Garden Party

Application audit

David Dérus EI - 24/11/2023

Sommaire

1 Introduction

The present document serves as a comprehensive report on the audit conducted to study the open-source Garden Party application.

It is focused around the existing codebase and aims at providing feedback around how it can improve and evolve.

Within this report, you will find detailed observations derived from the audit, accompanied by recommended actions to be undertaken. The report encompasses the following key aspects:

1. A presentation of the audit scope and goals.
2. A detail of the audit observations, around 5 topics:
    1. Documentation
    2. Security
    3. Data model
    4. Code
    5. UI & UX design
3. A conclusion summarising the auditor opinion of the application.

1.1 Goals

The objectives of this audit are the following:

- Ensure that the technical foundation and tooling adequately address the open-source project objectives.
- Assess the relevance of the software architecture and development methodologies.
- Confirm that the available documentation aligns with the expectations and requirements of the project.
- Evaluate the reliability of the data model used by the system.
- Define the necessary UI & UX design improvement tasks for the Garden Party application.

A two workdays' duration was provisioned for the analysis part of the audit.

1.2 Methodology

The audit observations are ranked as follows:

1. *Good practice*: An observation about a technical choice that stands out.

2. *Bad practice*: An observation signalling that something is missing or seems to be done in the wrong way.
3. *Improvement*: A suggestion regarding a specific observation.
4. *Remark*: A comment made by the auditor.

The audit observations about bad practices or suggested improvements are ordered by criticality:

1. *High*: This observation must be immediately prioritised and handled by the development team.
2. *Medium*: This observation is important and mustn't be dismissed. It should be prioritised in the next few months.
3. *Low*: This observation could be taken into account to improve the application.

1.3 Scope

The audit observations and recommendations cover the evaluation of the Garden Party application, including:

- Its Ruby on Rails API.
- Its Ruby on Rails admin interface.
- Its Vue.js front-end.

The audit observations and recommendations focus on assessing whether the development processes align with industry best practices and standards.

These observations and recommendations do not pertain to any future feature developments. Additionally, they do not replace an extensive application security and vulnerability assessment, or a full User Experience audit.

The audit focus was on the Garden Party codebase and documentation and required an additional dependency, the Rise UI design system.

Here are the specifications of the different codebase installed and studied during the audit:

Garden Party application

- Repository name: `garden-party`.
- Repository URL: GitLab.
- Code last tag: `0.13.3`.
- Code last SHA: `c206f578`.

Garden Party documentation

- Repository name: `garden-party-docs`.
- Repository URL: GitLab.
- Code last tag: `0.13.0`.
- Code last SHA: `b8fec3b8`.

Garden Party design system

- Repository name: `rise-ui`.
- Repository URL: GitLab.
- Code last tag: `0.0.4`.
- Code last SHA: `03f75395`.

1.4 Versions

- `24/11/2023`: Updated version after the audit presentation to Garden Party maintainers. This new version includes:
  - An updated title for the `2.5.3` recommendation.
  - An updated description and recommendation for the `2.3.6` and `2.6.4` recommendations.
  - A reduced criticality for the `2.6.4` observation (*previously* `2.6.3`), leading to a numbering change for the *UI & UX design* chapter observations.
- `22/11/2023`: Original public version of the audit report.

1.5 Glossary

In order to understand this document, the reader need to be aware of a few technical words and acronyms:

- **API:** Short for *Application Programming Interface*. A programming interface allowing data exchange between two codebases through the Internet.
- **Application:** A global naming referring to both a code project and its user interfaces, commonly accessible on the users' devices.
- **Back-end:** The part of an application that is not directly accessed by the user, responsible for storing and returning data.

- **CI:** Short for *Continuous Integration*. An automated process running tasks on a provider server. These tasks can be tests, linting, building or any other programmatic need a team can have.
- **Domain logic** or **Business logic:** The code part that defines the rules related to the operator running the software. It determines how data should be processed, transformed and stored.
- **Endpoint:** A URL address inside an API, allowing access to specific data.
- **Front-end:** The part of an application the user interacts directly with.
- **GPG:** Short for *GNU Privacy Guard*. An encryption program with public-key cryptographic support.
- **JavaScript:** A programming language able to run in the user Web browser or on a Web server.
- **Linting:** Code analysis allowing automated flagging of bad code patterns, missing best practices, security issues and other mistakes that can appear during development.
- **Node** or **Node.js**: A JavaScript runtime, usually used as a back-end server environment.
- **Postgres:** A SQL compliant relational database management system.
- **Repository:** A place where the code is stored, and each change done on a file is tracked. It could be hosted on the developer computer, on a private server or dedicated Web services, like Microsoft GitHub or GitLab.
- **Responsive:** The way a web application front-end adapts to different screen sizes.
- **RSpec:** A Ruby library use for automated testing.
- **Ruby on Rails** or **Rails:** A server-side web application framework used to build web applications and APIs in Ruby.
- **Ruby:** A programming language usually running on a Web server.
- **Test** or **Spec:** An automated way to check that the code, under a predetermined context, is behaving as expected.
- **UI Design:** Short for *User Interface Design*. A methodology related to the conception of user interfaces as displayed by an application.
- **UX Design:** Short for *User Experience Design*. A methodology defining the common design practices used to improve the user experience when using an application.
- **Vue.js:** A front-end framework used for the development of a web application.

