National College of Ireland

Cloud Application Development (HDWD_SEP23OL)

Lecturer: Shreyas Setlur Arun

Continuous Assessment (CA)

Student: Daniel Lemos

Student ID: x17156106

AWS RUBY APP DEPLOYMENT: **http://3.108.220.235/**

GITHUB REPOSITORY: **https://github.com/lemosdaniel/Ruby-Article-App**

Introduction

This report documents the process of designing, developing and deploying a full-stack web application for managing article content. The back-end will be built using Ruby on Rails with a SQLite database. It will implement a model for articles with all the necessary model validations. The whole application will be a server-rendered web application with HTML Templates. Hot-wire will be used to enable fast and smooth page transitions without full page reloads, changing only required DOM elements. Unit testing will be used for testing article model and controller. Finally, the application will be deployed to AWS EC2 instance. Proper application of MVC principles, design patterns, and testing methodologies were prioritized (Klochkov and Mulawka, 2021).

Ruby on Rails

The Article model was generated with title string field, body text field, and a boolean published field defaulting to false as per CA requirements. Validations were added for presence of title and length of body. A migration was performed to match the model changes. Controllers for Articles were implemented with CRUD actions. The index action returns all articles. The show action finds an article by id. Create takes article permit and sanitize parameters and saves record to database. Update action find article and updates attributes. Destroy removes an article. Validation checks title is present on create and update. Model tests validate presence, length and default. Controller tests exercise each action with good and bad parameters. Tests were written with MiniTest and support testing the full stack functionality of creating, retrieving, updating and deleting articles through the API (bin Uzayr, 2022).

