Final Project Presentation

CS3710-002: Full-Stack Web Development

William Maddock

12/06/2024

Web App Description

Overview: The AccessClearance Hub is a web application designed to streamline the process of managing and verifying access clearance levels for employees or contractors within an organization. Core functionalities include user authentication, clearance request submission, approval workflows, and automated clearance expiration notifications.

Audience: The app is designed for organizational administrators, security personnel, and employees who need to manage or request access clearances efficiently.

Tech Stack: The application leverages Ruby on Rails for backend development, Bootstrap for responsive frontend design, and Docker for environment consistency and deployment.

User Story

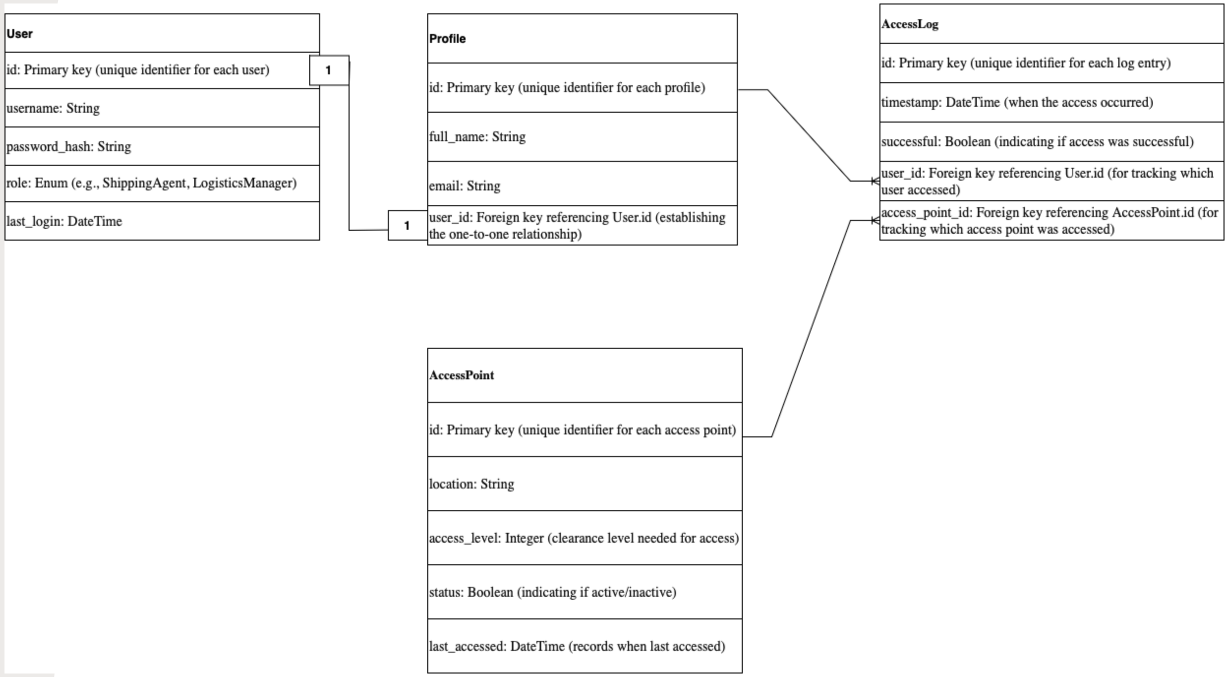
Content
As a Logistics Manager, I want to approve or deny clearance requests so that I can ensure only authorized personnel gain access to sensitive areas

Example Scenarios

Scenario 1: A Shipping Agent submits a request for clearance to access a restricted lab. The Logistics Manager reviews the request, checks the Shipping Agent's credentials, and approves it.

Scenario 2: A Shipping Agent's clearance has become inactive. So, either the administrator or the Logistics Manager revoked the clearance.

Visuals: UML Diagram

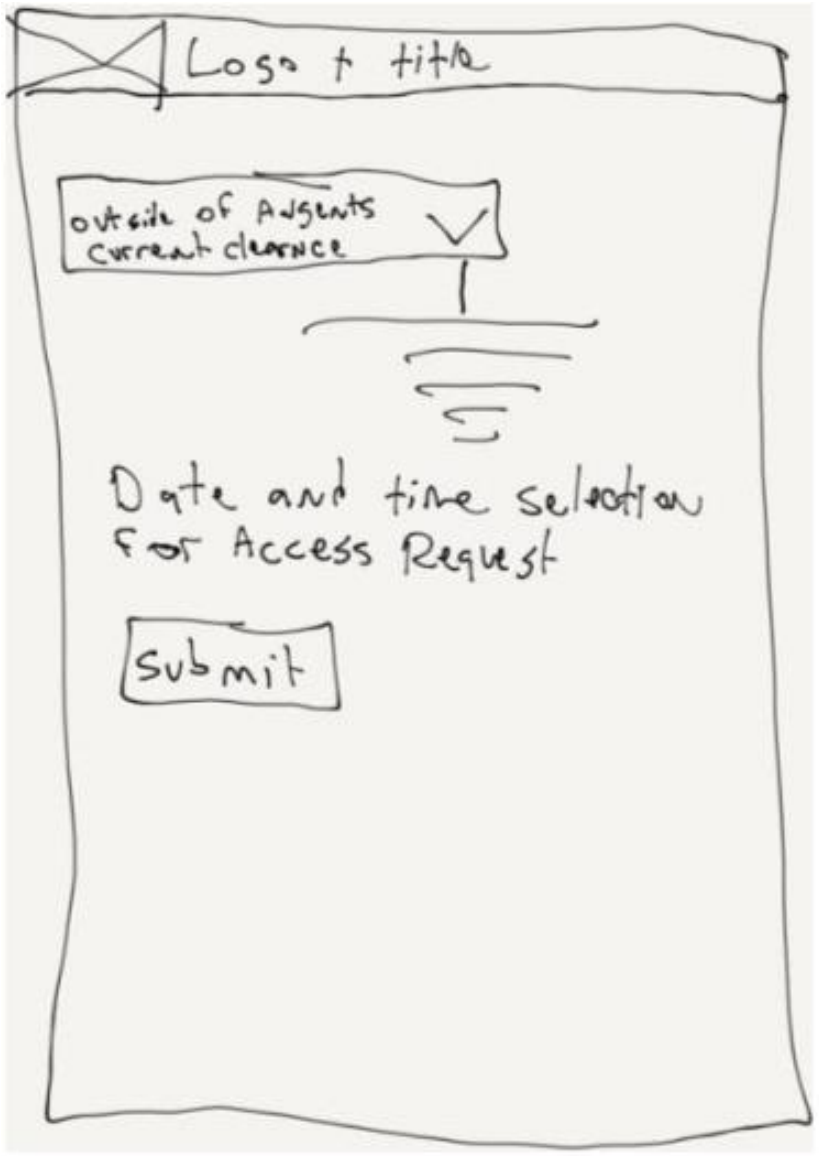

User
id: Primary key (unique identifier for each user)
username: String
password_hash: String
role: Enum (e.g., ShippingAgent, LogisticsManager)
last_login: DateTime
1
1
Profile
id: Primary key (unique identifier for each profile)
full_name: String
email: String
user_id: Foreign key referencing User.id (establishing the one-to-one relationship)
AccessLog
id: Primary key (unique identifier for each log entry)
timestamp: DateTime (when the access occurred)
successful: Boolean (indicating if access was successful)
user_id: Foreign key referencing User.id (for tracking which user accessed)
access_point_id: Foreign key referencing AccessPoint.id (for tracking which access point was accessed)
AccessPoint
id: Primary key (unique identifier for each access point)
location: String
access_level: Integer (clearance level needed for access)
status: Boolean (indicating if active/inactive)
last_accessed: DateTime (records when last accessed)

Lo-Fi Mockups

Mockup Screenshots: Include a wireframe of the “Create Access Request” page specifically designed for Shipping Agents. The wireframe should display a drop-down list for access points, a date and time selection widget, and a submit button.

