System Design and Implementation

Railway Administration and Information Logical System

RAILS for Model Railroads

David Bristow
Version 1.1.0
March 18, 2025

Contents

Chapter 1

Introduction

Railway Administration and Information Logical System (RAILS) is a software model and implementation of an automated system to assist the model railroader achieve realism in the operation of a model railroad. The model then drives the development of hardware and or software. The system provides, but not limited to the following functions:

- planing operating session resulting from the needs of industries to move products from one location to another
- simulating rolling stock failures
- controlling trains
- automating regular scheduled trains
- displaying the rail network and location of trains.

In addition, it provides:

- model railroad asset tracking and management by organizing the development of assets into projects
- rolling stock inventory
- reports

The system interacts with several key electronic subsystems to control the physical model railroad environment.

1.1 Original Approach

The original approach that was taken was use a standard software development methodology, starting with the development of requirements through the use of use-cases. The initial client/server design pattern was based on using:

- Java Enterprise Edition (JEE), which is a collection of Java technologies and standards for building enterprise applications. JEE applications are typically monolithic, meaning that all of the components of the application are packaged together and deployed as a single unit. They

can be developed using various Java frameworks such as Spring, JavaServer Faces (JSF), and Enterprise Java Bean (EJB), and they are typically deployed to an application server such as Apache Tomcat or Oracle WebLogic.

- Relational Database (RDB), which is a type of database that stores data in the form of related tables. These tables have rows and columns, and each row represents a single record, while each column represents a specific field of data. The tables are related to each other through the use of common fields, known as keys, which are used to link the data in different tables together. This allows for efficient querying and data manipulation. Examples of relational databases include MySQL, SQL Server, DB2, and Oracle.

The result would be a web application that would provide multiple users access to the developed system.

That approach was undertaken almost two decades ago and based on expertise at hand. Since technology evolved and more options were made available the focus switched from Java to JavaScript and from a relational database to non-relational database. The use cases were still relevant as they speak to the user interactions with the system.

1.2 Approach Implemented

The revised client/server design pattern that is being implemented is based on using (each of which is dicussed in the 1.2.2 Technologies subsection):

- Node.js see subsubsection 1.2.2.3
- Vue.js see subsubsection 1.2.2.5
- MongoDB see subsubsection 1.2.2.4

1.2.1 Architecture

Moving from a monolithic JEE application to a distributed set of components requires a different architecture design pattern. One such pattern is microservices architecture. The microservices architectural style is an evolution of the Services Oriented Architecture (SOA) architectural style.

It is a method of developing software applications in which a large application is built as a collection of small, independent services. These services communicate with each other through APIs, and each service can be developed, deployed, and scaled independently of the others. Microservices are often deployed in containerized environments and use technologies such as Kubernetes to manage and orchestrate the services. Microservices offers several advantages over monolithic architectures such as:

- Scalability: Each service can be scaled independently, so it's easy to add more resources to a specific service that's experiencing heavy load.
- Resilience: Because each service is isolated, a failure in one service will not bring down the entire application.

- Flexibility: Services can be written in different languages, which makes it easy to use the best tool for the job.
- Ease of Deployment: Since each service can be deployed independently, the deployment process can be simplified.

1.2.2 Technologies

In designing and implementing RAILS several technologies have been added to microservices architectural style that compliment it. They are:

1.2.2.1 Docker

In the context of microservices, containers are used to package the code, libraries, and configuration files needed for a microservice to run as a single, executable unit. This makes it easy to deploy and run microservices in different environments, such as on-premises servers or in the cloud. By packaging all the dependencies for a microservice in a container, it can be guaranteed to run the same way regardless of the environment it's deployed in. This helps to reduce issues caused by differences between development, testing, and production environments. Containers also make it easy to scale up or down the number of instances of a microservice running, and to update or rollback a microservice without affecting other services.

Docker is a platform that makes it easy to create, deploy, and run applications in containers. It provides a command-line interface (CLI) and a set of Application Program Interfaces (APIs) that make it simple to work with containers. Docker in conjunction with microservices is used to package and deploy each service in its own container. This allows each service to be managed, deployed, and scaled independently of other services. The Docker platform provides a number of tools to help manage and orchestrate the containers that make up a microservice-based application, such as Docker Compose and Docker Swarm.

Docker is an open-source platform that enables developers to automate the deployment, scaling, and management of applications using containerization. Containers are lightweight, isolated environments that encapsulate an application and all its dependencies, including libraries, frameworks, and other runtime components. Docker allows the user to package an application into a container image, which can then be run consistently across different environments, such as development, testing, and production.

Docker provides the following features:

- Docker containers are portable and can run on any system that supports Docker, regardless of the underlying operating system or infrastructure. This eliminates the "works on my machine" problem and ensures consistent behavior across different environments.
- Containers provide a high level of isolation, ensuring that applications and their dependencies are encapsulated and do not interfere with each other. This isolation improves security, as well as prevents conflicts between different software components.

- Docker containers are lightweight and share the host system's operating system kernel. This means they require fewer resources compared to traditional virtual machines (VMs). Multiple containers can run on the same host without significant performance overhead.
- Docker makes it easy to scale applications horizontally by running multiple instances of containers across different hosts or a cluster of machines. This enables efficient utilization of resources and helps handle increased workload demands.
- Docker allows versioning of container images, making it easier to track changes and roll back to a previous version if needed. This simplifies the deployment and update process, reducing the risk of application downtime.
- Docker has a large and active community, resulting in a vast ecosystem of pre-built container images available from Docker Hub and other registries. These images can be easily pulled and used as a base for building and deploying applications, saving development time.

For additional information on Docker see the official website.

1.2.2.2 Git & GitHub

Git is a distributed version control system (VCS) used for tracking changes in source code during software development. It was created by Linus Torvalds, the creator of the Linux operating system, in 2005. Git allows multiple developers to collaborate on a project by providing a mechanism to manage and merge their code changes efficiently.

GitHub is a web-based hosting service and collaboration platform for Git repositories. It provides a centralized location where developers can store, manage, and collaborate on their Git repositories. GitHub offers a variety of features and tools that enhance the functionality of Git and facilitate collaboration among developers and teams.

Microsoft acquired GitHub in June 2018 in a deal worth $7.5 billion. Following the acquisition, GitHub continues to operate as an independent platform, maintaining its brand, services, and user base while also benefiting from Microsoft's resources and support.

GitHub provides the following features:

- GitHub allows users to host their Git repositories on its servers. Developers can create repositories to store their code and version history.
- GitHub promotes social collaboration and networking among developers. Users can follow other developers, star and fork repositories, and contribute to open-source projects.
- GitHub includes an issue tracking system where users can create, assign, and track issues related to their projects. Issues can be used to report bugs, suggest enhancements, or discuss project-related topics.
- GitHub provides features such as wikis, project boards, and discussions to help teams collaborate effectively. Wikis allow for documentation and knowledge sharing, project boards help manage tasks and workflows, and discussions provide a platform for team communication.

- GitHub Pages is a feature that enables users to host static websites directly from their repositories. It simplifies the process of publishing and sharing web content.

GitHub offers both free and paid plans, with additional features and storage available in the paid tiers. It has become a popular choice for individual developers, open-source projects, and large organizations due to its user-friendly interface, strong community, and extensive features for code collaboration and project management.

1.2.2.3 Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to build server-side and network applications. Node.js enables developers to use JavaScript for server-side scripting, allowing them to write server-side code using the same language as client-side code, which is typically executed in web browsers. Key features of Node.js are:

- Node.js is designed to be non-blocking and event-driven. This means that it can handle many simultaneous connections efficiently, making it well-suited for applications with a large number of I/O operations, such as real-time web applications.
- Node.js operates on a single-threaded event loop, which handles asynchronous I/O operations. This allows it to handle many connections concurrently without the need for multi-threading, making it scalable and efficient.
- Node.js comes with a package manager called npm, which is the largest ecosystem of open-source libraries and modules for JavaScript. Developers can easily install, manage, and share packages and dependencies using npm.
- Node.js is cross-platform, meaning it can run on various operating systems, including Windows, macOS, and Linux.
- Node.js has a vibrant and active community, and its ecosystem is rich with modules and frameworks that simplify the development of web and server applications. Popular frameworks like Express.js make it easier to build web applications with Node.js.

Node.js is commonly used to build server-side applications, APIs, and real-time applications. It has gained popularity for its performance, scalability, and the ability to use JavaScript throughout the entire web application stack, from front end to back end.

For additional information on Node.js see the official website.

1.2.2.4 MongoDB

MongoDB is a popular open-source NoSQL database management system that falls under the category of document-oriented databases. Developed by MongoDB Inc., MongoDB stores data in flexible, JSON-like documents with a dynamic schema. This means that the fields within a document can vary from one document to another, and data can be nested within sub-documents or arrays.

Key features of MongoDB include:

- MongoDB stores data in BSON (Binary JSON) documents, which are binary representations of JSON-like documents. This flexible schema allows for the storage of data in a way that is more natural for certain types of applications, as it can accommodate changes in data structure over time.
- MongoDB is classified as a NoSQL database, which means it does not rely on the traditional relational database management system (RDBMS) model. NoSQL databases are designed to handle large volumes of unstructured or semi-structured data and can provide better performance and scalability for certain use cases.
- MongoDB is designed to scale horizontally by adding more servers to a database cluster, allowing it to handle increased load and larger datasets. It supports automatic sharding, a method of distributing data across multiple machines.
- MongoDB provides a rich set of query capabilities, allowing developers to perform complex queries on their data. It supports a variety of query operators, indexing, and aggregation frameworks.
- MongoDB is known for its high performance, especially for read and write-intensive applications. It uses memory-mapped files for storage, which can lead to faster read and write operations.
- MongoDB supports the creation of indexes on fields to improve query performance. Indexes can be created on single fields or compound indexes on multiple fields.
- MongoDB has a large and active community, and it is supported by a rich ecosystem of tools and libraries. It also provides official drivers for a variety of programming languages.