2 Observations

2.1 Summary

Category	Ranking	Count	
Documentation	Good practice	2	Link
Documentation	Bad practice	2	Link
Documentation	Improvement	3	Link
Documentation	Remark	1	Link
Security	Good practice	3	Link
Security	Bad practice	3	Link
Security	Improvement	1	Link
Data model	Good practice	2	Link
Data model	Bad practice	2	Link
Data model	Remark	1	Link
Code	Good practice	8	Link
Code	Bad practice	3	Link
Code	Improvement	13	Link
UI & UX design	Good practice	2	Link
UI & UX design	Bad practice	3	Link
UI & UX design	Improvement	12	Link

Please note that this audit methodology is all about discovering the Garden Party application, alerting about sensitive issues and suggesting improvements.

A low “Good practice” count or a few “Bad practice” for a category does not mean that the application is failing the audit.

It simply means that the auditor wants to put the focus on these observations.

2.2 Documentation

2.2.1 Developer documentation is available as a README

Ranking: Good practice

Description:

The `garden-party` repository has documentation in the form of a README file.

This README contains a lot of information for developers regarding the project setup, architecture, specificities and development methodologies.

Recommendation:

Even though the application basics are well documented, it is possible, in the optic of enabling contribution, to expand available information to include the following:

- A CONTRIBUTING.md guide with instructions regarding the test coverage expectations.
- A note pointing to issues previously documented and tagged with the `good-first-issue` tag.
- A valid `CODE_OF_CONDUCT.md` file.

All existing issues tags like `For:` must be kept and used for every new issue.

2.2.2 There is an official way to reach the developers

Ranking: Good practice

Description:

Reaching the application contributors is made easy by providing Matrix chatrooms links at the beginning of the project.

2.2.3 There is no documentation explaining the meaning and relationships between data models

Ranking: Bad practice

Criticality: High

Description:

Reading a data model and understanding the role of each entity and its relationship can prove to be a challenge without a proper documentation.

Moreover, the purpose of an entity named with generic terms like `Element` or `Resource` can be opaque without the application context.

Recommendation:

A brief introduction of the database structure and entities can help grasp the domain logic and differentiate the entities.

Furthermore, reusing the UML generator stored in the `lib` directory will generate an up-to-date diagram of the tables properties and relationships.

2.2.4 There is no instance deployment tutorial or one-click install

Ranking: Bad practice

Criticality: Medium

Description:

Garden Party is an open source project. As such, everyone should be able to host it on any provider.

For this to happen, a novice user, or someone with no knowledge of the Ruby and Rails ecosystem, will need some basic deployment instructions.

No such instruction is available in the project README, so only a few people may be able to deploy the application, which will slowdown its adoption.

Recommendation:

While exposing a full deployment documentation supporting multiple hosting providers can be costly to setup and maintain, some alternative do exist.

What could be easily done is adding a “one-click” install button for platforms like Scalingo, Render or Heroku. These few lines in the deployment README will point users to well-known Rails hosting providers, with no setup cost for the application maintainers.

In addition to that, using something like the kamal gem, an all-in-one Docker deployment tool built by Basecamp, or writing a basic `Dockerfile` and docker-compose setup, could help new users deploy the application.

2.2.5 There is a user documentation website, without any actual user documentation

Ranking: Improvement

Criticality: Medium

Description:

The user documentation website is succinct and does not explain the actual application usage. For a potential user, this could be a no-go from the start.

Recommendation:

While the documentation website is missing some explanation regarding the application usage, it is to be noted that the application does provide in-context help.

So what's missing here is a link between the Garden Party website, which acts more as a presentation website, and the actual help inside the application.

The website should explicitly redirect to a demo instance like the Sartre one, and emphasis that help will be provided inside the app.

Finally, maintainers could record a short walkthrough video, to guide users in building their first garden map.

2.2.6 The README is too long and contains instructions for both developers and people wanting to deploy the project

Ranking: Improvement

Criticality: Medium

Description:

While the Garden Party README is a great place to start learning about the project, it is also a very long read.

It covers topics from dev setup to testing while going through JavaScript application structure, CSS namespacing and much more. Going through all this information to locate a specific part could be tiresome.

Recommendation:

This README document should be split in at least 3 parts:

- Back-end and database setup and development.
  - (*including a link to the latest API documentation*).
- Front-end setup, development and conventions.
- Application deployment.

Each part will be stored in a dedicated file, and a table of content will be added the main README.

Note that such effort was started by a contributor in a merge request, which could be used as a base for implementing this recommendation.

2.2.7 The API documentation is auto-generated, but not published online

Ranking: Improvement

Criticality: Low

Description:

The project is using the rspec-rails-api gem to test the API and generate a Swagger file. This is great as this file can serve as an up-to-date API documentation.

However, it is not made accessible easily, even if GitLab is offering a good preview for it.

Recommendation:

At the very least, a link to the versioned Swagger file must be added to the README.

It will help front-end developers understand the API endpoints, and serve as a basis for future work on new Garden Party clients.

Such link should be located at the top of the README.

2.2.8 Online documentation is not up to date with the latest build

Ranking: Remark

Description:

For an unknown reason, the version shown on the user documentation website is the `0.13.2`, while the latest application version is the `0.13.3`.

2.3 Security

2.3.1 The application is using the latest Ruby on Rails major version available

Ranking: Good practice

Description:

The Ruby on Rails application is in its 7th version, which is the latest available.

Seeing you're using Ruby on Rails 7.0.5, you may as well upgrade to the latest Ruby on Rails 7.1, which brings security, performance and code improvements.

