introduction to Programming II (object-oriented concepts applied in JS, DOM events)
* COS2614 – Software Engineering (design of structured, modular web applications)
* COS3712 – Web Programming (HTML5, CSS3, client-side JavaScript, Fetch API, local storage)

# NetworkingSimulator

The project demonstrates network programming fundamentals with a basic TCP client–server chat system. It covers asynchronous communication, socket APIs, concurrent handling of connections, and I/O streams.

**Key features:**

* Socket Programming using System.Net.Sockets (TCP protocol)
* Asynchronous networking (ConnectAsync, SendAsync, ReceiveAsync) for non-blocking communication
* Client–server architecture (separate client and server programs)
* String encoding/decoding with UTF8 for message handling
* Graceful connection management (shutdown, disconnect handling)

**Technologies Used:**

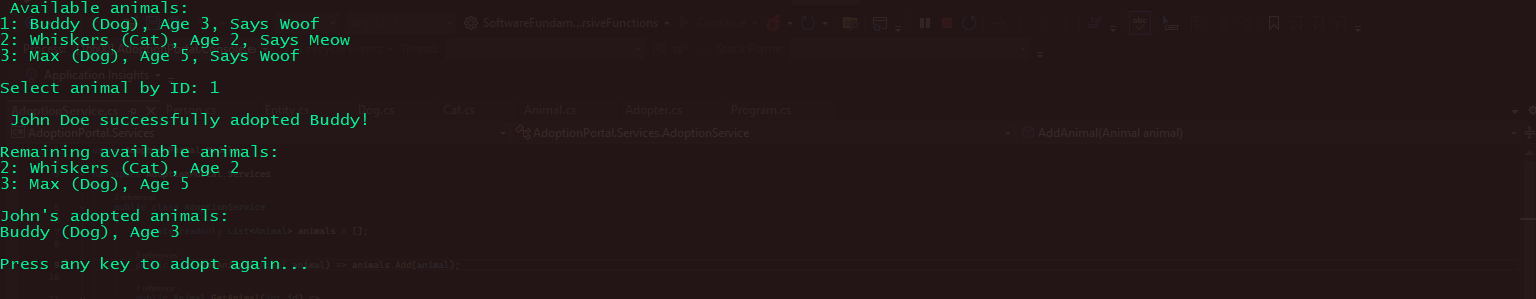
* C# / .NET 8
* System.Net & System.Net.Sockets namespaces
* Asynchronous programming (async/await)
* TCP/IP Networking fundamentals

**Relevant modules:**

* COS1511 – Introduction to Programming I (console input/output, program structure)
* COS1512 – Introduction to Programming II (event-driven programming, async handling)
* COS2661 – Programming: Concurrency (async/await, parallel handling of communication)
* COS3721 – Computer Networks (socket programming, TCP/IP communication)
* COS2614 – Software Engineering (modular design, separation of client & server components)

# CSharpFundamentals

## AdoptionPortal

This project is a simple adoption portal that lets users browse through a list of animals, choose one to adopt, and keep track of their adopted pets. It’s built with a strong focus on object-oriented programming, with a clear structure that separates the data models, the service layer handling the adoption logic, and the overall system flow.

**Key features:**

* Object-Oriented Design:
  + Base classes (Entity, Person, Animal) and derived classes (Dog, Cat, Adopter)
  + Inheritance (Animal → Dog/Cat, Person → Adopter)
  + Polymorphism (Speak() overridden in Dog and Cat)
  + Encapsulation (adoption state managed via methods, not direct property setting)
* Service Layer (AdoptionService) handles storage, filtering, and adoption logic
* Interactive Console UI for user input and feedback
* Collections & LINQ (List<>, Where, Any, FirstOrDefault, Single) for filtering available animals
* State management for tracking adopted vs available animals