MongoDB is commonly used in a range of applications, including content management systems, real-time big data analytics, mobile applications, and more. Its flexibility, scalability, and ease of use make it a popular choice for developers working on projects with evolving or complex data structures.

For additional information on MongoDB see the official website.

1.2.2.5 Vue.js

Vue.js, commonly referred to as Vue, is a progressive JavaScript framework used for building user interfaces. It is designed from the ground up to be incrementally adoptable, meaning that it can be easily integrated into existing projects and other libraries. Vue.js is often used for developing single-page applications (SPAs) where dynamic, responsive user interfaces are crucial.

Key features of Vue.js include:

- Vue.js uses a declarative approach to describe the structure and behavior of the UI. Developers can use simple template syntax to bind data to the DOM (Document Object Model) and let Vue.js handle the underlying manipulations.
- Vue.js promotes a modular and reusable architecture through components. Components encapsulate both the structure (HTML), behavior (JavaScript), and style (CSS) of a part of the UI. Components can be composed and reused throughout the application.

- Vue.js provides a reactive data-binding system. When the underlying data changes, the associated DOM updates automatically, and vice versa. This allows developers to build dynamic and responsive applications without directly manipulating the DOM.
- Vue.js includes a set of built-in directives that enable developers to add functionality to the DOM elements. For example, the 'v-if' directive is used for conditional rendering, and 'v-for' is used for rendering lists.
- Vue Router is the official routing library for Vue.js, allowing developers to build single-page applications with navigation. It enables the creation of navigation between different views or components in a Vue.js application.
- Pinia is the state management library for Vue.js applications. It provides a centralized state management pattern, helping manage the state of the application in a predictable and maintainable way, especially in larger applications.
- Vue CLI (Command Line Interface) is a tool for scaffolding and managing Vue.js projects. It helps developers set up a new project with a sensible default configuration, manage dependencies, and build production-ready applications.

Vue.js is known for its simplicity and ease of integration, making it a popular choice for both beginners and experienced developers. It can be used for building small to large-scale applications and has a growing community and ecosystem of libraries and tools. Vue.js is often compared to other JavaScript frameworks like React and Angular, and its flexible and progressive nature makes it an attractive option for a variety of projects.

For more information about Vue.js, visit the official website.

1.2.2.6 IoT

IoT refers to the network of physical devices, vehicles, buildings, and other items embedded with electronics, software, sensors, and connectivity which enables these objects to connect and exchange data.

1.2.2.7 MQTT

Message Queuing Telemetry Transport (MQTT) is a lightweight publish-subscribe messaging protocol that is commonly used for communication in IoT systems. It enables devices to send and receive messages over a network in a decentralized and efficient way. MQTT is often used in IoT because it is lightweight and requires minimal network bandwidth, making it well-suited for use on resource-constrained devices and low-bandwidth networks. Equally as important there a multiple implementations of the open source client software that are available for microcontrollers such as TI-Tiva, Arduino and Raspberry PI. The following are the high level features:

- Publish/subscribe: The MQTT protocol is based on the principle of publishing messages and subscribing to topics, which is typically referred to as a publish/subscribe model or sometimes abbreviated to pub/sub. Clients can subscribe to topics that are relevant to them. Then those clients only receive messages that are published to those topics. Alternatively, clients can publish messages to topics, which in turn makes them available to all subscribers of those topics.

- Quality of service levels: MQTT defines three quality of service (QoS) levels for message delivery, with each level providing a higher level of assurance the message gets delivered. However higher QoS levels are likely to consume more network bandwidth or subject the message to delays.
- Retained messages: With MQTT, the server keeps the message even after sending it to all current subscribers. If a new subscription is submitted for the same topic, any retained messages are then sent to the new subscribing client.
- Clean sessions and durable connections: When an MQTT client connects to the server, it sets the clean session flag. If the flag is set to true, all of the client's subscriptions are removed when it disconnects from the server. If the flag is set to false, the connection is treated as durable, and the client's subscriptions remain in effect after any disconnection. In this event, subsequent messages that arrive carrying a high QoS designation are stored for delivery after the connection is reestablished. Using the clean session flag is optional.
- Wills: When a client connects to a server, it can inform the server that it has a will , or a message, that should be published to a specific topic or topics in the event of an unexpected disconnection. A will is particularly useful in alarm or security settings where system managers must know immediately when a remote sensor has lost contact with the network.

A detailed explantion of the basics of MQTT can be found in the MQTT Essentials guide.

1.2.2.8 Microcontoller

A microcontroller is a small computer on a single integrated circuit that contains a processor core, memory, and programmable input/output peripherals. It is designed to control a specific function or set of functions within a larger system. Microcontrollers are used in a wide range of electronic devices and appliances, including automobiles, appliances, and consumer electronics. There are multiple advantages to using microcontrollers in electronic devices and systems:

- Cost-effective: Microcontrollers are relatively inexpensive and can be used to control a wide range of functions in a single device, reducing the overall cost of the system.
- Low power consumption: Microcontrollers are designed to consume minimal power, which makes them well-suited for use in battery-powered or low-power applications.
- Compact size: Microcontrollers are small in size, which allows them to be used in tight spaces or portable devices.
- Flexibility: Microcontrollers can be easily programmed to perform a wide range of functions, making them highly versatile and adaptable to different applications.
- Easy to interface with other devices: Microcontrollers have a variety of input/output (I/O) ports, which makes it easy to interface them with other devices such as sensors, actuators, and communication interfaces.
- Real-time performance: Microcontrollers can perform a variety of tasks concurrently, and also can handle real-time events which allows them to respond quickly to changing conditions in the system.

- Robustness: Microcontrollers are designed to operate in a wide range of environments, and can be used in harsh conditions like extreme temperatures and high levels of radiation.

In the context of the IoT, microcontrollers are used to collect data from sensors and control actuators, and to process and transmit that data over a network.

1.3 Software Development

Microservices development is a software development approach in which a software application is broken down into smaller, independent services that can be developed, deployed, and scaled separately. This approach allows for greater flexibility and scalability, as well as easier maintenance and testing. It also enables different parts of the application to be written in different languages and technologies, and allows for independent deployment and scaling of each service. However, it also requires careful planning and coordination to ensure that the different services work together effectively.

Microcontroller development is the process of designing, coding, testing, and debugging software and firmware for microcontrollers, which are small, low-power computer systems that can be integrated into a wide variety of devices and systems.

The development process for microcontroller-based systems involves the following steps:

- Hardware design: This includes selecting and specifying the microcontroller and any other components, such as sensors and actuators, that will be used in the system.
- Software development: This includes writing the firmware or embedded software that will run on the microcontroller. This step typically involves using a programming language such as C or Assembly and a development environment such as IAR Embedded Workbench or Keil uVision.
- Testing and debugging: This includes testing the system to ensure that it functions correctly and debugging any issues that are found. This step typically involves using tools such as logic analyzers, oscilloscopes, and in-circuit emulators (ICEs) to test and debug the system.
- Deployment: Once the system has been tested and debugged, it is ready to be deployed in the final application.

There are also development boards that are specifically designed for microcontroller development, these boards include the microcontroller, memory, and peripheral interfaces, and they make the development process easier by providing a ready-made platform for testing and debugging firmware.

1.3.1 Software Development Environment

Software development is accomplished using several tools (more details are discussed in):

- Visual Studio Code (VS Code) with the PlatformIO extension is used for c, c++ code for the Arduino platforms, including the ESP8266.
- VS Code is used for JavaScript for Node.js.

- VS Code is used for LaTex.
- Git is used for version control. The source code is stored on GitHub.

1.3.1.1 Visual Studio Code

A development environment, also known as an Integrated Development Environment (IDE), is a software tool or a set of tools that provides a comprehensive environment for developers to create, edit, test, and debug software applications. It encompasses various components, features, and functionalities that aid in the software development process.

A development environment typically includes the following key elements:

- Code Editor: A code editor is the central component of a development environment. It provides a text editor with features like syntax highlighting, code completion, code navigation, and formatting. It allows developers to write and modify code efficiently.
- Compiler/Interpreter: A development environment often includes a compiler or interpreter specific to the programming language being used. It translates the written code into executable or interpretable form.
- Build Tools: Build tools automate the process of compiling, linking, and packaging software applications. They help in managing dependencies, generating binaries, and performing other build-related tasks.
- Debugging Tools: Debugging tools enable developers to identify and fix issues in their code. They provide features like breakpoints, stepping through code, inspecting variables, and tracking program execution flow.
- Version Control Integration: Many development environments integrate with version control systems like Git, allowing developers to manage and track changes in their codebase, collaborate with others, and handle branching and merging.
- Project Management: Development environments often offer project management features to organize and manage multiple files and resources within a project. They may include features like project templates, file navigation, and project-specific settings.
- Testing Framework Integration: Some development environments integrate with testing frameworks, making it easier to write and run unit tests, perform automated testing, and generate test reports.
- Documentation Support: Development environments may provide features to assist in documenting code, such as auto-generating documentation, code commenting support, and integration with documentation generation tools.
- Integration with External Tools and Libraries: A good development environment allows seamless integration with external tools, libraries, and frameworks specific to the chosen programming language or platform. This makes it easier to utilize third-party libraries and leverage existing ecosystem resources.

- Customization and Extension: Development environments often offer extensibility through plugins, extensions, or a package management system. This allows developers to enhance the functionality of the environment by adding new features or integrating with additional tools.