It will also bring you new features, like ActiveRecord CTE and async queries (read more).

2.3.2 The application is using Devise

Ranking: Good practice

Description:

By using the Devise gem, the application is choosing a well-known and battle-tested authentication system.

A lot of sensitive defaults are built in this gem, as well as security mechanism preventing common attacks. It is actively maintained and regularly updated.

2.3.3 Permissions and policies are in place to restrict users' actions

Ranking: Good practice

Description:

Adding an authorisation layer on top of a robust authentication system guarantees that every user action is valid and that only authorised data are accessible by said user.

The Pundit gem is one of the best libraries available to do so and Garden Party is using it to protect each action and entity with a dedicated and **tested** policy.

Even the scopes used inside the policies are tested, leading to a great overall security.

2.3.4 Garden Party Ruby version is not maintained anymore

Ranking: Bad practice

Criticality: High

Description:

The Garden Party application is using Ruby 2.7, which is not maintained since April 2022 and has reached end-of-life in March 2023.

Furthermore, the project documentation, `garden-party-docs`, is using another version, Ruby 3.1.2.

Recommendation:

The application must be updated to Ruby 3, which may not be an issue since the latest Ruby on Rails version is used.

Moreover, it could make sense to use the same Ruby version for all Garden Party's repositories.

2.3.5 The application Node version is obsolete

Ranking: Bad practice

Criticality: High

Description:

The Garden Party front-end is using Node v16, which have no security support since September 2023, and no active support since October 2022.

Recommendation:

The application must be updated to Node 18, or even Node 20, depending on the CI and deployment constraints.

2.3.6 The users search endpoint is accessible without authentication

Ranking: Bad practice

Criticality: High

Description:

Garden Party offers a search feature, so that a user can add another to its team by looking them up with their username or email.

This feature is only available for logged-in users, but its endpoint is accessible without authentication.

While this endpoint does not return a list of emails matching a query, it does validate that a given email exists on the instance. This public search endpoint also returns matching usernames and the users' IDs.

Recommendation:

The email search allows an attacker to check if a full email address exists on the instance. This is considered a bad security practice as it gave a potential attacker more knowledge about its targets.

But, perhaps more importantly, the username search, which only takes three chars to look for all matching usernames, could lead to a GDPR breach, as the username could include personal information (*e.g. a search for* `dav` *could return* `david_derus`).

This endpoint must be behind an authentication rule.

2.3.7 The application data sources could be GPG signed

Ranking: Improvement

Criticality: Low

Description:

Garden Party instances can sync data from a public repository owned by Garden Party maintainers.

To improve trustfulness around this unique source of information, we recommend allowing only GPG signed commits from trusted developers.

This way, the data source content authenticity can be continuously validated by any instance maintainer before pulling any update.

2.4 Data model

2.4.1 The application database architecture is straightforward

Ranking: Good practice

Description:

The global database structure of the Garden Party application is well thought and easy to grasp.

It does not include convoluted relationship, bad naming or obscure database extensions, which make it a good database architecture.

2.4.2 The database tables have reasonable attributes

Ranking: Good practice

Description:

When designing a database, it can be easy to forget about indexing and foreign key constraints.

The Garden Party application is using them whenever pertinent, adding index on important keys and foreign keys and even enforcing cascade deletion for linked tables.

2.4.3 Using bitfields is adding complexity between the code and database layer

Ranking: Bad practice

Criticality: High

Description:

Bitfields are a programmatic way of storing information. Their behaviour is not immediately intuitive when shown in there raw form, as integers, but make sense in systems with a very limited space or trying to get a small memory or storage footprint.

In Garden Party case, there is no need to go to such length. Moreover, there is no existing Postgres database method to query this kind of information.

With bitfields, the Garden Party application must load the raw value and then use some custom logic on the back and front-end sides.

This also prevents the application from making use of basic database features like indexes and where clause, thus limiting the opportunity around the data stored as bitfields.

Recommendation:

Instead of using bitfields, the application can make use of Postgres arrays. They are natively supported by both Postgres and Active Record, and offer very good performances.

They're indexable and can be queried directly from the database, without requiring a gem.

This opens opportunity to enhance existing queries and introduce new queries like "*What resources are harvestable in November?*", without loading all resources on the front-end side.

2.4.4 Content migration classes should be extracted from database migrations for easier testing

Ranking: Bad practice

Criticality: Medium

Description:

Ruby on Rails migrations are supposed to allow modifying the **structure** of the database, not its content. So, when content migration is required, there is no built-in mechanism to handle it.

To answer the content migration scenario, the Garden Party application uses raw Ruby code inside its migration to insert or update existing data. This code is written directly in the migration file, which is later required in a spec to test its behaviour.

This leads to a lot of responsibilities in a migration file, and a tight coupling between the `ActiveRecord::Migration` code and its test.

Recommendation:

Instead of writing the content migration code directly in the migration and requiring the latter in the test file, one can introduce simple Ruby content migration classes, and require them inside the migration.

These simple Ruby classes can be tested without requiring the migration code, thus decoupling both classes.

Another concept, *chores*, may be worth a look, as they remove the content migration step from the migration. Instead, chores make content migration another task, generally ran after a `rails db:migrate`, from Rake or with the help of a gem.

Have a look at gems like data-migrate for more chores examples.

2.4.5 Using an abstract model class to handle `GeometryRecord` is not the "Rails way"

Ranking: Remark

Description:

While an abstract model class may work to handle shared methods and validations, it is not the "official Rails way".

