Software Engineering Company Policy

RD Research

Author Max Durrant

Updated May 2023

Contents

1 Introduction & Capabilities

RD Research is a small but highly professional bespoke software development company based in Norwich. We specialise in providing customised database, CRM, and ERP development solutions tailored to the specific needs of our clients. Our expertise lies in managing multiple data sources, including various databases, spreadsheets, and other third-party data sources into a single, centralised database and CRM.

We have a proven track record of delivering projects on time and within budget, consistently meeting and exceeding our clients' expectations. This reputation for reliability and excellence extends to our work with the NHS, Banks, insurance companies and major on-line retailers including Amazon, where we have developed systems to streamline their processes.

Our solutions are designed to meet unique requirements, yet are competitively priced. At RD Research, we pride ourselves on understanding our clients' needs and delivering exemplary bespoke software solutions that drive business growth and success. Our commitment to delivering on time and on budget, coupled with our ability to create tailored solutions, has solidified our reputation as a trusted partner in software development and data management.

The purpose of this document is to ensure a consistent, secure, and high-quality approach to our software development lifecycle, from code creation, review, and version control, to agile methodologies and DevOps practices. We cover best practices for our primary programming languages and frameworks, as well as essential services such as Amazon Web Services (AWS) and Laravel Forge.

The document serves as a single source of truth for our software development operations and is to be updated regularly. It ensures that all team members are on the same page, contributing to our shared mission with clarity and confidence.

By consistently following these guidelines, we are not only ensuring the smooth and efficient functioning of our operations but also enhancing the quality of our products, benefiting our clients and business in the long run.

2 Software and Services Utilized

At RD Research, we use a diverse range of software and services to facilitate our software development and business operations. This section outlines the primary tools we use:

- **Code Development:** We use IDEs and text editors like Visual Studio Code, Php-Storm, and WebStorm. For frontend development, we use libraries and frameworks such as React, Vue.js, most commonly this would be NuxtJS.
- **Version Control:** We use Git for version control, with repositories hosted on GitHub. We also maintain a local repository on our TrueNAS system.

- **Project Management:** We manage our projects and track issues using JIRA.
- **Continuous Integration/Continuous Deployment (CI/CD):** Our CI/CD pipeline is facilitated through Jenkins, and we use Docker for containerization.
- **Database Management:** We use MySQL Workbench for managing and designing our databases.
- **Server Management:** Our servers are managed through Laravel Forge, and we use AWS services such as EC2 for server hosting.
- **Backup and Storage:** We utilize AWS S3 for backups and AWS EFS for large file storage.
- **DNS Management:** We manage our Domain Name System (DNS) configurations using Cloudflare. Our domains are registered through registrars like 123-reg and Directnic.
- **Email:** Our email services are managed through MXRoute.
- **Collaboration and Communication:** For team communication and collaboration, we use Microsoft Teams.
- **Documentation:** Our technical documentation is created and maintained using platforms Overleaf and Confluence.
- **Security:** We use various tools and services to ensure our software's security, including SSL certificates, firewall configurations, and security auditing tools.

These tools form the backbone of our software development and operational processes, enabling us to work efficiently and deliver high-quality products.

3 AWS Practices and Guidelines

At RD Research, we leverage AWS (Amazon Web Services) for our hosting and storage needs. We utilize various AWS services like EC2, RDS, S3, and EFS. Here are some guidelines for working with these AWS services:

- **EC2 (Elastic Compute Cloud):** EC2 instances should be chosen based on the requirements of the application. Ensure that instances are properly secured with appropriate security group rules. Regularly monitor the CPU, Memory, and Disk usage of your instances and adjust the instance type as necessary.
- **RDS (Relational Database Service):** Use RDS for managed database services. Regularly back up your databases and monitor the database performance. Enable Multi-AZ deployment for high availability and read replicas to offload read traffic.
- **S3 (Simple Storage Service):** S3 is used for backups and storing static files. Enable versioning to keep track of and retrieve all versions of an object. Use lifecycle policies to automatically move older backups to cheaper storage classes and delete them after a certain period.
- **EFS (Elastic File System):** Use EFS for storing large files that need to be accessed across multiple EC2 instances. Always enable lifecycle management on EFS to move infrequently accessed files to a cost-effective storage class.

- **Security:** Ensure that all AWS resources are secured using AWS Identity and Access Management (IAM). Regularly review and update the security policies.
- **Cost Optimization:** Regularly review the usage and costs associated with your AWS resources. Consider reserved instances for EC2 and RDS if you have predictable usage to save on costs. Utilize cost-saving storage classes in S3 and EFS.
- **Monitoring and Alerts:** Use CloudWatch to monitor your AWS resources and set up alerts for any unusual activity or thresholds.