Explanation: This mockup aligns with the user story by enabling Shipping Agents to request access to areas outside their current clearance. It supports the scenario where the agent needs temporary access, streamlining the approval process by forwarding the request to their supervisor.

Visuals: Wireframe for Access Request Page


Logo + title
outside of Agents current clearnce
Date and time selection for Access Request
Submit

Docker Setup

Purpose: Docker was used to ensure environment consistency across development, testing, and deployment. By containerizing the application, the setup process becomes streamlined, reducing the likelihood of issues due to differences in developer environments. It also simplifies deployment and provides an isolated workspace for running the application.

Setup: The Docker image was built using the following **command:**

docker buildx build -t bigwill12/msucs3710_name_it_whatever .

The application is run using the following **command** to map the local directory and ports:

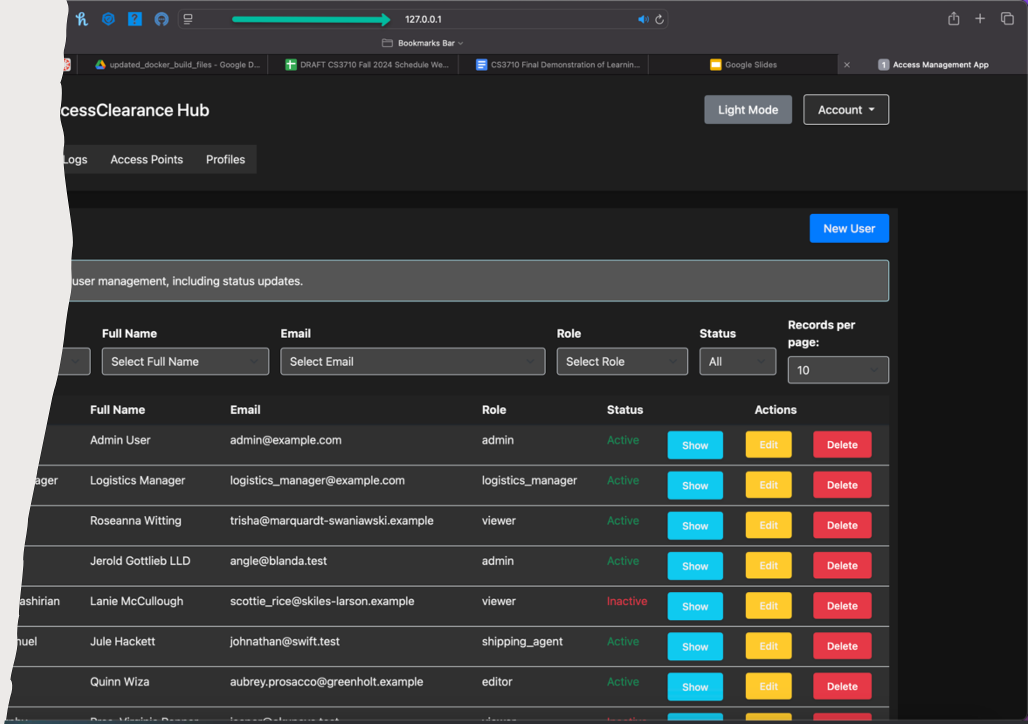
docker run -it -p 3000:3000 -v "$(pwd):/workspace" bigwill12/msucs3710_name_it_whatever

Inside the Docker container, the Rails server is started using this **command:**

rails server -b 0.0.0.0

Visuals: Screenshot of Docker in action.

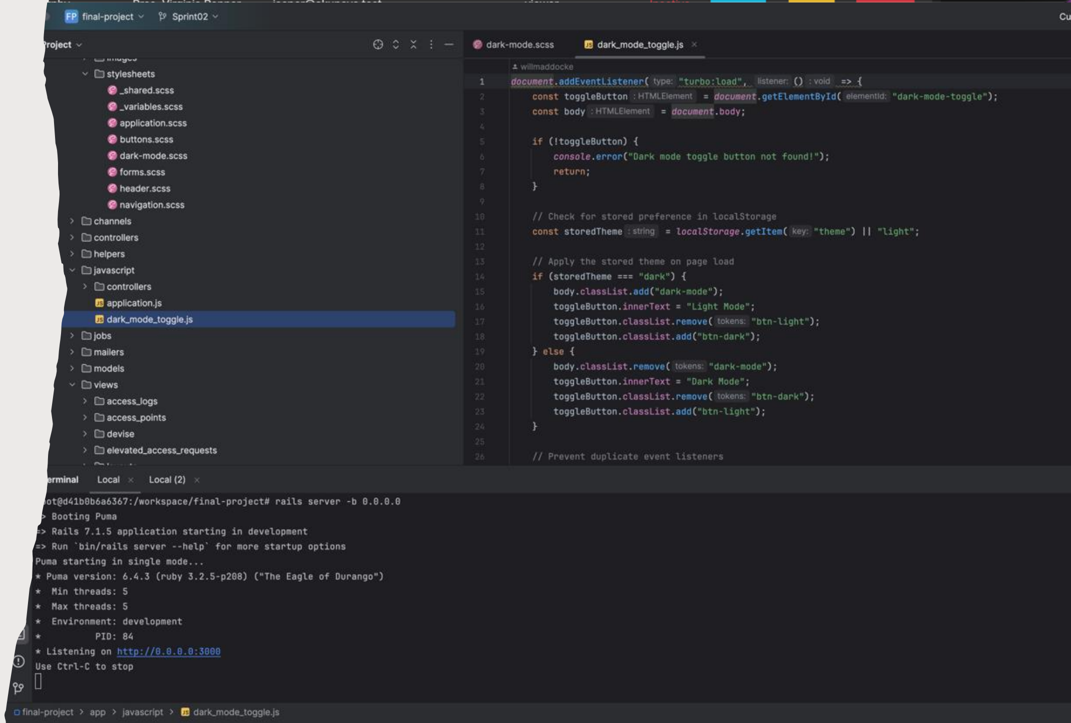
Note: Docker and Ruby on Rails setup followed the instructions provided in the linked Google Drive resource: Docker and Rails Setup.



user management, including status updates.