Instead, concerns should be used, and included in each model currently inheriting from the abstract class.

2.5 Code

2.5.1 There is a valid test setup with a very good test coverage for the front and back ends

Ranking: Good practice

Description:

Using RSpec, code coverage for the Ruby code is around 81% with 1202 examples. For the JavaScript code, vitest shows a 90% coverage, with 161 tests.

While not included in the RSpec coverage score, Cucumber is running 99 scenarios which covers multiple front-end features.

Both coverage scores are great, and prove that a majority of the application code is tested. Such high coverage scores must be maintained in the next application releases.

2.5.2 Code linting is thorough and automated in CI

Ranking: Good practice

Description:

In order to guarantee a good code quality, the Garden Party project maintainers have defined a large set of rules for all the code components, including Ruby, JavaScript, SCSS and HAML code.

If said rules are not validated, the code is automatically rejected, with an error log pointing to the faulty code.

2.5.3 Security checks are run against each push to the CI

Ranking: Good practice

Description:

The same way code linting is run against each push and rejects bad code practices, the Garden Party maintainers are using the brakeman security gem to validate that known bad security practices are not included in code changes.

This help prevent security issues like SQL injection, cross-site request forgery, authentication issues and much more.

2.5.4 Specific code and methods are commented

Ranking: Good practice

Description: While a vast majority of the code logic is straightforward and well tested, some very specific part of it requires more information.

To answer this need, developers of Garden Party added comments, which are frequently documenting the more intricate part of controllers, models, validators and components.

2.5.5 There is validations around data models properties and nested content

Ranking: Good practice

Description:

The application relies heavily on Ruby on Rails validators to check that every model attribute is present and well formatted.

For more complex tests, the application even includes its own set of validators, which are even validating the nested structure of some JSON attributes.

2.5.6 Models are tied to tests describing their validation conditions

Ranking: Good practice

Description:

Garden Party application models are using validators to test the data before it is persisted. Each of these models is also well tested, with cases covering multiple model validation scenarios.

2.5.7 The code project includes all requirements for new contributors to start coding

Ranking: Good practice

Description:

Aside from the documentation requirements, new contributors also need to have a set of guidelines to follow while joining a new project.

The Garden Party application includes a lot of these rules, helping configure the contributor development environment with an Editor Config file and an Overcommit Git hook configuration.

Once the contributor starts writing code, they can count on a well-configured group of linters dedicated to every language used by the project.

2.5.8 Logic is extracted from the JavaScript code into VueX stores

Ranking: Good practice

Description:

In Garden Party, the front-end state logic is handled by VueX, a well-maintained store utility.

This prevents the state logic to be duplicated across components and provide a unique and global source of state for all of them.

This way, components remains dedicated to the data visualisation and user interaction and do not manage the domain logic.

2.5.9 Routing specs are duplicates of requests specs and testing Rails inner working

Ranking: Bad practice

Criticality: Medium

Description:

Testing routes is like testing Ruby on Rails inner code. It's redundant in most of the case.

It Garden Party case, routes are already tested by requests tests, as well as covered by some of the acceptances tests.

Furthermore, removing the `spec/routing` directory does not lead to any change in the test coverage score.

Recommendation:

In Garden Party codebase, routing tests are redundant and must be removed.

2.5.10 Some domain logic code is duplicated

Ranking: Bad practice

Criticality: Low

Description:

Scopes and validations of the `Family`, `Genus` and `Resource` look the same.

Recommendation:

This duplicate code should be extracted to a `Geometry` concern introduced by this recommendation.

2.5.11 Avoid nesting tests to three levels or deeper

Ranking: Bad practice

Criticality: Low

Description:

Some JavaScript tests, like `_TeamMate.spec.js`, are nesting `describe` instructions in an iteration block.

This decrease the test code readability, and produces longer and sometime truncated test log lines, which may themselves be hard to read.

Recommendation:

Inline the tests context and assertions whenever possible. When nesting is required, do not go deeper than three levels down.

Regarding this specific test, using something like a shared example and reusing it for each scenario may increase readability while keeping the code short.

2.5.12 Front-end and back-end code are tightly coupled

Ranking: Improvement

Criticality: High

Description:

The Garden Party application is, in itself, a Ruby on Rails application which is building and serving a Vue application as part of its assets. It is also testing the front-end behaviour thanks to Ruby tools.

Developing a front-end feature requires using the entire project Ruby development and testing stack, as well as some knowledge to set this up.

This is not required, as both the front-end and back-end could live as separate applications, with reduced requirements and coupling.

It will allow deploying front-end updates without changing the Rails application code, and vice versa.

The Ruby on Rails API will truly become standalone and open to people wanted to use Garden Party data with their own server.

Recommendation:

The front-end code should be extracted in a separate project / repository.

All front-end tests done by the Ruby on Rails application will be rewritten with vitest or another integration testing utility, which will bring speed to both the back-end deployment pipeline.

This will require a new API authentication system for the JavaScript client application to work, but will greatly improve both JavaScript and Ruby developers experiences.

2.5.13 The full Ruby and JavaScript test suites are taking some time to run

Ranking: Improvement

Criticality: High

Description:

The test coverage, as described a few paragraphs above, is a good tool to measure that a project has tests, and that said tests cover a good part of the application.

As far as Garden Party is concerned, its entire test suite is running in under 12 minutes on its GitLab CI, which may be an impediment in the long run.