By following these guidelines, we ensure that our AWS resources are managed effectively, securely, and cost-efficiently Amazon Web Services, 2023.

4 Forge Practices and Guidelines

At RD Research, we use Laravel Forge for server management and automated deployments. Forge connects to your repositories and deploys your applications with ease, while also simplifying many common server management tasks. Here are our practices and guidelines for using Forge:

- **Server Setup:** Use Forge to set up your servers. Forge supports multiple providers including DigitalOcean, AWS, and Linode, and it automatically installs necessary software such as Nginx, PHP, MySQL, and more.
- **Automated Deployments:** Link your GitHub repositories with Forge to enable automatic deployments whenever you push to a designated branch. This ensures your applications are always running the latest code.
- **Environment Management:** Use Forge's .env file management to configure environment variables for your applications. This ensures sensitive information is kept out of your codebase.
- **Monitoring and Alerts:** Use Forge to set up monitoring services and alerts for your servers and applications. This helps to quickly detect and respond to any issues that arise.
- **Security:** Forge automatically configures firewall settings and updates server software packages regularly to keep your server secure. Always review and follow the security best practices provided by Forge.
- **Backups:** Configure regular backups for your databases using Forge to prevent data loss.
- **SSH Key and Sudo User:** Use Forge to manage SSH keys and sudo users on your server. This provides a secure and efficient way to manage access to your server.

By adhering to these guidelines, we can manage and deploy our applications effectively and securely with Laravel Forge Laravel, 2023.

5 Laravel Practices and Guidelines

At RD Research, we use Laravel as one of our primary backend frameworks due to its elegant syntax, feature-rich ecosystem, and robust performance. Here are some guidelines for working with Laravel:

- **Adhere to Laravel's Directory Structure:** Laravel's directory structure is well organized and adheres to PHP's PSR-4 autoloading standard. Always follow this structure to keep the application maintainable and scalable.
- **Use Eloquent ORM:** Make full use of Eloquent ORM for database operations. Eloquent provides a simple and readable way to perform database operations and ensures that the application is secure from SQL injection attacks.

- **Leverage Laravel's Artisan Commands:** Laravel comes with a command-line tool called Artisan, which provides a number of helpful commands for common tasks. This includes creating controllers, models, migrations, and running tests.
- **Utilize Laravel's Middleware:** Use middleware to filter HTTP requests entering your application. Middleware provides a convenient mechanism for inspecting and filtering HTTP requests entering your application.
- **Take Advantage of Dependency Injection:** Laravel's service container is a powerful tool for managing class dependencies and performing dependency injection. It helps to keep the code flexible, maintainable, and testable.
- **Follow MVC Pattern:** Laravel is an MVC framework, so ensure that code is correctly organized into Models, Views, and Controllers. This pattern ensures the separation of concerns where each component of the application has a specific role.
- **Database Migrations:** Use Laravel's migration feature to version control your database. This approach ensures consistency across different environments.
- **Validation:** Use Laravel's built-in validation features to validate data before it reaches the controller. This can help to protect your application from invalid data and security vulnerabilities.
- **Use Laravel's Built-in Testing Tools:** Laravel provides a suite of testing tools to help ensure that new changes don't break existing functionality. Writing tests can help to catch bugs early in the development process.

By following these guidelines, we ensure that our Laravel applications remain robust, maintainable, and efficient Community, 2022.

6 Frontend Languages Practices and Guidelines

At RD Research, our primary frontend languages are JavaScript and TypeScript, which we use in combination with modern frameworks and libraries for creating intuitive and responsive user interfaces. Here are our practices and guidelines for using these languages:

- **Framework and Library Usage:** Make efficient use of the capabilities provided by the frontend framework or library in use (like React, Vue, or Angular). Understand their core concepts, follow their best practices, and utilize their established patterns for consistency and maintainability.
- **Component-Based Development:** Develop reusable components to improve code maintainability and consistency across the user interface. This approach facilitates testing and helps to keep the codebase DRY (Don't Repeat Yourself).
- **Asynchronous Programming:** JavaScript and TypeScript are inherently asynchronous, and understanding how to work with Promises, async/await, and callbacks is crucial. Always handle asynchronous operations properly to avoid unhandled promise rejections and callback hell.

- **State Management:** Depending on the complexity of your application, consider using a state management solution like Redux or Vuex. This can help you manage shared states more predictably and effectively across your components.
- **Accessibility:** Build interfaces that are accessible to all users, including those with disabilities. Follow the Web Content Accessibility Guidelines (WCAG) to ensure your web pages are accessible.
- **Performance Optimization:** Pay attention to frontend performance. Minimize the use of blocking operations, optimize render performance, and consider techniques like lazy loading to improve user experience.
- **Cross-Browser Compatibility:** Ensure that your code works consistently across different browsers. Use feature detection, polyfills, and transpilation (with tools like Babel) where necessary.
- **Error Handling:** Always implement robust error handling. This can include input validation, try/catch blocks for runtime errors, and proper handling of failed network requests.

