Student Name	Jonathan Kadiri	**Student Number**	22364026
Supervisor	Keith Maycock	**ECTS Credits**	
Project Title	VeriCV		

1) Project Objectives

VeriCV is a secure digital professional directory platform functioning as a "phone book" of verified professionals. Users create digitally signed CVs with cryptographically verified credentials from issuing institutions. Once verified, professionals are added to a searchable public directory, enabling employers to discover candidates with authenticated qualifications.

The system addresses resume fraud through cryptographic verification whilst creating a trusted marketplace for talent discovery. Unlike traditional platforms relying on self-reported information, VeriCV implements verification-gated directory access where only users with verified credentials appear publicly. Employers can search by degree, institution, skills, or experience, with each profile including digital signatures proving credential authenticity.

Expected deliverables include a functional web application with authentication, CV management with digital signatures, automated credential verification, a public searchable directory with filtering capabilities, a binary verification status system, RESTful API backend (Spring Boot), responsive React frontend, comprehensive API documentation (Swagger), and Docker containerisation.

2) Description of work completed

Over the initial project phase, significant foundational work was completed. A comprehensive Spring Boot 3.2 architecture was designed with PostgreSQL 15 for persistence. The authentication system was fully implemented with JWT token-based security. Development tools including Docker, Maven, and Git were configured. The project approach was revised to incorporate a "phone book" directory model, recognising that core value extends beyond individual CV verification to creating a trusted ecosystem of verified professionals.

2.1) Evidence of work completed

A complete Spring Boot backend structure was established following standard Java package conventions with clear separation across configuration, controller, service, repository, model, security, and DTO layers. PostgreSQL 15 was configured using Docker Compose. The database schema was designed with eight entity models: User, CV, Education, Experience, VerificationRequest, Institution, Company, and CVAccessPermission.

A comprehensive authentication system was implemented featuring user registration with BCrypt password hashing, JWT token generation using JJWT 0.12.3, Spring Security configuration with stateless session management, custom

UserDetailsService, role-based access control, and 24-hour token expiration. Key implementations include User entity with roles and timestamps, UserRepository with custom queries, JwtTokenProvider for HMAC-SHA256 token handling, CustomUserDetailsService for Spring Security integration, SecurityConfig for filter chain configuration, AuthService for business logic, and AuthController providing /api/auth/register and /api/auth/login endpoints.

Swagger/OpenAPI 2.3.0 was integrated for interactive API documentation accessible at http://localhost:8080/swagger-ui.html. The development environment was configured with Java 17, Maven 3.9+, Docker Desktop, VS Code with Java extensions, and Git version control. Comprehensive documentation was created including README, setup guides, implementation checklists, and Git workflow documentation.

Several technical challenges were encountered and resolved including Maven POM XML corruption, package declaration errors with incorrect namespace prefixes, Docker connectivity issues, YAML syntax errors from CRLF line endings, and Maven dependency resolution for JWT libraries. The Git repository was successfully initialised with multiple commits documenting progress, culminating in successful code push to GitHub at https://github.com/jonathankadiri0/VeriCV.

2.2) Literature review

Research was conducted into RSA cryptographic signature schemes for document verification, identifying Bouncy Castle as the implementation framework for SHA256withRSA signatures. JWT standards (RFC 7519) were reviewed for stateless authentication implementation. Spring Security 6.x documentation informed authentication provider and filter chain configuration.

JPA/Hibernate best practices were researched for entity modelling, particularly bidirectional relationships, cascade operations, and fetch strategies to optimise queries and prevent N+1 problems. RESTful API design principles were studied, including HTTP method usage, status codes, and resource naming. OpenAI 3.0 specification guided endpoint documentation.

Existing CV platforms (LinkedIn and Indeed) were analysed. None implement cryptographic verification, representing a market gap that VeriCV addresses through its verified directory approach.

