**CSE 310 – Applied Programming**

# W01-Prove: Create Course Plan

Name: MATHEUS FELBER

Date: 5/8/2025

1. After looking at the Module Description document in I-Learn, select three modules that you want to complete during this course. You cannot repeat a module. You will be responsible for learning the material and creating software to demonstrate what you learned. You can change your mind later in the course based on your experiences. You should not select a module that you have already used before. Mark only one “X” in each of the three columns.

**Note: You will have less time to complete module #1 than the other modules. Consider selecting a module you feel you can learn quickly for module #1.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Modules** | **Module**  **#1** | **Module**  **#2** | **Module**  **#3** |
| Cloud Databases |  |  |  |
| Data Analysis |  |  |  |
| Game Framework |  |  |  |
| GIS Mapping |  |  |  |
| Mobile App |  |  |  |
| Networking |  |  |  |
| SQL Relational Databases |  |  |  |
| Web Apps |  |  |  |
| Language – C++ |  | ✅ |  |
| Language – Java |  |  |  |
| Language – Kotlin |  |  |  |
| Language – R |  |  |  |
| Language – Erlang |  |  |  |
| Language - JavaScript |  |  |  |
| Language – C# | ✅ |  |  |
| Language - TypeScript |  |  |  |
| Language – Rust |  |  | ✅ |

1. Complete the following learning skill survey regarding your learning habits:

|  |  |
| --- | --- |
| **Learning Skill** | **Ranking**  **1 = I don’t do this**  **2 = I do this sometimes**  **3 = I do this frequently** |
| **RESEARCH** |  |
| **I use multiple sources of information.** | 3 |
| **I document what I am learning.** | 1 |
| **I prepare well-thought-out questions.** | 1 |
| **I use the scientific method (research, hypothesis, experiment, and conclusion) to solve computing problems.** | 3 |
| **I share what I am learning with others (i.e. collaboration).** | 2 |
| **TIME MANAGEMENT** |  |
| **I manage my time between all my responsibilities.** | 2 |
| **I effectively estimate task duration for assignments and projects based on my previous performance.** | 2 |
| **I create a schedule for all assignments and projects.** | 1 |
| **RISK MANAGEMENT** |  |
| **I always start with identifying what I do not know.** | 3 |
| **I effectively identify what could fail based on my previous experiences.** | 3 |
| **I create mitigation plans for risks related to missing knowledge or potential failures.** | 1 |
| **CONTINUOUS IMPROVEMENT** |  |
| **I honestly identify the mistakes I have made in my work.** | 3 |
| **I develop improvement plans to support future assignments and projects.** | 3 |
| **I am driven by a vision of whom I can become by the creation of goals.** | 3 |

1. Based on your responses in the survey above, write a plan below to improve one behavior starting at the beginning of this course.

**CSE 310 Development Plan**

**Module 1: C# - Audio Player**

**Task List:**

1. **Research Phase:**
   * Review C# audio libraries: NAudio and System.Media.
   * Read official documentation and tutorials for NAudio.
   * Explore simple WinForms or WPF UI examples.
2. **Setup Project Structure:**
   * Create a new C# project in Visual Studio.
   * Set up folders for audio files, source code, and documentation.
3. **File Selection and Playback Controls:**
   * Implement file selection dialog to load WAV/MP3 files.
   * Integrate play, pause, and stop buttons.
   * Test playback functionality with sample audio files.
4. **Audio Playback Logic:**
   * Implement audio playback using NAudio.
   * Add error handling for unsupported file formats.
   * Implement a playback progress bar.
5. **User Interface Design:**
   * Design a simple UI with buttons, labels, and progress bar.
   * Ensure responsive design for various window sizes.
6. **Testing and Debugging:**
   * Test with multiple audio files (WAV, MP3).
   * Debug and resolve any playback or UI issues.
7. **Documentation and Video Demo:**
   * Write README.md with project overview and usage instructions.
   * Record video demonstrating the application in action.

**Objective:**  
Create a simple audio player that can load and play WAV/MP3 files with basic playback controls (play, pause, stop) and a simple user interface. This project will introduce key C# concepts including event-driven programming, file handling, and basic UI design.

**Why This Module?**

* C# is highly effective for building desktop applications with interactive user interfaces.
* This project is small enough to complete within the limited time frame but offers a comprehensive introduction to C#'s capabilities.

**Expected Learning Outcomes:**

* Implement basic file I/O operations in C#.
* Create a functional user interface using WinForms or WPF.
* Handle audio playback using NAudio or System.Media.

**Development Steps:**

1. Research and review C# audio libraries (NAudio, System.Media).
2. Implement file selection and playback controls.
3. Integrate audio playback and control logic.
4. Test with multiple audio formats.
5. Create a simple, user-friendly UI.
6. Document the project and record the video demonstration.