By following these practices, we can create high-quality, robust, and user-friendly frontend applications Lindley, 2020.

7 Local Development Practices and Guidelines

Local development involves writing and testing code on individual developers' machines before committing changes to the version control system. At RD Research, we encourage practices that enhance productivity, ensure code quality, and maintain consistency across different local environments.

Here are our guidelines for local development:

- **Setup a Consistent Development Environment:** All developers should maintain a local development environment that closely mirrors our production environment. This includes using the same versions of languages, libraries, and databases. Docker is recommended for managing and isolating the development environment.
- **Use of Integrated Development Environment (IDE):** Choose an IDE that best suits your needs and the technology stack of the project. Configure it to match our coding standards and styles.
- **Testing:** Write unit tests and integration tests for your code. Run these tests locally to catch and fix errors before pushing your changes. This improves the quality of your contributions and reduces the time spent on fixing bugs later.
- **Performance:** Monitor the performance of your code locally. Use profiling tools to identify potential bottlenecks and optimize your code where necessary.
- **Security:** Implement secure coding practices from the start of local development. Regularly check your code for security vulnerabilities.

- **Version Control:** Regularly commit your changes to a separate branch in the version control system. This ensures your work is saved and allows you to track your progress.
- **Collaboration:** Keep your team members informed about what you are working on and any challenges you encounter. Seek feedback early and often.

These practices help to ensure high-quality, efficient local development that aligns with the team's overall goals and workflows Various, n.d.

8 Version Control Practices and Guidelines

Version control is crucial in managing changes to our codebase, tracking the history of modifications, and enabling collaboration among team members. At RD Research, we use GitHub for our production repository and a local repository on the TrueNAS system.

Our guidelines for version control practices are as follows:

- **Branching Strategy:** Every feature, bugfix, or task should be developed in a dedicated branch, created from the main branch. This prevents the main branch from having unstable code and enables isolated testing for each task.
- **Commit Often:** Commit your changes often. This creates a more granular history of the code, which can be useful for tracking changes and identifying the cause of bugs.
- **Informative Commit Messages:** Write clear, informative commit messages that explain what changes were made and why. Each commit message should include the corresponding JIRA issue code for traceability.
- **Pull Requests:** When a task is completed, create a pull request. This allows team members to review the changes before they are merged into the main branch.
- **Code Reviews:** Before a branch can be merged into the main branch, it must be reviewed and approved by at least one other team member.
- **Merge Carefully:** When merging branches, ensure you're not overriding or conflicting with changes made by others. Resolve any merge conflicts carefully to prevent bugs.
- **Sync Frequently:** Regularly sync your local repository with the remote repositories to stay updated with the latest changes made by others.
- **Backup:** All code is backed up on our local TrueNAS system to prevent data loss in case of any issues with the remote repository.

Following these practices helps us maintain an organized, traceable version history and facilitates effective collaboration GitHub, n.d.-b.

9 Code Structures Practices and Guidelines

In RD Research, we understand that well-structured code is vital for software maintenance, scalability, and collaboration. Our code structure practices and guidelines aim to ensure our codebase is logically organized and easy to understand.

- **Modularity:** Code should be separated into independent modules or components. Each module should have a single responsibility and encapsulate the related functionality. This makes the code easier to understand, test, and reuse.
- **Consistent Directory Structure:** The project's directory structure should be consistent and intuitive. Similar files (like models, controllers, or tests) should be grouped in dedicated directories. This helps team members to quickly locate and understand the context of a particular code file.
- **Use of Design Patterns:** Use appropriate design patterns where they make sense. Design patterns provide proven solutions to common software design problems and can make the code more flexible and easier to maintain.
- **Separation of Concerns:** Different aspects of the application, such as business logic, data access, and user interface, should be separated. This improves the maintainability and scalability of the software.
- **Commenting and Documentation:** Code should be as self-explanatory as possible. Use comments to explain why certain decisions were made or describe complex parts of the code where needed. Always document the purpose and usage of each module, class, method, and function.
- **Refactoring:** Regularly refactor the code to improve its structure and readability. Remove duplicate code, simplify complex parts, and ensure the code aligns with the current best practices.

These practices align with the general recommendations on code structures and software architecture McConnell, 2004.

10 Continuous Integration/Continuous Deployment (CI/CD)

At RD Research, we use GitHub, Docker, Jenkins, and our sandbox server in our CI/CD pipeline. Here's how we use these tools:

- **GitHub:** We use GitHub for version control and code hosting. All code changes are stored in branches and merged into the main production branch after being reviewed and tested GitHub, n.d.-a.
- **Docker:** Docker allows us to package our applications along with their dependencies into a container. This makes our applications portable and ensures they run consistently across different environments Docker, n.d.

