Final Project Sprint 01

Effort: **Individual**
Points: 50 pts
Deliverables: Upload this document as Word document or PDF containing answers and separate video file . Do not upload both video and document in a zip file.

Goals:

ABET Outcome 2: Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.

- **Create** effective and efficient web app **designs** that include.
  - Develop solutions utilizing OOP principles and relationship
  - Develop unit, integration and system testing that are automated following industry best practices
- **Implements** the **solution** effectively using agile methodologies, web app theories, design patterns and architectures.
- **Evaluate** and **test** to ensure it meets requirements and functions as intended.

Final Web App Project

1 Web App Functionality

App should contain the following **unless you have permission** from me to do something different
Highlight below what you completed in iteration 01 when you submit

- Create new Rails App
- App must have at least 2 models that contain
  - at least 2 required attributes for each model
  - At least 4 different data types
  - One of the following relationships

        - One to one
        - One to many
- One stakeholder: A user that has an association with at least another item (model) where the user can
        - Register
        - Logout
        - Login
        - Create a new item
        - Edit item
        - Destroy Item
- Another stakeholder: Should be able to view and search items (do not need to log in but can have requirement that other stakeholder also must log in)
    - App should show list of all items on page
    - Display information about each item individually
- Implement UI
    - Using Bootstrap
    - Responsive to different size screens
    - Design for Accessibility
        - ARIA Labels
        - Dos and don'ts on designing for accessibility
- Implement Rspec Testing
    - 4 unit tests for model
    - 4 System Tests with Capybara
        - 2 different user story tests with happy and sad path scenarios
- Feature Research, Design and Implementation

2 Project Planning and Design (10)

Create a public github Repository
Link to repository:

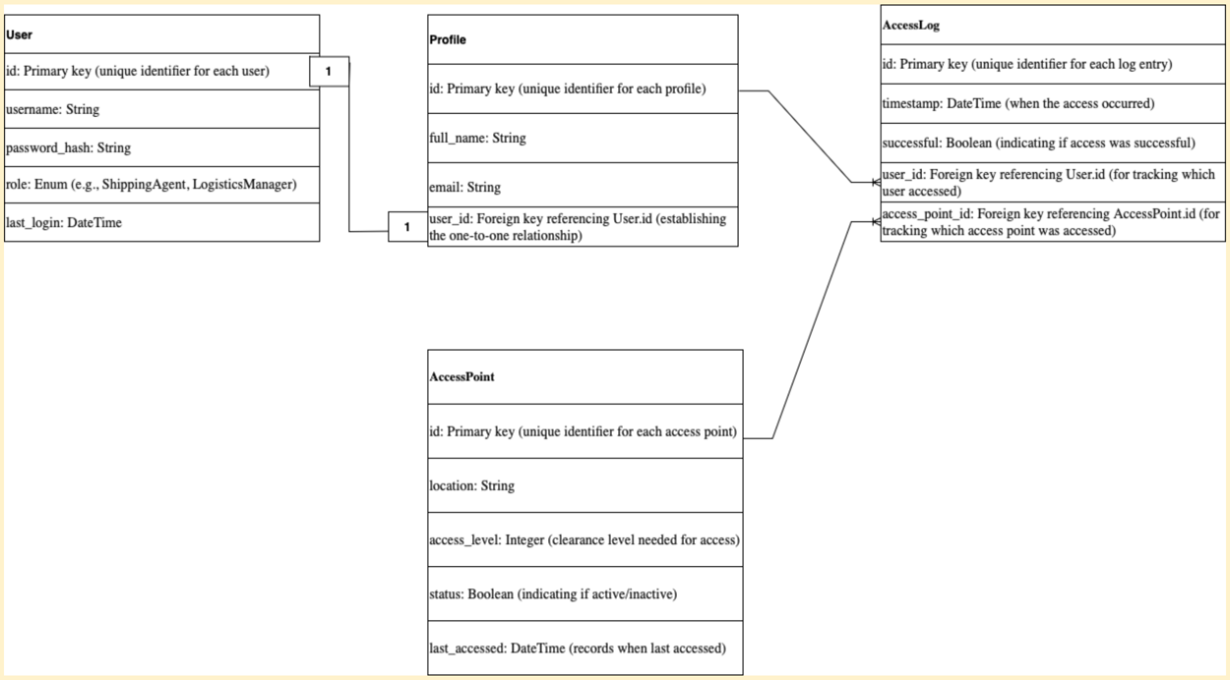
https://github.com/willmaddock/final-project/tree/Sprint01