Recommendation:

There are no issues with “too many tests”, but there could be an issue if these tests were slowing down the project development and releases.

It is to be said that most of this test time is due to Cucumber features tests, which take around 6 minutes to run. Switching to more vitest tests may help reduce this duration while keeping a good front-end testing behaviour, without being coupled to some Ruby code.

This switch could be made now, or after the front-end and back-end codes are split into two repositories.

2.5.14 Front-end API requests are using the XMLHttpRequest method

Ranking: Improvement

Criticality: Medium

Description:

It's been 16 years since the `XMLHttpRequest` introduction. While this method is a valid solution to make requests from a JavaScript client, to a server, it has fallen favour since the introduction and widespread adoption of the Fetch API and associated `.fetch()` method.

Generally speaking, there is no more reason to favour the `XMLHttpRequest` boilerplate code as written in the Garden Party JavaScript client, as the Fetch API offer the same features with a **simpler and cleaner syntax**.

It also offers a modern approach to HTTP requests, with `Promise` or `async` / `await` support, without the need to write events listeners.

Recommendation:

The Garden Party `api` client should be rewritten to use the `fetch` method.

2.5.15 The JavaScript application is mixing rise-ui and storybook

Ranking: Improvement

Criticality: Medium

Description:

The Garden Party JavaScript code is using both the `rise-ui` and `storybook` projects for components documentation.

While both projects purpose does differ, there is no clear explanation on why they're mixed, and what role each of them has for the application.

Recommendation:

The front-end documentation should explain the `rise-ui` and `storybook` projects purpose and actual usage.

In the long run, the `storybook` documentation could be completely replaced by `rise-ui`.

2.5.16 Policies tests can be DRYed

Ranking: Improvement

Criticality: Medium

Description:

There is a lot of duplication inside policies' tests, mainly around two contexts `with no authenticated user` and `with authenticated user`.

Recommendation:

Use a shared example to rewrite these tests.

2.5.17 The `database_cleaner` gem may not be required

Ranking: Improvement

Criticality: Medium

Description:

Using the `database_cleaner` gem may not be required as RSpec already include a database transaction mechanism, thus truncating the database after each test suite.

This mechanism is already enabled in Garden Party, with the `use_transactional_fixtures` configuration option and should work out-of-the-box.

Recommendation:

As the transactional fixtures option is already enabled and the project is using System specs instead of Features specs, with a Rails version greater than 5.1, it should not encounter issues when removing the `database_cleaner` gem.

Related:

- RSpec 3.7 has been released!
- Is DatabaseCleaner still necessary with Rails system specs?

2.5.18 The HAML rendering gem should be replaced by hamlit for better performances

Ranking: Improvement

Criticality: Medium

Description:

The admin part of the application is relying on the HAML abstraction language to ease views code writing.

In order to convert this HAML code to HTML, the Ruby on Rails server relies on the `haml-rails` gem to render each view when requesting.

This gem uses the standard `haml` implementation which, since its inception, has been outperformed by the hamlit gem, with a 1.5x performance increase.

Recommendation:

If you want to keep the view generator, replace the `haml-rails` gem with the hamlit-rails gem.

Otherwise, you could just replace it with the `hamlit` gem for view processing.

2.5.19 Extract domain logic from controllers, models and validators

Ranking: Improvement

Criticality: Medium

Description:

The Garden Party application is following the MVC pattern and the underlaying *"Rails Way"* offered by the Rails framework.

This is considered good practice as this came with a standard set of conventions that each Rails developer knows, but it can be difficult to scale once the codebase reach a certain size.

In most cases, the application models came to carry most of the domain logic, and relies heavily on callbacks and validations to control the data flow.

With time, these models start to grow longer and harder to read. Their behaviour becomes harder to test and can lead to unexpected errors and cause chain reactions.

Concurrently, controllers may become heavier as they are riddled with more and more domain actions or external services code.

One way to overcome such situation is to extract the domain (*or business*) logic outside of the application logic, leaving the controller to handle the requests and the models to persist the data.

Recommendation:

There is multiple way to decouple the application logic from the domain logic. Each project and developer may favour one or the other depending on their preferences and requirements.

Here are some well-known methodologies:

- Domain Driven Design, a design approach.
  - Arkency's book on the matter.
- Hexagonal Architecture, a design pattern, sometimes used alongside DDD.
  - An article about it.
- Trailblazer by Nick Sutterer, a gem, a book and a methodology.
  - It's main website.
- Rails-oriented patterns by Selleo, a set of patterns from multiple philosophies.
  - The pattern gem, regrouping them all.
- Service Objects, a common love-or-hate pattern in the Rails world, usually targeted to fat controllers.
  - An article about them from Toptal.

Related:

- Architecture the Lost Years by Robert Martin.
- Jim Weirich on Decoupling from Rails.
- GitLab refactoring proposal around Hexagonal architecture.

2.5.20 The application is requiring the whole Ruby on Rails stack, even if it is not using parts of it.

Ranking: Improvement

Criticality: Medium

Description:

In its `application.rb` configuration, the Garden Party application is requiring the `rails/all` dependency. While this is the default value in a new Rails application, the application is not using all the resources included.

For example, the `ActionCable` dependency, both imported from this statement and in the JavaScript application (`@rails/actioncable`), is not used at all by the application.

Recommendation:

The maintainers should only include the Rails parts they use, and remove all `require` statements, gems and related boilerplate files from the Ruby and JavaScript code.