- **Jenkins:** Jenkins is our CI/CD automation server. It's responsible for pulling the latest code from GitHub, building Docker containers, running tests, and deploying the code to our sandbox or production server if all tests pass Jenkins, n.d.
- **Sandbox Server:** Before deploying to production, new features and updates are first deployed to our sandbox server. This allows us to test our changes in a production-like environment without impacting our live users.

Our typical workflow is as follows:
1. Developers commit their code changes to a new branch on GitHub and create a pull request. 2. Once the pull request is created, Jenkins automatically builds a Docker container with the new changes and runs all automated tests. 3. If the tests pass, Jenkins deploys the new container to the sandbox server for further testing. 4. If the sandbox testing is successful and the pull request is approved, Jenkins deploys the new container to the production server. 5. The team monitors the application for any issues following the deployment.

11 Coding Standards and Styles

11.1 PHP 8 Coding Standards and Styles

11.1.1 General Guidelines

- Follow the PSR-1 and PSR-2 guidelines developed by the PHP Framework Interoperability Group "PSR-12: Extended Coding Style", n.d. These standards provide guidelines for basic coding standards and coding style guide respectively.
- Use UTF-8 encoding.
- Use Unix-style line endings (LF).
- End all PHP files with a single blank line.
- Make use of PHP's built-in linter ('php -l') before committing code to catch any syntax errors.

11.1.2 Indentation and Whitespace

- Use 4 spaces for indentation. Do not use tabs.
- Avoid trailing whitespace at the end of lines.
- Use a single blank line to separate logical blocks of code.
- Insert a space after control flow keywords like 'if', 'while', 'for', etc. Insert a space before and after the opening parenthesis and before the opening curly brace.
- Do not insert spaces after opening and before closing parentheses in function calls.

11.1.3 Naming Conventions

- Use camelCase for variables and methods. Use PascalCase for class names. Avoid using underscores as prefixes or suffixes.
- Choose descriptive and meaningful names. Variable and function names should clearly represent their purpose.
- Use uppercase letters for constants defined with the ‘define‘ function or the ‘const‘ keyword.

11.2 Control Structures

- Always use curly braces to delimit control structures, even if they only contain one line.
- Control structure keywords must have one space after them, followed by a space-separated condition.

11.2.1 Function and Method Calls

- Do not put spaces between the function or method name and the opening parenthesis, nor between the closing parenthesis and the semicolon.
- Put a space after each comma in the argument list.

11.2.2 PHPDoc

- All PHP files should start with a file-level docblock that contains a brief description of the contents and licensing information.
- Classes and methods should have a docblock that explains their purpose, inputs, outputs, and exceptions.

Remember, these are general guidelines and there may be some exceptions based on your specific use case or team preferences. It’s important to maintain consistency across your codebase, so ensure that everyone on your team understands and follows these conventions.

11.3 JavaScript Coding Standards and Styles

11.3.1 General Guidelines

- Follow the Airbnb JavaScript Style Guide “JavaScript Style Guide”, n.d. This guide is a widely accepted industry standard.
- Use UTF-8 encoding.
- Use Unix-style line endings (LF).
- End all JavaScript files with a single blank line.

11.3.2 Indentation and Whitespace

- Use 2 spaces for indentation. Do not use tabs.
- Avoid trailing whitespace at the end of lines.
- Use a single blank line to separate logical blocks of code.
- Add a space before the opening curly brace in blocks and functions.

11.3.3 Naming Conventions

- Use camelCase for variables and functions. Use PascalCase for class names. Avoid using underscores as prefixes or suffixes.
- Choose descriptive and meaningful names. Variable and function names should clearly represent their purpose.
- Constants should be in uppercase with underscores separating words.

11.3.4 Control Structures

- Always use curly braces to delimit control structures, even if they only contain one line.
- Control structure keywords must have one space after them, followed by a space-separated condition.

11.3.5 Function and Method Calls

- Do not put spaces between the function or method name and the opening parenthesis, nor between the closing parenthesis and the semicolon.
- Put a space after each comma in the argument list.

11.3.6 JSDoc

- All JavaScript files should start with a file-level docblock that contains a brief description of the contents and licensing information.
- Classes and methods should have a docblock that explains their purpose, inputs, outputs, and exceptions.

11.4 TypeScript Coding Standards and Styles

11.4.1 General Guidelines

- Follow the TypeScript Coding Guidelines provided by Microsoft.
- Use UTF-8 encoding.
- Use Unix-style line endings (LF).
- End all TypeScript files with a single blank line.