Create a Backlog of Issues for your User Stories
Link to GitHub Issues:

https://github.com/willmaddock/final-project/issues

2.1 Design the Data Structures and Routes

Include UML model and describe


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)

- Model Validations

Model Validations

- **User Model**:
  username: Presence and uniqueness validation.
  password_hash: Presence validation.
  role: Inclusion validation to ensure it matches predefined roles.

- **Profile Model**:
  full_name and email: Presence validation.
  email: Uniqueness and format validation.
  user_id: Presence validation with uniqueness constraint to enforce one-to-one relationship.

- **AccessPoint Model**:
  location: Presence and uniqueness validation.
  access_level: Presence validation, ensuring it is an integer.
  status: Presence validation.

- **AccessLog Model**:
  timestamp: Presence validation.
  user_id and access_point_id: Presence validation, ensuring valid associations.

Model Relationships

User - Profile: One-to-One (User has one Profile).

User - AccessLog: One-to-Many (User has many AccessLogs).

AccessPoint - AccessLog: One-to-Many (AccessPoint has many AccessLogs).

Routes

User Routes:
GET /users: List all users.
POST /users: Create a new user.
GET /users/:id: Show user details.
PATCH /users/:id: Update user information.
DELETE /users/:id: Delete user.

Profile Routes:
GET /profiles/:id: Show a user's profile.
PATCH /profiles/:id: Update a user's profile.

AccessPoint Routes:
GET /access_points: List all access points.
POST /access_points: Create a new access point.
GET /access_points/:id: Show access point details.
PATCH /access_points/:id: Update access point.
DELETE /access_points/:id: Delete access point.

AccessLog Routes:
GET /access_logs: List all access logs.
POST /access_logs: Record a new access attempt.
GET /access_logs/:id: Show specific access log details.

2.2 Rspec Testing

Identify the 4 unit tests you will write for a model

The model being tested in the spec/models/user_spec.rb file is the User **model**. This model includes the following functionalities and attributes:

1. Validation of Essential Fields

Ensures that required fields, like username and email, are present for the model to be valid.

2. Uniqueness of Fields

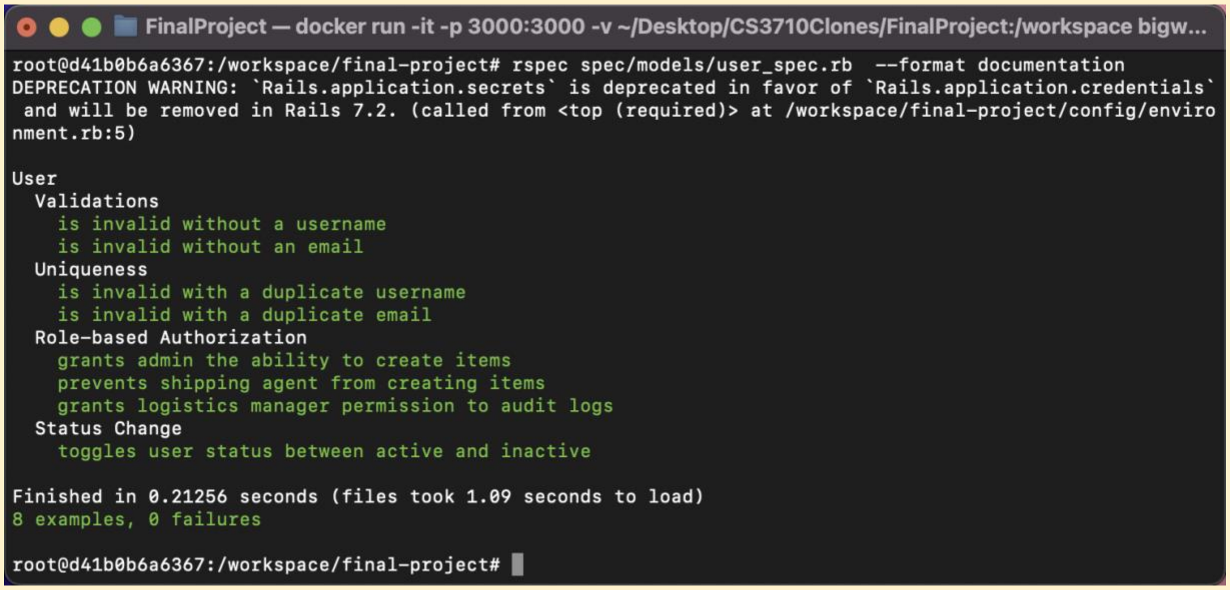
Checks that fields such as username and email are unique, ensuring no duplicate usernames or emails.

