User Documentation: QUAD Seller Dashboard

How to install software

- If someone wanted to deploy your application on their own environment, what should they do?
    - Step 1: “docker compose build”
    - Step 2: “docker compose up”
    - Step 3: visit localhost:3000 to interact with the development environment of our Web APP
- What software is needed?
    - Text editor
    - Docker
    - Postgres
    - Maybe Cygwin if on Windows (none of us developed on Windows so this may be incorrect)

How to use each completed feature (User Documentation)

SEE USER DOCUMENTATION AT THIS LINK

How to modify/extend software

- Changes to the software can be made by modifying the contents of the “app” folder. The HTML views can be found in “app/views”, the controllers can be found in “app/controllers”, the models can be found in “app/models”, and the CSS stylesheets can be found in “app/assets/stylesheets”. Other assets such as images and fonts also live in “app/assets”.
- This application was built on Ruby on Rails 3.2.3 using Docker. Git was used for version control. A full list of dependencies can be found in the Gemfile, however, these should be automatically at build time.
- The project backlog can be found with the rest of our project deliverables on https://quad-marketplace-umber.vercel.app/
- Some users and listings are generated at build time. These can be modified via the file located at “db/seeds.rb”.

FAQs

The brand my clothing is from is not one of the brands to choose from, what should I do?

> The brand field is not required to create a listing, so just add the brand name into your title and we will do our best to update supported brands to include yours in the future.

If my listing is sold or rented out, how will I get paid?

> All payments are made in the Quad Marketplace store.

Does my email need a @crimson.ua.edu domain to shop Quad Marketplace?

> Yes, Quad Marketplace is a platform for students to sell and rent their clothing. We are only at the University of Alabama and need your college email to verify your student status.

How can I set the admin credentials?

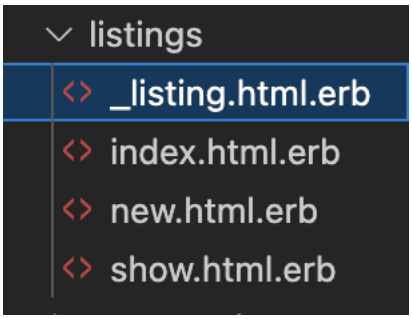
> The admin credentials can be set through the .env file in the root directory. If this file does not exist, it must be created.

Why isn't my listing posting to Shopify?

> You must be running the web app in production mode and have a valid Shopify storefront connected to the web app to post to Shopify.

Software Documentation

Listing View



There are four different views for the listing pages.

The new file is for creating a new listing.

HTML Structure

The HTML document is structured into several key sections:

Head Section: Includes external libraries such as jQuery and Select2, CSS for Select2, custom styles, and embedded JavaScript for specific functionalities.

Body Section: Contains the main form used for creating new listings, along with a logo and headers.

Key Elements

Form: The form is set up to handle new listings, with fields for photos, title, category, size, brand, color, and other listing-specific details.

JavaScript Events: Event listeners are set up to enhance form inputs and selections, including custom behaviors for dropdown menus and file inputs.

CSS Styles

General Styles: Basic styles for hiding elements and customizing file inputs.

Custom File Upload: Styled to look like a button for a more user-friendly interface.

Upload Instructions: Provides a styled informative block that lists acceptable picture types for uploads.

JavaScript Functionality

Style Tags Selection: Limits the selection of style tags to a maximum of three and allows toggling selection via mouse click.

Disable First Option in Select: Prevents reselection of the default prompt option in dropdown menus after another option has been chosen.

Price Fields Visibility: Adjusts the visibility of price fields based on whether the item is for sale, rent, or both.

Suggested Prices Calculation: Dynamically suggests prices based on the original price and whether the item is new with tags.

Dynamic Style and Size Dropdowns: Updates style and size options based on selected category.

Select2 Integration: Enhances the brand selection dropdown with the Select2 library for a searchable interface.

Key Functions

updatePriceFields(): Shows or hides price input fields based on the selected selling option.

updateSuggestedPrices(): Calculates and displays suggested pricing based on the item's original price and condition.

updateStyleDropdown(): Updates the available style options based on the selected category.

updateSizeOptions(): Updates the available size options based on the selected category.

External Libraries Used

jQuery: Used for general DOM manipulations and handling events.

Select2: Applied to the brand selection dropdown to enable search functionality and improve user interface.

The show file is to show listing information after clicking on it.

HTML Structure

The HTML document consists of a single div block that encapsulates all components related to the listing's details:

Head Section: Contains the page title and a link to the stylesheet specific to this page.

Body Section: Includes components that display the listing's title, photos, and various attributes such as size, brand, color, and pricing details.

