Advance Rails Training

Day 1: Dive into JavaScript and Styling Updates

Learning Goal: By the end of the day, you should be comfortable with the new JavaScript integrations in Rails 7, especially Hotwire and Import Maps, and understand the basics of styling with TailwindCSS in a Rails environment.

Introduction to Hotwire:

- Set up a basic Rails 7 application with Hotwire.
- Create a simple feature to understand Turbo Drives and Turbo Frames.
- Add Stimulus controllers to the mix and see how it interacts with Turbo.

Transitioning to Import Maps:

- Set up a new Rails 7 application and explore the default JavaScript setup.
- Import and use a few popular JavaScript libraries via import maps.
- Compare and contrast the experience with Webpacker.

Exploring TailwindCSS:

- Add TailwindCSS to a Rails 7 application.
- Create a few UI components using TailwindCSS utilities.
- Discuss the advantages and potential challenges.

Further Reading and References:

- Hotwire Official Documentation
- Understanding Import Maps in Rails 7
- TailwindCSS Documentation

Day 2: Mastering Navigation with Turbo Drive

Learning Goal: By the end of the day, you should be adept at using Turbo Drive to enhance the navigation and interactivity of Rails 7 applications, understanding its benefits over traditional full-page reloads.

Introduction to Turbo Drive:

- Set up a basic Rails 7 application with Turbo Drive.
- Create features that leverage Turbo Drive to handle link navigation and form submissions without full page reloads.

Turbo Drive and Page Updates:

- Learn how Turbo Drive handles page updates using partial page replacement.
- Experiment with Turbo Frames to update sections of a page independently.

Turbo Drive Event Handling:

- Use Turbo Drive lifecycle events to manage page rendering and form submissions dynamically.
- Add custom JavaScript callbacks for *turbo:load*, *turbo:before-cache*, and form submission events.

Optimizing Forms with Turbo Drive:
- Implement form submissions via Turbo Drive, demonstrating asynchronous updates without a full-page turnaround.
- Explore how Turbo Stream tags can be used to update the DOM in response to form submissions.

Comparing Full-Page Reloads vs. Turbo Drive:
- Contrast the user experience and performance between traditional full-page reloads and Turbo Drive navigation.
- Analyze network activity to understand the efficiency gains with Turbo Drive.

Further Reading and References:
Turbo Drive Documentation
Leveraging Turbo Drive in Rails 7

Day 3 & 4: Engaging Interfaces with Turbo Frames and Streams

Learning Goal: Gain proficiency in enhancing user interfaces with Turbo Frames for partial page updates and Turbo Streams for live content changes, understanding the full spectrum of methods and options available.

Turbo Frames:
- **Turbo Frame Tags and Attributes:**
  - Explore the *<turbo-frame>* tag and its attributes such as *id*, *src*, *loading*, and *target*.
- **Handling Forms with Turbo Frames**
- **Turbo Frame Navigation and Links:**
  - Learn how Turbo Frames handle internal link navigation using *data-turbo-frame* attribute.
- **Dynamic Content Loading in Turbo Frames:**
  - Explore *lazy-loading* with Turbo Frames using the src attribute and how it affects performance.
- **Turbo Frame Events:**
  - Learn about the custom events fired by Turbo Frames, such as *turbo:frame-load*.
  - Attach event listeners to Turbo Frames for custom interactivity.
  - Set up Turbo Frames and explore attributes like *src*, *loading*, and *target*.
  - Use *Turbo.frame()* JavaScript methods to interact with frames programmatically.

Turbo Streams:
- **Understanding Turbo Stream Tags:**
  - Delve into the syntax and usage of Turbo Stream tags.
- **Server-Side Stream Responses:**

    - Learn how to broadcast server-side updates using Turbo Streams in a Rails controller.
- **Turbo Stream Actions:**
    - Explore each Turbo Stream action (*append*, *prepend*, *replace*, *update*, *remove*) and their use cases.
    - Hands-on exercise: Create a dynamic to-do list that uses all Turbo Stream actions.
- **Client-Side Turbo Stream Events:**
    - Discuss handling client-side events triggered by Turbo Stream updates.
- **Integrating WebSockets with Turbo Streams:**
    - Overview of using Action Cable with Turbo Streams for WebSocket connections.
- **Advanced Techniques and Troubleshooting:**
    - Address common issues and advanced techniques for optimizing Turbo Stream usage.