2.5.21 Rails admin views could be componentised to avoid code repetition and improve maintainability

Ranking: Improvement

Criticality: Medium

Description:

Currently, the Garden Party admin front-end is not part of the Vue.js application. It's a distinct Rails route, with HAML views and dedicated controllers.

This segmentation results in a lot of components being used as Vue.js components and as Rails partials or helpers.

Additionally, while writing a new admin view, one may have to add a lot of HTML structure (*forms*, *alerts*, *tables*, *filters*...) and CSS classes to their HAML code, as all the view components are not available as helpers.

Recommendation:

A good first step would be to remove existing repetition in the admin codebase, by using more view helpers or a distinct Ruby solution like GitHub's ViewComponents.

If this is not enough, another solution could be to migrate the existing admin application to a Vue.js application, and reuse existing Vue.js components, thus migrating the Ruby on Rails project to an API-only mode.

Finally, if all admin actions are or could be available in the Vue.js app, the admin codebase should as well be removed.

2.5.22 The VueX store is accessed directly from the front-end components

Ranking: Improvement

Criticality: Low

Description:

The Garden Party Vue.js code is accessing the VueX store directly from its components, with code like `this.$store.dispatch()` or `this.$store.getters.something`.

Recommendation:

While accessing the VueX store through the `this.$store` public instance is not a bad practice, it can be improved by using VueX helpers like mapGetters and mapActions.

These helpers offer the following advantages:

- A shorter syntax for similar results.
- Less verbosity if the code is calling the property multiple times.

This is also an efficient way to:

- Read which data come from the store.
- Track all declarations, as they're all in one place.

And avoid redundant code like:

```
familiesToSync () { return this.$store.getters.familiesToSync }
```

2.5.23 The GardenMap state is updated through an event bus

Ranking: Improvement

Criticality: Low

Description:

The `GardenMap` component is not a Vue.js reactive component. Due to its dependency to the OpenLayers map system, it is coded as a JavaScript class, which itself is embedded in a `Garden` Vue.js component.

To guarantee the `GardenMap` update, the `Garden` component introduces an EventBus, with multiple listeners calling `GardenMap` methods.

This is the only place where this EventBus pattern is used but, as it is central to the Garden Party front-end, code tied to it can be found elsewhere, like in the main `App` component.

Recommendation:

The Garden Party front-end is already using VueX for central state management. It should not rely on another third party to update its components, as it adds complexity with no apparent benefit.

At first, it should be possible to centralise all the `GardenMap` state modifiers in the VueX state. Once done, it will be possible to replace the pub-sub system with watchers or subscription triggered when state's properties change.

Then, if the maintainers think it's a pertinent, the `GardenMap` component could be rewritten to be full reactive and directly using the VueX store to update itself.

2.5.24 Picture analysis could be delayed through ActiveJob

Ranking: Improvement

Criticality: Low

Description:

On a new picture upload, the Ruby on Rails code is doing a synchronous `picture.analyze` call.

This method relies on third-party libraries to extract metadata from the file, and can be slow depending on the server performances and the file size.

Recommendation:

The application should use the `analyze_later` method instead, so that the analysis task is enqueued in ActiveJob.

Please note that this is the default ActiveStorage behaviour, so manual calls to `analyze` or `analyze_later` **may not be required in the first place**.

2.6 UI & UX design

2.6.1 Some of the application components are documented and exposed through a design system

Ranking: Good practice

Description:

Using a design system helps continuous improvement and reutilisation of components, while enforcing the organisation distinct look.

It also allows designers and integrators to work on improving usability at a high level, while developers can focus on updating the component once it is ready.

2.6.2 The application is responsive

Ranking: Good practice

Description:

Most of the Garden Party application UI components seems to be responsive. The application even includes a manifest, allowing a user to add a valid shortcut on their phone home screen.

2.6.3 Irreversible action alerts are not helpful

Ranking: Bad practice

Criticality: High

Description:

When removing a map layer, the confirmation message below is shown. It does not remind the user about what is going to be deleted, and is not listing the deletion side effects.

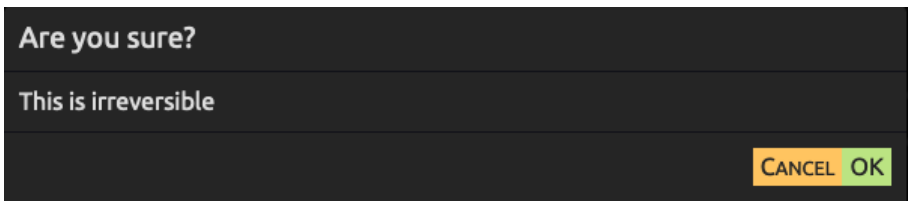

Figure 1: A confirmation message

Recommendation:

A definitive action like a deletion must, at the very least, recall the subject of the deletion, and strongly emphasis if related entities are going to be removed too (*if this is the case*).

The application alerts should be updated accordingly.

2.6.4 The application contrast ratio must be increased

Ranking: Bad practice

Criticality: Medium

Description:

Garden Party is using its own design system, with a default light theme, and an optional dark theme, automatically chosen depending on the user system settings.

While implementing a dark theme is not a bad practice per se, it must follow basic design requirements to remain readable and accessible.

For Garden Party, the main issue is the overall application contrast ratio.

At a minimum, a designer or integrator must make sure the contrast ratio between colours is no lower than `4.5:1`. And for custom foreground and background colours, they must strive for a contrast ratio of `7:1`, especially in small text.