Full Name: Select Full Name
Email: Select Email
Role: Select Role
Status: All
Records per page: 10

	Full Name	Email	Role	Status	Actions
	Admin User	admin@example.com	admin	Active	Show Edit Delete
ager	Logistics Manager	logistics_manager@example.com	logistics_manager	Active	Show Edit Delete
	Roseanna Witting	trisha@marquardt-swaniawski.example	viewer	Active	Show Edit Delete
	Jerold Gottlieb LLD	angle@blanda.test	admin	Active	Show Edit Delete
ashirian	Lanie McCullough	scottie_rice@skiles-larson.example	viewer	Inactive	Show Edit Delete
uel	Jule Hackett	johnathan@swift.test	shipping_agent	Active	Show Edit Delete
	Quinn Wiza	aubrey.prosacco@greenholt.example	editor	Active	Show Edit Delete



```javascript
document.addEventListener( type: "turbo:load",  listener: () : void  => {
    const toggleButton : HTMLElement  = document.getElementById( elementId: "dark-mode-toggle");
    const body : HTMLElement  = document.body;

    if (!toggleButton) {
        console.error("Dark mode toggle button not found!");
        return;
    }

    // Check for stored preference in localStorage
    const storedTheme : string  = localStorage.getItem( key: "theme") || "light";

    // Apply the stored theme on page load
    if (storedTheme === "dark") {
        body.classList.add("dark-mode");
        toggleButton.innerText = "Light Mode";
        toggleButton.classList.remove( tokens: "btn-light");
        toggleButton.classList.add("btn-dark");
    } else {
        body.classList.remove( tokens: "dark-mode");
        toggleButton.innerText = "Dark Mode";
        toggleButton.classList.remove( tokens: "btn-dark");
        toggleButton.classList.add("btn-light");
    }

    // Prevent duplicate event listeners
```

```
ot@d41b0b6a6367:/workspace/final-project# rails server -b 0.0.0.0
> Booting Puma
=> Rails 7.1.5 application starting in development
=> Run `bin/rails server --help` for more startup options
Puma starting in single mode...
* Puma version: 6.4.3 (ruby 3.2.5-p208) ("The Eagle of Durango")
*  Min threads: 5
*  Max threads: 5
*  Environment: development
*          PID: 84
* Listening on http://0.0.0.0:3000
Use Ctrl-C to stop
```

Version Control

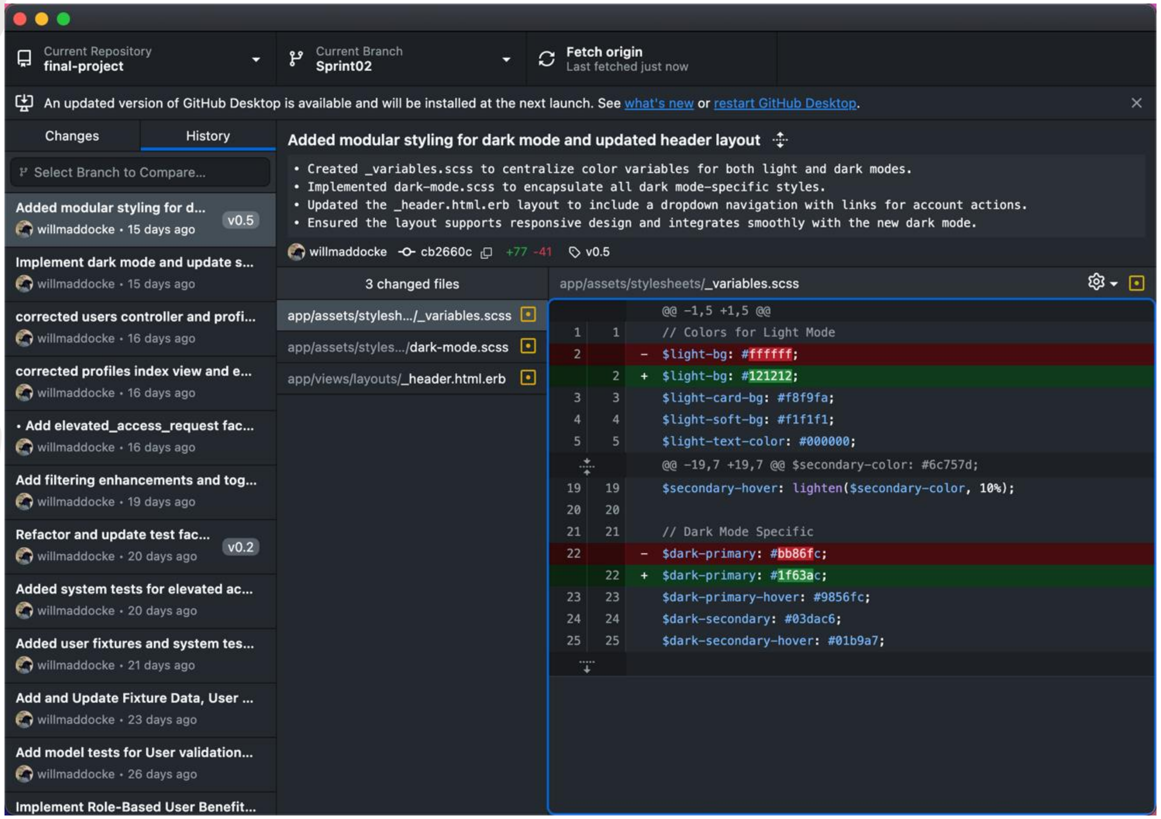
Purpose: Version control is essential for tracking changes and maintaining a reliable development history. It ensures that any mistakes can be reverted without losing progress.

Usage: The project is hosted on GitHub, where meaningful commit messages describe each change. Commit messages follow a clear format, such as:

> “Add user authentication functionality”
>
> “Fix bug in access request form validation”

Workflow: The branching strategy used includes:

> **Main Branch:** Stable version of the project, ready for deployment.
>
> **Sprint Branches:** Separate branches for Sprint01 and Sprint02.
>
> **Pull Requests:** Feature branches are reviewed and tested before merging into the main branch to ensure stability.



An updated version of GitHub Desktop is available and will be installed at the next launch. See what's new or restart GitHub Desktop.

Added modular styling for d... v0.5
willmaddocke • 15 days ago
Implement dark mode and update s...
willmaddocke • 15 days ago
corrected users controller and profi...
willmaddocke • 16 days ago
corrected profiles index view and e...
willmaddocke • 16 days ago
• Add elevated_access_request fac...
willmaddocke • 16 days ago
Add filtering enhancements and tog...
willmaddocke • 19 days ago
Refactor and update test fac... v0.2
willmaddocke • 20 days ago
Added system tests for elevated ac...
willmaddocke • 20 days ago
Added user fixtures and system tes...
willmaddocke • 21 days ago
Add and Update Fixture Data, User ...
willmaddocke • 23 days ago
Add model tests for User validation...
willmaddocke • 26 days ago
Implement Role-Based User Benefit...

Added modular styling for dark mode and updated header layout

```
• Created _variables.scss to centralize color variables for both light and dark modes.
• Implemented dark-mode.scss to encapsulate all dark mode-specific styles.
• Updated the _header.html.erb layout to include a dropdown navigation with links for account actions.
• Ensured the layout supports responsive design and integrates smoothly with the new dark mode.
```

willmaddocke cb2660c +77 -41 v0.5

app/assets/stylesh.../_variables.scss
app/assets/styles.../dark-mode.scss
app/views/layouts/_header.html.erb

app/assets/stylesheets/_variables.scss

```
@@ -1,5 +1,5 @@
 // Colors for Light Mode
-$light-bg: #ffffff;
+$light-bg: #121212;
 $light-card-bg: #f8f9fa;
 $light-soft-bg: #f1f1f1;
 $light-text-color: #000000;
@@ -19,7 +19,7 @@ $secondary-color: #6c757d;
 $secondary-hover: lighten($secondary-color, 10%);

 // Dark Mode Specific
-$dark-primary: #bb86fc;
+$dark-primary: #1f63ac;
 $dark-primary-hover: #9856fc;
 $dark-secondary: #03dac6;
 $dark-secondary-hover: #01b9a7;
```

Testing with RSpec

BDD and TDD: The application follows both Behavior-Driven Development (BDD) and Test-Driven Development (TDD). Using RSpec, BDD is applied by writing system tests that simulate user interactions with the web application, ensuring the behavior aligns with user stories. TDD is demonstrated by creating model tests to validate business logic and application integrity before implementing features.