Overall, a development environment provides a unified and streamlined workflow for software development, bringing together essential tools and features needed to write, test, and debug code effectively. It aims to improve productivity, code quality, and collaboration among developers.

VS Code is such an IDE that is used to develop and maintain RAILS.

VS Code is developed and maintained by Microsoft and has gained significant popularity among developers worldwide. The following factors and attributes have contributed to the widespread adoption of VS Code:

- VS Code supports many different programming languages, including JavaScript, TypeScript, Python, C++, C#, Java, and many others. Its flexibility and extensive language support make it appealing to a broad range of developers.
- VS Code offers excellent support for web development. Its integration with web technologies, such as HTML, CSS, and JavaScript, combined with the availability of various extensions, makes it a preferred choice for building web applications.
- VS Code is an open-source project with an active community of contributors. This open nature encourages collaboration, fosters innovation, and enables developers to extend and customize the editor to suit their specific needs.
- VS Code is available for Windows, macOS, and Linux, making it accessible to developers across different operating systems. Its consistent user interface and features across platforms contribute to its popularity.
- VS Code is known for its speed and performance. It is lightweight compared to many other IDEs, making it quick to start up and responsive even when working with large codebases.
- VS Code provides a rich ecosystem of extensions, allowing developers to enhance their development environment. The marketplace offers a wide range of extensions for various programming languages, frameworks, and tools, enabling developers to customize their setup and improve their productivity.
- VS Code includes an integrated terminal, eliminating the need to switch between the editor and an external command-line interface. This seamless integration enhances the development workflow, allowing developers to execute commands, run scripts, and interact with their projects without leaving the editor.
- VS Code Code offers strong integration with version control systems like Git. It provides a built-in version control interface, allowing developers to manage their code repositories, track changes, and resolve conflicts directly within the editor.
- The VS Code community is active and vibrant. Developers can find help, tutorials, and resources through official documentation, community forums, and online platforms. This active support system contributes to the growth and adoption of the editor.

- Microsoft and the open-source community continue to actively develop and improve VS Code. Regular updates introduce new features, performance enhancements, and bug fixes, ensuring that the editor remains up-to-date and meets the evolving needs of developers.
- VS Code provides a rich and extensible IDE experience with features like code highlighting, autocompletion, code navigation, and debugging.
- VS Code has excellent integration with Git, enabling version control management within the editor. The Git functions commit, pull, push, and resolve merge conflicts are provided by VS Code.

VS Code with PlatformIO is a powerful combination for developing embedded systems and IoT projects. PlatformIO is an open-source ecosystem that provides a unified development platform for different microcontrollers, development boards, and frameworks. When used together, Visual Studio Code and PlatformIO offer a range of features and capabilities for embedded development. Here are some things that can be done with VS Code and PlatformIO:

- PlatformIO supports various development platforms, including Arduino, ESP-IDF, mbed, STM32, and many more. It is easy to switch between different platforms and frameworks within VS Code.
- PlatformIO integrates with a vast library repository, making it easy to search, install, and manage libraries for projects. It simplifies the process of adding external libraries to the projects code.
- PlatformIO provides a powerful build system that handles the compilation and linking of the project's code. It supports different build configurations and allows the user to customize the build process.
- PlatformIO includes a serial monitor feature that allows the user to communicate with the user's embedded device over a serial interface. The user can send and receive data, monitor logs, and troubleshoot issues.

VS Code provides excellent support for Node.js and web development, including frameworks like Vue 3.

- VS Code offers excellent support for Vue development. With the "Volar" extension installed, it provides features like IntelliSense, syntax highlighting, code snippets, and error checking for Vue templates, JavaScript, and CSS.
- VS Code offers a range of features to enhance the JavaScript code editing experience. It provides syntax highlighting, code formatting, and auto-completion out of the box.
- VS Code has built-in support for code formatting and linting. It is possible to configure any project's formatting and linting rules using tools like Prettier and ESLint.
- VS Code has an integrated terminal that allows the ability to run Node.js commands and scripts without switching to an external terminal.

For additional features, VS Code has a rich ecosystem of extensions that can be installed to enhance the development experience. The following are some useful extensions for Node.js and web development:

- Volar - This extension provides advanced features for Vue development, including syntax highlighting, IntelliSense, code snippets, and error checking.
- Prettier - This extension provides code formatting for JavaScript, TypeScript, and CSS.
- ESLint - This extension provides linting for JavaScript and TypeScript.
- Debugger for Chrome - This extension allows the user to debug JavaScript and TypeScript code in the Google Chrome browser.
- ive Server - This extension provides a live preview of the user's web application in the browser.
- GitLens - This extension provides Git integration and allows the user to view and manage Git repositories within VS Code.
- Code Spell Checker - This extension provides spell checking for the user's code.

For additional information about VS Code and its features, visit the official website and Getting Started with Visual Studio Code. For more information about PlatformIO, visit the official website.

1.3.1.2 MQTTX

MQTTX is a free and open-source MQTT client tool that provides a graphical user interface (GUI) for working with MQTT. It is designed to facilitate the testing, debugging, and monitoring of MQTT-based applications. MQTT is a lightweight messaging protocol commonly used in IoT and other applications that require efficient, reliable, and real-time communication between devices or clients and a server.

MQTTX offers the following features:

- MQTTX connects to MQTT brokers (servers) and manages multiple connections simultaneously. It supports connecting to brokers using various authentication methods, such as username/password or certificates.
- With MQTTX, enables the user to publish messages to MQTT topics and subscribe to topics to receive messages. It provides an intuitive interface to define topic names, payloads, and QoS (Quality of Service) levels for both publishing and subscribing.
- MQTTX maintains a message history, which allows the user to review and analyze previously sent and received messages. It provides a payload preview feature to visualize the content of messages, helping the user understand the data being transmitted.
- MQTTX displays a topic tree that provides a hierarchical view of MQTT topics. This helps the user navigate through the topics and easily subscribe to or unsubscribe from specific topics. It also supports filtering messages based on topic patterns, making it easier to manage and monitor specific topics of interest.
- MQTTX includes built-in tools for encoding and decoding payload data in various formats, such as JSON, Base64, and Hex. This is useful for analyzing and manipulating payload data during testing and debugging.

- MQTTX supports secure communication using TLS/SSL encryption. It allows the user to configure and connect to MQTT brokers that require secure connections, providing an additional layer of data protection.

MQTTX is available for different operating systems, including Windows, macOS, and Linux. Its user-friendly interface and feature-rich environment make it a valuable tool for developers working with MQTT-based applications, enabling efficient testing, debugging, and monitoring of MQTT communications. For more information about MQTTX, visit the official website.

1.3.1.3 MongoDB Compass

MongoDB Compass is a graphical user interface Graphic User Interface (GUI) tool provided by MongoDB Inc. It is designed to simplify the process of working with MongoDB databases. Compass allows users to interact with their MongoDB databases visually, providing an intuitive way to explore, analyze, and manipulate data.

Some key features of MongoDB Compass include:

- Compass provides an easy-to-use interface for navigating and exploring MongoDB databases and collections. Users can view documents, query data, and understand the structure of their data through a graphical representation.
- Compass includes visual query and aggregation builders that allow users to construct MongoDB queries and aggregations without writing any code. This feature helps users who may not be familiar with the MongoDB query language to easily interact with their data.
- Compass provides real-time monitoring and performance analysis tools to help users optimize their MongoDB deployments. It offers insights into query execution times, index usage, and other performance metrics, allowing users to identify and address performance bottlenecks.
- Compass allows users to create, modify, and delete indexes on their MongoDB collections. It provides recommendations for index creation based on query patterns and can help users improve the performance of their database queries.
- Compass includes a schema validation feature that enables users to define rules for the structure and content of their MongoDB documents. It helps maintain data integrity by validating incoming documents against predefined schemas.
- Compass supports importing and exporting data to and from MongoDB databases. Users can import data from various file formats, such as JSON or CSV, and export data to these formats as well.

Overall, MongoDB Compass is a powerful tool that simplifies the management and interaction with MongoDB databases, providing a visual and user-friendly interface for developers, database administrators, and data analysts. For more information about MongoDB Compass, visit the official website.

Chapter 2

Use Case Survey Model

A use case survey model is a method of identifying and documenting the specific tasks and activities that a particular group of users (such as modelers or operators) performs with a product or system. The goal of a use case survey is to understand how the users interact with the product or system, and to identify opportunities for improvement or potential areas of confusion or frustration. The survey typically includes a series of questions that ask the users to describe their tasks and activities, as well as any problems or issues they have encountered. The results of the survey are then analyzed to identify patterns and trends, which can be used to inform design and development decisions.

Use cases are used in software development to describe how a system or product should behave in response to certain inputs or events. They provide a clear and detailed understanding of the requirements for a system, and serve as a communication tool between developers, stakeholders and users. These are depicted as ellipses in the survey model.

Some of the key benefits of using use cases include:

- They help to identify the specific functional requirements of a system, which can be used to guide the design and development process.
- Use cases provide a clear and detailed description of how a system should behave in response to different inputs, which can be used to test the system and ensure that it meets the requirements.
- Use cases can be used to identify and document the different types of users and their specific needs, which can be used to inform user-centered design decisions.
- They provide a clear and consistent way to communicate the requirements of a system to different stakeholders, including developers, business analysts, and project managers.
- Use cases can be used to identify potential risks and problems early in the development process, which can help to minimize the impact of these issues on the overall project.

In use case modeling, an actor is a person or system that interacts with the system being developed. Actors represent the external entities that interact with the system in order to achieve a specific goal or accomplish a specific task.

Each actor is typically defined by a set of characteristics, such as their role, responsibilities, and

the specific tasks or activities they perform with the system. Actors can be either human users or other systems that interact with the system being developed.

