My ePortfolio showcases my academic accomplishments and honor rolls from high-quality outcomes in the SNHU Computer Science program.   
This course emphasizes professional communication with coworkers and stakeholders as well as coding and debugging. In my ePortfolio, I intentionally write code reviews and narratives for each artifact upgrade. In the workplace, I will need to communicate key features to stakeholders like users or clients to incentivize the features I can develop, critically review a code base without bias, and create robust arguments for areas that need improvement without appearing hostile or deconstructive. The fundamental goal is to develop the most effective, safe, and efficient software I can, which entails tackling severe backlog issues.   
A well-rounded developer needs hard abilities like critical thinking and problem solving in addition to soft talents. Data structures, algorithms, software engineering, databases, and security are important hard skills for software development. My studies have taught me that data structures involve knowing and using primitive and nonprimitive data types, when to utilize them, and how to format them for a product's language and development environment. These data structures are linked by algorithms, which execute data-manipulating operations that must be organized to be understood and debugged. Software engineering combines these two approaches to develop fully realized applications for a specific audience. A program can generate, read, update, or remove data in a database, which persists even after the software is off. Finally and most significantly, security locks down all the previous principles so only certain users or tools can access certain data or functions. This takes a constant effort to study and apply industry standards, how to implement or maintain them in a code base, and what security features to prioritize based on the program and conditions. Consider constant threats from external and internal malevolent actors. (Secure Development, 2017) A software application that uses Internet of Things and a SQL database must protect sensitive company or user data from online attacks like SQL injection.   
My ePortfolio combines the soft and hard skills I've learned at SNHU and is shown via the artifacts I've chosen and improved. My ePortfolio is a coherent, technically sound, and audience- and context-appropriate demonstration of my skills.   
  
  
  
The first artifact is a CS-360 project I produced. databases for inventory and user credentials. The app aims to streamline procedures. I chose this artifact after careful consideration since it shows my front-end and back-end programming skills. I want to improve this product.   
I carefully developed and built the front-end user interface, creating user-friendly and attractive layouts. These designs made inventory management easy. My work on the primary backend component ensured the program's reliability. This contained class hierarchies, CRUD operations, and important algorithms. I improved this category's documentation, commentary, design, and structure. I implemented exception handling to reduce errors. I also improved application performance by reducing functions, eliminating redundancies, and removing obsolete variables and functions. However, I kept certain commented-out sections for future usage. I initially felt an alternative artifact may meet course goals. Despite this, I chose the Inventory App for software design and engineering, algorithms and data structures, and databases. This was because it met all course objectives and showed my software development progress. The project has taught me the value of thorough organization and documentation. This opportunity increased my awareness of the need of functionality monitoring, structural refinement, and incentive to improve code and application design.   
  
  
My CS-320 project is the second artifact. This project created a Java-based Contact class and repository class for in-memory contact repository and service CRUD operations. Unit tests for Contact.java and ContactService.java were created. Using Java HashMap to create an in-memory data store simplifies contact management. It also provides several unit tests to assess data input and expected functionality, improving artifact quality. This application was written with Eclipse and Java JDK 18. JUnit Test version 5 validates boundary conditions, functionality, and data input. The Entity class creates and operates Contact objects using data encapsulation, constants, and robust methods. The repository class used HashMap, a fast Java clection structure for insertion, retrieval, and deletion.   
  
We chose two Java classes and associated JUnit tests to check their functionality. The program's lack of a text-based or graphical user interface indicates user interaction issues. The product followed Java standards, naming conventions, and secure coding best practices. The application is created in alignment with Software Design and Development frameworks based on the requirement analysis performed on the issue description and requirements supplied throughout the training module. The analysis is converted into a Java POJO class and collection or repository class that manages contacts in memory.   
  
Through its development and enhancement, this software development and quality assurance application shows CS320 course skills. Automated unit tests verify application accuracy and functionality. Dynamic application testing improves robustness, accuracy, performance, efficiency, and quality assurance of the developing application. Unit testing and quality assurance can give software applications many competitive advantages, including improved efficiency and operational performance, lower maintenance costs, consistent performance, improved brand reputation, fewer bugs, and regulatory compliance. Unit testing validates functionality and boundary conditions to ensure a software produces the expected outcomes given a dataset.   
  
  
I developed the third artifact in CS-340. An application-specific document-oriented MongoDB NoSQL database will be created. This database will use a shelter's animal data. Students learn how to plan, design, and create client/server applications in CS340. A selected artifact is used to construct a NoSQL database using the supplied dataset. Simple and Complex indexes are proposed for this database application upgrade. This upgrade improves database search capabilities. MongoDB database systems use indexes to speed up data selection and projection queries. Indexes help execute select queries efficiently, allowing queries to retrieve and filter data, improving performance.   
  
  
Queries are usually rendered ineffective when indexes are not available, which is especially true when dealing with datasets that are sufficiently large. Indexes significantly reduce query time, improving database performance. MongoDB generates output results using brute force, which is inefficient and time-consuming. The approach requires checking each database page without indexes. MongoDB indexes restrict documents connected with an index. Runtime and database performance improve as temporal complexity decreases. This allows for the building of effective, dependable, and real-time systems without lag. In MongoDB, field values are saved in indexes in the sequence indicated by the simple or complicated index. Simple and sophisticated indexes hold values. Ordered indexes filter information quickly and accurately based on query criteria. They output queries faster because they are made up of smaller datasets from the indexes rather than the entire database. The following graphic shows a 10,000-record database. Each paper offers information on a Wildlife Shelter animal. Four indexes improved database query execution and search.   
  
  
Backend databases are of the largest relevance in client/server application settings since frontend programs are heavily dependent on them. Thus, backend database performance and efficiency are crucial. The AnimalShelter database will effectively optimize and accelerate query performance. This will improve user experience and speed up client/server apps. This goal will be achieved via indexes. Indexes provide compact lookup tables, eliminating the need to search whole databases. Thus, they accelerate and refine datasets for frontend rendering, visualization preparation, and other dataset and query result presentation, giving them an edge over simple databases. This makes them outperform basic databases.   
  
  
This Computer Science Capstone course uses optimal database design methods to create and upgrade the AnimalShelter database. This course aims for better database design and implementation. Non-relational databases are versatile and scalable. MongoDB is a document-oriented database, meaning each document represents a record. The data model must be carefully planned before constructing a NoSQL database. A dataset was submitted for the project. Each CSV record imported into the database is converted into its corresponding document. This is the Animal Record. After transforming the basic data into Animal documents, indexes are developed to maximize query processing and database speed. These methods can help create an effective and successful backend database system for client-server applications, RESTful APIs, websites, and other database-driven applications. Through this exposure and experience, I may improve my ePortfolio and competence, allowing me to leverage NoSQL databases for many applications.