Examples:

1. System Test:

   The spec/system/restricted_area_access_spec.rb file includes tests for:

   Requesting elevated access and receiving approval or denial.

   Viewing elevated access request details.

   Handling pagination and filtering of requests.

   Example 1:

   ```
   scenario "Shipping agent requests access and gets approved" do
     visit root_path
     # Test steps...
     expect(page).to have_text "Approved"
   End
   ```

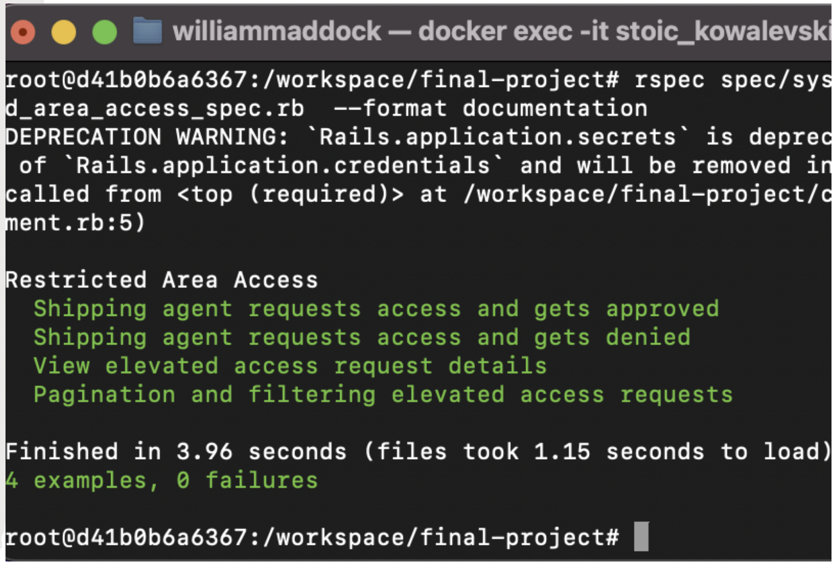
   This test validates end-to-end workflows, ensuring that roles interact with the system as intended.

2. Model Test:

   The spec/models/elevated_access_request_spec.rb file validates the ElevatedAccessRequest model for required attributes such as user, access_point, and reason.

   Example 2:

   ```
   context "without an access point" do
     it "is not valid" do
       elevated_access_request.access_point = nil
       expect(elevated_access_request).not_to be_valid
     end
   End
   ```



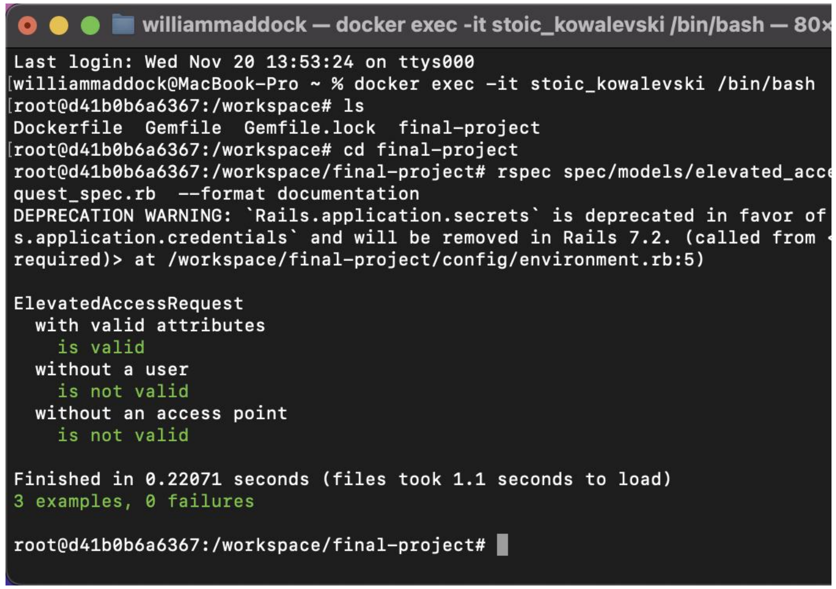
williammaddock — docker exec -it stoic_kowalevski

```
root@d41b0b6a6367:/workspace/final-project# rspec spec/sys
d_area_access_spec.rb  --format documentation
DEPRECATION WARNING: `Rails.application.secrets` is deprec
 of `Rails.application.credentials` and will be removed in
called from <top (required)> at /workspace/final-project/c
ment.rb:5)

Restricted Area Access
  Shipping agent requests access and gets approved
  Shipping agent requests access and gets denied
  View elevated access request details
  Pagination and filtering elevated access requests

Finished in 3.96 seconds (files took 1.15 seconds to load)
4 examples, 0 failures

root@d41b0b6a6367:/workspace/final-project# 
```



williammaddock — docker exec -it stoic_kowalevski /bin/bash — 80×

```
Last login: Wed Nov 20 13:53:24 on ttys000
[williammaddock@MacBook-Pro ~ % docker exec -it stoic_kowalevski /bin/bash
[root@d41b0b6a6367:/workspace# ls
Dockerfile  Gemfile  Gemfile.lock  final-project
[root@d41b0b6a6367:/workspace# cd final-project
root@d41b0b6a6367:/workspace/final-project# rspec spec/models/elevated_acce
quest_spec.rb  --format documentation
DEPRECATION WARNING: `Rails.application.secrets` is deprecated in favor of
s.application.credentials` and will be removed in Rails 7.2. (called from 
required)> at /workspace/final-project/config/environment.rb:5)

ElevatedAccessRequest
  with valid attributes
    is valid
  without a user
    is not valid
  without an access point
    is not valid

Finished in 0.22071 seconds (files took 1.1 seconds to load)
3 examples, 0 failures

root@d41b0b6a6367:/workspace/final-project# 
```

HTML/CSS

Purpose: The user interface for the web stack demonstrates the foundational design and responsiveness of the application, showcasing basic but functional styling. The design ensures clear navigation and intuitive interaction with the application's core features while adhering to industry standards for web development.

Examples:

1. **Navigation Bar:**

   Simple navigation bar styled for clarity, featuring links to key areas such as "Home," "Access Requests," and "Account."

   Example: Bootstrap's navbar component is used for responsive navigation across different screen sizes.