11.4.2 Indentation and Whitespace

- Use 2 spaces for indentation. Do not use tabs.
- Avoid trailing whitespace at the end of lines.
- Use a single blank line to separate logical blocks of code.
- Add a space before the opening curly brace in blocks and functions.

11.4.3 Naming Conventions

- Use camelCase for variables and functions. Use PascalCase for class names and interfaces. Avoid using underscores as prefixes or suffixes.
- Choose descriptive and meaningful names. Variable and function names should clearly represent their purpose.
- Constants should be in uppercase with underscores separating words.

11.4.4 Control Structures

- Always use curly braces to delimit control structures, even if they only contain one line.
- Control structure keywords must have one space after them, followed by a space-separated condition.

11.4.5 Function and Method Calls

- Do not put spaces between the function or method name and the opening parenthesis, nor between the closing parenthesis and the semicolon.
- Put a space after each comma in the argument list.

11.4.6 TSDoc

- All TypeScript files should start with a file-level docblock that contains a brief description of the contents and licensing information.
- Classes and methods should have a docblock that explains their purpose, inputs, outputs, and exceptions.

11.5 Python (PEP-8) Coding Standards and Styles

11.5.1 General Guidelines

- Follow the PEP-8 style guide for Python “PEP 8 – Style Guide for Python Code”, n.d. This is the official style guide for Python and is widely used in the Python community.
- Use UTF-8 encoding.
- Use Unix-style line endings (LF).

- Python files should end with a single newline character.

11.5.2 Indentation and Whitespace

- Use 4 spaces for indentation. Do not use tabs.
- Line length should not exceed 79 characters.
- Use spaces around operators and after commas, but not directly inside bracketing constructs: ‘a = f(1, 2) + g(3, 4)‘.
- Avoid trailing whitespace at the end of lines.
- Use a single blank line to separate functions, classes, and larger blocks of related functions.
- Use two blank lines to separate top-level functions and classes.

11.5.3 Naming Conventions

- Use snake_case for variables and functions. Use PascalCase for class names.
- Use ALL_CAPS for constants.
- Use self as the first method argument, for instance, methods and cls as the first method argument for class methods.

11.5.4 Imports

- Always put imports at the top of the file.
- Imports should be grouped in the following order: standard library imports, related third-party imports, and local application/library-specific imports.
- Each import should be on a separate line.
- Avoid wildcard imports (‘from mod import *‘).

11.5.5 Comments and Docstrings

- Use inline comments sparingly.
- Write comments in complete sentences.
- Use docstrings for all public modules, functions, classes, and methods.
- Docstrings should follow the conventions outlined in PEP-257.

Remember, a style guide aims to achieve greater readability and consistency in your code. While following this guide as closely as possible is good practice, don’t hesitate to make exceptions if it improves readability and understandability.

12 Code Review Practices

Code review is an integral part of the software development process at RD Research. It provides a means to maintain the quality of the codebase, share knowledge among team members, and prevent bugs from reaching the production environment. Here are the practices we follow:

- **Review Requirement:** All code must be reviewed and approved by at least one other person before it can be merged into the production branch. This applies to everyone, regardless of position within the company.
- **Pull Requests:** Code reviews are conducted via pull requests. When a developer has completed a task, they should open a pull request against the target branch.
- **Descriptive Messages:** Each pull request should have a descriptive title and a comprehensive description. The description should explain the purpose of the changes, how they were implemented, and any potential impact on the existing codebase.
- **Timely Reviews:** Team members are expected to review pull requests assigned to them promptly. This ensures a smooth and fast development process.
- **Constructive Feedback:** Code reviews should be constructive. Reviewers should provide clear, concise, and respectful feedback.
- **Actionable Comments:** Feedback should be actionable. Comments such as ”This needs to be fixed” should be avoided in favour of more specific instructions, like ”Consider refactoring this function to improve readability.”
- **Reviewer Approval:** If the reviewer approves the changes, they can authorize merging the pull request. If they request changes, the original developer should make the requested adjustments and then request another review.
- **Continuous Integration:** Use a CI system to automatically build and test the code when a pull request is opened. This helps to catch any integration errors or failing tests early.

These practices align with the recommendations given in Software, n.d.

13 Database Management and Migrations

13.1 General Guidelines

Proper database management is essential for maintaining data integrity and optimizing system performance. Here are some general best practices for managing databases in our environment:

- **Schema Design:** Carefully design your schema before implementing it. Consider the relationships between different data entities and normalize your data to eliminate redundancy.

- **Indexing:** Use indexes wisely to improve query performance. However, avoid over-indexing as it can slow down insert and update operations.
- **Backups:** Regularly back up your database to prevent data loss in case of system failures. Test your backups periodically to ensure they can be restored successfully.

13.2 MySQL Workbench

We use MySQL Workbench "MySQL Workbench Manual", n.d. as our standard database design and management tool. This versatile tool supports data modelling, SQL development, and comprehensive administration tools for server configuration, user administration, and much more.