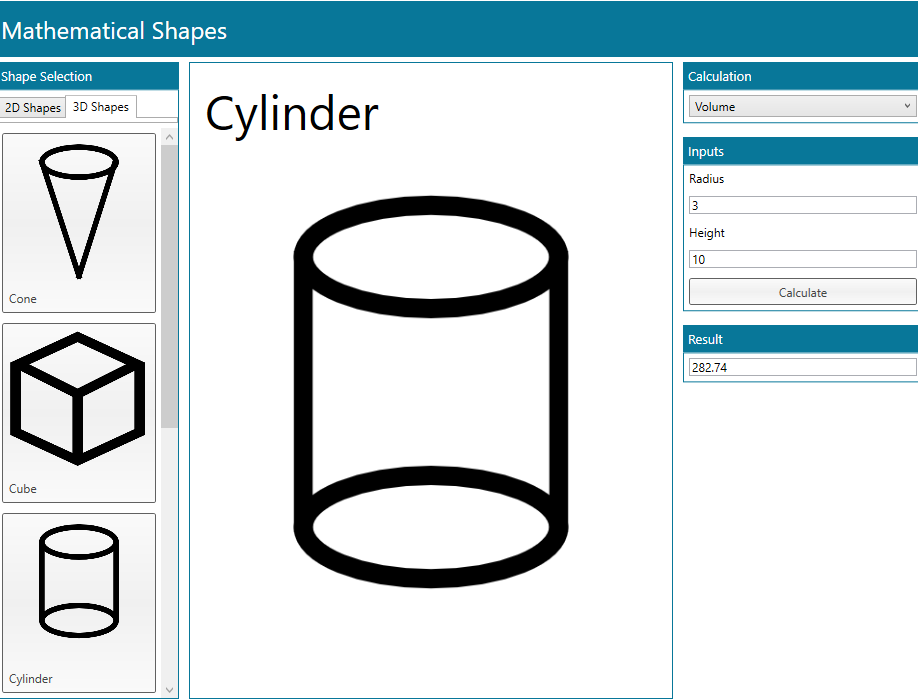
**Technologies Used:**

* C# / .NET 8
* OOP principles (encapsulation, inheritance, polymorphism)
* Collections & LINQ queries
* Console-based UI

**Relevant modules:**

* COS1511 – Introduction to Programming I (basic program flow, input/output)
* COS1512 – Introduction to Programming II (object-oriented programming: inheritance, polymorphism, encapsulation)
* COS2611 – Data Structures and Algorithms (use of collections like lists, filtering, and LINQ queries)
* COS2614 – Software Engineering (service-oriented design, separation of concerns between models, services, and UI)

## OOPProject

This project demonstrates an object-oriented geometry calculator capable of handling both 2D and 3D shapes. It allows users to calculate areas, circumferences, and volumes depending on the selected shape. The design applies interfaces (I2D, I3D), polymorphism, and a centralized calculation service to ensure flexibility and maintainability. The solution is implemented as a WPF desktop application, where users can select a shape, provide input values, and view results instantly, with validation and error handling included. Unit tests and performance benchmarks were also developed to confirm correctness and reliability.

**Key Features:**

* OOP Design:
* I2D and I3D interfaces define contracts for 2D and 3D shapes.
* Shape classes (Circle, Rectangle, Cube, Cylinder, Cone, etc.) implement these contracts.
* Calculator class uses dynamic polymorphism and pattern matching (switch expression) to perform calculations.
* Error Handling: Graceful handling of runtime errors (division by zero, invalid parameters, unsupported shapes).
* Extensibility: Adding a new shape only requires implementing its class and updating the Shape enum.
* Constants & Metadata: A Constants dictionary maps shapes + calculations to expected inputs (e.g., Circle → Radius).
* Testing: Unit tests (MSTest) validate correctness and performance across many iterations.
* WPF GUI:
* Models: (ShapesModel, ImageOptionModel, ShapeInputModel, etc.) encapsulate UI state.
* User Controls: (ImageOptionUserControl, ShapeInputUserControl, ShapesUserControl) make the UI modular.
* Interactive selection of shapes, dynamic loading of required inputs, and real-time result display.