Optimization and Performance:
- Examine the network and rendering performance using browser tools.
- Compare Turbo's methods and options with traditional full-page reloads and AJAX for both user experience and resource consumption.

Further Reading and References:
Turbo Frames: Methods & Options
Turbo Streams: Methods & Options

Day 5 & 6: Building Interactive UIs with Stimulus and Strada

Learning Goal: By the end of the day, you should be able to create interactive and maintainable front-end features in a Rails application using Stimulus, with an emphasis on state management within controllers.

Understanding Stimulus:
Review the principles of Stimulus and its conventions.
Initialize Stimulus in a Rails application and discuss its lifecycle.

Controllers and Actions:
- Develop Stimulus controllers and connect them to HTML with a *data-controller*.
- Interactive exercise: Bind events to controller actions using *data-action*.

In-Depth Event Handling:
- **Binding Events:** Discuss how to bind events to DOM elements using *data-action*.
- **Parameterized Actions:** Practice passing parameters to actions for more complex event handling.

- **Global Events:** Learn to listen for global events (e.g., *window* resize, *keypress*) in a Stimulus controller.

State Management with Stimulus:

- **Initializing State:** Set up state within a Stimulus controller and bind it to the UI.
- **Reactivity:** Explore how changes in state can automatically update the UI.
- **Persisting State:** Implement techniques to persist state across page loads using the session or local storage.

Advanced Event and State Interaction:

- **Complex State Changes:** Handle events that trigger multiple state changes and UI updates.
- **Throttling and Debouncing:** Use throttling and debouncing to manage rapid event firing for smoother state transitions.
- **State and DOM Synchronization:** Ensure that the DOM always reflects the current state, even after asynchronous updates.

Strada Integration for Mobile Experiences:

- **Mobile Optimization:** Introduce Strada for enhancing mobile user experiences and its integration with Stimulus.
- **Strada Features:** Explore Strada's capabilities such as smooth page transitions and mobile hardware interactions.
- **Strada and Stimulus Synergy:** Practice building mobile-optimized features using both Strada and Stimulus.

Advanced Interaction Patterns:

- Explore advanced Stimulus patterns for handling complex state and interactions, such as syncing state with server or local storage.
- Exercise: Build an interactive form that saves partial state, with validations and live feedback.

Further Reading and References:

Stimulus Handbook
Stimulus Reference: Working with State
Stimulus Handbook - Managing State

Day 7: Styling with Tailwind CSS

Learning Goal: Become proficient in using Tailwind CSS to style Rails applications with a utility-first CSS framework for rapidly building custom designs.

Tailwind CSS Fundamentals:

- Introduction to Utility-First: Understand the philosophy behind a utility-first CSS framework.
- Explore server side manner of TailwindCSS and how to manage Tailwind classes on Javascript side(if needed)
- Setting Up Tailwind CSS in Rails: Go through the installation process and configuring Tailwind CSS with Rails.

Building Layouts with Tailwind CSS:
- **Responsive Design:** Learn to create responsive designs using Tailwind's utility classes.
- **Grid and Flexbox:** Practice building complex layouts with Tailwind's grid and flexbox utilities.

Customizing Tailwind CSS:
- **Theming:** Explore how to customize Tailwind's default theme to match your design requirements.
- **Plugins and Components:** Dive into extending Tailwind with plugins and crafting reusable components.

Advanced Tailwind CSS Techniques:
- **Optimization for Production:** Implement strategies to optimize Tailwind CSS for production, including purging unused styles.
- **Interactive UI Elements:** Create interactive elements such as dropdowns, modals, and tabs using Tailwind CSS.
- JIT classes, purge, Forman?

Further Reading and References:
Tailwind CSS Documentation

Day 8: Active Record and Authentication

Learning Goal: Dive deep into the data handling features introduced in Rails 7. By the end of the day, you should understand encrypted attributes, the basics of horizontal sharding, and the advantages of delegated types.

Async Query Methods:
- Introduction to asynchronous querying with load_async and the addition of *async_sum*, *async_pluck*, and *async_count_by_sql*.
- **Resetting Singular Associations:** How to use reset on *has_one* associations to clear cached results.
- **Enum Method Generation Control:**
  Utilize the new *instance_methods*: false option to prevent Rails from generating all the helper methods for enums.

Support for Common Table Expressions (CTEs):
- Learn to use the *.with* method for more readable and maintainable complex queries.