Key Elements

Listing Title: Displays the title of the listing at the top of the page.

Listing Photos: Conditionally rendered if photos are attached to the listing, displaying each photo with alt text set to the listing's title.

Listing Information: A series of paragraphs detailing the listing's attributes including category, size, brand, color, and prices.

Navigation Button: Provides a button to navigate back to the user's profile or another designated root path.

ERB Tags

Embedded Ruby (ERB) tags are used throughout the document to integrate Ruby code with HTML:

<%= %>: Executes Ruby code and inserts the result into the HTML.

<% %>: Executes Ruby code but does not insert the result.

CSS Styles

Stylesheet Link: The stylesheet_link_tag helper includes a CSS file named show, which is presumed to contain specific styles for this page.

JavaScript

Data-Turbolinks-Track: This attribute in the stylesheet link tag ensures that the CSS is reloaded appropriately when using Turbolinks, aiding in faster page transitions that fetch only the assets that have changed.

Functional Components

Listing Photos

Conditional Rendering: Photos are only displayed if they are attached to the listing, ensuring that no broken images or empty photo containers appear.

Iterative Display: Each photo attached to the listing is displayed using a loop, with appropriate alt text for accessibility.

Listing Attributes

Dynamic Content: All attributes of the listing (like category, size, brand, color, etc.) are dynamically inserted into the page based on the current listing's data.

Conditional Text: The display of certain text, such as whether the item is new with tags, and availability for sale or rent, is conditionally rendered based on the listing's properties.

Navigation Button

Back to Profile Button: Allows users to return to a main page or profile page, enhancing the navigational flow of the website.

The index file is the user home page and _listing just shows every listing attached to the user/admin

Listing Controller

Actions

new

Path: GET /listings/new

Purpose: Initializes a new Listing object and renders the form for creating a new listing.

Variables: @listing - a new instance of Listing.

clear_flash

Purpose: Clears all flash messages and redirects to the previous page.

Flash Handling: Uses flash.discard to clear flash messages without waiting for the next action.

index

Path: GET /listings

Purpose: Displays all listings or filtered listings based on a search query.

Search Functionality: If a search parameter is present, it performs a case-insensitive search for first_name in the users table, joined with listings.

Variables: @listings - a collection of listings based on the presence of a search parameter.

create

Path: POST /listings

Purpose: Creates a new listing associated with the current user, using provided form data.

Form Handling: Uses listing_params to filter permissible parameters. It sets sell and rent based on the presence of listing_price and rental_price, respectively.

Response: Redirects to the listing's show page on success or re-renders the new form on failure with appropriate error messages.

show

Path: GET /listings/:id

Purpose: Displays a single listing.

Variables: @listing - the listing found by id.

update

Path: PATCH/PUT /listings/:id

Purpose: Updates the status of a listing to "Verified".

Post-Conditions: After successful update, redirects to an admin path with a success notice; otherwise, redirects with an error notice.

destroy

Path: DELETE /listings/:id

Purpose: Removes a listing from the database.

Post-Conditions: Redirects to an admin path with either a success or error notice based on the outcome of the delete operation.

Private Methods

set_listing

Purpose: Sets @listing based on the listing's id from parameters.

Usage: Intended to be used as a before_action for any method that requires loading a listing from the database.

listing_params

Purpose: Specifies the list of allowed parameters for listing operations to prevent mass assignment vulnerabilities.

Parameters: Allows and requires :listing key in params, and permits specific attributes including nested attributes for style_tags and file attachments for photos.

Filters

require_login: Ensures the user is logged in before allowing access to any action within this controller, executed as a before_action filter.

Listing Model

Model Relationships

Belongs To: Each Listing is associated with a User who owns the listing. This relationship is essential for tracking who created and manages each listing.

Has Many Attached: Listings can have multiple photos attached, utilizing Active Storage to manage these attachments.

Validations

The model includes several validations to ensure that data integrity is maintained:

Presence Validations

Title: Must be present to ensure each listing can be adequately identified.

Category: Required to classify the listing for filtering and search purposes.

Size: Necessary to provide potential buyers or renters with fitting information.

Numericality Validations

Original Price: Must be a number greater than or equal to zero; can be nil to allow for flexibility when the original price is unknown or not applicable.

Listing Price: Must also be a number greater than or equal to zero; can be nil to accommodate listings that are not for sale.

Rental Price: Ensured to be a number greater than or equal to zero; can be nil for listings that are not for rent.

Custom Validations

price_based_on_sell_or_rent_option

Purpose: Ensures that appropriate pricing information is provided based on whether the listing is available for sale, rent, or both.

Logic:

If the item is for sale only (sell is true and rent is false), the listing_price cannot be blank.