That's not the case for Garden Party, where contrast ratio is as low as `1.38:1` on some "Legend" elements, and `1.11:1` for some "default" buttons, checkboxes or tabs.

Generic text contents are reaching a higher `7.5:1` contrast ratio, even if some alert texts have a `2.21:1` contrast ratio.

Recommendation:

The `rise-ui` design system and the Garden Party application must be updated to improve accessibility around the dark theme.

By using tools like Coolors Contrast Checker, the browsers *Accessibility* tools or extensions like WCAG Contrast checker, a developer can easily check its components for compliance and improve them for a greater accessibility.

The Garden Party application should also provide a way to toggle this dark theme.

Related:

- Apple - Dark mode.

- Apple - Accessibility - Color and effects.
- Material System - Dark theme.
- Web Content Accessibility Guidelines (WCAG).

2.6.5 There is no way to go back to top on the observations timeline

Ranking: Bad practice

Criticality: Medium

Description:

When scrolling down the timeline, or reaching a specific year, there is no easy way to go back up.

Recommendation:

Add a "Scroll back to top" button to the timeline component.

2.6.6 The onboarding process is lacking some visible and understandable instructions

Ranking: Improvement

Criticality: High

Description:

The Garden Party onboarding is the first part a user encounter when starting to use the application. It must be easy to grasp and use.

Instead:

1. On the first step, the "Picture selection", no validation button is visible unless a picture is uploaded. There is no obvious information explaining why
2. On the "Center definition" step, the help message is not immediately visible. A user may wonder what's expected from them and why the "Save this map" button is hatched.
3. Globally, it does not explain why a user must do what they are doing.

Recommendation:

The onboarding steps must be re-crafted with great care, while keeping in mind that not all users are computer-savvy.

A specific UX study must be run to understand what the user expectations are and how to help them initiate their first map with reduced friction.

2.6.7 Avoid covering the whole website with a loader

Ranking: Improvement

Criticality: High

Description:

When switching from the map tab to the observations tab, a loader screen is shown, covering the whole screen, including the navigation bar.

This give the impression that the whole page is reloading, make the tab bar disappear and if the loading time is short, cause a “flickering” feeling.

Recommendation:

Do not cover the whole screen with the loader component. Instead, use the loader component as the default tab content while the real content is loading.

You may also use a skeleton loader to act as a visual placeholder for the content while it is loading. This improves user experience by reducing load time frustration and ensuring responsiveness and a sense of activity while avoiding a blank dark screen.

2.6.8 Use CSS cursors to help understand available actions

Ranking: Improvement

Criticality: High

Description:

Many parts of the Garden Party front-end can benefit from custom cursors as offered by the CSS `cursor` property, to enhance understanding of a button or menu action.

Recommendation:

Use the following `cursor` CSS values for the described use cases:

- `not-allowed` for a disabled button.
- `grab` and `grabbing` for the drag-and-drop map side menu.
- `help` for a help button.

- `wait` for any waiting action, like clicking a button or loading a new screen.
- `context-menu` for a contextual menu.

2.6.9 Context menu navigation is using two icons to say the same thing

Ranking: Improvement

Criticality: High

Description:

The Garden Party front-end is showing both a “triple-dot” and an arrow to help users identify its contextual menu.

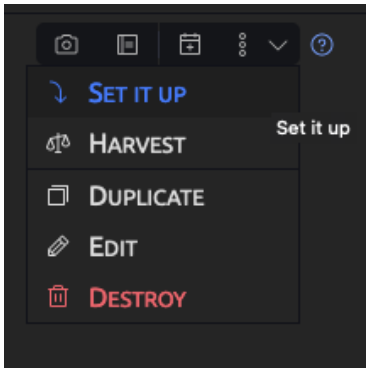

Figure 2: A context menu

Recommendation:

Both symbols are the representation of a contextual menu. The application must only show one of them, with the context-menu cursor on hover.

2.6.10 In guest mode, the “observations” tab is duplicated, while referring to a different feature

Ranking: Improvement

Criticality: High

Description:

While browsing the app, a guest user can notice that two “observations” actions are shown. One is a real button triggering a modal window while the other is a tab.

Both are showing a different content.

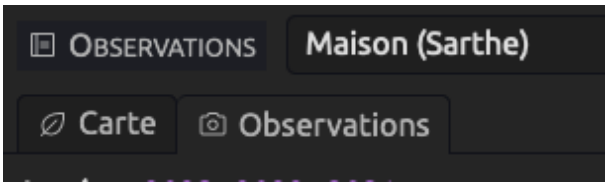

Figure 3: The observations tab and button

Recommendation:

Two actions with different results and content must not be close to each other.

They should not be named in the same way. One can imagine that the button can keep its name while the tab is renamed “Timeline” or “History”.

2.6.11 Deleting a path hides all its resources

Ranking: Improvement

Criticality: Medium

Description:

After deleting a path that includes resources, said resources are no longer visible on the map. It was not obvious that they were going to be deleted, and they are still listed in other tabs.

Recommendation:

If the deletion is making the resources “orphans”, they should be gathered in a specific and visible part of the map or below the layer panel.

If they are really deleted, they should no longer appear in other tabs.

In all cases this behaviour should be described in the deletion confirmation message.

2.6.12 Some buttons are hiding the tab bar

Ranking: Improvement

Criticality: Medium

Description:

The breadcrumb appears to be an important part of the application navigation. Yet, it can be hidden, for example when a user clicks on the “Library” button, which is included in the tab bar.