- **Modeling:** Use MySQL Workbench for designing and modifying database schemas. It provides an intuitive graphical interface for designing database structures.
- **Development:** Use its powerful SQL editor for creating and executing SQL scripts. Its debugging tools help you debug and optimize your SQL code.
- **Administration:** Use MySQL Workbench for server configuration and administration. Its user management tools let you manage database access permissions.

13.3 Database Migrations

Database migrations are crucial for maintaining and updating the database schema. They enable version control for databases and make it possible to move databases across different systems.

- **Version Control:** Just like source code, database schema changes should be tracked using version control. Each change to the database schema should correspond to a specific migration script.
- **Automated Migrations:** Use automated migration tools where possible to reduce the risk of errors. Automated migration tools can generate migration scripts automatically when you modify the database schema using a GUI.
- **Testing:** Always test your migrations on a staging environment before applying them to the production database. This will help you catch and fix any potential issues before they affect your production environment.

13.4 Within Laravel

13.4.1 General Guidelines

Laravel "Laravel Documentation", n.d. provides a powerful suite of tools for database management. Here are some key practices to follow:

- **Eloquent ORM:** Laravel's Eloquent ORM provides a simple and fluent interface for querying and manipulating database records. It also allows for a more readable and maintainable codebase. Use it whenever possible.

- **Validation:** Always validate user data before it reaches your database. Laravel provides several ways to do this including form request validation, manual validation, and route model binding.
- **Security:** Be aware of potential SQL injection attacks. Laravel's query builder uses PDO parameter binding which protects your application from SQL injection attacks.

13.4.2 Migration Practices

In Laravel, migrations are like version control for your database, allowing a team to modify the database schema and stay up to date on the current schema state. Here are some recommended migration practices:

- **Generating Migrations:** Use the Artisan CLI provided by Laravel to generate migrations. The command 'php artisan make:migration create_users_table' will create a new migration file for a "users" table.
- **Running Migrations:** Use the command 'php artisan migrate' to run outstanding migrations. Always check for any database errors that may occur during this process.
- **Rolling Back Migrations:** If you need to undo a migration, you can use the 'php artisan migrate:rollback' command. However, be aware that this should be used carefully, especially in a production environment, as it can lead to data loss.
- **Adding Columns:** When adding columns to an existing table, create a new migration with the 'php artisan make:migration' command instead of editing an existing migration.

13.4.3 Seeding

Database seeding is a convenient way to populate your database with test data using seed classes. Seeding can be performed with the 'php artisan db:seed' command.

- **Generating Seeders:** Use the Artisan command 'php artisan make:seeder UsersTableSeeder' to generate a seeder for the "users" table.
- **Running Seeders:** After writing your seeder, use the 'php artisan db:seed' command to run your seeders. You can also use the '–class' option to specify a specific seeder class to run individually.

14 Secure Coding Practices

Secure coding is a set of techniques designed to prevent security vulnerabilities in code. Following these practices helps to protect a system against threats such as unauthorized access, data breaches, and cyber attacks.

- **Input Validation:** Always validate user input to prevent injection attacks. User input should never be trusted implicitly.

- **Error Handling:** Avoid revealing sensitive information in error messages. Use exception handling to catch and manage errors effectively.
- **Password Security:** Store passwords securely, ideally hashed with a salt value.
- **Secure Dependencies:** Regularly update and review the libraries and dependencies you use in your software to prevent security vulnerabilities.
- **Principle of Least Privilege:** Each part of the system should operate with the least amount of privilege necessary to complete its function.

14.1 Secure Communication

Secure communication is critical for protecting data in transit.

- **SSL/TLS:** Use Secure Socket Layer (SSL) or Transport Layer Security (TLS) to encrypt the communication between the client and the server. This prevents potential attackers from reading or modifying the data in transit. “Let’s Encrypt”, n.d.
- **HTTPS:** Always use HTTPS (HTTP over SSL/TLS) instead of HTTP for all web communications. This ensures the confidentiality and integrity of data between the client and server.

14.2 Cross-Origin Resource Sharing (CORS)

CORS is a security feature that can restrict how and when a document can request resources from a different origin.

- **CORS Policy:** A server defines its CORS policy in the HTTP response headers it sends back to the client. This policy specifies who (i.e., which origins) can access its resources. Contributors, n.d.
- **CORS Implementation:** When implementing CORS, ensure that your policy is as strict as possible, and only allows trusted web origins to request your server’s resources.

14.3 Environment Configuration Management

- **Environment Variables:** Keep all sensitive information like API keys, database credentials, and so forth in environment variables, not hard-coded in your application Wiggins, n.d.
- **Isolated Environments:** Keep development, staging, and production environments isolated from each other. Never mix data or configuration between them.
- **Version Control:** Environment configuration should be under version control, but without any sensitive information.
- **Automatic Configuration:** Automate the environment configuration process to avoid manual errors and to make it easy to recreate the environment when needed.