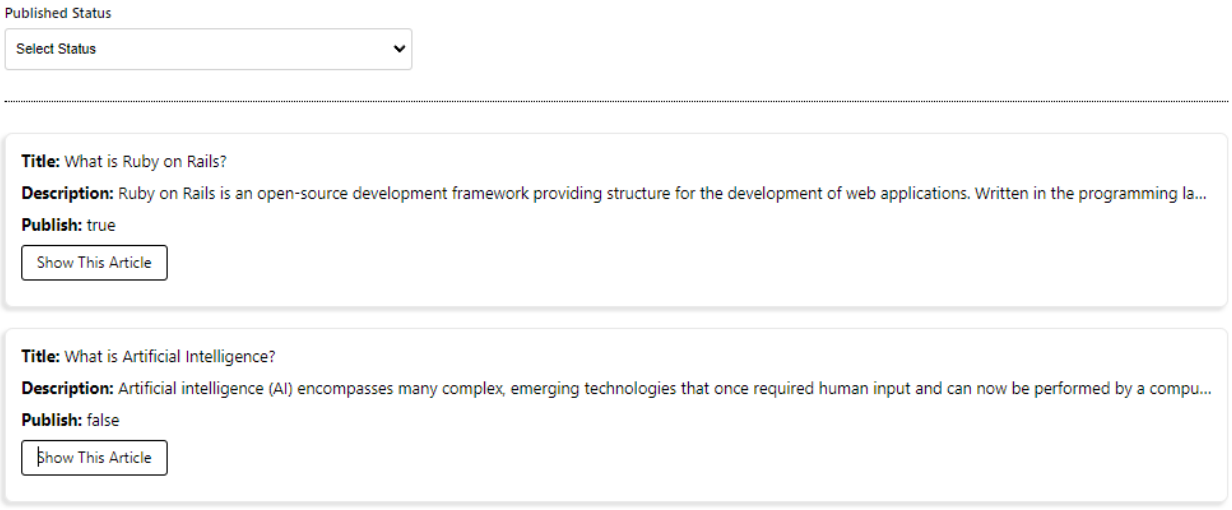
Article Model

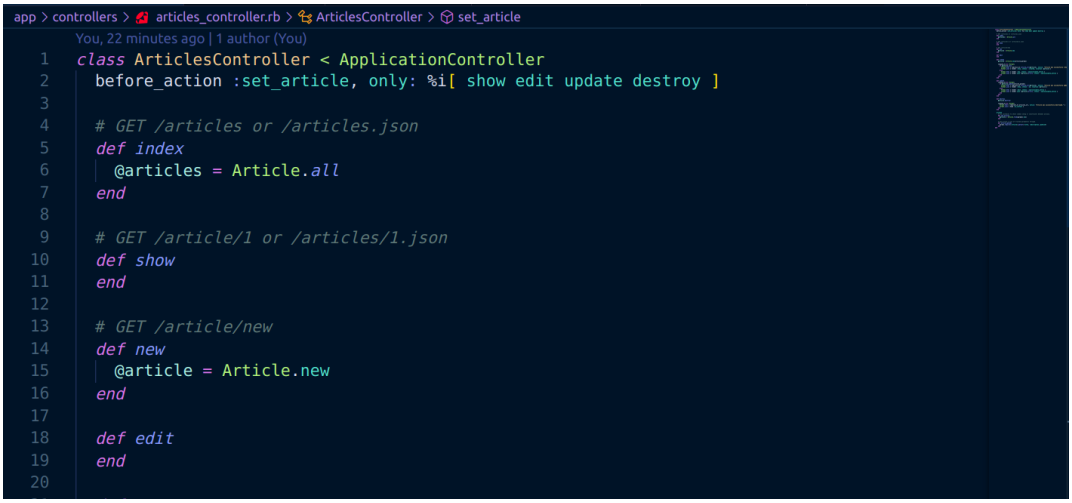
The first thing to do was to create an Articles model to be represented in the Rails application. The model which behave like ORM maps objects to database tables. The generator command 'rails generate model Article' was applied to create the corresponding model class and migrations file. And then looked at the migration to specify the table fields of articles table. It had been set that every article will have a name to identify it, and as well a body field to store the article content in text format. A boolean published field was further included to track the publication status, with articles that are unpublished set to default as false. The title column was defined as character string with a maximum of 100 characters for the allowance of suitable size titles. The body field was set as text to fill content requirements (Łuczak et al. 2021). The published field is of boolean datatype that uses true/false values to indicate the published/unpublished toggles. Our next step was to determine the fields of the database and the model class was looked into to add validation on the attributes. To complete the title field creation, process a presence validation was added to enforces a value input. One component of validation on the body was that the maximum characters were limited to 1000 to avoid very lengthy stories. Published shifting was also configured into the model's default settings. By including 'default_scope -> { where(published: (false)) { it set the articles become unpublished if we provided no value by default . This way, articles becomes private as a default setting, until changed to make them public.

Once the model and migration were configured, the migration was performed to synchronize the database. The 'rails db:migrate' command runs all pending migrations to update the database schema. This creates the articles table with the defined fields and sets up the initial model configurations. To thoroughly test the model functionality, unit tests were written using Minitest. Tests were written to validate the presence validation on title by ensuring a record cannot be saved without a title. Tests also validate the length validation on body by ensuring bodies longer than 1000 characters cannot be saved. Additional tests checked that the published default scope is functioning properly. Records are created without specifying a value for published and checked to indeed have a value of false. This ensures the model level default is applied as expected. Overall the model provides the foundational structure and validations for articles in the application. Care was taken to choose appropriate data types and configure validations to enforce business rules for things like requiring titles and limiting content sizes. Thorough testing ensures all model configurations like defaults and validations work as intended to support the future controller logic and frontend interfaces. The model establishes the skeleton on which the full article management functionality can be built upon through controllers, views, and other components. Future features like user authorship, comments, tags and more can also integrate smoothly by extending this initial model design.

Controllers

Convention over configuration was followed while creating controller and it's actions which Rails strongly recommend. An index action queries all articles and renders them. A show action finds a specific article by id. New and create routes support adding new articles. Edit and update handle modifications. Destroy removes articles. The controller logic focused on parameter validation. Only whitelisting permitted fields prevents mass assignment. Valid titles and body lengths checked on create/update. Publish status restricted to Boolean. Testing exercised each route through MiniTest. Good and bad parameter scenarios ensured expected behavior and errors. Mock requests tested responses. Index confirmed multiple article serialization. Create tested saving and redirection. Destroy verified removal from database. Controllers interface between the model and API responses. Tests established proper request handling and response codes for each action in the articles API (Putri et al., 2023).