If the item is for rent only (rent is true and sell is false), the rental_price cannot be blank.

If the item is available for both selling and renting (sell and rent are both true), neither listing_price nor rental_price can be blank.

Private Methods

price_based_on_sell_or_rent_option

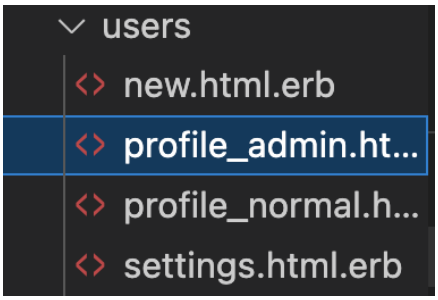
This method adds custom error messages to the listing object if pricing information does not meet the conditions set by the sell and rent attributes.

style_tags_are_valid

Purpose: Validates that the style_tags attribute, if present, is an array of strings. This ensures that all style tags are stored consistently and can be managed uniformly.

Validation: Checks that style_tags is an array and that each element within the array is a string.

Admin View



There is one file under the users view for creating the admin page called profile_admin.

HTML Structure

The HTML document is structured into several key sections:

Head Section: Contains the header images and labels indicating you are on the admin page.

Body Section: Contains the necessary functions for displaying all listings of all users and the ability to search and filter the listings.

Key Elements

Sort form: The form is set up to allow the admin to filter by user using the first and last name of the user.

Dropdown: allows the user sort using normal rankings such as date and/or alphabetically

Listing.each do |listing|: Render the partial from the listing view folder to display all the listings from quad marketplace sellers.

Verify button allows the admin to check the listing fields and send the listing to shopify website.

Delete button allows the admin to delete the listing if it does not meet standards.

CSS Styles

Stylesheet Link: The stylesheet_link_tag helper includes a CSS file named show, which is presumed to contain specific styles for this page.

General Styles: Basic styles for rendering listings.

Buttons: Button styling for verify and delete buttons.

ERB Tags

Embedded Ruby (ERB) tags are used throughout the document to integrate Ruby code with HTML:

<%= %>: Executes Ruby code and inserts the result into the HTML.

<% %>: Executes Ruby code but does not insert the result.

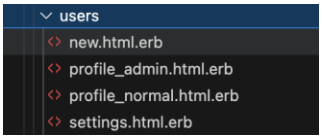
Admin Controller

There is no specific admin controller, it is a subsection of the user controller. See documentation below.

Admin Model

There is no specific admin model, it is a subsection of the user model. See documentation below.

User View



There are four different views for the user pages.

The new file is for new users to sign up.

HTML Structure

The HTML document is structured into several key sections:

Head Section: Includes "Create Account" Title and the stylesheet link tag we used for CSS.

Body Section: Contains a div container with the logo, header, and the form for signing up for an account.

Key Elements

Form: The form is set up to handle new users, with fields for first name, last name, email, phone number, crimson email, username, password, and password confirmation.

CSS Styles

General Styles: Basic styles to make page for entering information easy..

The profile admin file is explained above.

The profile normal file is a view that creates the user's view.

HTML Structure

The HTML document is structured into several key sections:

Head Section: Contains the header images and labels indicating you are on the user page.

Body Section: Contains the necessary functions for displaying all listings custom to the user and the ability to filter the listings.

Key Elements

Sort form: The form is set up to allow the user to filter by date, title, and price.

Dropdown: allows the user sort using normal rankings such as date and/or alphabetically

Listing.each do |listing|: Render the partial from the listing view folder to display all the listings custom to the current user.

CSS Styles

Stylesheet Link: The stylesheet_link_tag helper includes a CSS file named show, which is presumed to contain specific styles for this page.

General Styles: Basic styles for rendering listings.

ERB Tags

Embedded Ruby (ERB) tags are used throughout the document to integrate Ruby code with HTML:

<%= %>: Executes Ruby code and inserts the result into the HTML.

<% %>: Executes Ruby code but does not insert the result.

Dropdown Menu

Dropdown menu is a dropdown that you can hover over that allows you to create a new listing, go to account settings, and sign out.

The settings page is a file used to be able to change your password and email address.

HTML Structure

The HTML document is simply just a form structured by different divs.

Key Elements

Form: The form is set up to handle changing emails and passwords with fields for current email, new email, password, and password_confirmation.

CSS Styles

General Styles: Basic styles for changing the account settings.

Buttons:

Buttons included for this view included a button for going back to the profile.

User Controller

Actions

Path: Users controller actions will depend on user authentication status and admin privileges.