Async Bulk Record Destruction:
- Configuration of *destroy_association_async_batch_size* for managing the destruction of large numbers of records.

Working with Encrypted Attributes:
- Set up a model with encrypted attributes.
- Perform CRUD operations and understand the encryption and decryption process.

Understanding Horizontal Sharding:
- Learn the theory behind horizontal sharding.
- Set up a mock environment to simulate sharding. This may be complex, so the primary focus should be on understanding the concept and the configurations involved.

Delegated Types:
- STI (Single Table Inheritance) vs Abstract Controller
- Refactor it to use Rails 7's delegated types.

User authentication is now Managed by Rails Itself!
- Rails 7 authentication
    - Rails Authentication From Scratch

Further Reading and References:
- Encrypted Attributes in Rails 7
- A Deep Dive into Horizontal Sharding
- Introducing Delegated Types

Day 9 & 10: Meta-Programming Mastery in Ruby

Learning Goal: By the end of the day, you should have a solid grasp of meta-programming concepts in Ruby, enabling you to write more expressive and concise code. You'll understand how to dynamically define methods, leverage method_missing, and utilize other meta-programming techniques to enhance the flexibility and power of your Ruby applications.

Introduction to Meta-Programming:
- Discuss Ruby's capabilities for meta-programming and why it's a powerful tool.
- Explore the use of *self* and *contexts* where meta-programming shines.

Dynamic Method Definition:
- Learn to define methods dynamically with *define_method*.
- Practice using *send* to invoke methods dynamically.

Hands-on Exercise: Create a dynamic report generator that defines methods based on various report types at runtime.

Advanced Meta-Programming Techniques:
- **The Power of *method_missing*:**
    - Understand the implications of using *method_missing* and best practices.
    - Implement custom behavior for undefined methods to handle graceful fallbacks.
- **Using *const_missing* and *const_set*:**
    - Discover how to dynamically handle missing constants and set them at runtime.

Hands-on Exercise: Build a mock interface that simulates API calls, using *method_missing* to handle dynamic method names representing different API endpoints.

Meta-Programming in Active Record:
- Utilize *method_missing* and *respond_to_missing?* to create a flexible query interface for Active Record models.
- **Eigenclasses and Singleton Methods:** Examine the concept of eigenclasses (also known as singleton classes) and how to define methods on individual object instances.

Hands-on Exercise: Extend a Rails model with custom eigenclass methods that adjust the behavior of the model instance on the fly, such as adding attribute-specific validation methods.
Evaluating Meta-Programming Practices:

- **Best Practices and Pitfalls:**
  - Explore the trade-offs of using meta-programming, including **readability vs. flexibility**.
  - Review code maintainability and the impact of meta-programming on debugging.
- **Performance Considerations:**
  - Analyze the performance implications of meta-programming and how to mitigate potential issues.

Further Reading and References:
https://www.rubyguides.com/guides/metaprogramming-guide.pdf
https://medium.com/codex/meta-programming-in-ruby-87afa97db59e
https://www.shakacode.com/blog/metaprogramming-in-ruby/

Day 11 & 12: Rails Asset Pipeline Mastery

Learning Goal: By the end of the day, you should have a solid understanding of the Rails Asset Pipeline and how to optimize front-end assets using *esbuild*, *Rollup*, and *Vite*.

Asset Pipeline Fundamentals:

- Introduction to the Asset Pipeline and its core components.
- Exercise on structuring assets within a Rails app and using manifest files.

Integrating Modern Bundlers:

- **Choosing the Right Bundler:** A comparison of *esbuild*, *Rollup*, and *Vite* to help decide which is best for your project needs.
- **Setting Up *esbuild*:**
  - **Installation:** Guide through adding *esbuild* to your Rails app.
  - **Configuration:** Customize the *esbuild* settings to work with Rails' asset pipeline.
- **Building and Deploying:** Implement the build process and integrate with Rails deployment strategies.
- **Working with Rollup:**
  - **Rollup Basics:** Introduce Rollup and its core concepts.
  - **Plugins and Extensions:** Explore the rich plugin ecosystem of Rollup for asset management.
  - **Optimization:** How to leverage Rollup for optimizing asset delivery.
- **Leveraging *Vite* for Development:**
  - **Hot Module Replacement (HMR):** Experience the fast update cycle with *Vite's* HMR.
  - **TypeScript and Other Languages:** Incorporate TypeScript and pre-processors with *Vite*.
  - **Production Builds:** Set up *Vite* for production builds and understand the advantages over traditional methods.