**Technologies Used**:

* C# / .NET 8
* WPF (Windows Presentation Foundation) for GUI
* OOP Principles: abstraction, inheritance, polymorphism, encapsulation
* MSTest Framework for unit testing
* LINQ & Collections for managing input sets and formulas
* Dynamic typing (dynamic) for flexible calculation delegation

**Relevant Modules:**

* COS1511 – Introduction to Programming I (basic flow, operators, methods)
* COS1512 – Introduction to Programming II (OOP: interfaces, inheritance, polymorphism)
* COS2611 – Data Structures and Algorithms (use of collections, LINQ, efficient data mapping via dictionaries)
* COS2614 – Software Engineering (modular design, separation of concerns between core logic, services, UI, and tests)
* COS3711 – Programming: Graphical User Interfaces (WPF, event-driven programming, MVVM-style modeling)
* COS3721 – Advanced Software Development (error handling, testing, performance optimization, extensibility)

# WebAppPortal

## Api

This project is a .NET Core 8.0 Web API designed as a portal for managing users, products, and system logs. It implements database seeding, authentication, and data persistence, providing a fully functional backend for applications requiring user management, product cataloging, and logging capabilities.

The solution demonstrates full-stack backend development skills, including API design, Entity Framework Core integration, seeding and migrations, and secure user management



**Key Features**

* User Management
  + CRUD operations on users.
  + Roles and registration status support using enums.
  + Passwords stored securely with hashing and salts.
  + Tracking of creation date and last sign-in timestamp.
* Product Management
  + Products associated with specific users.
  + Support for storing images in binary format.
  + Metadata storage (name, description, file extension, creation timestamp).
* Logging
* Request logging with details: URL, HTTP method, request/response bodies, status codes, headers, and timestamps.
* Event logging with log levels, exception tracking, and stack traces.
* Database & Seeding
* Uses SQL Server as the backend database.
* Automatic database migration on startup.
* Pre-populates the database with sample users and products during development.
* API Features
* RESTful endpoints for dashboard summary retrieval.
* Mediator pattern for request handling (via MediatR).
* JWT or cookie-based authentication integration (authorization attributes applied).
* Development Convenience
* Development-time seeding for testing and demonstration.
* Enum to string conversion in EF Core for cleaner database storage.

**Technologies Used**

* Backend & API
* .NET 8.0 / C#
* ASP.NET Core Web API
* MediatR (CQRS/Mediator pattern)
* Database
  + Microsoft SQL Server
  + Entity Framework Core (EF Core)
  + EF Core Migrations & Seeding
* Security & Identity
* Role-based authorization
* Password hashing with salt
* Logging
* Request and event logging for auditing purposes
* Development Tools
* Visual Studio
* NuGet packages