Filters

authenticate_user!: Ensures the user is logged in before allowing access to profile_admin, profile_normal, settings, destroy, and update actions.

redirect_if_authenticated: Redirects users who are already authenticated away from the create and new actions.

verify_admin: Ensures the user has admin privileges before accessing the profile_admin action.

create

Path: POST /users

Purpose: Creates a new user account.

Form Handling: Validates and creates a user using create_user_params. Sends a confirmation email upon successful creation.

Response: Redirects to the login path with a notice on success; renders the new form with errors on failure.

clear_flash

Purpose: Clears all flash messages.

Flash Handling: Uses flash.discard or flash.clear to remove flash messages.

Response: Redirects to the previous page using redirect_back with a fallback location as root_path.

destroy

Path: DELETE /users/:id

Purpose: Deletes the current user's account and clears the session.

Post-Conditions: Redirects to the login path with an alert indicating account deletion.

profile_normal

Path: GET /users/profile_normal

Purpose: Displays the normal user profile including listings filtered by status or sorted by certain criteria.

Variables: @user - the current user; @listings - user's listings, potentially filtered or sorted; @active_sessions - active sessions ordered by creation time.

profile_admin

Path: GET /users/profile_admin

Purpose: Displays the admin user profile with access to all listings, with optional filters and sorting.

Variables: @user - the current admin user; @listings - all listings, potentially filtered or sorted; @active_sessions - active sessions ordered by creation time.

settings

Path: GET /users/settings

Purpose: Displays user settings and active sessions.

Variables: @user - the current user; @active_sessions - active sessions ordered by creation time.

new

Path: GET /users/new

Purpose: Renders the form for creating a new user.

Variables: @user - a new instance of User.

update

Path: PATCH/PUT /users/:id

Purpose: Updates the current user's account settings.

Form Handling: Validates the user's current password and updates the account based on update_user_params. Sends a confirmation email if the email is changed.

Response: Redirects to the account settings path with a notice on success; renders the settings form with errors on failure.

Private Methods

verify_admin

Purpose: Verifies that the current user has admin privileges.

Usage: Redirects to user account path with notice if not an admin.

create_user_params

Purpose: Specifies allowed parameters for user creation.

Parameters: Permits attributes including name, email, phone number, username, password, and admin status.

update_user_params

Purpose: Specifies allowed parameters for updating user details.

Parameters: Permits password, email updates, and associated confirmations.

User Model

Model Relationships

Has Many: Each User can have multiple active sessions and listings. These relationships allow tracking user activity and management of listings respectively.

Active Sessions: Managed via a dependent relationship, ensuring sessions are destroyed when the user is deleted.

Listings: Allows each user to own multiple listings.

Validations

Presence Validations

Email: Must be present and formatted as a crimson.ua.edu account to ensure user authenticity and uniqueness.

Username and Phone Number: Must be present and unique to ensure each user is identifiable and contactable.

First Name and Last Name: Required for basic identification and user profile completeness.

Format Validations

Email: Validates that the email is in the correct format specific to crimson.ua.edu domain.

Unconfirmed Email: Checks the format using a standard email regular expression, allowing it to be blank.

Constants

CONFIRMATION_TOKEN_EXPIRATION: Set to 10 minutes to limit the time frame for confirming the email address.

PASSWORD_RESET_TOKEN_EXPIRATION: Also set to 10 minutes, ensuring security during the password reset process.

MAILER_FROM_EMAIL: The default sender email address for all outgoing mails from this model.

Custom Methods

confirm!

Purpose: Confirms the user's email, updating the confirmed_at field.

Post-Conditions: Returns false unless the email can be updated or confirmed, effectively managing the confirmation status.

generate_confirmation_token

Purpose: Generates a secure token for email confirmation.

Functionality: Uses Rails' signed_id to create a token that expires according to CONFIRMATION_TOKEN_EXPIRATION.

generate_password_reset_token

Purpose: Generates a secure token for resetting the user's password.

Functionality: Similar to generate_confirmation_token, but for password reset purposes.

send_confirmation_email!

Purpose: Sends an email with the confirmation token to the user's email address.

Action: Invokes UserMailer to deliver the email immediately.

send_password_reset_email!

Purpose: Sends a password reset email with a secure token.

Action: Similar to send_confirmation_email!, but for password reset.

Private Methods

downcase_fields

Purpose: Ensures all relevant user fields are stored in lowercase to prevent case-sensitive issues in the database.

Fields Affected: email, first_name, last_name, username.

downcase_unconfirmed_email

Purpose: Converts any unconfirmed email to lowercase, ensuring consistency across user input data.

Role Management

admin?

Purpose: Checks if the user has an admin role.

Usage: Typically used to grant access to admin-specific areas of an application.

normal?

Purpose: Checks if the user has a normal role.

Usage: Can be used to restrict or allow access to general user functionalities.