Recommendation:

Hiding or replacing the tab bar should be avoided at all costs. Instead, modal windows or side panels can be used to display categorised or hierarchical information.

2.6.13 The map layout should show plants information on hover

Ranking: Improvement

Criticality: Medium

Description:

The garden map is using alphabetical information to help associate a resource with its meaning, through the legend panel or by clicking an item.

The problem is, the legend panel is hidden by default, so there is not immediate way to read a map.

Recommendation:

Legends are very useful to read a map in the physical world. In the digital space, they still serve their purpose, but they can be complemented with the hover behaviour.

For example, when hovering a resource, a small tooltip can display its name and properties.

2.6.14 Map scale can be 200 km and can’t be limited

Ranking: Improvement

Criticality: Low

Description:

A user can create a map that is more than 200 kilometres wide. This does not seem realistic.

Figure 4: A very large map scale

Recommendation:

The application should limit the maximum size a map can reach. It should also allow the user to input realistic limits for his own property, and to update them at a later time.

2.6.15 Avoid showing the lightbox arrows if there is no previous or next

Ranking: Improvement

Criticality: Low

Description:

In the observations tab, a user can browse a timeline of events related to their garden.

If a timeline entry has one or more picture attached, it is possible to click on the picture, and it will be displayed in full-screen.

If there is more than one picture, arrows are available to browse them all. But if there is only one, the arrows are still displayed, even if they are disabled.

Figure 5: A lightbox with disabled arrows

Recommendation:

Displaying arrows when they are not needed adds unnecessary visual information to the screen. Moreover, even if the buttons are disabled, the white arrows are still highly noticeable, which could lead users to click on them.

Removing the arrows when there is only one picture available will fix this issue.

2.6.16 The map zoom capacities should be improved

Ranking: Improvement

Criticality: Low

Description:

Visible zoom features are the zoom in and zoom out buttons. There is no way to reset or recentre the zoom.

Moreover, there is no clear indication stating that zoom is also possible with the mouse wheel.

Globally, the map usage requires that the user as previous knowledge of similar map UI.

Recommendation:

A reset button should be listed next to the zoom buttons. Upon a user action, it will recentre the map and reset the zoom.

To facilitate the map usage, some contextual help should also be added near it.

One can imagine that the onboarding steps can also include a “How-to use” explanation or that the user documentation could display a video tutorial to help learn the basics of “garden mapping”.

2.6.17 Hide withdrawn items from the main inventory

Ranking: Improvement

Criticality: Low

Description:

In the inventory screen, resources are all shown, even if they are withdrawn. There are marked as `Removed at` and are shown with reduced opacity.

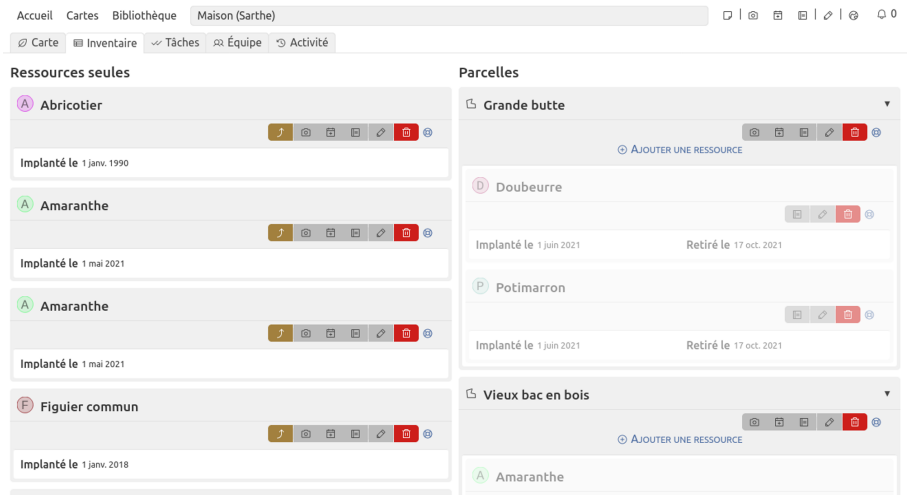

Figure 6: An inventory with removed items

Recommendation:

When a user access its inventory, they may not need immediate information regarding withdrawn items.

Showing these items leads to information overload and confusion, because the withdrawn items are listed in the same way as normal ones.

Withdrawn items should be hidden by default, and only shown when the user clicks on a dedicated “Show withdrawn items” toggle.

3 Conclusion

Garden Party is a fine example of an open source application developed in accordance with best practices.

From a technical point of view, it combines simple, efficient data structuring, a modern, secure Ruby on Rails code base, and an effective Vue.js front-end, albeit a little more difficult to access.

Everything is carefully tested and any complex code is documented. Modifying and making this codebase evolve should be a pleasure.

Admittedly, a few black spots have been identified, but these will be easily rectified and in no way jeopardise the application future.

From a user's point of view, however, we have noted a lack of project deployment documentation, which seems to us to be an obstacle to the adoption of Garden Party within the Open Source community.

In addition, the user interface can be greatly improved, both in terms of good accessibility practices and of ease of use for new users.

Here too, we think that the project's contributors will be able to adapt and greatly improve the application.

To sum things up, we think Garden Party is a stable and production-ready open-source project, which will, once our more critical recommendations will have been taken into account, benefit from being shared and used by the open source community.

David Dérus El
davidderus.fr
24/11/2023