Home

To maximize efficiency within our teams, it is essential that everyone adheres to the same standardized processes. Consistency in our workflows not only enhances collaboration but also streamlines our operations, allowing us to achieve our goals more effectively. We kindly ask you to familiarize yourself with the following procedures to ensure that we are all aligned and working towards a common objective. Your cooperation in this matter is greatly appreciated.

Our Process

Your work is available on Jira. You can setup dashboards to make it simple to see the work that's been assigned to you. Additionally, you can use the filters to view the work that has been assigned to you.

Please access the Repository (see page 5) for the project and checkout the dev branch to begin your work.

When you start an issue remember to move the issue to **In Progress**.

You should always start with the dev branch and perform:

```
git fetch && git pull
```

Once you're on a clean and updated branch, you can create a new branch as shown in the Repository (see page 5).

Please remember to keep all work in it's respective branch. Do not make additional changes to any other code that's unrelated to completing the task in full.. If you see code that should be updated, please inform your project lead so we can document the bug within Jira and it can be handled on it's own branch and handled based on it's priority.

We will merge branches into the dev branch for Quality Assurance before releasing to production. Releases will be in accordance with our Release Schedule.

Once issues pass Quality Assurance they will be moved into the **Done** column and given a version number.

After you complete your work, leave a comment of the work that's been changed, and please transition the issue to **Quality Assurance**.

The completed work should be documented within the respective project within the same page. This will allow us to have an additional source for work that has been completed overtime and allow us to gather additional feedback from the development teams, such as process details being understood, how to gain efficiency, issues that were encounter, etc…

We use Jira automations to unassign and reassign issues to various please that control various roles of the Development Process such as Quality Assurance.

Documentation

Effective documentation is crucial for the long-term success and sustainability of any project. It serves as the backbone of project communication, ensuring that all team members remain aligned, have access to essential information, and can make informed decisions throughout the project's lifecycle. Without proper documentation, important knowledge can be lost, miscommunicated, or inaccessible when needed most.

Importance of Staying Aligned

Maintaining up-to-date and accessible documentation is key to keeping everyone on the same page. As a project evolves, so do its components, requirements, and objectives. Therefore, it is important that any updates, changes, or decisions made are consistently captured in real time and shared across the team. This ensures that no matter when or where someone joins the project, they have the necessary information to understand its current state and can contribute effectively.

Documentation as a Team Effort

Documentation should not be seen as the responsibility of a single individual but rather a collective team effort. Each team member brings unique insights and knowledge, and sharing that knowledge through documentation benefits the entire project. The best way to ensure comprehensive and accurate documentation is for everyone involved to actively contribute to it. Whether a developer is working on a feature, a designer is updating the interface, or a project manager is refining timelines, their contributions should be reflected in the project's documentation.

Confluence as a Central Hub

To keep all documentation organized and accessible, each project should have a dedicated **space in Confluence**, which acts as the central hub for all project-related information. This space should serve as a repository where the team can store and manage documentation on everything from project scope and goals to individual component details and ongoing discussions.

Within this Confluence space, new pages should be created for specific areas of the project, including individual components, features, or significant milestones. This allows team members to break down the project into more manageable parts and document the finer details of what they are working on. These component-specific pages foster collaboration and allow the team to collectively brainstorm, troubleshoot, and discuss potential improvements, leading to a more refined final product.

Recording Changes, Decisions, and Timelines

Project documentation should not be static. As changes occur, whether due to a shift in project scope, new information, or decisions made during meetings, they must be promptly documented. This includes updates to timelines, feature changes, new risks, or solutions identified during development. It is vital to regularly update and review documents to reflect these changes accurately. This ensures that the documentation evolves alongside the project, preventing outdated or incorrect information from leading to confusion or delays.

Linking Jira Issues to Confluence

For projects utilizing both Jira for task management and Confluence for documentation, it is essential to create a seamless link between the two platforms. **Linking Jira issues to Confluence documents** is a best practice that helps bridge the gap between task tracking and project documentation. By doing so, you gain a more complete understanding of a component, feature, or task by viewing both the related development tasks and the documentation that provides context and insight.

For instance, if a team member is working on a specific feature, they can create a new page in Confluence to document their progress, technical details, and design decisions. Then, by linking the corresponding Jira issue, the team can quickly access both the development tasks and the supporting documentation in one place. This approach not only enhances transparency but also makes it easier for future team members to understand the history and context of each component.