2. **Forms:**

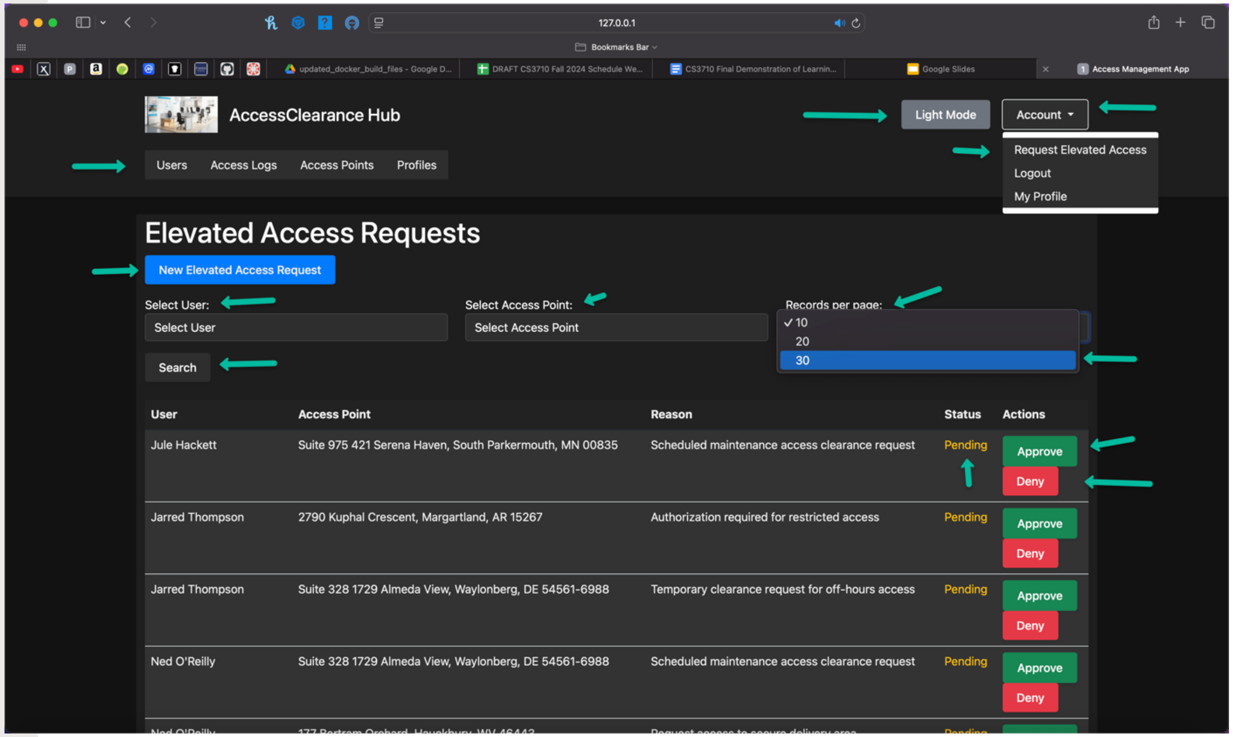
   Access request forms styled with Bootstrap's form-control and btn classes to ensure a consistent and user-friendly look.

   Example: Dropdown menus for access points and date pickers for request scheduling are styled for ease of use.

3. **Layouts:**

   The layout is built with Bootstrap's grid system, ensuring that the application adapts to various screen sizes, providing a responsive and clean user experience.

   Example: Single-column layout for mobile devices and multi-column layouts for desktop views.



Elevated Access Requests

New Elevated Access Request

Select User: Select User

Select Access Point: Select Access Point

Records per page: ✓ 10, 20, 30

Search

User	Access Point	Reason	Status	Actions
Jule Hackett	Suite 975 421 Serena Haven, South Parkermouth, MN 00835	Scheduled maintenance access clearance request	Pending	Approve Deny
Jarred Thompson	2790 Kuphal Crescent, Margartland, AR 15267	Authorization required for restricted access	Pending	Approve Deny
Jarred Thompson	Suite 328 1729 Almeda View, Waylonberg, DE 54561-6988	Temporary clearance request for off-hours access	Pending	Approve Deny
Ned O'Reilly	Suite 328 1729 Almeda View, Waylonberg, DE 54561-6988	Scheduled maintenance access clearance request	Pending	Approve Deny
Ned O'Reilly	177 Bertram Orchard, Hawkbury, WV 46443	Request access to secure delivery area	Pending	

Accessibility

Principles: The application follows the Web Content Accessibility Guidelines (WCAG) to ensure inclusivity and usability for all users, including those with disabilities. Key principles include:

Perceivable: Ensuring all information is presented in ways users can perceive (e.g., text alternatives for non-text content).

Operable: Designing interfaces usable by keyboard and assistive technologies.

Understandable: Creating predictable navigation and error handling.

Robust: Ensuring compatibility with a wide range of user agents, including assistive technologies.

Examples:

1. **ARIA Labels:**

Added ARIA labels to form elements for users relying on screen readers.

Example: Buttons and dropdowns include descriptive labels such as aria-label="Submit Access Request".

2. **Keyboard Navigation:**

Ensured all interactive elements (e.g., links, buttons, and form fields) are accessible via keyboard navigation (e.g., Tab and Enter keys).

Focus indicators are visible to guide users through the interface.

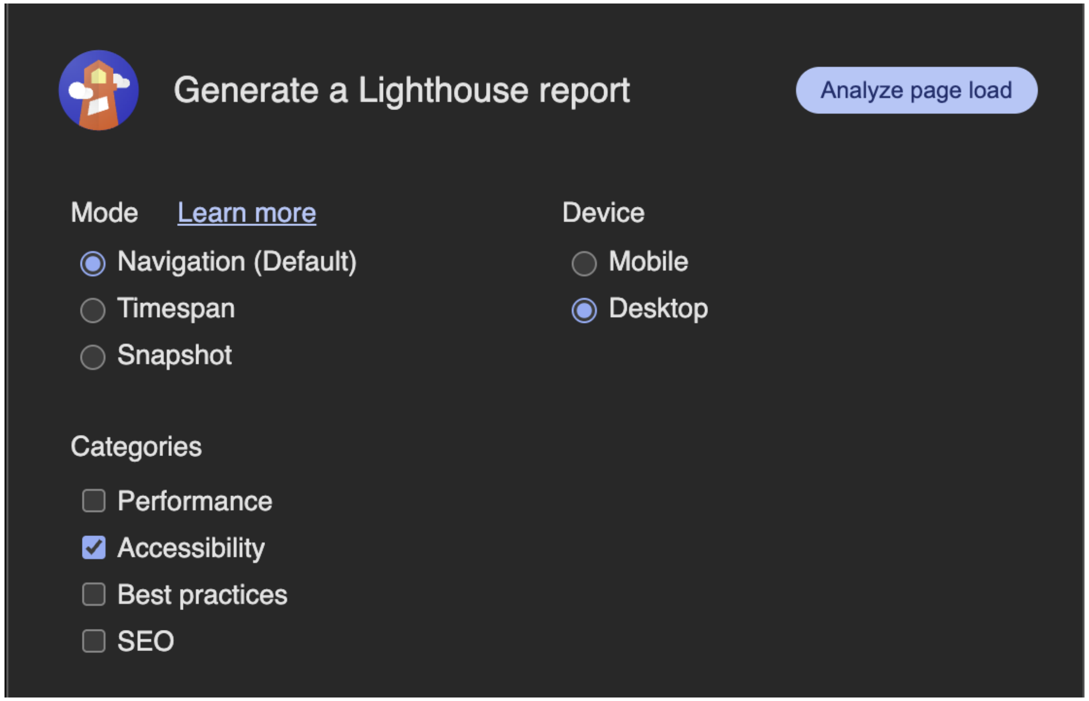
3. **Color Contrast:**

Used Bootstrap's accessible color palette to maintain sufficient contrast ratios for text and UI components.

Example: Buttons and text meet the contrast requirements for readability.

4. **Error Feedback:**

Clear error messages provided in forms, with alerts accessible via screen readers.