3. Role-Based Authorization

Verifies that specific roles (e.g., admin, shipping_agent, and logistics_manager) have the appropriate permissions. For instance:
Admins can create items.
Shipping agents cannot create items.
Logistics managers can audit logs.

4. Status Change

Tests the active attribute toggle functionality, which switches the user's status between active and inactive.


FinalProject — docker run -it -p 3000:3000 -v ~/Desktop/CS3710Clones/FinalProject:/workspace bigw...

```
root@d41b0b6a6367:/workspace/final-project# rspec spec/models/user_spec.rb  --format documentation
DEPRECATION WARNING: `Rails.application.secrets` is deprecated in favor of `Rails.application.credentials`
 and will be removed in Rails 7.2. (called from <top (required)> at /workspace/final-project/config/enviro
nment.rb:5)

User
  Validations
    is invalid without a username
    is invalid without an email
  Uniqueness
    is invalid with a duplicate username
    is invalid with a duplicate email
  Role-based Authorization
    grants admin the ability to create items
    prevents shipping agent from creating items
    grants logistics manager permission to audit logs
  Status Change
    toggles user status between active and inactive

Finished in 0.21256 seconds (files took 1.09 seconds to load)
8 examples, 0 failures

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

Identify the System Tests
List the 2 user stories with the steps for happy and sad path scenarios

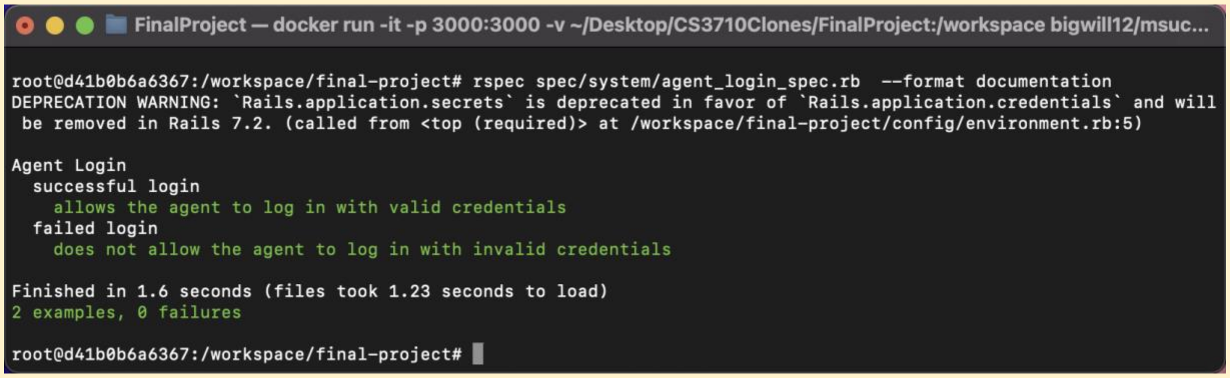
1st system test (completed both spec and test):
spec/system/agent_login_spec.rb
test/system/agent_login_test.rb

System Test 1: User Story - Agent Logs in with Secure Credentials

1. **Scenario 1: Agent successfully logs in (Happy Path)**
    Given the agent has valid login credentials
    When they enter their username and password
    Then they gain access to the dashboard

2. **Scenario 2: Agent enters incorrect password (Sad Path)**
    Given the agent has invalid login credentials
    When they enter an incorrect password

Then they receive an “Invalid credentials” message and cannot proceed.


FinalProject — docker run -it -p 3000:3000 -v ~/Desktop/CS3710Clones/FinalProject:/workspace bigwill12/msuc...

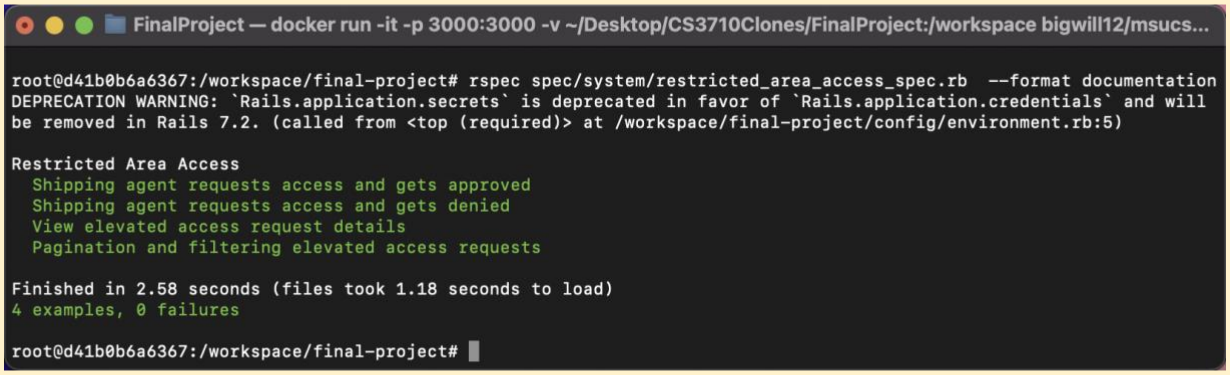
```
root@d41b0b6a6367:/workspace/final-project# rspec spec/system/agent_login_spec.rb  --format documentation
DEPRECATION WARNING: `Rails.application.secrets` is deprecated in favor of `Rails.application.credentials` and will
 be removed in Rails 7.2. (called from <top (required)> at /workspace/final-project/config/environment.rb:5)