Collaboration and Continuous Improvement

One of the key benefits of maintaining well-organized documentation is the ability to facilitate ongoing conversations about improving the project. When documentation is up to date and easily accessible, team members can engage in meaningful discussions on how to enhance specific components, fix issues, or optimize workflows. These conversations can happen directly within Confluence, allowing teams to comment, share feedback, and iterate on ideas in real-time.

Moreover, as the project progresses, this wealth of documented knowledge becomes a valuable resource that can be referenced and built upon. By documenting not just the "what" but the "why" behind decisions, teams can better understand the rationale behind each choice and ensure that future work aligns with the project’s overall goals.

Conclusion

In summary, **documentation is not just a requirement but a strategic tool** for ensuring that projects run smoothly and effectively. By using Confluence as the central hub for all documentation and linking it with Jira for task tracking, teams can maintain clear communication, foster collaboration, and ensure that valuable information is always accessible. As a team, continuously contributing to and updating this documentation will ensure that the project remains transparent, organized, and on track, regardless of changes or future challenges.

Release Schedule

In our commitment to minimizing disruption during updates, our CI/CD workflows will incorporate a status page to indicate when an application is undergoing maintenance. We will deploy to a temporary folder within the CI/CD pipelines, ensuring that the transition to production is seamless and that users either see the status page or experience no interruptions. Should any pipeline fail, it is crucial to consult the logs for troubleshooting. After merging your changes, please verify their functionality on both development and production sites. If any issues arise, promptly create a new branch to address and resolve the problem. This structured approach promotes stability and reliability in our deployment processes.

Release to Deployment

We currently implement a continuous integration and continuous deployment (CI/CD) strategy to facilitate regular releases to our development environment. Upon opening a Merge Request for the relevant branch, the associated issue is transitioned to the Quality Assurance status, where it undergoes thorough testing to ensure compliance with our quality standards before moving forward in the deployment pipeline. This streamlined process enhances collaboration and maintains the integrity of our development workflow.

Release to Production

We currently implement a weekly release cycle from development to production in conjunction with the conclusion of each sprint. Issues that have successfully passed Quality Assurance are merged into the production environment. Items marked as "Ready For Release" are integrated into the main branch, and upon successful merging, they are updated in Jira, transitioning to the "Done" column. Additionally, these issues are included in the latest version release.

Please be advised that all production releases will occur between 9:00 PM and 10:00 PM Eastern Standard Time on Fridays. This schedule is designed to ensure optimal performance and availability with as little interruption to our client as possible. We appreciate your understanding and cooperation as we implement these updates. Thank you for your continued support.

Merge Request Process

A Merge Request is a fundamental concept in Git-based version control systems like GitLab and Bitbucket. It allows developers to propose changes to a codebase and request that those changes be incorporated into the main branch. This process involves code review, testing, and ultimately merging the changes into the main codebase. Merge Requests are essential for collaboration and maintaining code quality in software development projects.

Before creating a merge request ensure the follow:

- Ensure that the code changes are thoroughly reviewed and tested.
- Verify that the changes align with the project's coding standards and guidelines.
- Confirm that the merge request includes a clear description of the proposed changes.
- Check for any potential conflicts with the main branch.
- Ensure that all necessary approvals are obtained before merging.
- Verify that any associated issues or tasks are linked to the merge request.
- Double-check that the merge request title and description are informative and concise.
- Confirm that the merge request is targeting the correct branch for integration.
- Ensure that any relevant documentation updates are included in the merge request.
- Verify that the automated tests pass successfully.
- Confirm that the merge request has been assigned to the appropriate reviewers.

When you are satisfied that your code is ready to be published please open a Merge Request to the **dev** branch. If a dev branch does not exist, please notify your project lead.

- The merge request should have a clear and concise description of what changes have been completed. You can copy the commit text into the description.
- The merge request should not delete the branch.
- Please make sure if possible that any new models or database changes are created in a migration and the migration files exist, otherwise, please create and include the a sql file in the sql directory and include it in the commit before opening a Merge Request. Additionally if using an sql file please make sure that the description gives reference to the file and in Jira and that a database change needs to be made.

In the event that a merge request is open and has identified issues, please close the existing Merge Requests before opening a new one.

Support Processes

Repository

Our current code is hosted at https://gitlab.com/paradynamix

Please ask @Stephan Shere for access to the repository.