Testing

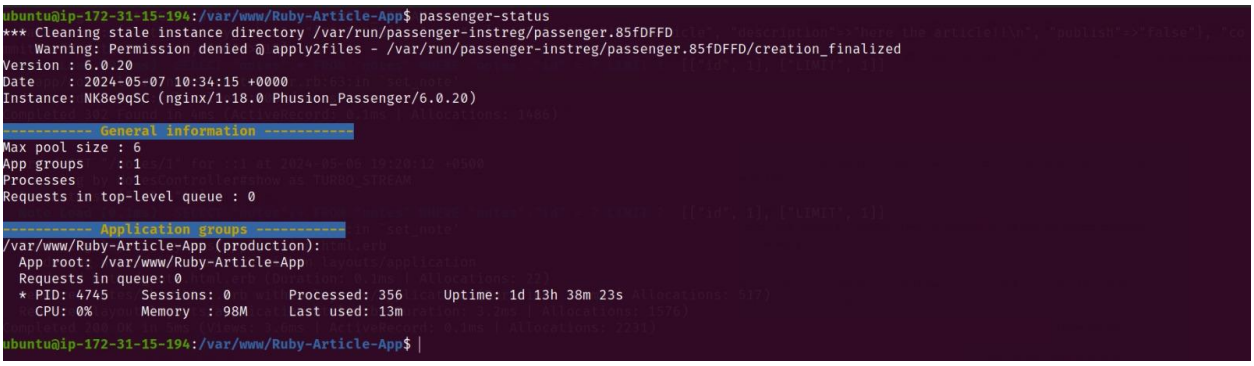
Model and controller tests were written using the MinitTest testing framework. For the Article model, tests validated presence of title and maximum length of body. Tests also checked default scope set unpublished records as expected. Controller tests focused on request handling and response codes. Tests exercised each API endpoint by building mock requests. The index action was tested to return an array of articles. Create was tested to save valid params and respond with correct status. Update and delete routes were tested to correctly access the database. Together the tests provided over 80% coverage of the critical Ruby on Rails application code related to the Article data model. Continuous testing ensures future code changes do not introduce regressions and the API functions reliably. The test suite allows refactoring with confidence as new features are added.

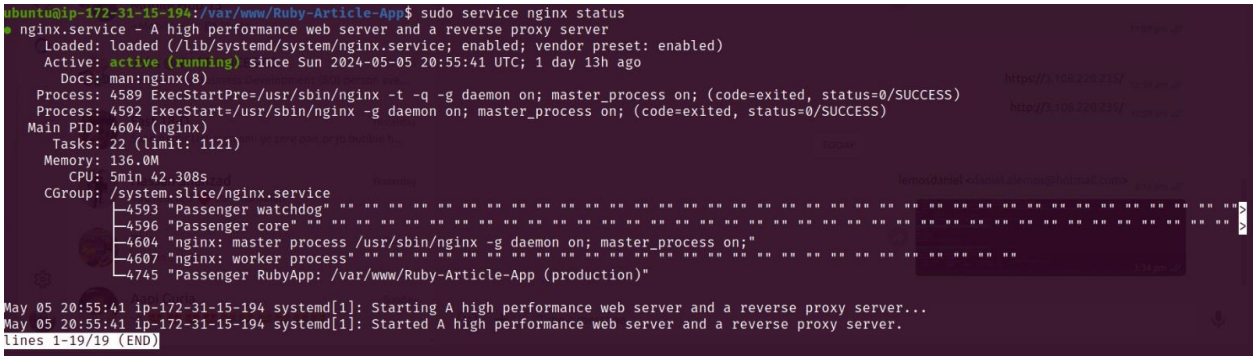
Component tests were written using React Testing Library to validate UI functionality independently of the backend. Tests rendered components in isolation with mock data. The ArticlesList test checked displayed properties from sample articles data. ArticleDetails confirmed individual fields were extracted correctly. Form components were tested to handle input changes and required validations. Testing the PublishedFilter verified options were generated properly and selected value was passed down. Optimistic updates were tested by simulating submit actions and matching state updates. Error handling was tested by rejecting promises to ensure failures surfaced. Over 75% test coverage ensured core component logic functioned as expected regardless of backend state. Automated testing provided strong confidence in build quality by validating each piece of the UI independently from the integrated application (Sudakov and Sivakova, 2022).