**Relevant Modules**

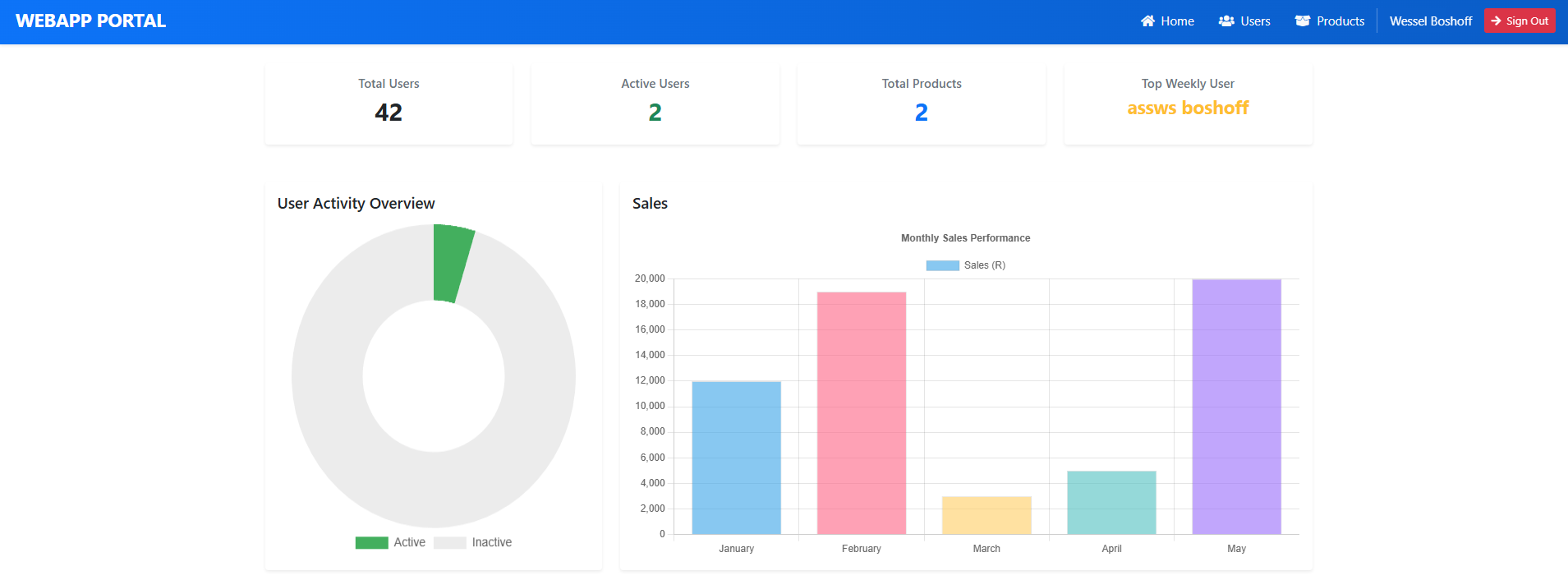
* COS2601 / COS2605 – Software Development & Design Principles
* COS3601 / COS3605 – Database Systems / Database Design
* COS2610 / COS3621 – Web Development and API Integration
* COS2700 – Programming in C# / .NET Core
* COS3608 – Security in Software Systems (Password hashing, role-based access)
* COS3701 – Application Integration and Middleware Patterns (Mediator pattern)
* COS3622 – Advanced Data Handling & Logging

## Site

The WebApp Portal Site is a secure ASP.NET Core MVC web application that acts as the frontend for the WebAppPortal API.

It provides an administrative dashboard, user management, and product management features, supported by role-based authentication and session handling.

The site is integrated with the API but remains a standalone web application that focuses on user experience, data visualization, and secure access control.



**Key Features**

* Authentication & Authorization
  + Cookie-based auth integrated with JWT from the API.
  + Role-based access control (Admin, Root) with authorization policies.
  + Secure cookie/session handling and auto-logout middleware.
* Session Management
  + Custom PrimarySession storage using ISession.
  + Middleware to check session validity and redirect to login if expired.
* Dashboards
  + Home page dashboard with key stats
    - Total Users
    - Active Users
    - Total Products
    - Top Weekly User
  + Interactive charts using Chart.js for user activity and sales trends.
* User Management
  + Create, edit, delete users.
  + Assign roles via dropdown.
  + Confirmation modal for deletions.
  + Login, Register, Set Password pages with validation.
* Product Management
  + Add, edit, delete products with images (Base64 storage).
  + Product thumbnails rendered in tables.
  + Modal confirmation for deletion.
* Logging
  + Standardized error logging with extension methods.
* UI & UX
  + Responsive layout with Bootstrap 5.
  + FontAwesome icons.
  + Razor Components (User Greeting, NavBar with role-based visibility).
  + Modern card-based design and clean error handling pages.

**Technologies Used**

* ASP.NET Core MVC (Razor Views)
* Authentication & Security
  + ASP.NET Identity cookies
  + JWT token integration
  + Role-based authorization policies
  + Secure session management
* Custom session-check, request logging, error handling
* UI/Styling: Bootstrap 5, FontAwesome, custom CSS
* Charts & Visuals: Chart.js
* Data Binding: Strongly typed ViewModels, extension mapping between API models and site models.

**Relevant Modules**

* COS2614 – Programming (C#)
* COS3711 – Software Engineering
* COS3712 – Web Application Development
* INF3707 – Systems Analysis & Design
* COS3721 – Information Security
* COS3811 – Databases (linked via API)