In the use case diagram, actors (human) are represented by stick figures and actors (systems) are represented by subsystem packages. Actors are usually connected to the use cases they interact with by an arrow.

Use case actors are important because they help to identify the different types of users or systems that interact with the system, which can be used to inform user-centered design decisions. Actors also help to identify the specific requirements of the system, which can be used to guide the development process and ensure that the final product meets the needs of all stakeholders.

Figure 2.1 depicts the use case survey model used to design and develop RAILS. In figure 2.1 the line color:

- blue indicates the actor use case relationship
- orange indicates an inheritance relationship where the open arrow is the inherited.
- purple indicates a uses relationship where the open arrow is the used use case.

2.1 Actors

2.1.1 Human Actors

- Dispatcher is responsible to develop an operating session plan in which rolling stock is assigned to consists and those consists are routed and scheduled.
- Model Railroad Manager is responsible to establish and maintain railroad network; rolling stock inventory; defining Digital Command and Control (DCC) parameters for appropriate rolling stock; industries and their cargo types; schedule rolling stock and railroad network components for maintenance; initial operating states for all turnouts, signals and DCC components.
- Model Railroad Modeler is responsible to create, modify, and dispose of model railroad assets.
- Session Operator is responsible to execute part or all an operating session plan, outside of the rail yard, using the system and the DCC Subsystem.
- Session Supervisor is responsible to supervise the execution of part or all an operating session plan by the Session Operators and Yard Masters. This person may also function as a Session Operator.
- Yard Master is responsible to execute part of an operating session plan, by assembling the consists inside the rail yard, using the system and the DCC Subsystem.

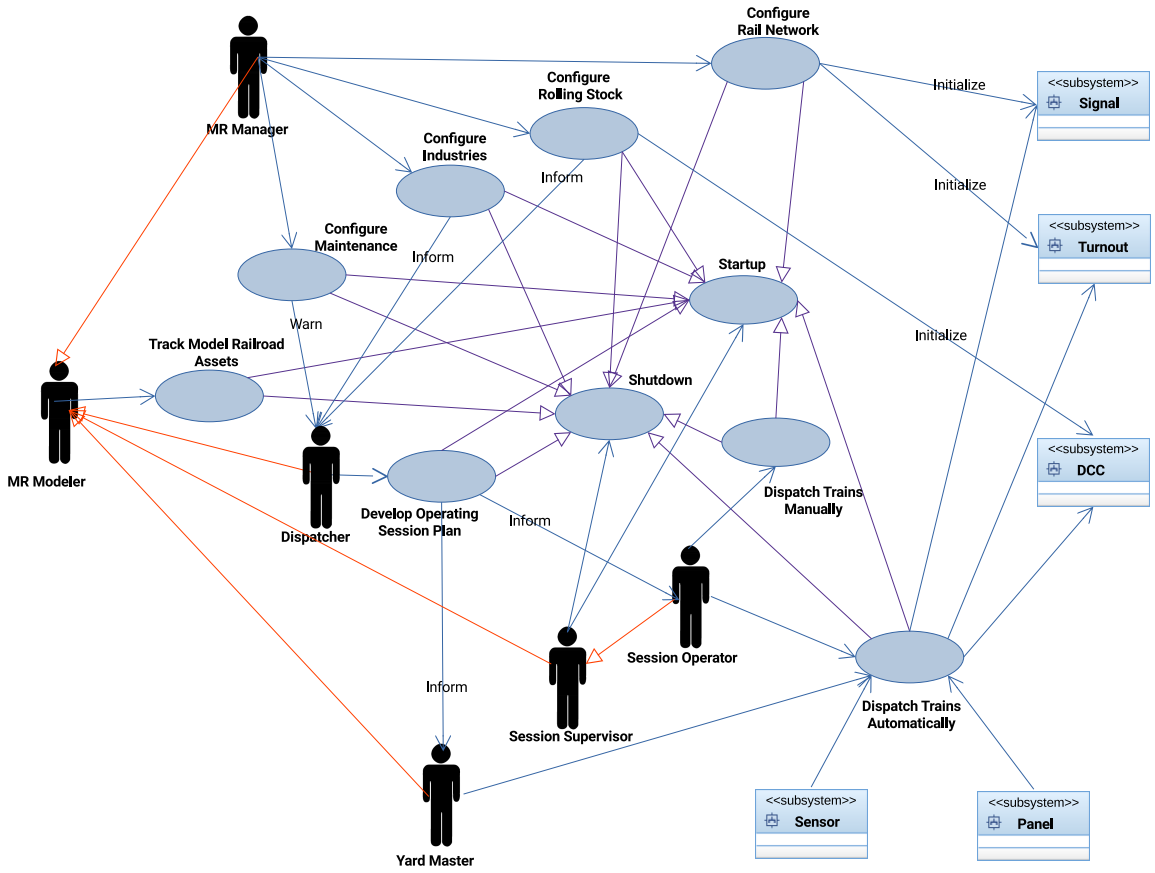

Figure 2.1: Use Case Survey Model

2.1.2 Subsystem Actors

- DCC Subsystem is responsible to transmit messages to rolling stock equipped with DCC receiver/controller, as commanded by the system and to warn the system of possible malfunctions.
- Sensor Subsystem is responsible to detect the presence of rolling stock in a railroad network block/segment and provide notification to the system.
- Signal Subsystem is responsible to change the state of specified signals located on the rail network as commanded by the system.
- Turnout Subsystem is responsible to indicate the state of turnouts on the rail network; change the state of turnouts as commanded by the system; and indicate operators' action to change the state of turnouts.
- Panel Subsystem is responsible for displaying status information about turnouts and providing operator input to change turnout position.

2.2 Use Cases

- Configure Industries is responsible to establish and maintain the Industry database, allowing the manager to add, modify or delete industries, simulate production/consumption of goods as cargo, and inform the dispatcher of industry cargo requirements.
- Control Model Railroad is responsible to assist the operator in managing the execution of an operating session plan. By:
  - automatically executing the parts of an operating session plan selected by the operator for regular scheduled trains by commanding the Signal, Turnout and DCC subsystems; and
  - assisting the operator executing parts of an operating session plan by commanding the Signal, Turnout and DCC subsystems.
- Configure Maintenance is responsible to establish and maintain the maintenance database, allowing the manager to schedule rolling stock or railroad network components maintenance, simulate component failures, and warn the dispatcher of needed maintenance
- Configure Network is responsible to establish and maintain the railroad network database, allowing the manager to add, modify or delete railroad network components, and to initialize the Signal and Turnout Subsystems automatically at startup and as requested by the manager.
- Configure Rolling Stock is responsible to establish and maintain the rolling stock database, allowing the manager to add, modify or delete rolling stock and inform the dispatcher of the rolling stock attributes.
- Develop Operating Session Plan is responsible to assist the dispatcher in developing an operating session plan.

- Dispatch Trains Automatically is responsible for executing the operating session plan autonomously.
- Dispatch Trains Manually is reponsible to assist the operators execute the operating session plan.
- Shutdown Model Railroad is responsible to bring the system down gracefully ensuring the state be stored so that when the system Startup occurs the system returns to the state when the system was shutdown.
- Startup is responsible to bring the system up in either a distributed environment or standalone mode.
- Track Model Railroad Assets is responsible to establish and maintain the model railroad asset database, allowing the modeler to add, or modify assets and manage modeling projects.

Chapter 3

Microservices Design Components

Figure 3.1 shows the microservices components that make up the design of RAILS.

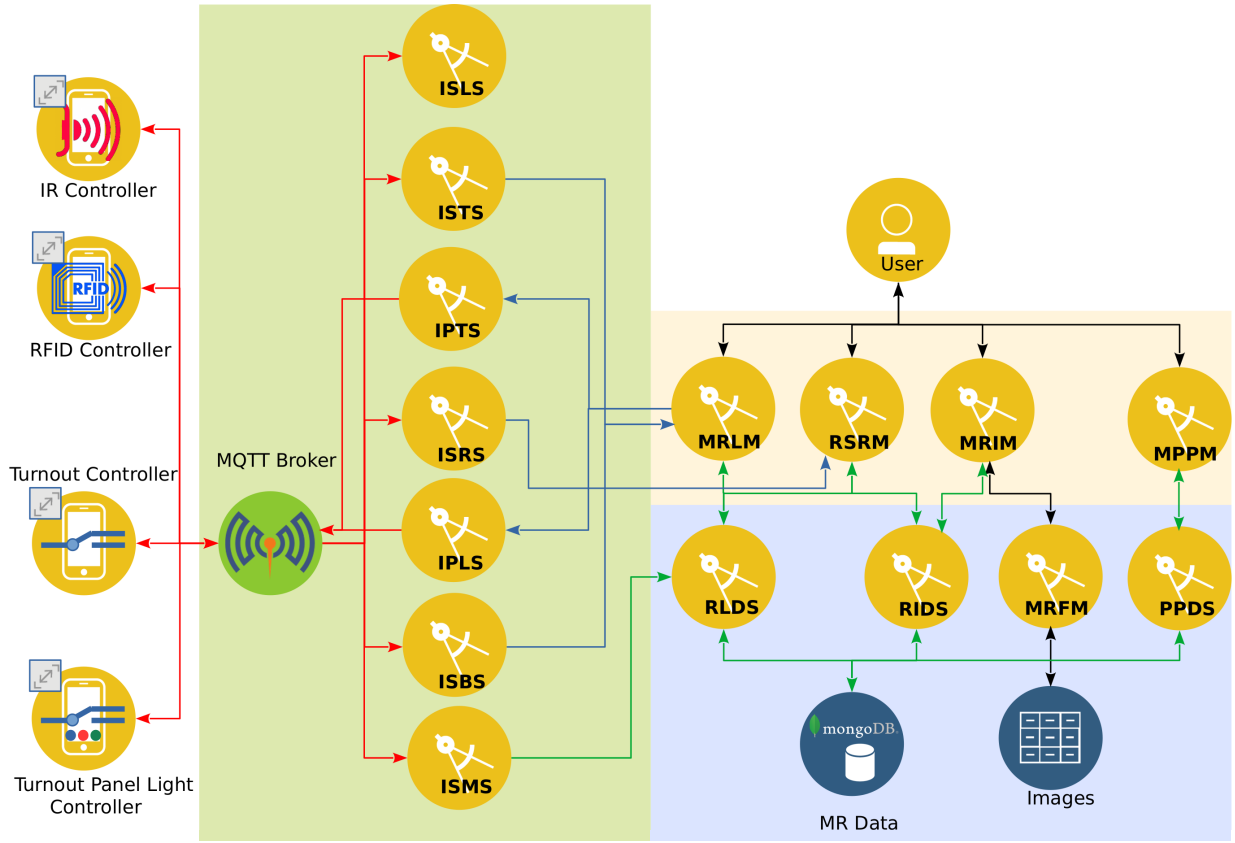

Figure 3.1: Microservices Component Architecture

The microservices design components are divided into three sets:

