Network Analyzer

This document outlines the Software Development Lifecycle (SDLC), the technical stack, and the instrumentation used for the project. This plan serves as the foundation for development and documentation.

Project Overview

This project aims to develop a software-controlled network analyzer using an oscillator, an oscilloscope, and a computer. The analyzer will automate frequency sweeps, measure phase and amplitude response, and generate a Bode plot. The system will integrate Python for signal generation, a two-channel oscilloscope for measurement, and a web-based UI for data visualization. This project will work with IEEE 488.2 (USB-TMC) for USB-based instrument control and apply signal processing and data visualization.

Key Features

- Includes the IEEE 488.2 standard, which is an enhancement of IEEE 488.1 that standardizes instrument communication
  - It mandates SCPI (Standard Commands for Programmable Instruments) and standardized data formatting
  - The oscilloscope used, the TDS 1002B, does not have a GPIB interface, but it is IEEE 488.2 compliant via USB-TMC using SCPI commands
  - SCPI over USB-TMC provides structured communication, error handling, and automation, making it equivalent to traditional GPIB control
- Automated signal generation using Python (SciPy + Sounddevice)
- Automated frequency sweeps (e.g., 20 Hz – 300 kHz)
- Oscilloscope measurements to capture phase and amplitude response
- Supports real-time signal generation and USB-based signal output for oscilloscope integration
- Real-time data visualization using Chart.js in Angular
- Bode Plot (Gain & Phase vs. Frequency)
- Backend API in Node.js to interface with Python and store measurements
- OAuth-based authentication for user access
- API for backend communication
  - Potentially using RESTful API structure
- Kubernetes integration for CI/CD and deployment
- Deployment on AWS, Firebase, or a self-hosted server

- In-accordance with industry standards
    - Follows Agile principles (Sprint-based development, continuous testing)
    - Incorporates CI/CD with Kubernetes, aligning with DevOps best practices
    - Uses GitHub for version control, daily logs, and backlog tracking

Instruments & Tools

Hardware:

- Computer with Visual Studio (development environment)
- TDS 1002B, Two-channel oscilloscope (for measuring phase & amplitude)
- USB-based signal generation (compatible with USB-TMC or external DAC for digital-to-analog conversion)

Software & Frameworks:

- Python (SciPy + Sounddevice) → Signal generation
- Node.js + Express.js → Backend API
- Angular + Chart.js → UI visualization
- JSON → Data storage
- OAuth → Secure authentication
- Kubernetes → CI/CD and deployment

Hosting & Deployment Options:

- AWS EC2 or Firebase (alternative to App Store deployment)
- Kubernetes for container orchestration

Software Development Lifecycle (SDLC)

NOTE: ALL DELIVERABLES WILL BE DOCUMENTED

<u>Phase 1: Design (Sprint 1)</u>

Key Tasks

- Define system architecture:
    - Backend: Node.js with RESTful API to communicate with Python
    - Frontend: Angular with Chart.js for Bode plot visualization
    - Database: JSON for simple data storage
- Define hardware integration plan (oscilloscope, Python-based signal generation using USB-TMC or an external USB-based function generator
- Plan the authentication flow with OAuth
- Plan Kubernetes integration for CI/CD
- API Specifications

- RESTful API structure will be used for backend communication unless performance bottlenecks arise
    - Endpoints for controlling the signal generator and retrieving oscilloscope data will follow RESTful conventions
    - If WebSockets are needed for real-time data streaming, they will supplement REST API
- GitHub Version Control and Documentation
    - Set up GitHub repository
    - Create an initial README.md with project overview
    - Document system architecture (backend, frontend, API structure)
    - Repository structure:

```
/software-network-analyzer
├── backend/           # Node.js backend
├── frontend/          # Angular frontend
├── scripts/           # Python signal generation
├── docs/              # Documentation
├── test/              # Test scripts
├── .github/workflows/ # GitHub Actions for CI/CD
├── README.md          # Overview of the project
├── LICENSE            # Open-source license
├── .gitignore         # Ignore unnecessary files
```