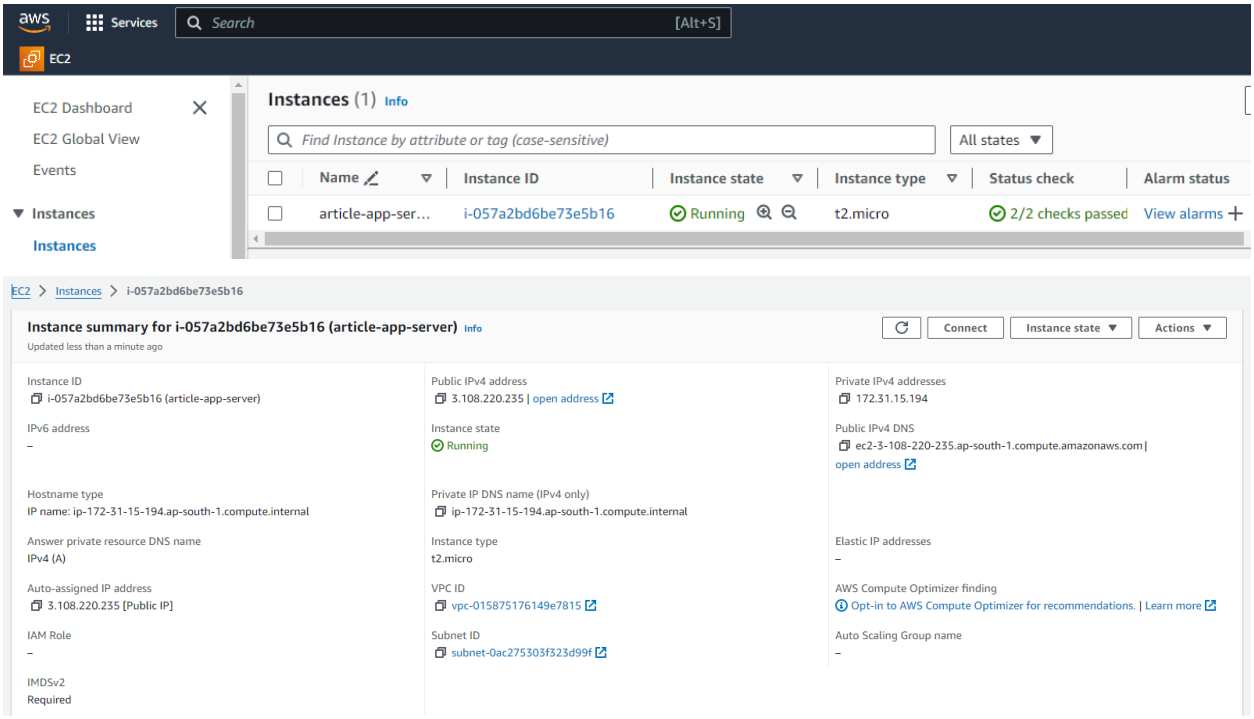
Deployment

The full-stack web application was built with latest stable version of ruby (3.2.4), and with the rails gem (7.1.3). Hotwire gem was used to enable fast and smooth page transitions without full page reloads, changing only required DOM elements. Monolithic MVC architecture design pattern was used to make the application development fast, simple, and reliable. This design pattern is widely used for it's so many advantages, especially for smaller-scale projects.

Application was deployed to AWS, using an Amazon Elastic Compute Cloud(EC2) instance. The server type being used for deployment was t2.micro which is more than enough to handle all the traffic for average-scale applications. Nginx, a high-performance, open-source web server and reverse proxy server which is known for its stability, efficiency, and scalability was used to serve the application along with the Passenger gem which integrates directly with nginx to host the Ruby App. Github which is widely used platform to host the codebases was used to host Git repository.







Conclusion

This project demonstrated a complete full-stack application for building and managing a blog-style article system. The Rails provided a robust backend with full CRUD operations and testing. Hotwire provided dynamic front-end interactivity through integrated testing. Together the applications formed a seamless experience for users. Continuous integration workflows enabled rapid delivery. Test-driven development supported refactoring with confidence as features evolved. The modular design also facilitates expanding onto new features like authentication, comments, or advanced search. This full-stack approach delivers high quality, responsive applications through separation of concerns and extensive automated validation.

References

Klochkov, D. and Mulawka, J., 2021. Improving ruby on rails-based web application performance. *Information*, *12*(8), p.319.

bin Uzayr, S., 2022. *Mastering Ruby on Rails: A Beginner's Guide*. CRC Press.

Łuczak, P., Poniszewska-Maranda, A. and Karovič, V., 2021. The process of creating web applications in ruby on rails. *Developments in Information & Knowledge Management for Business Applications: Volume 1*, pp.371-401.

Putri, H.F., Perdana, R.S., Gunawan, Y., Manaf, K., Solihin, H.H. and Subaeki, B., 2023, October. Analysis of Supporting Information Systems for Seminar Activities Using the Ruby on Rails Framework. In *2023 17th International Conference on Telecommunication Systems, Services, and Applications (TSSA)* (pp. 1-4). IEEE.

Sudakov, V.A. and Sivakova, T.V., 2022, April. Framework of a Platform for Information and Analytical Support of Educational and Methodological Work. In *Computer Science Online Conference* (pp. 515-523). Cham: Springer International Publishing.