- IoT components, which are highlighted with the light green colored background in Figure 3.1.
- Data Services (DS) components, which are highlighted with the light blue colored background in Figure 3.1.

- Single Page Application (SPA) components, which are highlighted with the buff colored background in Figure 3.1.

3.1 IoT Components

The components of the IoT set of design components are subdivided into:

- Micro Controllers using the MQTT protocol:
  - Radio Frequence Identification (RFID) Controller processes RFID tags obtained from a RFID reader and then publishes the value
  - Turnout Controller subscribes to turnout commands then to act on the command to cause the turnout to move. It then publishes the state of the turnout
  - infrared (IR) Controller (in planning) processes IR sensors and publishes their values
- MQTT Broker is the heart of any publish/subscribe protocol, is responsible for receiving messages, posting to designated topics and sending messages to clients subscribing to topics.
- The subscribers and publishers bridge the MQTT elements with the GUI applications:
  - IoT Publisher Turnout Panel Light Services (IPLS) publishes turnout panel light commands a Turnout Panel Controller
  - IoT Publisher Turnout Services (IPTS) publishes turnout commands to a Turnout Controller
  - IoT Subscriber Turnout Panel Button Services (ISBS) subscribes to push button events and pushes them via a web-socket to the MRLM component
  - IoT Subscriber Location Services (ISLS) (in planning) IoT subscribes to topics that provide location information i.e., IR Sensors and RFID sensors
  - IoT Subscriber Micro-controller Services (ISMS) subscribes to micros and adds or updates micros collection in RAILS. It also subscribes to micro heartbeats.
  - IoT Subscriber RFID Services (ISRS) subscribes to RFID tags and pushes them via a web-socket to the Rollingstock RFID Manager (RSRM) component
  - IoT Subscriber Turnout Services (ISTS) subscribes to turnout switch closures and pushes them via a web-socket to the Model Railroad Layout Manager (MRLM) component

3.2 DS Components

DS consist of all the components that handle and or store the model railroad data:

- MR Data – the document repository, MongoDB, to store complete collections of items such as rolling stock, industries (producers and consumers), track elements, turnouts, projects, purchases, etc.
- Railroad Inventory Data Services (RIDS) provides Representational State Transfer (REST) access to railroad inventory documents

- Plans and Purchases Data Services (PPDS) provides REST access to model railroad projects and purchases documents
- Railroad Layout Data Services (RLDS) provides REST access to model railroad layout documents
- Model Railroad File Manager (MRFM) provides the user the ability to upload image files for the use by the Model Railroad Inventory Manager (MRIM) component
- Images is the file store for the images uploaded by MRFM component and used by the MRIM component

3.3 SPA Components

GUI applications that provide user access to RAILS:

- RSRM, this SPA allows a user to match a RFID value to a rolling stock road name and number
- MRIM, this SPA allows a user to create, update and delete model railroad assets, such as rolling stock
- Model Projects and Purchase Manager (MPPM), this SPA allows a user to enter information about their projects and purchases
- MRLM, this SPA allows a user to enter information about their layout and control elements of it

Chapter 4

IoT Design

In software architecture, the publish/subscribe pattern is a messaging pattern where senders of messages (publishers), do not program the messages to be sent directly to specific receivers (subscribers), but instead categorize published messages into topics without knowledge of which subscribers, if any, there may be. Subscribers express interest in one or more topics, and only receive messages that are of interest, without knowledge of which publishers, if any, there are. This pattern decouples senders and receivers, allowing for greater scalability and a more dynamic communication environment.

Eclipse Mosquitto is an open-source message broker software that implements the MQTT protocol, a standard for publish/subscribe messaging for the IoT. Mosquitto provides a lightweight way for IoT devices to communicate with each other, allowing them to publish and subscribe to messages on topics. With Mosquitto, devices can publish information, such as sensor readings or status updates, to the message broker and receive updates from other devices. Mosquitto helps to facilitate efficient and effective communication between IoT devices in a scalable manner.

Table 4.1 identifies functions excuted by microcontrollers and the IoT messaging attributes.

Micro Function	**Pub/Sub**	**Topic**	**Message**
RFID Reader	Pub	sensors/rfid	RFID
Turnout Contacts	Pub	sensors/toc	Turnout Position
Turnout Panel Push Button	Pub	sensors/pb	TP Button
Heartbeat	Pub	micros	Micro Info
Turnout	Sub	acts/to/cntrlr id	Turnout Cmd
Turnout Panel Light	Sub	acts/tpl/cntrlr id	TP Light Cmd
Micro Info	Pub	micros	Micro Info

Table 4.1: IoT Micro Function Table

Figure 4.1 depicts the flow of MQTT messages through the MQTT broker from and to microservices components.

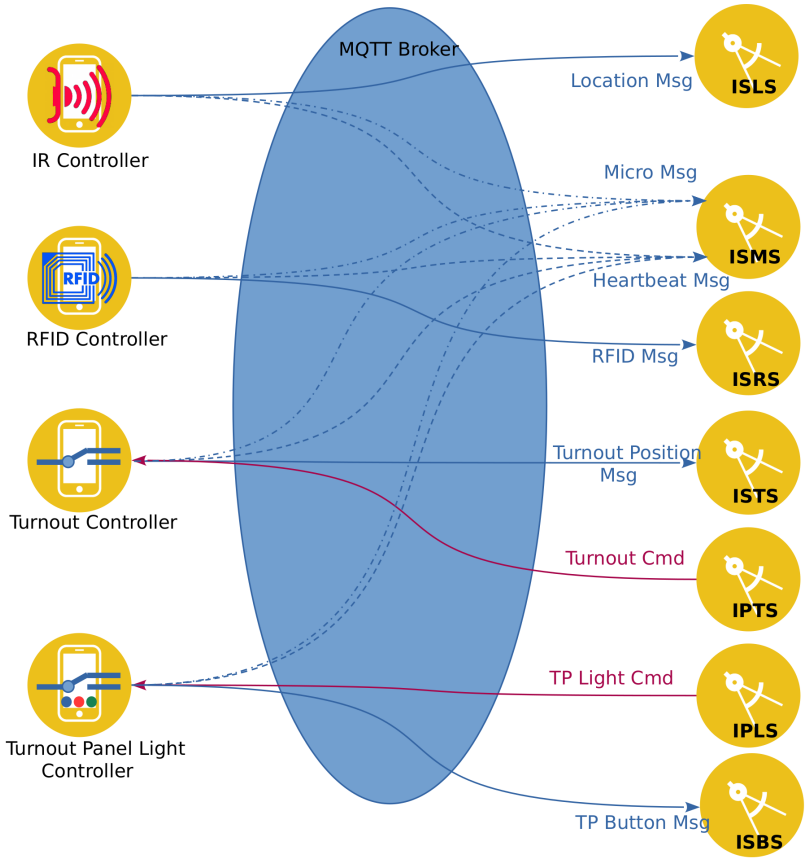

Figure 4.1: IoT Design

4.1 Message Definitions

- RFID message format: {“et”:“epoch time”,”mcntrlr”:”sensor id“,”rfid”:”rfid tag value“}
  - example {”et”:”1590463450”,”mcntrlr”:”rfidRdr01”,”reader”:”1”,”rfid”:”1C0044CF23”}
- **Turnout Position** message format: {“et”:”epoch time“,”mcntrlr”:”turnout controller id“, ”to”:”turnout number“,”state”:”THROWN|CLOSED|ERROR“}
  - example {”et”:”1588827073”,”mcntrlr”:”TrnCtlr01”,”to”:”1”,”state”:”THROWN”}
- **Turnout Panel Button** message format: {“et”:”epoch time“,”mcntrlr”:”tp controller id“, ”pb”:”push button number“}
- **Turnout Command** message format: {”mcntrlr”:”turnout controller id“,”to”:”turnout number“, ”cmd”:”THROW|CLOSE|STATUS“}
  - example {”mcntrlr”:”TrnCtlr01”,”to”:”1”,”cmd”:”CLOSE”}
- **Turnout Panel Light Command** message format: {”mcntrlr”:”turnout panel controller id“, ”tpl”:”light number“,”color”:”RED|GREEN|BLUE“,”type”:”BUTTON”}

- **Micro Info** message format: {“et”:”epoch time“,”mcntrlr”:microcontroller id“, ”msgType”:”initial|heartbeat”,”ip”:”Address of Micro”}
  - ’heartbeat’ example: {”et”:”1590462747”,”mcntrlr”:”RfidRdr00”,”msgType”:”heartbeat”}
  - ’initial’ example: {”et”:”1590462747”,”mcntrlr”:”RfidRdr00”,”msgType”:”initial”,”ip”:”192.168.0.19”}

Chapter 5

Data Design

MongoDB is a popular not only SQL (NoSQL) database that is widely used for various types of applications. A NoSQL database, is a type of database management system that provides an alternative approach to storing and retrieving data compared to traditional relational databases that use Structured Query Language (SQL). NoSQL databases are designed to handle large volumes of unstructured or semi-structured data, and they offer more flexibility and scalability for certain types of applications.