Agent Login
  successful login
    allows the agent to log in with valid credentials
  failed login
    does not allow the agent to log in with invalid credentials

Finished in 1.6 seconds (files took 1.23 seconds to load)
2 examples, 0 failures

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

2nd system test (completed both spec and test):
spec/system/restricted_area_access_spec.rb
test/system/restricted_area_access_test.rb

System Test 2: User Story - Agent Accesses Restricted Areas for Delivery

1. **Scenario 1: Agent has the necessary clearance (Happy Path)**
   Given the agent has clearance for elevator and stairwell access
   When they request access to a restricted area (elevator or stairwell)
   Then the shipping agent waits for the logistics manager to approves it.
   And logistics manager grants them access, allowing them to proceed without delays.

2. **Scenario 2: Agent lacks necessary clearance (Sad Path)**
   Given the agent does not have the required clearance for certain floors or areas
   When they attempt access to these areas
   Then the system checks their credentials
   And logistics manager denies access with a notification explaining the clearance issue.


FinalProject — docker run -it -p 3000:3000 -v ~/Desktop/CS3710Clones/FinalProject:/workspace bigwill12/msucs...

```
root@d41b0b6a6367:/workspace/final-project# rspec spec/system/restricted_area_access_spec.rb  --format documentation
DEPRECATION WARNING: `Rails.application.secrets` is deprecated in favor of `Rails.application.credentials` and will
be removed in Rails 7.2. (called from <top (required)> at /workspace/final-project/config/environment.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 2.58 seconds (files took 1.18 seconds to load)
4 examples, 0 failures

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

2.3 UI

Design for accessibility

Users	**Design Practice to be Implemented**
autistic spectrum	**Design Practices to Implement:** Avoid overloading pages with information or distracting elements. Use simple, clear language and concise instructions. Create a predictable layout with consistent navigation and visual hierarchy. Avoid animations or auto-playing media that may be overwhelming. **Design Benefit:** This reduces cognitive load, making the interface easier for users to understand and navigate.
screen readers	**Design Practices to Implement:** Provide descriptive and unique labels for all interactive elements (e.g., buttons, forms). Use semantic HTML to allow screen readers to interpret the content correctly.

	Structure headings logically (H1 for page title, H2 for sections) to help users navigate. Add alt text for images and ensure links make sense out of context (e.g., avoid “click here”). **Design Benefit:** Ensures that visually impaired users can navigate and understand content accurately.
low vision	**Design Practices to Implement:** Use high-contrast color schemes for text, buttons, and backgrounds. Allow font resizing or provide zoom options for better readability. Avoid using color as the only way to convey information (e.g., use icons or patterns). Ensure that links and buttons are large enough to be easily clicked. **Design Benefit:** Enhances readability and interaction for users who have limited vision.
physical or motor disabilities	**Design Practices to Implement:** Create large, easy-to-click buttons and provide ample spacing between clickable elements. Ensure full keyboard navigation for all elements (form fields, menus, buttons). Avoid features requiring precise control, such as drag-and-drop or swiping. Limit the need for timed responses to prevent added pressure.