Deliverables

- System architecture diagram (backend, frontend, hardware interactions)
- API specifications
- Security design (OAuth implementation)
- Hardware-software interaction plan

Phase 2: Implementation (Sprint 2-4)

Key Tasks

- Backend Development (Node.js, Python)
    - Develop Node.js REST API to:
        - Control Python-based signal generator
        - Interface with the oscilloscope to collect measurement data
        - Store measurements in JSON format
    - Implement OAuth authentication
    - C++ will be used in cases where low-level performance, hardware communication, or optimization is required
    - Backend logic requiring computational efficiency will use C++ integrated into the Node.js API using C++ addons (node-gyp)
    - Python Signal Generation

        - Use SciPy + Sounddevice to create real-time signals
        - Implement waveform selection (sine, square, triangle)
        - Python will generate signals and output them via USB-TMC direct control-based signal generation to send signals to the oscilloscope
            - Automated Frequency Sweeps in Python
        - Implement frequency stepping from 20 Hz to 300 kHz
            - Python API Integration with Node.js
        - Use Flask or FastAPI as a Python bridge for Node.js API
    - SCPI Command Execution
        - SCPI Automation
            - Use PyVISA to send SCPI commands to TDS 1002B via USB-TMC
        - Error Handling Implementation
            - Implement IEEE 488.2-compliant error handling using SCPI
            - Ensure API correctly formats SCPI responses into JSON
- Frontend Development (Angular + Chart.js)
    - Build Angular dashboard:
        - Frequency control UI (user inputs frequency, amplitude, waveform)
        - Real-time oscilloscope data display
        - Chart.js Bode Plot Visualization
- GitHub Version Control and Documentation
    - Maintain API documentation as backend endpoints are developed
    - Add code comments and inline documentation in Node.js, Python, and C++
    - Document SCPI command integration and oscilloscope control

Deliverables

- API documentation (if using REST)
- Code structure & dependencies
- OAuth authentication setup
- Backend & frontend communication flow

Phase 3: Testing (Sprint 5)

Key Tasks

- Unit testing:
    - Verify Python-generated signals are correct when transmitted over USB and received by the oscilloscope
    - Validate oscilloscope readings

    - Ensure API endpoints return accurate data
- Integration testing:
    - Ensure frontend can fetch data from backend
    - Verify real-time updates in Chart.js
    - Lissajous patterns will be used to verify phase difference measurements between input and output signals
    - TDS 1002B supports X-Y mode for Lissajous patterns, which can be used as a secondary verification method for phase shift calculations
- Security testing:
    - Check for OAuth authentication vulnerabilities
- GitHub Version Control and Documentation
    - Write test cases & expected results in `/docs/testing.md`
    - Keep a bug tracking log in GitHub Issues

Deliverables

- Test cases & expected results
- Bug tracking log
- Performance benchmarks

<u>Phase 4: Debugging & Troubleshooting</u>

Key Tasks

- Fix API integration issues
- Resolve frontend visualization errors
- Optimize performance for real-time updates
- GitHub Version Control and Documentation
    - Update error logs & resolutions in `/docs/debugging.md`
    - Document performance optimizations

Deliverables

- Error logs & resolutions
- Debugging tools used
- Optimizations implemented

<u>Phase 5: Deployment (Kubernetes + CI/CD)</u>

Key Tasks

- Hardware and Software configuration
    - Ensure correct USB signal output configuration, whether through USB-TMC for direct oscilloscope communication or an external USB-based signal generator
- Deploy the backend on AWS, Firebase, or self-hosted server
- Use Kubernetes for:

    - Scalability (managing multiple containers)
    - Automated deployment (CI/CD pipeline)
- Set up OAuth authentication for multi-user access
- GitHub for Version Control
    - Finalize full documentation in GitHub repo
    - Ensure README.md explains setup, API usage, and deployment steps
    - Make sure all tests, logs, and optimization notes are accessible

Deliverables

- Deployment strategy
- Hosting setup
- Kubernetes configuration