MongoDB was choosen for the data repository for RAILS for the following factors:

- Model railroad systems often involve a variety of data types and relationships. NoSQL databases, especially document-oriented ones like MongoDB, allow you to store data in a more flexible and schema-less manner. This is beneficial when dealing with diverse information such as train schedules, routes, stations, user preferences, and more.
- As the model railroad system expands, it might need to accommodate a larger volume of data and potentially more users. NoSQL databases are designed to scale out horizontally, making it easier to handle increased workloads and data growth.
- Model railroad systems often involve real-time operations, such as tracking train locations, monitoring schedules, and responding to user interactions. NoSQL databases are optimized for low-latency operations, allowing you to retrieve and update data quickly.
- MongoDB offers geospatial capabilities. This can be highly beneficial for a model railroad system, as you could store and query data related to train positions, routes, and geographic locations.
- Model railroad systems may include various types of data, some of which might not fit neatly into a structured format. NoSQL databases excel at handling semi-structured and unstructured data, allowing you to store and retrieve information like images, sensor data, and user-generated content.
- NoSQL databases often use formats like JSON, which are more intuitive and familiar to developers. This can streamline development and reduce the need for complex data transformations.

- Model railroad systems might undergo changes in data requirements as the system evolves or new features are added. NoSQL databases make it easier to adapt to these changes without requiring a rigid schema.
- MongoDB has a wide range of libraries, drivers, and tools available, which can facilitate development and integration with other components of your system.
- If a developer encounters challenges or has questions while developing the model railroad system, NoSQL databases often have active communities and extensive documentation that can provide assistance.

MongoDB stores data in a flexible, semi-structured document format, using JavaScript Object Notation (JSON).

5.1 Document Collections

MongoDB stores data in a flexible, semi-structured document format, using JSON. RAILS organizes the data in three[1] groupings, which correspond to the three data microservices; RIDS, PPDS and RLDS.

5.1.1 Railroad Inventory

5.1.1.1 aarcodes

An Association of American Railroads (AAR) code refers to a classification code used by the AAR to identify different types of equipment and rollingstock used in the North American railway industry. The AAR codes are a standardized way of categorizing and labeling various railway vehicles, making it easier to identify and track equipment across different railways and organizations.

Table 5.1: AarCode Collection Fields Table

Field	Type	Description	Req'd
aarCode	String	A classification code used by the AAR to identify different types of rollingstock	Yes
rollingstockType	String	A category of rollingstock having common characteristics, such as; Auxiliary, Freight, Gondola, Hopper, Locomotive, Non Revenue, and Passenger.	No
description	String	A written representation that aims to make vivid the characteristics of the AAR Code.	No

5.1.1.2 dccs

DCC is a system used in model railroading to control and operate model trains and their accessories. It is a standardized method of transmitting control signals digitally to locomotives and other rollingstock. DCC allows for more precise and independent control of multiple trains on the same track.

[1]The current design has only three but as additional functionality is added additional groups may be added

Table 5.2: Dccs Collection Fields Table

Field	Type	Description	Req'd
locomotiveID	ObjecctId	The identity object of the dccs document assigned by MongoDB at the time the rollingstock (locomotive) document is created.	Yes
mfg	String	The company that made the DCC decoder installed in a rollingstock, typically a locomotive.	No
family	String	A group of DCC decoders produced by a specific manufacturer that share common features, specifications, and compatibility.	No
model	String	A DCC decoder offering specific features or specifications within the same product family	No
address	String	A unique identifier assigned to each DCC decoder, which is a numerical or digital label that distinguishes one rollingstock from another in the DCC system. DCC addresses range from 1 to 9999.	No

5.1.1.3 images

A set of photographs or illustrations of the rollingstock and structures of the model railroad present in RAILS. These images may capture different designs, configurations, and features of rollingstock, providing a visual representation of the diversity within the railway transportation system.

Table 5.3: Images Collection Fields Table

Field	Type	Description	Req'd
fileName	String	A unique identifier assigned to a computer file in order to distinguish it from other files within a file system.	Yes
title	String	The name or label given to the image to identify and distinguish it from others.	No
notes	String	A brief record of events or observations about the image.	No
category	String	A classification or group of images that share common characteristics, features, or attributes.	No

5.1.1.4 industries

Sectors of economic activity that involve the production of goods or the provision of services. These sectors are characterized by similar business activities and are often grouped together based on their common features. Railways service a wide range of industries, facilitating the transportation of goods and passengers. The industries collection is used to identify the industries that are present in the model railroad system.

Table 5.4: Industries Collection Fields Table

Field	Type	Description	Req'd
shortName	String	A concise or abbreviated version of a longer name. It could be a nickname, an acronym, or a shortened form that is used for convenience or brevity.	Yes
longName	String	A word or set of words by which an industry is identified and distinguished from others, legal name.	No
industryType	String	The various categories or sectors of economic activity in which businesses and organizations engage.	No
industryLocation	String	A specific place or position in space, often designated by its geographical coordinates or described in relation to other landmarks.	No

5.1.1.5 rollingstocks

rollingstock refers to the vehicles that move on a railway track. These vehicles are an essential part of any railway system and include various types of vehicles used for transporting goods and passengers. Rollingstock can be broadly categorized into two main types: passenger rollingstock and freight rollingstock. Locomotives are considered a type of rollingstock in the context of railway transportation.

Table 5.5: Rollingstock Collection Fields Table

Field	Type	Description	Req'd
roadName	String	The road name also reffered to as reporting marks are a set of letters used to identify the owner or operator of a railcar or locomotive.	Yes
roadNumber	String	The road number is an identifier for a railcar or a locomotive and is numerical.	Yes
color	String	The exterior paint or livery applied to the rollingstock. The color of rollingstock serves several purposes, including identification, branding, and visibility.	No
aarCode	String	See paragraph 5.1.1.1	No
description	String	The informative account that provides information about the characteristics, features, qualities, or attributes of the rollingstock.	No
numberBlt	String	The total quantity or count of a particular rollingstock.	No
inSvcDate	Date	The specific date on which a peice of rollingstock is officially put into operation or use. It marks the beginning of its active service life, during which it is expected to perform its intended functions.	No

Continued on next page

Table 5.5 – *Continued from previous page*

Field	Type	Description	Req'd
insideLength	String	For passenger cars it is measured from one end of the passenger compartment to the other. It defines the available space for seating and other passenger-related amenities. For freight cars it is measured from the inside faces of the end walls, excluding any bulkheads or projections that might reduce the usable length. It provides an indication of the space available for loading and transporting goods.	No
insideHeight	String		No
insideWidth	String		No
loadTypes	String		No
capacity	String	Capacity refers to the maximum amount of cargo, typically measured in weight or volume, that it can safely and efficiently transport.	No
bldr	String		No
bltDate	Date		No
notes	String	A brief record of events or observations about the rollingstock.	No
ltWeight	String	The weight of the car itself without any cargo or load.	No
loadLimit	String	The maximum weight that a freight car can carry, including both the weight of the car itself (tare weight or lightweight) and the weight of the cargo (payload).	No
lastMaintDate	Date		No
locationNow	String		No
homeLocation	String		No
rsStatus	String	The current condition, availability, or operational state of the rollingstock used on a railway.	No
imageID	String		No
modelWeight	String	The mass of the model rollingstock in either grams or ounces.	No
modelLength	String	The distance measure between the middle of the front coupler knuckle to the middle of the rear coupler knuckle of the model rollingstock.	No
rfid	String	The 10 character identifier emitted by the RFID tag attached to the model rollingstock.	No

5.1.1.6 structures

Field	Type	Description	Req'd
title	String		Yes
structureUse	String		Yes
description	String		No
owner	String		No
location	String		No
construction	String		No
builtDate	String		No
size	String		No
image	String		No
isIndustrial	Boolean		No
rawMaterials	String		No
rMCapacity	String		No
conRate	String		No
priority	String		No
aarCodeIn	String		No
product	String		No
productCap	String		No
prodRate	String		No
aarCodeOut	String		No
unloadDuration	String		No
loadDuration	String		No
sidingCap	String		No
notes	String		No

Table 5.6: Structures Collection Fields Table

5.1.2 Railroad Projects and Purchases

5.1.2.1 mrcompanies

Field	Type	Description	Req'd
name	String		Yes
type	String		Yes
website	String		No
email	String		No
phone	String		No
address	String		No
notes	String		No

Table 5.7: MrCompanies Collection Fields Table

5.1.2.2 projects

Field	Type	Description	Req'd
title	String		Yes
type	String		Yes
description	String		No
startdate	Date		No
enddate	Date		No
roadname	String		No
roadnumbers	String		No
notes	String		No

Table 5.8: Projects Collection Fields Table

5.1.2.3 purchases

Field	Type	Description	Req'd
num	String		No
date	String		No
store	String		Yes
item	String		No
desciption	String		Yes
manufacturer	String		No
unitcost	Number		No
qty	Number		No
project	String		No
roadname	String		No
roadnumbers	String		No
notes	String		No

Table 5.9: Purchases Collection Fields Table

5.1.3 Railroad Layout

5.1.3.1 micros

Field	Type	Description	Req'd
microID	String		Yes
microIP	String		Yes
et	String		No
purpose	String		No
status	String		No
sensorLoc	String		No

Table 5.10: Micros Collection Fields Table

5.1.3.2 tplights

Field	Type	Description	Req'd
id	String		Yes
tplNum	String		Yes
controller	String		Yes
panelName	String		No
panelNum	String		No

Table 5.11: TPLights Collection Fields Table

5.1.3.3 turnouts

Field	Type	Description	Req'd
toID	String		Yes
toNum	String		Yes
controller	String		Yes
state	String		Yes
type	String		No
lock	String		No
notes	String		No
lastUpdate	String		No
toLoc	String		No

Table 5.12: Turnouts Collection Fields Table

Chapter 6

Microcontrollers Design

6.1 Introduction

Figure 3.1 shows the microservices components that make up the design of RAILS. The microservices, highlighted with the light green colored background in Figure 3.1 are shown connected to the IoT components on the left side of the figure. The IoT components are the microcontrollers that are used to control the model railroad layout. The microcontrollers are connected to the MQTT broker via Wi-Fi. The microcontrollers are programmed to provide model railroad sensors and actuators. A conceptual diagram of the microcontroller design is shown in Figure 6.1 where the PC is the platform the MQTT broker and the other microservices are running.