Generate a Lighthouse report

Analyze page load

Mode Learn more

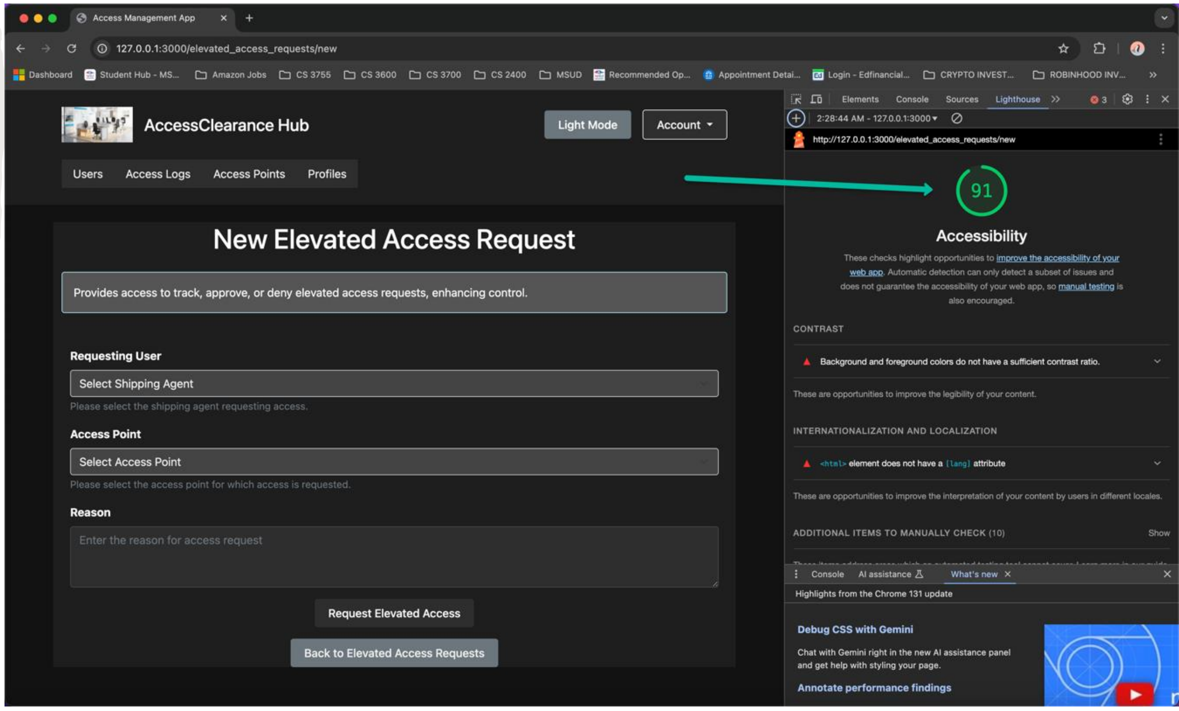
- [✓ Navigation (Default)]
- Timespan
- Snapshot

Device

- Mobile
- [✓ Desktop]

Categories

- Performance
- [✓ Accessibility]
- Best practices
- SEO



Access Management App

127.0.0.1:3000/elevated_access_requests/new

Dashboard · Student Hub - MS... · Amazon Jobs · CS 3755 · CS 3600 · CS 3700 · CS 2400 · MSUD · Recommended Op... · Appointment Detai... · Login - Edfinancial... · CRYPTO INVEST... · ROBINHOOD INV...

AccessClearance Hub

Light Mode · Account

Users · Access Logs · Access Points · Profiles

New Elevated Access Request

Provides access to track, approve, or deny elevated access requests, enhancing control.

Requesting User

Select Shipping Agent

Please select the shipping agent requesting access.

Access Point

Select Access Point

Please select the access point for which access is requested.

Reason

Enter the reason for access request

Request Elevated Access

Back to Elevated Access Requests

Elements · Console · Sources · Lighthouse

2:28:44 AM - 127.0.0.1:3000

http://127.0.0.1:3000/elevated_access_requests/new

91

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.

CONTRAST

▲ Background and foreground colors do not have a sufficient contrast ratio.

These are opportunities to improve the legibility of your content.

INTERNATIONALIZATION AND LOCALIZATION

▲ `<html>` element does not have a `[lang]` attribute

These are opportunities to improve the interpretation of your content by users in different locales.

ADDITIONAL ITEMS TO MANUALLY CHECK (10) Show

Console · AI assistance · What's new

Highlights from the Chrome 131 update

Debug CSS with Gemini

Chat with Gemini right in the new AI assistance panel and get help with styling your page.

Annotate performance findings

Security

Principles: The application incorporates essential security measures to protect user data and ensure a secure user experience:

User Authentication: Implemented using the devise gem for secure login, registration, and session management.

Data Validation: Enforced model validations to prevent invalid or malicious data input.

Role-Based Access Control (RBAC): Defined user roles (admin, viewer, shipping_agent, etc.) to restrict permissions and safeguard sensitive operations.

Examples:

1. **Authentication with Devise:**

Code Snippet (from app/models/user.rb):

```
devise :database_authenticatable, :registerable,
     :recoverable, :rememberable, :validatable
```

This configuration ensures secure password handling, account recovery, and session persistence.

Registration Form (from app/views/devise/registrations/new.html.erb):

The form incorporates proper validations and secure password entry fields:

```
<%= f.password_field :password, autocomplete: "new-password", class: "form-control" %>
```

```
<%= f.password_field :password_confirmation, autocomplete: "new-password", class: "form-control" %>
```

2. **Role-Based Access Control (RBAC):**

User roles are defined using enum in the User model, ensuring controlled access to various functionalities:

```
enum role: { admin: 0, editor: 1, viewer: 2, shipping_agent: 3, logistics_manager: 4 }
```

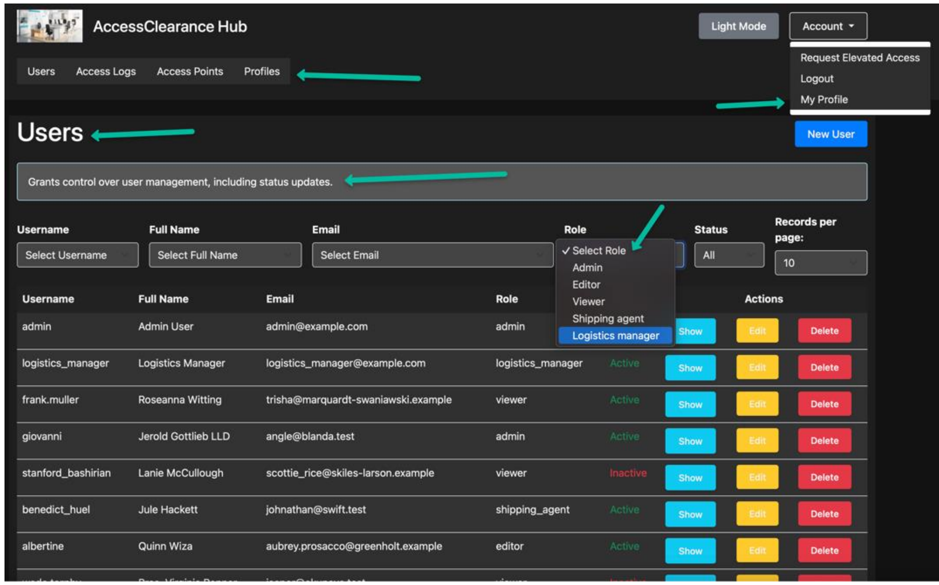
Custom methods like can_request_access? and can_edit_items? enable granular permission checks, protecting sensitive actions.

3. **Data Validation:**

The User model enforces strict validation of critical fields:

```
validates :email, presence: true, uniqueness: true
validates :username, presence: true, uniqueness: true
validates :role, inclusion: { in: roles.keys }
```

This prevents duplicate records, invalid emails, or unauthorized role assignments.