15 Documentation Practices

Documentation is a crucial part of software engineering. It helps the development team understand and maintain the codebase, and allows users to understand how to use the software. This section outlines best practices for code and written documentation Overflow, n.d.

15.1 Code Documentation

- **Code Comments:** Use comments to explain the purpose of complex code blocks, especially algorithms and business logic. Comments should be concise and directly related to the code beneath them.
- **Function/Method Comments:** All functions/methods should have a comment explaining what they do, what their inputs are, what they return, and any side effects they may have.
- **Class Comments:** Each class should have a comment at the top describing its purpose and how it interacts with other classes.
- **Variable Naming:** Use descriptive variable names to make the code self-explanatory. This is often better than a comment for understanding what a particular piece of code does.
- **Consistency:** Keep the style of your comments consistent throughout the codebase. This helps other developers read and understand your comments more quickly.

15.2 Written Documentation

- **User Manuals:** Create comprehensive user manuals for your software. These should include step-by-step instructions for using each feature of your software, as well as troubleshooting guides for common issues.
- **Developer Guides:** Write detailed developer guides. These should include instructions on setting up the development environment, explanations of the code structure and key algorithms, and guidelines for contributing to the project.
- **API Documentation:** If your software has an API, provide thorough documentation. This should include descriptions of each endpoint, the request/response formats, and example requests/responses.
- **Updates:** Keep your documentation updated. Whenever you make changes to your software, make sure to update the relevant documentation as well.
- **Accessibility:** Ensure your documentation is accessible and easy to navigate. Use clear headings, include a table of contents, and use plain language as much as possible.

16 DevOps Practices

DevOps combines software development (Dev) and IT operations (Ops) to shorten the system development lifecycle and deliver high-quality software continuously. At RD Research, we adhere to the following key practices in our DevOps culture:

- **Continuous Integration/Continuous Deployment (CI/CD):** We use CI/CD pipelines to automate the testing and deployment of our software. This allows us to detect issues earlier, deliver updates faster, and reduce the risk of deployment failures Fowler and Foemmel, n.d.
- **Infrastructure as Code (IaC):** We manage and provision our computing resources programmatically and version control these scripts. This reduces human error and enables consistent and reproducible environments "What is Infrastructure as Code (IaC)?", n.d.
- **Monitoring and Logging:** We continuously monitor our applications and systems to detect anomalies and performance issues. Logging helps us in troubleshooting and understanding system behavior.
- **Incident Response:** We have procedures in place to manage and respond to system incidents efficiently, minimizing downtime and impact on users.
- **Collaboration and Communication:** Effective communication and close collaboration between development and operations teams are central to our DevOps culture. This ensures that everyone has a comprehensive view of the system from end to end.
- **Security:** We incorporate security practices into every step of the development process. This includes conducting security reviews, using secure coding practices, and regularly updating and patching our systems.

17 Agile/Scrum Practices

17.1 General Overview

We at RD Research follow Agile development principles to promote sustainable development. Our Agile practices encourage a flexible response to change which is crucial in a dynamic business environment.

17.2 Scrum

Our preferred Agile framework is Scrum due to its simplicity and efficiency. We have weekly Scrum meetings to enhance communication, collaboration, and product quality.

- **Scrum Meetings:** Scrum meetings, or 'sprints', are held once a week. Each sprint starts with a planning meeting, where tasks for the week are defined and assigned to team members. During the sprint, team members work on the assigned tasks and meet daily for a quick status update (Daily Scrum/Standup).

- **End of Week Cool Down:** At the end of each week, we have a cool down period. This is a time for the team to review the work completed during the week, to reflect on what worked well and what improvements could be made for the next sprint. It's also a time for the team to unwind and recharge before the next sprint begins.

17.3 Integration with JIRA

We use JIRA "Jira Software: The #1 software development tool used by agile teams", n.d., a project management tool, to help manage our Agile workflow. This allows us to easily create and assign tasks, track progress, and manage changes.

- **Task Management:** Tasks for each sprint are created as 'issues' in JIRA. During the sprint planning meeting, these issues are assigned to team members based on their skills and availability.
- **Progress Tracking:** JIRA's Agile boards provide a visual representation of the progress of the sprint. Team members update the status of their tasks as they work on them, giving everyone a real-time view of the sprint progress.
- **Collaboration:** JIRA facilitates communication and collaboration among team members. Team members can comment on issues, attach files, and tag each other for updates.
- **Reporting:** JIRA provides robust reporting tools that can help the team understand their performance over time. These include velocity charts, burn-down charts, and cumulative flow diagrams.