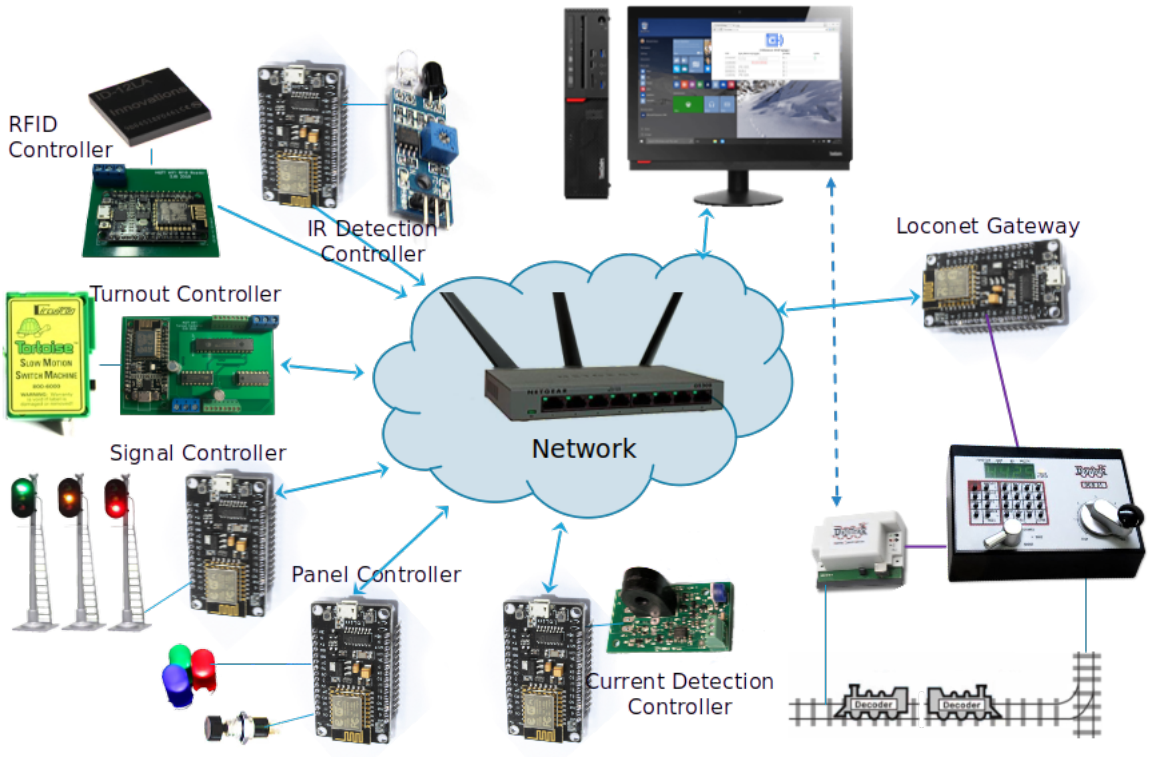

Figure 6.1: Components in the design of RAILS

6.2 Microcontrollers

An ESP8266 development board is a small, breadboard-friendly circuit board that houses the ESP8266 Wireless Fidelity (Wi-Fi) microcontroller chip along with additional components to make it easy to use for programming and prototyping. It's a popular choice for hobbyists and makers working on IoT projects. The key features of the ESP8266 are:

- 32-bit Reduced Instruction Set Computer (RISC) Central Processing Unit (CPU): Tensilica Xtensa LX106 running at 80 MHz
- 64 KiB of instruction Random Access Memory (RAM), 96 KiB of data RAM
- Institute of Electrical and Electronics Engineers (IEEE) 802.11 b/g/n Wi-Fi
- 16 General Purpose Input/Output (GPIO) pins
- Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), Inter-IC Sound (I2S) interfaces with Direct Memory Access (DMA) (sharing pins with GPIO)
- Universal Asynchronous Receiver-Transmitter (UART) on dedicated pins
- 1 10-bit Analog to Digital Converter (ADC)
- Pulse Width Modulation (PWM) on all GPIO pins with DMA (sharing pins with GPIO)
- 1 8-bit Digital to Analog Converter (DAC)
- 10 µA deep sleep current

The benefits of using an ESP8266 development board, such as the NodeMCU are:

- Breadboard-friendly
- Versatile and easy to use
- ESP8266 boards are some of the cheapest Wi-Fi development boards available
- Easy to program and debug, either with the Arduino IDE or VS Code with PlatformIO.
- Easy to connect to a network and the Internet

6.3 RFID Microcontroller

The RFID microcontroller is responsible for reading the RFID tags attached to the rolling stock. The RFID microcontroller is programmed to:

- initializes Wi-Fi and MQTT paramters are set at compile time using values kept in a params.h file
- connects to an MQTT broker via Wi-Fi
- publishes info about this reader to the topic "micros" in the format: {"et":"1590462747","mcntrlr":"RfidRdr01","msgType":"initial","ip":"192.168.0.19"}

- publishes a heartbeat to the topic “micros” in the format: {“et”:“1590462747”,“mcntrlr”:“RfidRdr01”,“msgType”:“heartbeat”}
- reads values from a single ID-12LA RFID reader, formats the results as a JSON string, gets Epoch time from an Network Time Protocol (NTP) server and then publishes the JSON string to the topic “sensors/rfid” in the format: {“et”:“1590463450”,“mcntrlr”:“rfidRdr01”,“reader”:“1”,“rfid”:“1C0044CF23”}

Figure 6.2 depicts the circuit diagram of the RFID reader. The RFID reader is connected to the ESP8266 microcontroller via a serial connection.

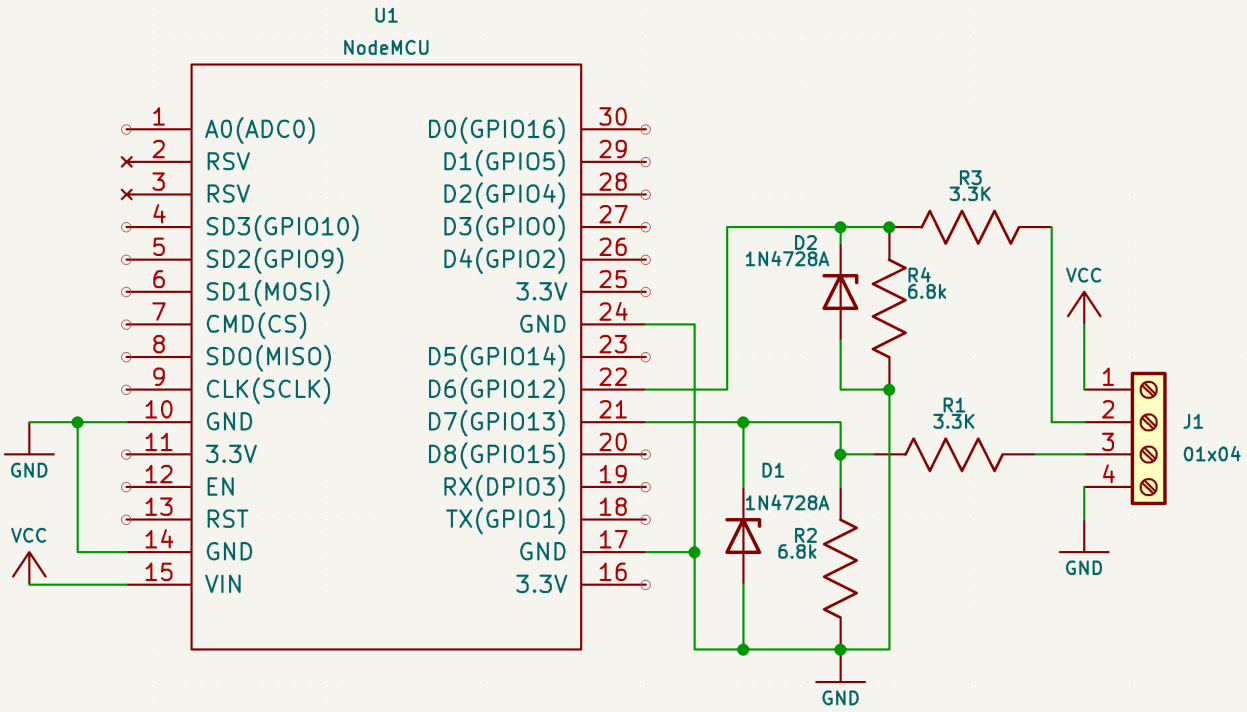

Figure 6.2: RFID Reader

6.4 Turnout Microcontroller

The turnout microcontroller is responsible for controlling the turnouts on the model railroad layout. The turnout microcontroller is programmed to:

- initializes Wi-Fi and MQTT paramters are set at compile time using values kept in a params.h file
- connects to an MQTT broker via Wi-Fi
- publishes info about this reader to the topic “micros” in the format: {“et”:“1590462747”,“mcntrlr”:“ToCntlr01”,“msgType”:“initial”,“ip”:“192.168.0.19”}
- publishes a heartbeat to the topic “micros” in the format: {“et”:“1590462747”,“mcntrlr”:“ToCntlr01”,“msgType”:“heartbeat”}

- subscribes to the topic acts/to/ToCntlrxx where xx is the this controller in the format: {”to”:”1|2|3|4”,”command”:”THROW|CLOSE|STATUS”} which either sets the turnout using an L293 to closed or thrown based on the command received
- responds to status command for a turnout by publishing the status of the turnout to the topic ”sensors/toc” in the format: {“et”:“1590463450”,“mcntrlr”:“ToCntlr01”,“to”:“1”,“dir”:“THROWN”}

Figure 6.3 depicts the circuit diagram of the turnout controller.