	Design Benefit: Improves ease of use for users with limited fine motor skills or tremors.
D/deaf or hard of hearing	**Design Practices to Implement:** Provide captions for all video content and transcriptions for audio-only media. Avoid relying solely on sound to convey information; use visual cues as well. For alerts or notifications, use visual indicators in addition to or instead of sounds. **Design Benefit:** Ensures that important content and feedback are accessible without sound.
dyslexia	**Design Practices to Implement:** Use simple, readable fonts and avoid overly decorative typefaces. Allow for customization of text spacing to improve readability. Avoid large blocks of text; use bullet points, images, or icons to break up content. Left-align text and avoid justified text to prevent uneven spacing. **Design Benefit:** Makes reading content easier and reduces visual strain for dyslexic users.

2 Sprint 01 Project Development and Demonstration

2.1 Project Task Management and Version Control (10)

Break into Tasks
Decide what user stories you will implement for this sprint and the tasks.
Suggestion: focus on functionality and unit tests for two models

List User Story tasks and development tasks for this spring. Include a time estimate to help you budget your time. You can either put it below or create a task board that contains a user story number with a task and time estimate.

Example Tasks
- Models/Database set up,
- Model validations and relationships
- TDD rspec model unit testing
- Routes and Controller
- Views
- Search
- Seed Database
- UI and Accessibility
- Model Testing
- Rspec System testing

User Story and Development tasks	Tasks	Time Estimate
Development Set Up		
1 User story		

Create a **public** Project board in github for Sprint 01 with the following columns

Sprint Backlog	To Do	In Process	Done

Add tasks for the user stories

Link to public project github task board: https://github.com/users/willmaddock/projects/2

You will also create a new github repository for using git to version your project. Follow version control practices.

Link to public project github repository https://github.com/willmaddock/final-project/tree/Sprint01
How you used version control with git and github 1. **Commit Frequency:** I commit frequently after completing a feature or significant change, typically once per day. 2. **Commit Messages:** I use clear messages like: "Add [feature]" "Fix [issue]" "Refactor [code]" "Update [doc/config]" 3. **Branches:** I create a new branch for each feature or bug fix to keep the main branch clean. 4. **Managing and Merging Changes:** I pull the latest changes regularly and resolve conflicts. I use pull requests for review before merging.

5. **Collaboration:**
I use pull requests for new features or fixes and GitHub Issues to track tasks and bugs.

First Push to Version v0.0:
I created the repository and initialized the project, then made the first commit:
“Initial commit with basic project setup”
This commit is labeled as v0.0, marking the starting point of version control.
The final version of sprint01 will be v0.4.

2.2 Development and Testing (20)

Give a description and the commands for the following

How you created a new docker instance.

1. **Project Directory Setup**: First, I created a new folder for the project (e.g., Final-Project) and placed updated Docker build files in this directory.

2. **Building the Docker Image**: In the terminal, I navigated to my project directory and built the Docker image with:

