HephIA

Scalable Intelligence

NoSQL

Réalisé par Brice FOTZO | *15/11/2023*

Best Practices?

Flemme

GoCod

Terminal

- Autocompletion via touche tab
- Ouvrir un dossier/fichier dans VSCode via commande code
- Touche haut pour revenir vers les commandes précédentes
- history | grep "command/mot" pour retrouver une commande

Code standards

Code standards

GoCod

General

- Adopt standard formatting or corporate ones
- Use formatting and Linting tools. Ex: Black(format) and Blake8(lint) for python
- Consistency: keep same prefix/suffix pattern in one project
- Avoid code duplication
- Prefer reusable functions, classes, components, services
- Pre-commits: Ensure your code quality before pushing

Code sharing/versioning

Branching strategy

GoCod

Dev (Development) Branch: dev

- **Role**: This is where all the new features, enhancements, and bug fixes are first merged.
- **Who**: All developers.
- **How**: Never commit directly to dev. Always create a Pull Request (PR).

Staging Branch: staging

- **Role**: Pre-production environment. Code here should be feature complete and undergo thorough testing.
- **Who**: DevOps, Team leads or QA engineers.
- **How**: Merge from dev to staging for final testing. Again, use PRs for merging.

Production Branch: prod

- **Role**: Reflects the codebase currently in production.
- **Who**: Only team leads or DevOps engineers.
- **How**: Merge from staging to prod after successful testing and validation. Use PRs and ensure proper reviews.

Supporting branches

GoCod

Feature Branches: **feature/***

- **From**: dev
- **Merge Back Into**: dev
- **Purpose**: New features.

Enhancement Branches: **enhance/***

- **From**: dev
- **Merge Back Into**: dev
- **Purpose**: Improvements to existing features.

Bugfix Branches: **bugfix/***

- **From**: staging or prod (depending on the urgency and nature of the bug)
- **Merge Back Into**: dev, staging, and prod as applicable
- **Purpose**: To fix bugs and issues.

Commit messages

GoCod

Prefixes

Use prefixes to categorize the type of change:

feat: for new features
fix: for bug fixes
chore: for maintenance tasks
docs: for documentation changes
test: for tests

Title

Limit: Keep the title to 50 characters or less.
Case: Capitalize the first letter.
Punctuation: Do not end the title with a period.
Imperative: Use the imperative mood ("Add" not "Added").

Body

Wrap Lines: Wrap lines at 72 characters.
Context: Explain the "what" and "why", not the "how".
Bullet Points: Use hyphens or asterisks for bullet points.

```
feat: Add cart feature with animation

Enhanced the CSS layout of the cart section, addressing text alignment issues and
refining the layout for improved aesthetics and readability.
```

Structure of PR content

GoCod

Description:
- Start with a short summary of the changes.
- You can provide context on what led to the change, especially if the PR is complex.

Changes Made:
- List the main changes made in bullet points or a brief paragraph.
- If the list is extensive, group them under subheadings like "Backend Changes", "Frontend Changes", etc.

Screenshots/GIFs:
- Whenever possible, add screenshots or GIFs to show visual changes or flow of new features.

Testing:
- Briefly explain how the change was tested or provide steps for reviewers to test it.
- Mention any new tests added.

Related Issue(s):
- Link any related issues or user stories.
- Use the format Related to #issueNumber.

Pre-commits

Structure of PR content

GoCod

Ensure Code Quality:
- Automatically format code and catch style issues
- Ensure that incoming code adheres to the project's style guidelines.

Prevent Bugs:
- Identify issues early in the development cycle
- Reduce the chances of bugs

Streamline Reviews:
- Reduce the manual effort in code reviews
- Allow the team to focus on more complex aspects of the code

Enforce Policies:
- Prevent secrets from being pushed
- Enforce file naming conventions
- Ensure commit message formats

Developer Convenience:
- Speed up the development process by catching issues locally,
- Save both time and mental energy.

Config files?