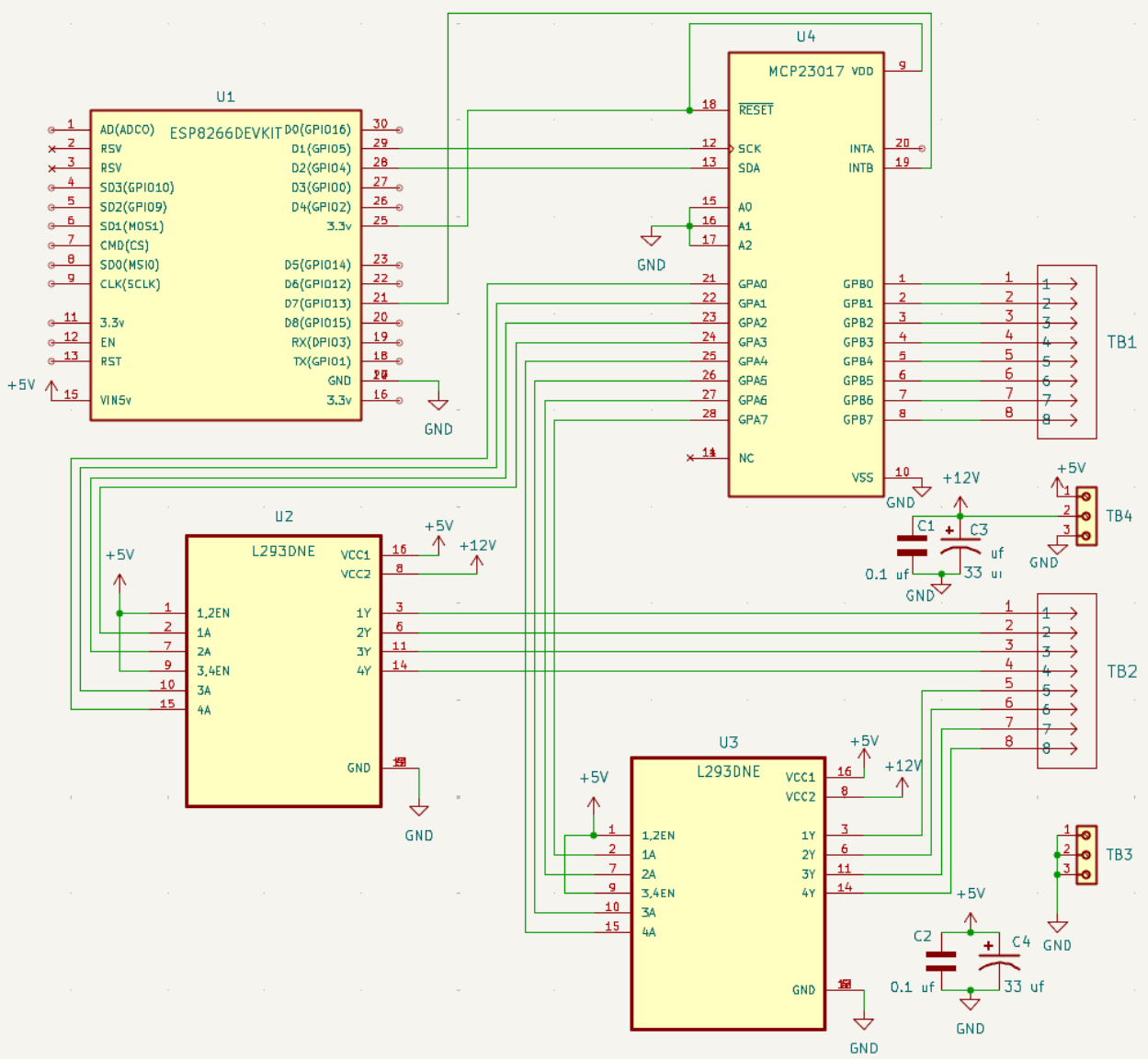

Figure 6.3: Turnout Controller

Glossary

Analog to Digital Converter (ADC) . 36

Application Program Interface (API) acts as a middleman between different software applications. It defines a set of rules and specifications that allow applications to talk to each other and exchange data or functionality. 4

Association of American Railroads (AAR) . 28

Central Processing Unit (CPU) . 36

command-line interface (CLI) is a a text-based way to interact with a computer program by typing commands and seeing text-based responses. 4

Data Services (DS) is a class of RAILS components that provides an API permitting access to the MongoDB that stores the documents used by RAILS. 21, 22

Digital Command and Control (DCC) . 17, 19, 28, 29

Digital to Analog Converter (DAC) . 36

Direct Memory Access (DMA) . 36

Enterprise Java Bean (EJB) . 3

General Purpose Input/Output (GPIO) . 36

Graphic User Interface (GUI) is the visual way you interact with a computer program using elements like windows, icons, menus, and buttons, instead of typing commands. GUIs are the standard way most people interact with computers today. 15, 22, 23

in-circuit emulator (ICE) . 10

infrared (IR) Infrared radiation is a type of electromagnetic radiation that lies just beyond the red end of the visible light spectrum. It's invisible to the human eye, but we can feel it as heat. 22

input/output (I/O) . 9

Institute of Electrical and Electronics Engineers (IEEE) . 36

Integrated Development Environment (IDE) . 11, 12, 13, 36

Inter-IC Sound (I2S) . 36

Inter-Integrated Circuit (I2C) . 36

Internet of Things (IoT) The Internet of Things refers to a vast network of physical devices embedded with sensors, software, and other technologies that allows them to connect and exchange data with other devices and systems over the internet or other communication networks. 1, 8, 10, 13, 14, 21, 22, 24, 35, 36

IoT Publisher Turnout Panel Light Services (IPLS) . 22

IoT Publisher Turnout Services (IPTS) . 22

IoT Subscriber Location Services (ISLS) . 22

IoT Subscriber Micro-controller Services (ISMS) . 22

IoT Subscriber RFID Services (ISRS) . 22

IoT Subscriber Turnout Panel Button Services (ISBS) . 22

IoT Subscriber Turnout Services (ISTS) . 22

Java Enterprise Edition (JEE) . 2, 3

JavaScript Object Notation (JSON) . 28, 37

JavaServer Faces (JSF) . 3

Message Queuing Telemetry Transport (MQTT) MQTT is a lightweight messaging protocol designed for machine-to-machine (M2M) communication in resource-constrained environments, like those found in the Internet of Things (IoT). 8, 9, 14, 15, 22, 24, 35, 36, 37

Model Projects and Purchase Manager (MPPM) . 23

Model Railroad File Manager (MRFM) . 23

Model Railroad Inventory Manager (MRIM) . 23

Model Railroad Layout Manager (MRLM) . 22, 23

Network Time Protocol (NTP) . 37

not only SQL (NoSQL) . 27

Plans and Purchases Data Services (PPDS) . 23, 28

Pulse Width Modulation (PWM) . 36

quality of service (QoS) . 9

Radio Frequence Identification (RFID) is technology that uses radio waves to wirelessly identify and track objects. 22, 25, 36, 37

Railroad Inventory Data Services (RIDS) . 22, 28

Railroad Layout Data Services (RLDS) . 23, 28

Railway Administration and Information Logical System (RAILS) . 2, 4, 12, 17, 21, 27, 28, 29, 35

Random Access Memory (RAM) . 36

Reduced Instruction Set Computer (RISC) . 36

Relational Database (RDB) . 3

Representational State Transfer (REST) is a set of architectural principles for designing web services. It's not a protocol itself (like HTTP), but rather a guideline for creating web services. 22, 23

Rollingstock RFID Manager (RSRM) . 22, 23

Serial Peripheral Interface (SPI) . 36

Services Oriented Architecture (SOA) . 3

Single Page Application (SPA) is a web application that loads a single HTML page in the user's browser and dynamically updates the content based on user interaction, without reloading the entire page. This creates a more fluid and responsive user experience, similar to what you might expect from a native mobile app. 22, 23

Structured Query Language (SQL) . 27

Universal Asynchronous Receiver-Transmitter (UART) . 36

version control system (VCS) . 5

virtual machine (VM) is a software program that acts like a physical computer. It creates a virtualized environment that can run its own operating system (OS) and applications, just like a physical computer would. 5

Visual Studio Code (VS Code) . 10, 11, 12, 13, 14, 36

Wireless Fidelity (Wi-Fi) . 36, 37