Grants control over user management, including status updates.

Username	Full Name	Email	Role		Actions		
admin	Admin User	admin@example.com	admin		Show	Edit	Delete
logistics_manager	Logistics Manager	logistics_manager@example.com	logistics_manager	Active	Show	Edit	Delete
frank.muller	Roseanna Witting	trisha@marquardt-swaniawski.example	viewer	Active	Show	Edit	Delete
giovanni	Jerold Gottlieb LLD	angle@blanda.test	admin	Active	Show	Edit	Delete
stanford_bashirian	Lanie McCullough	scottie_rice@skiles-larson.example	viewer	Inactive	Show	Edit	Delete
benedict_huel	Jule Hackett	johnathan@swift.test	shipping_agent	Active	Show	Edit	Delete
albertine	Quinn Wiza	aubrey.prosacco@greenholt.example	editor	Active	Show	Edit	Delete

Model-View-Controller

Explanation:

The Model-View-Controller (MVC) architecture organizes application logic into three interconnected components:

1. **Model**: Manages the data, logic, and rules of the application. In this application:

User: Represents users with attributes and methods for authentication and permissions.

Profile: Stores detailed user information such as personal and professional data.

ElevatedAccessRequest: Handles access request data, statuses, and associations with users and access points.

2. **View:** Presents data to the user and collects input. In this application:

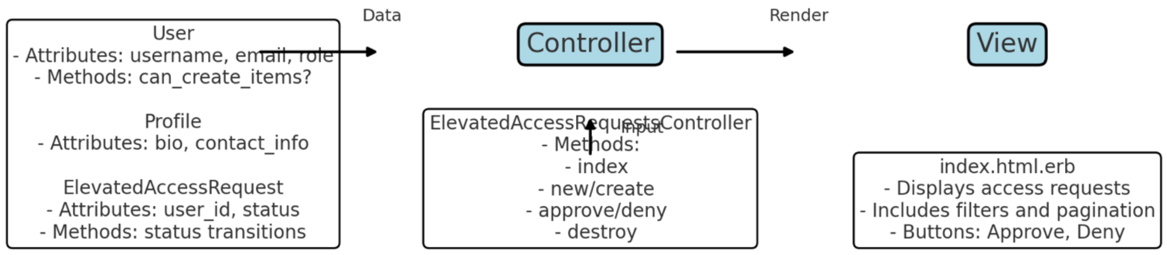
index.html.erb: Displays a list of elevated access requests, forms for filtering/searching, and actions like approve/deny.

3. **Controller:** Processes user inputs, interacts with the model, and selects the view to render. In this application:

ElevatedAccessRequestsController: Manages CRUD operations for elevated access requests, ensures user permissions, and handles approvals/denials.

This separation improves code modularity, making the application easier to maintain and scale. For instance, the ElevatedAccessRequestsController processes user actions (e.g., approve/deny), while the ElevatedAccessRequest model ensures data integrity, and the index.html.erb view ensures that users only see what they need.

Diagram: Custom diagram showing interactions between Model, View, and Controller.



Data
Controller
Render
View
User
- Attributes: username, email, role
- Methods: can_create_items?
Profile
- Attributes: bio, contact_info
ElevatedAccessRequest
- Attributes: user_id, status
- Methods: status transitions
ElevatedAccessRequestsController
Input
- Methods:
- index
- new/create
- approve/deny
- destroy
index.html.erb
- Displays access requests
- Includes filters and pagination
- Buttons: Approve, Deny

Database Overview

Purpose: The database schema serves as the backbone of the application, organizing and structuring data to support functionality. It facilitates secure access management, user profiles, and record tracking. The key components of the schema include:

Users: Stores user information such as credentials, roles, and access levels, enabling role-based permissions and authentication.

Profiles: Extends user data with personal details like bio and avatar.

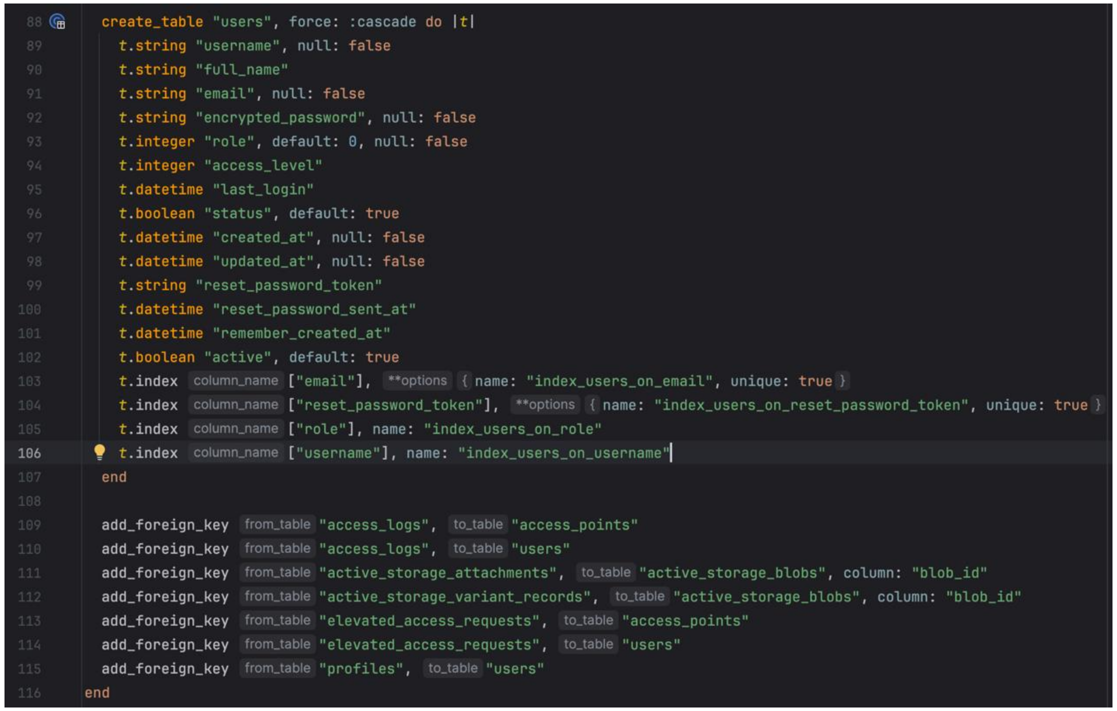
Elevated Access Requests: Tracks requests for access to restricted areas, associating each request with a user and an access point. Includes status management for approval workflows.

Access Points: Represents physical or virtual locations with an associated access level, aiding in access control logic.

Access Logs: Records user access attempts with timestamps and success status, enabling audit trails and monitoring.

These tables are interconnected through foreign key relationships, ensuring data integrity and enforcing referential constraints.