Advanced Techniques and Best Practices:

- Asset Minification and Optimization:
    - **Compression:** Techniques for compressing assets to reduce load times.
    - **CDN Hosting:** Best practices for serving assets through CDNs for improved performance.
- **Asset Versioning and Cache Busting:**
    - **Fingerprinting:** Implement fingerprinting to manage cache validation.
    - **Cache Control:** Strategies for controlling how assets are cached in the browser.
- **Debugging and Troubleshooting:**
    - **Console Tools:** Using browser developer tools to debug asset loading issues.
    - **Logs Analysis:** How to interpret Rails logs to troubleshoot compilation errors.
- **Security Considerations:**
    - **Dependency Management:** Keeping dependencies up-to-date to avoid security vulnerabilities.
    - **Content Security Policy (CSP):** Configuring *CSP* headers to enhance asset loading security.

Day 13 & 14: Rails Generators, Rack Middleware, Rails Engines

Rails Generators:

- What are generators?
    - Model vs. Resource vs. Scaffold
- **Using Built-in Generators**
    - Utilize link_to within scaffold templates.
    - Apply content_for to inject custom scripts in scaffolded views.
- **Creating Custom Generators**
    - Implement file_name and class_name helper methods in custom generators.
- **Enhancing Generators with Helpers**
    - Use `indent` to format scaffolded generator code.
    - Create a module_namespacing helper to handle namespaced file paths.

Practical Exercise: Write a generator for a blog resource that includes helper methods for customizing the scaffold view files with dynamic links and scripts.

Rails on Rack:

- What's a rack? Web Servers vs Application Server
- **Building Custom Middleware**
    - Define a *redirect_to_https* helper within a middleware to enforce SSL.
- **Middleware Stack Manipulation**
    - Write a *remove_middleware* helper method to dynamically adjust the middleware stack.
- **Rack Environment Helpers**
    - Create *request_ip* and *request_path* helpers to extract information from the Rack environment hash.

Practical Exercise: Develop a middleware that uses helper methods to log custom analytics data about each request. **WITHOUT USING RAILS** just using ruby

Rails Engines

- What are Rails Engines? How does it work?
- **Engines Helpers Introduction**
    - Use *main_app* and *engine_name* helper methods to navigate between engine and application routes.

- **Sharing Helpers Between Engines and Application**
    - Implement *shared_form_for* to create form builders that can be used both in the main app and the engine.
- **Isolating Engine Code**
    - Develop a decorate helper method for presenting model data within an engine.

Practical Exercise: Build an engine with a shared navigation view component using a helper method that renders appropriately whether in the engine or the main application.

Further Reading and References:
https://guides.rubyonrails.org/generators.html
https://guides.rubyonrails.org/engines.html
https://guides.rubyonrails.org/rails_on_rack.html

Day 15: Miscellaneous Features and Best Practices

Learning Goal: Explore the miscellaneous updates in Rails 7, including response rendering and routing, and gain a strong grasp of the testing improvements. Understand the rationale behind the deprecated features and get acquainted with the best practices in Rails 7.

Response Rendering and Routing Improvements:

- Experiment with the new at: option.
- Directly respond with various types (HTML, JSON, etc.).
- Explore the updated URL helpers.

Testing in Rails 7:

- Set up system tests using the :webdrivers gem.
- **Introduction to Parallel Testing:**
    - Overview of parallel testing in Rails and its advantages.
    - Configuring *test_helper.rb* to enable parallel testing with processes or threads.
- **How Parallel Testing Works:**
    - Deep dive into the default parallelization process and database management.
    - Hands-on exercise to adjust the parallel worker count and observe test suite behavior.

- **Tackling Common Pitfalls:**
    - Discussion on compatibility issues with RSpec and potential solutions.
    - Strategies to manage overhead with smaller test suites and adjust the parallelization threshold.
- **Gradual Adoption Techniques:**
    - Implement a modular approach to apply parallel testing to specific test classes.
    - Create a base test class for parallelization to inherit from for selected tests.

Discussion on Deprecated Features:

- List out significant deprecations in Rails 7.
- Discuss the reasons behind them and the recommended alternatives.

Wrap-up and Best Practices:

- Discuss the best practices when adopting a new Rails version.
- Explore potential challenges and solutions when upgrading from a previous version.