18 Recommended Practices/Guidelines for a Software Engineering Company

Beyond the specific technical practices and guidelines covered in this document, there are broader software engineering principles and methodologies that we at RD Research adhere to. These guiding principles inform our approach to software development and contribute to a productive, collaborative, and high-quality software engineering environment:

- **Continuous Learning and Improvement:** The field of software engineering is ever-evolving. We encourage all team members to continuously learn and adapt to new technologies, tools, and practices. Regular training and learning opportunities should be provided to the team.
- **Collaboration and Communication:** Software development is a team effort. Open and transparent communication fosters a culture of collaboration and mutual respect, enhancing overall productivity and morale.
- **Quality Assurance:** Rigorous testing practices, including unit testing, integration testing, and end-to-end testing, should be in place to ensure the reliability and quality of our software.

- **User-Centric Design:** All software should be developed with the end-user in mind. We should strive to understand our users' needs and design our software to be intuitive and user-friendly.
- **Sustainability and Maintainability:** We strive to write clean, readable, and well-documented code that can be easily maintained and updated over time. This involves adhering to principles like DRY (Don't Repeat Yourself) and KISS (Keep It Simple, Stupid).
- **Security:** Security should be a top priority in all stages of software development. We must adhere to best practices in secure coding, data handling, and configuration management to protect our applications and users' data.
- **Performance:** We aim to build software that is not just functional, but also efficient. Regular performance profiling and optimization should be part of our development process.
- **Ethical Considerations:** As software engineers, we have a responsibility to consider the ethical implications of our work. We strive to build software that respects user privacy, promotes inclusivity, and does not harm the environment.

By adhering to these principles, we strive to build a software engineering culture that is focused on continuous improvement, collaboration, quality, and ethics Council, 2020.

19 Conclusion

This document serves as a comprehensive guide to the capabilities, practices, and procedures we adhere to at RD Research. From code creation to agile methodologies, local development to deployment, and database management to security, we have outlined our approach across all aspects of the software development lifecycle.

Maintaining and following these guidelines will facilitate a consistent, efficient, and high-quality approach to our work.

This document serves to guide existing team members and provides a reference for onboarding new members, ensuring a unified understanding of our practices. It may also be passed to clients so they can better understand how we work.

In a dynamic field like software and database development, it is important that a structured approach is used. As such, this document should be updated and augmented as necessary to stay in sync with emerging trends and best practices. Our ultimate goal is to maintain a software engineering culture that fosters learning, collaboration, quality, and innovation.

By adhering to these guidelines, we are confident in our ability to deliver outstanding software solutions that meet and exceed our clients' needs.

References

Amazon Web Services, I. (2023). Aws best practices [Accessed: 2024-05-24].
Community, L. (2022). Laravel best practices [Accessed: 2024-05-24].
Contributors, M. (n.d.). Cross-origin resource sharing (cors) [Accessed: 2023-05-24].
Council, F. T. (2020). Best practices for a successful software engineering culture [Accessed: 2024-05-24].
Docker. (n.d.). Empowering app development for developers [Accessed: 2024-05-24].
Fowler, M., & Foemmel, M. (n.d.). Continuous integration [Accessed: 2024-05-24].
GitHub. (n.d.-a). Github: Where the world builds software [Accessed: 2024-05-24].
GitHub. (n.d.-b). Understanding the github flow [Accessed: 2024-05-24].
Javascript style guide [Accessed: 2023-05-22]. (n.d.).
Jenkins. (n.d.). Jenkins: Build great things at any scale [Accessed: 2024-05-24].
Jira software: The #1 software development tool used by agile teams [Accessed: 2023-05-22]. (n.d.).
Laravel. (2023). Laravel forge documentation [Accessed: 2024-05-24].
Laravel documentation [Accessed: 2023-05-22]. (n.d.).
Let's encrypt [Accessed: 2023-05-24]. (n.d.).
Lindley, C. (2020). Front-end developer handbook [Accessed: 2024-05-24].
McConnell, S. (2004). *Code complete: A practical handbook of software construction*. Microsoft Press.
Mysql workbench manual [Accessed: 2023-05-22]. (n.d.).
Overflow, S. (n.d.). Best practices for writing code comments [Accessed: 2023-05-22].
Pep 8 – style guide for python code [Accessed: 2023-05-22]. (n.d.).
Psr-12: Extended coding style [Accessed: 2023-05-22]. (n.d.).
Software, S. (n.d.). Best practices for code review [Accessed: 2024-05-24].
Various. (n.d.). Local development best practices [Accessed: 2024-05-24].
What is infrastructure as code (iac)? [Accessed: 2024-05-24]. (n.d.).
Wiggins, A. (n.d.). The twelve-factor app: Iii. config [Accessed: 2023-05-24].