Diagram: Entity-Relationship Diagram (ERD) or **schema screenshot**.



```
  create_table "users", force: :cascade do |t|
    t.string "username", null: false
    t.string "full_name"
    t.string "email", null: false
    t.string "encrypted_password", null: false
    t.integer "role", default: 0, null: false
    t.integer "access_level"
    t.datetime "last_login"
    t.boolean "status", default: true
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "reset_password_token"
    t.datetime "reset_password_sent_at"
    t.datetime "remember_created_at"
    t.boolean "active", default: true
    t.index column_name ["email"], **options { name: "index_users_on_email", unique: true }
    t.index column_name ["reset_password_token"], **options { name: "index_users_on_reset_password_token", unique: true }
    t.index column_name ["role"], name: "index_users_on_role"
    t.index column_name ["username"], name: "index_users_on_username"
  end

  add_foreign_key from_table "access_logs", to_table "access_points"
  add_foreign_key from_table "access_logs", to_table "users"
  add_foreign_key from_table "active_storage_attachments", to_table "active_storage_blobs", column: "blob_id"
  add_foreign_key from_table "active_storage_variant_records", to_table "active_storage_blobs", column: "blob_id"
  add_foreign_key from_table "elevated_access_requests", to_table "access_points"
  add_foreign_key from_table "elevated_access_requests", to_table "users"
  add_foreign_key from_table "profiles", to_table "users"
end
```

Object and Database Relationship

Explanation:

In Rails, Object-Relational Mapping (ORM) is managed through Active Record, which connects Ruby objects to database tables. Each model represents a table, and attributes correspond to columns in that table. Relationships between objects are defined using methods like belongs_to, has_many, and has_one, enabling complex interactions.

For example, in our application:

The User model represents the users table.

The ElevatedAccessRequest model represents the elevated_access_requests table.

Relationships like "a user has many elevated access requests" are defined in the model files, bridging the database and object layers.

Visuals: Code snippet showing a model's relationship definitions.

Code Example:

The following snippet demonstrates ORM relationships and definitions for User and ElevatedAccessRequest models.

User Model (app/models/user.rb):

```
class User < ApplicationRecord
  # Associations
  has_many :elevated_access_requests
  has_one :profile

  # Enum for roles
  enum role: { standard: 0, logistics_manager: 1, shipping_agent: 2 }

  # Validations
  validates :username, presence: true, uniqueness: true
  validates :email, presence: true, uniqueness: true
end
```

ElevatedAccessRequest Model (app/models/elevated_access_request.rb):

```
class ElevatedAccessRequest < ApplicationRecord
  # Associations
  belongs_to :user
  belongs_to :access_point

  # Scopes for status filtering
  scope :pending, -> { where(status: 'pending') }
  scope :approved, -> { where(status: 'approved') }
  scope :denied, -> { where(status: 'denied') }

  # Validations
  validates :reason, presence: true
End
```

Relationship Mapping:

1. **Database Relationship:**

users table has a one-to-many relationship with the elevated_access_requests table.

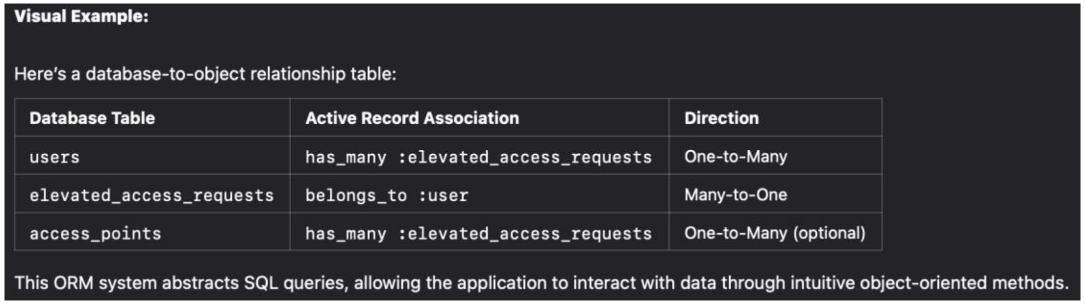
elevated_access_requests connects to access_points via foreign keys.

These relationships are enforced with t.references and add_foreign_key in migrations.

2. **Object Interaction:**

A User object can fetch its related elevated access requests with user.elevated_access_requests.

ElevatedAccessRequest retrieves its associated User with request.user.



Visual Example:

Here's a database-to-object relationship table:

Database Table	Active Record Association	Direction
`users`	`has_many :elevated_access_requests`	One-to-Many
`elevated_access_requests`	`belongs_to :user`	Many-to-One
`access_points`	`has_many :elevated_access_requests`	One-to-Many (optional)

This ORM system abstracts SQL queries, allowing the application to interact with data through intuitive object-oriented methods.

Reflection

Learning Outcomes: Throughout the development of this application, I gained significant experience in several key areas of software development:

1. **Full-Stack Development:**

I worked on both the front-end and back-end of the application. This included setting up models, migrations, and controllers for the database, as well as implementing user interfaces with HTML, CSS, and JavaScript.

I also utilized Rails as a backend framework, including its Active Record ORM to manage data, and incorporated Devise for user authentication.

2. **Database Management:**

I developed and maintained an efficient database schema to support the app's features, with a focus on relationships between models, data integrity, and optimization.

I gained proficiency in writing migrations and managing relationships using Active Record associations like has_many, belongs_to, and has_one.

3. **Testing:**

I gained experience in writing and running tests using tools like RSpec and Capybara to ensure the application functions correctly.

I also applied Test-Driven Development (TDD) principles to ensure that new features were implemented with a high level of code reliability.

4. **Agile Development:**

The development process was iterative, with frequent feedback loops that allowed for continuous improvement. I practiced agile methodologies, using tools like Trello to manage tasks and user stories.

5. **Security Practices:**

Implementing secure user authentication with Devise taught me the importance of password encryption, session management, and other security practices such as protection against SQL injection and cross-site scripting (XSS).

Future Improvements:

1. **Real-Time Notifications:**
Implementing a notification system that alerts users about changes to their access requests (approved, denied, etc.) in real-time using technologies like ActionCable or WebSockets.
2. **Role-Based Dashboards:**
Developing more customized dashboards based on user roles (e.g., administrators, logistics managers, standard users) would enhance the user experience and provide more focused access control.
3. **Improved Analytics and Reporting:**
Adding more sophisticated reporting features, such as generating reports on user access history, request statuses, and frequent access points, would provide more insights for administrators and help in decision-making.
4. **Mobile Optimization:**
Optimizing the user interface for mobile devices could enhance usability, particularly for users who need to access the application on the go. Using frameworks like Bootstrap or Tailwind CSS could aid in achieving this.
5. **Multi-Factor Authentication (MFA):**
Adding multi-factor authentication to improve security for users, particularly for those with elevated access levels, would be a valuable addition to the app.

By reflecting on these experiences and challenges, I can continue to evolve as a developer and refine the features and user experience of the app for future iterations.

Challenges and Solutions:

1. **Challenge:** Managing complex relationships between tables, such as users and access requests, was initially difficult, especially in terms of ensuring data integrity and proper foreign key constraints.
Solution: I took time to understand how Active Record associations and foreign key constraints work. I then refactored and structured the migrations to ensure that the relationships were properly defined and maintained across the app. Additionally, I used tools like rails db:rollback to iteratively test and adjust the schema.
2. **Challenge:** Implementing user authentication with Devise posed a challenge, particularly with configuring password resets and managing user sessions securely.
Solution: I closely followed Devise documentation and used online forums and tutorials to troubleshoot specific issues. I also wrote tests to ensure that the authentication system worked seamlessly, which helped pinpoint any bugs or configuration issues.
3. **Challenge:** Implementing real-time access log tracking for users was more complex than expected due to managing timestamps and frequent updates.
Solution: I solved this by breaking down the process into smaller steps and using background jobs to handle logging asynchronously. I also utilized database indexes to improve the speed of read operations on log data.
4. **Challenge:** Ensuring the application remained responsive and user-friendly, especially with dynamic forms for access requests.
Solution: I used JavaScript and AJAX to improve user experience by asynchronously submitting forms and updating UI elements without needing full-page reloads.