Config files

GoCod

What are configuration files?
- External to the code
- Centralize settings like DB connections, environment variables

Common file formats:
- JSON, YAML, .env, etc…

Importance in (NoSQL) development:
- Ensure consistency across different environments (dev, test, prod)
- Key for maintaining and evolving code efficiently

Best practices
- Externalize all configurations from codebase
- Use separate files for different environments
- Implement version control on configuration templates

Recommended Tools:
- dotenv for Node.js
- python-dotenv, ConfigParser for Python

Security considerations:
- Avoid storing sensitive data directly in the code
- Employ environment variables and encryption tools

```
#.env
DB_HOST=”35.128.45.12”
DB_USER=”admin“
DB_PASSWORD=“admin”
```

NEVER PUSH SECRETS TO THE REMOTE REPO

gitguardian bot commented yesterday

⚠️ **GitGuardian has uncovered 1 secret following the scan of your pull request.**

Please consider investigating the findings and remediating the incidents. Failure to do so may lead to compromising the associated services or software components.

▼ 🔎 Detected hardcoded secret in your pull request

GitGuardian id	Secret	Commit	Filename	
8714156	MongoDB Credentials	65ed8c5	Mongo_DB_URI.env	View secret

▼ 🛠️ Guidelines to remediate hardcoded secrets

1. Understand the implications of revoking this secret by investigating where it is used in your code.
2. Replace and store your secret safely. Learn here the best practices.
3. Revoke and rotate this secret.
4. If possible, rewrite git history. Rewriting git history is not a trivial act. You might completely break other contributing developers' workflow and you risk accidentally deleting legitimate data.

To avoid such incidents in the future consider

- following these best practices for managing and storing secrets including API keys and other credentials
- install secret detection on pre-commit to catch secret before it leaves your machine and ease remediation.

Test your code!

HephIA

Why?

- Code exactness today and tomorrow
- Prevent from past bugs
- Safely refactorize

How?

- Unit Tests
- Integration Tests
- Functional Tests
- Performance Tests

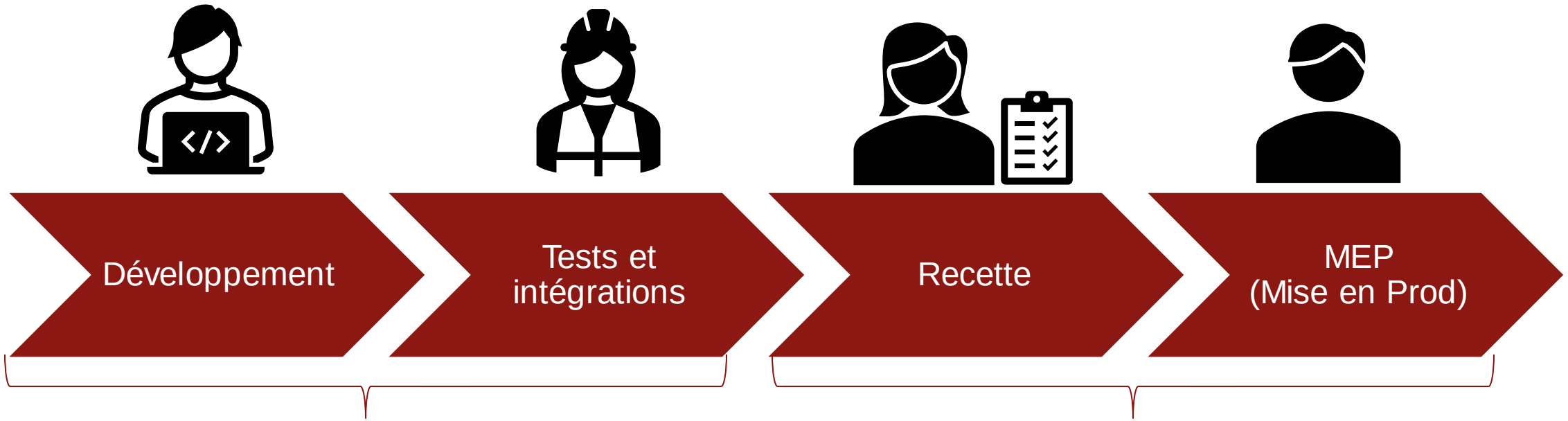
Best practices

- Isolation
- Repetability
- Speed
- Keep it simple
- Automate it

CI/CD?

CI/CD

From Dev to Prod



Intégration Continue(CI)

- Codes gérés et centralisé par SCM(GitHub, GitLab)
- Build, Tests(Unitaires, Intégration, QA, dépendances)

Déploiement Continue(CD)

- Tests (recette, qualif, pré-prod)
- MEP rapide une fois tests validés