**Module 2: C++ - Audio Synthesizer**

**Task List:**

1. **Research Phase:**
   * Study JUCE and PortAudio libraries for C++ audio processing.
   * Review basic digital signal processing (DSP) concepts.
2. **Setup Project Structure:**
   * Create a new C++ project with a clear folder structure.
   * Set up CMake or Makefile for compilation.
3. **Waveform Generation:**
   * Implement sine, square, and sawtooth wave generators.
   * Allow user to input frequency and amplitude values.
4. **Command-Line Interface:**
   * Develop CLI for user interaction (input frequency, amplitude, waveform type).
   * Implement input validation and error handling.
5. **Real-Time Audio Processing:**
   * Integrate PortAudio for audio output.
   * Implement multi-threading for continuous playback.
6. **Testing and Audio Output:**
   * Test waveform generation and playback.
   * Adjust parameters and observe audio changes.
7. **Documentation and Video Demo:**
   * Write README.md with usage instructions and code structure.
   * Record a video demonstrating waveform generation and playback.

**Objective:**  
Develop a simple audio synthesizer that generates basic waveforms (sine, square, sawtooth) and allows the user to adjust frequency and amplitude through a command-line interface.

**Why This Module?**

* C++ provides the low-level control necessary for precise audio processing and real-time waveform generation.
* The project is challenging but manageable within the two-week period.

**Expected Learning Outcomes:**

* Implement audio signal generation using JUCE or PortAudio.
* Apply basic DSP techniques to generate and manipulate audio waveforms.
* Integrate multi-threading to handle real-time audio processing.

**Development Steps:**

1. Research C++ audio libraries (JUCE, PortAudio).
2. Develop basic waveform generation (sine, square, sawtooth).
3. Implement frequency and amplitude controls.
4. Add simple command-line interface for user interaction.
5. Implement multi-threaded audio processing for real-time playback.
6. Document the project and record the video demonstration.

**Module 3: Rust - Simple Blockchain Ledger**

**Task List:**

1. **Research Phase:**
   * Study basic blockchain concepts (blocks, transactions, mining).
   * Review Rust libraries: serde for serialization, sha2 for hashing.
   * Understand simple Proof-of-Work (PoW) mechanisms.
2. **Setup Project Structure:**
   * Create a new Rust project using Cargo.
   * Organize folders for source code, data files, and documentation.
3. **Blockchain Data Structure:**
   * Implement basic structs for Block and Transaction.
   * Include fields for timestamp, data, previous hash, and current hash.
4. **Hashing and Block Linking:**
   * Calculate SHA-256 hashes for each block using sha2.
   * Link blocks by referencing the previous block’s hash.
5. **Proof of Work Implementation:**
   * Implement a basic PoW algorithm with adjustable difficulty.
   * Allow mining to confirm and add blocks to the chain.
6. **Transaction Management:**
   * Add CLI commands to create transactions and add them to a block.
   * Display pending transactions and confirmed transactions.
7. **Testing and Debugging:**
   * Test block creation, mining, and transaction processing.
   * Handle invalid data inputs and missing blocks.
8. **Documentation and Video Demo:**
   * Write README.md with project overview and command usage.
   * Record video demonstrating transaction creation, mining, and block linking.

**Objective:**  
Implement a simple blockchain ledger in Rust that allows users to create transactions, mine blocks, and verify the chain integrity using a basic proof-of-work mechanism.

**Why This Module?**

* Provides a foundational understanding of blockchain concepts such as hashing, block linking, and proof of work.
* Covers essential Rust concepts including data structures, serialization, and command-line interfaces.

**Expected Learning Outcomes:**

* Implement data structures for blocks and transactions.
* Calculate and verify block hashes.
* Apply basic proof-of-work for mining.
* Integrate CLI commands for transaction management and mining.

**Task List:**

1. **Research Phase:**
   * Study Rust networking libraries: tokio, pnet.
   * Review basic networking and packet analysis concepts.
2. **Setup Project Structure:**
   * Create a new Rust project using Cargo.
   * Organize folders for source code, data logs, and documentation.
3. **Packet Capture Implementation:**
   * Implement packet capture using pnet.
   * Parse HTTP, TCP, and UDP headers.
4. **Real-Time Data Analysis:**
   * Display packet data in real-time using ASCII charts.
   * Log packet data to a local file (e.g., CSV or JSON).
5. **Command-Line Interface:**
   * Develop CLI for starting/stopping packet capture.
   * Add options for filtering specific protocols (HTTP, TCP, UDP).
6. **Testing and Debugging:**
   * Test packet capture with various network activities (web browsing, file download).
   * Debug and handle unexpected packet structures.
7. **Documentation and Video Demo:**
   * Write README.md with project overview and command usage.
   * Record video demonstrating packet capture and data analysis.

**Objective:**  
Create a network traffic analyzer that captures and logs packets, categorizes traffic by protocol (HTTP, TCP, UDP), and displays traffic statistics in real-time.

**Why This Module?**

* Rust is well-suited for networking and concurrency, providing safe and efficient data handling.
* This project covers key Rust concepts like async programming, error handling, and packet parsing.

**Expected Learning Outcomes:**

* Implement asynchronous data processing with Tokio.
* Capture and analyze network packets using pnet.
* Display real-time data using a command-line interface or simple visualization.

**Development Steps:**

1. Research Rust networking libraries (tokio, pnet).
2. Implement packet capture and parsing logic.
3. Develop CLI for viewing real-time traffic statistics.
4. Categorize and log traffic by protocol.
5. Display traffic analysis in real-time using ASCII charts.
6. Document the project and record the video demonstration.