2.3) Use of GenAI and tools

Claude AI was used primarily for debugging assistance and resolving technical challenges during development. Three key use cases demonstrate its effectiveness:

Complex Debugging: When encountering "duplicate class" compilation errors, Claude systematically diagnosed incorrect package declarations (package main.java.com.vericv instead of package com.vericv), providing both explanation of the error mechanism and solution approaches including automated sed commands for batch correction.

Configuration Troubleshooting: When YAML configuration files caused application startup failures, Claude identified CRLF line ending issues and provided solutions for converting to LF format. Similarly, Maven POM XML corruption was diagnosed and resolved through Claude's analysis of error messages.

Dependency Resolution: Maven dependency conflicts for JWT and validation libraries were resolved through Claude's suggestions for compatible version combinations and proper dependency scope configuration.

Additional tools included GitHub Copilot for code completion, VS Code IntelliSense for syntax checking, Maven for dependency management, and Docker Compose for database provisioning. GenAI tools were employed judiciously, primarily to accelerate problem resolution when encountering unfamiliar error messages or configuration issues, whilst core implementation remained manual to ensure deep understanding of the codebase and architecture.

3) Future Work

Core System Development: CV management requires completion including CRUD operations, REST endpoints, and DTOs. Education and experience entities need full implementation with controllers and validation logic. Digital signature implementation involves RSA key pair generation, CV content hashing, signature generation using Bouncy Castle, and verification endpoints.

Verification System: A comprehensive verification workflow needs development for requesting credential verification from institutions. This includes email-based invitations, verification tokens with expiration, a verifier portal interface for institutions to confirm or reject credentials, and automatic directory eligibility updates upon successful verification. Institution integration requires organisation profiles, verification contact management, email domain verification, and trust relationship establishment. Users will have a binary verification status (verified/unverified) determining directory inclusion, with individual credentials displaying verification status from their respective issuing institutions.

Phone Book Directory System: A separate Directory Entry entity must store publicly searchable profiles with automatic creation upon verification. Search functionality requires full-text search across names, qualifications, and skills, filtering by industry, location, and experience, ranking algorithms favouring verified profiles, and autocomplete suggestions. Users need granular visibility controls including directory toggle, credential visibility settings, and contact information management. Employer features include saved searches with alerts, candidate bookmarking, direct contact requests, and verification code checking.

Frontend Development: React frontend requires authentication flows, CV builder interface, public directory search with filters, profile pages with verification indicators, directory settings panels, and responsive Tailwind CSS design. Public-facing pages include homepage, search results, individual profiles, and employer landing pages.

Testing and Deployment: Comprehensive testing includes unit tests for services, integration tests for APIs, and security testing for authentication. Directory search requires load testing, query performance benchmarking (target <200ms), and anti-scraping measures. Production deployment involves Docker containerisation, CI/CD pipelines, and potential Elasticsearch integration for search scalability. User documentation must cover CV management, directory optimisation, and employer search guides.

Current Challenges: VS Code Java Language Server occasionally exhibits synchronisation issues though builds succeed. The directory model introduces search performance complexity requiring careful indexing and potential Elasticsearch integration. Balancing privacy controls with discoverability presents UX challenges. Additional directory functionality increases scope, necessitating feature prioritisation for MVP delivery.

The project follows a phased approach with initial PostgreSQL full-text search implementation, reserving Elasticsearch as post-MVP enhancement if performance demands. Regular checkpoints against timeline, consistent Git commits, and comprehensive documentation support risk mitigation.

Spring Framework Documentation. "Spring Boot Reference Documentation." Version 3.2.0. https://docs.spring.io/spring-boot/docs/3.2.0/reference/html/

JSON Web Token (JWT) Specification. RFC 7519. Internet Engineering Task Force, 2015.

Bouncy Castle. "The Legion of the Bouncy Castle Java Cryptographic APIs." Version 1.78. https://www.bouncycastle.org/java.html

PostgreSQL Global Development Group. "PostgreSQL 15 Documentation." https://www.postgresql.org/docs/15/