Git Workflow

Please Work on small code changes. Commit often and ensure your branches are separated as well as possible. Use different branches for each fix and/or issue.

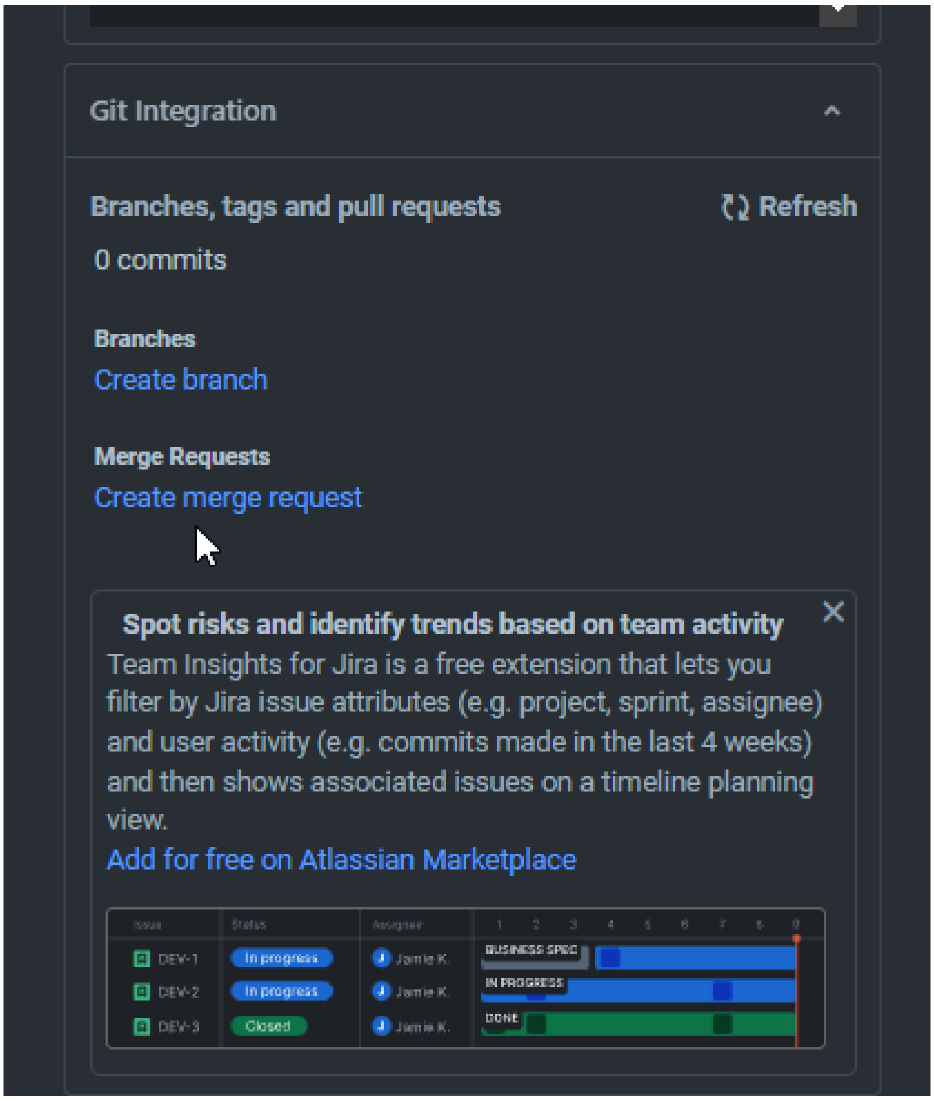
Perform a merge request when you are comfortable that the current branch is completed.

Branches before main/master are based off releases- branches.

To start work, create a new branch off development

Each branch for a new issue should be a fix/ or a feature/ branch.

Branches should be made using the Jira Integrations within the issue.