```
docker buildx build -t your_dockerhub_username/your_app_name .
```

3. **Running the Docker Container**: I started the container using:

```
docker run -it -p 3000:3000 -v "$(pwd):/workspace"
your_dockerhub_username/your_app_name
```

Here is the link to my M06 technical documentation:
https://github.com/rileymck/WebDev-2024/blob/main/M06-William-Creating-A-Branch.pdf

How you installed rails

Rails was installed using the Rails gem within Docker, using updated Docker files to ensure all dependencies aligned with Rails setup:

```
rails new access_management_app --skip-bundle
cd access_management_app
bundle install
```

Later updated it using Deb’s way. Here is the link in canvas.
https://msudenver.instructure.com/courses/98620/discussion_topics/1386406

How you created your models and set up database

Generating Models: I created models using rails generate model commands, based on the project requirements.

User Model, View, and Controller Scaffold:

```
rails generate scaffold User username:string password_hash:string
full_name:string email:string role:string access_level:integer
last_login:datetime status:boolean
```

Profile Model, View, and Controller Scaffold:

```
rails generate scaffold Profile user:references bio:text location:string
avatar:string
```

Access Point Model, View, and Controller Scaffold:

```
rails generate scaffold AccessPoint location:string access_level:integer
description:text status:Boolean
```

Access Log Model, View, and Controller Scaffold:

```
rails generate scaffold AccessLog user:references access_point:references
timestamp:datetime successful:Boolean
```

Elevated Access Request Model, View, and Controller Scaffold:

```
rails generate scaffold ElevatedAccessRequest user:references
access_point:references status:string
```

Database Setup: I ran the following commands to set up the database:

```
rails db:create
rails db:migrate
```

This process created necessary tables and relationships for the User, AccessPoint, and other models.

Problems Solving

Describe an issue you encountered and explain what you did to resolve the issue.

Issue: One issue I encountered was an error with ActiveModel::UnknownAttributeError for attributes not defined in my models.

Resolution: To fix this, I verified attribute names and confirmed they matched those in the schema and migration files, which resolved the issue.

What resources helped you the most for completing sprint 01?

Helpful Resources: The Rails documentation and Docker setup guides were especially useful, as well as Stack Overflow for debugging errors.

Here are links to some resources that provide guidance on resolving ActiveModel::UnknownAttributeError and understanding attributes in Rails models:

1. **Rails Documentation on Active Record Validations and Callbacks**
Active Record Basics - Rails Guide
This guide covers Active Record validations, how models interact with the database, and tips on defining attributes correctly.

2. **Stack Overflow: Common Causes of UnknownAttributeError in Rails**
UnknownAttributeError in Rails Models - Stack Overflow

This discussion includes various common causes and solutions for UnknownAttributeError, such as mismatches between migration files and model definitions.

3. **Rails Migration Guide**
Active Record Migrations - Rails Guide
Understanding migrations is key to managing attributes in Rails. This guide explains how to create, modify, and roll back migrations to keep the database schema in sync with model definitions.

4. **Rails Database Schema Management and Schema.rb File**
Rails Schema Management - Thoughtbot Blog
Thoughtbot's blog covers tips for working with schema.rb, a key file to verify that attributes in the database align with model definitions, which can prevent attribute-related errors.

5. **Debugging Tips from RSpec Tests**
RSpec Rails Guide
The RSpec Rails guide provides detailed information on setting up model tests, which can help verify that attributes and validations are correctly implemented.

What concepts do you understand better after doing sprint 01?

This sprint enhanced my understanding of Docker, Rails model relationships, and testing with RSpec.

How were your estimates for your tasks in comparison to the actual time?

The actual time for tasks was longer than estimated, especially for configuring Docker and debugging attribute errors. However, these steps provided valuable insights and will help improve future task estimations.

2.3 Customer Demonstration (10)

Create a video screencast to demonstrate the requirements for sprint 01 as if you are presenting to your customer.

- Give brief overview of app and problem statement
- Demonstrate behavior of app: A user that has an association with at least another item (model) where the user can
    - Register
    - Logout
    - Login
    - Create a new item
    - Edit item
    - Destroy Item