Git Integration

Branches, tags and pull requests ⟳ Refresh

0 commits

Branches

Create branch

Merge Requests

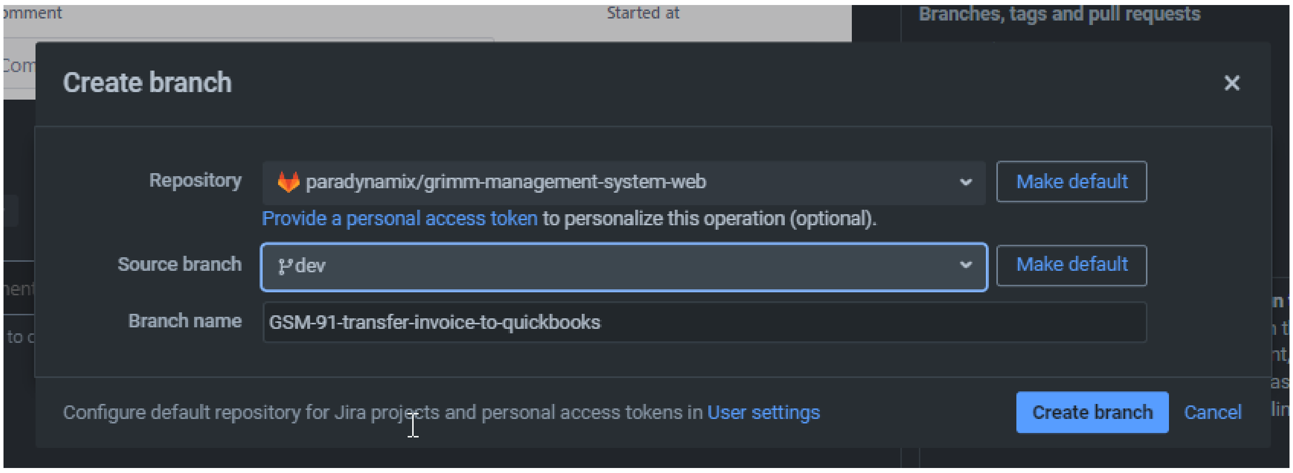
Create merge request

Spot risks and identify trends based on team activity

Team Insights for Jira is a free extension that lets you filter by Jira issue attributes (e.g. project, sprint, assignee) and user activity (e.g. commits made in the last 4 weeks) and then shows associated issues on a timeline planning view.

Add for free on Atlassian Marketplace

Use Create Branch to create a new branch quickly and then pull it on your local environment.



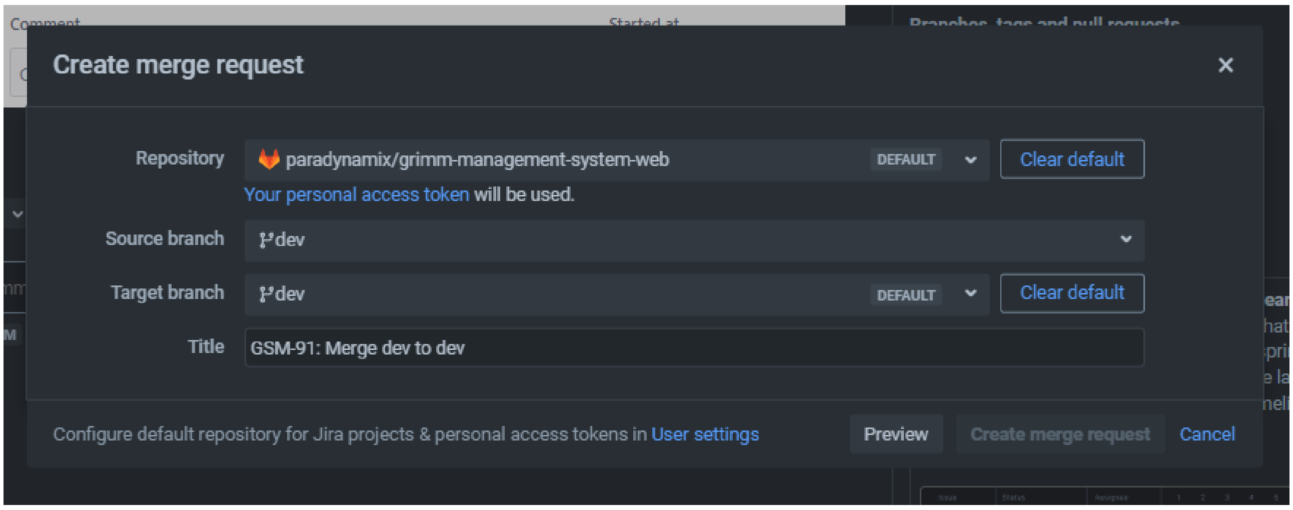
Create branch

Repository	paradynamix/grimm-management-system-web	Make default
	Provide a personal access token to personalize this operation (optional).	
Source branch	dev	Make default
Branch name	GSM-91-transfer-invoice-to-quickbooks	

Configure default repository for Jira projects and personal access tokens in User settings

This allows us to easily track the issue against the source code.

When an issue has been completed and is ready to be merged, please open a merge request for the issue and request that it's merged into the dev branch.


Create merge request

Repository: paradynamix/grimm-management-system-web DEFAULT

Your personal access token will be used.

Source branch: dev

Target branch: dev DEFAULT

Title: GSM-91: Merge dev to dev

Configure default repository for Jira projects & personal access tokens in User settings

Merge Requests

Merge Requests should be opened for each branch when the code has been fully tested with the branch.

Merge requests will be merged in as quick as possible.

Merges to master or main will automatically be closed if they are not a hot fix.

Merges to master will be scheduled to occur once a week unless they are a hot fix.

Hot fixes will be released they pass Quality Assurance or have been moved to Done.

Creating New Issues

Creating a well-crafted Jira issue description is essential for clear communication and efficient task management. A good description ensures that team members, stakeholders, and developers fully understand the issue at hand. Here are the key elements to include in a Jira issue description:

1. **Summary or Title**:

- Start with a concise and informative summary. This should give a high-level overview of the problem, task, or feature. Avoid ambiguous terms and be as specific as possible.

Example: "User unable to reset password on mobile version of the app."

2. **Description**:

- **Context/Background**: Provide the necessary context so the issue is clearly understood. Mention why this issue is being raised and its importance.
  Example: "Users have reported that they cannot reset their password using the mobile app. This impacts a critical feature as it prevents users from regaining access to their accounts."
- **Steps to Reproduce**: For bug reports, provide step-by-step instructions on how to reproduce the issue. Be precise and list the exact actions that lead to the problem.
  Example:

a. Open the mobile app.
b. Click on “Forgot Password.”
c. Enter the registered email and tap “Submit.”

- **Expected Result**: Describe what should happen if everything works correctly.
  Example: “An email with a password reset link should be sent to the user’s email.”
- **Actual Result**: Describe what actually happens.
  Example: “No email is sent, and the user receives an error message stating ‘Request failed.’”
- **Environment**: Mention the environment in which the issue was observed, such as the operating system, browser, app version, or any other relevant details.
  Example: “iOS 15, mobile app version 2.3.1.”

3. **Acceptance Criteria**:

- Clearly state what needs to be achieved for the issue to be considered resolved. These are measurable outcomes that define when the issue is complete.
  Example:
  - Users should successfully receive a password reset email.
  - No error messages should appear after submitting the password reset form.

4. **Attachments/Screenshots**:

- Attach any relevant files like screenshots, logs, or screen recordings that help illustrate the issue.

5. **Priority and Labels**:

- Define the urgency of the issue using priority levels and apply relevant labels to make it easier to filter and categorize the task.

6. **Additional Notes**:

- If necessary, add any other relevant details such as links to related issues, user feedback, or investigation results.

By including these components in a Jira issue description, you provide a clear, structured, and comprehensive understanding of the issue, leading to faster resolution and better collaboration.

Development Check List

It is imperative that developers take full ownership of the issues assigned to them, ensuring both functionality in development and production environments. To facilitate a seamless transition into Quality Assurance, we strongly encourage adherence to our best practices. This includes verifying that the code operates as intended, does not introduce new bugs, and maintains the integrity of other functions. Additionally, ensure that no alerts or console logs are present, that the code compiles without errors and with minimal warnings, and that all pipelines for your commit, branch, and merge request are successful. By following these guidelines, developers can significantly enhance the quality and reliability of their contributions.

- Check that the code functions as intended.
- Check that the code does not introduce any new bugs.
- Check that other areas of the function perform as intended.
- Check that the code does not introduce any alerts or console.log's
- Check that the code compiles as intended without any error and with minimal warnings.
- Check that there pipelines for your commit, branch, and merge request are successful.

Technology Stacks

Our team utilizes a diverse array of technology stacks to effectively address a range of projects. While we often engage with applications that have been developed previously, we also dedicate resources to the creation of new solutions. Although the specific languages and frameworks may vary from project to project, we maintain a consistent approach that ensures efficiency and adaptability in our development processes. This balance allows us to leverage past insights while embracing innovation in our work.

Databases

- MySQL
- MS SQL Server

Languages

- C#
- JavaScript
- Typescript

Technologies

- Nuxt
- Vue
- React Native

Servers

Tailscale