

Documenter efficacement

De quoi on parle exactement

- Cible : les développeurs
- But : garder la connaissance du projet

Comment faire ?

- Documenter la partie technique du projet
- Ancrer la connaissance au plus près du code
- Garder la documentation vivante

Plusieurs types de documentation

- Au niveau du projet
- Au niveau du code source

Technique

Au niveau projet :

- README : point d'entrée
- BUILD : pour les infos de build précises
- CONTRIBUTING : pour les consignes de développement
- ADR : documenter les choix importants

README

- Le point d'entrée
- Comprendre rapidement ce qu'est le projet
- Comment l'utiliser, l'installer
- Renvoie vers de la documentation plus précise si besoin

README : Dans le projet

- Un fichier texte `README.md` à la racine du projet
- Généralement au format Markdown ou AsciiDoc

README : Modèle

- Nom du projet, logo
- Badges : CI, Coverage, Version, License...
- Description
- Pré-requis
- Utilisation / Exemples

README : Modèle

- Installation / Build ⇒ BUILD.md
- Contribuer ⇒ CONTRIBUTING.md
- Licence ⇒ LICENSE
- Autres ressources ⇒ Documentation d'entreprise, ADR

README : Examples

- Grimoire Club
- Code Decorator

BUILD

Décrit en détail comment lancer le projet depuis les sources :

- compiler
- configurer
- exécuter

BUILD : Dans le projet

- Un fichier `BUILD.md` à la racine
- Généralement au format Markdown ou AsciiDoc

BUILD : Modèle

- Pré-requis
 - Langage (Java 21, Node.js 20, ...)
 - Outils (Maven, Yarn, Docker, ...)
- Installation des dépendances
- Instructions pour la compilation

BUILD : Modèle

- Configuration (.env, app.properties, ...)
- Lancement de l'application
- Troubleshooting

BUILD : Exemple

- Grimoire Club

CONTRIBUTING

- Explique le processus de contribution
- Facilite l'intégration de nouveaux contributeurs
- Fige les règles de contribution

CONTRIBUTING : Dans le projet

- Un fichier `CONTRIBUTING.md` à la racine du projet
- Généralement au format Markdown ou AsciiDoc

CONTRIBUTING : Modèle

- Bienvenue
- Pré-requis
- Règles (conventions, linters, tests obligatoires)
 - Style (formatage, conventions de nommage)
 - Linters
 - Format des commits (ie.: conventionnal commits)

CONTRIBUTING : Modèle

- Processus pour contribuer
 - Exemple : fork → pull request → validation
 - Politique de revue de code
 - Fournir un modèle de pull request
- Tests
 - Lancement des tests en local
 - Tests à fournir avec la contribution

CONTRIBUTING : Modèle

- Communication
 - Où trouver la communauté (issues, slack, discord, ...)
 - Règle de conduite
- Liens externes si besoin

CONTRIBUTING : Exemple

- JHipster Lite

Architecture Design Records (ADR)

- Permettent de tracer les décisions techniques
- *Pourquoi avons-nous fait ce choix à ce moment-là ?*
- À réaliser en cas de changement structurel
 - Choix du framework, BDD, architecture
 - Après une discussion technique importante


ADR : dans le projet

- Un dossier dédié, par exemple `docs/adr`
- Nomenclature des fichiers : `0001-title.md`
- Format simple : Markdown ou AsciiDoc
- Outillage : `adr-tools`, `log4brains`

ADR : Modèle





- Titre court et clair
- Date et status
- Contexte : situation, problème à résoudre
- Décision : choix et justification, alternatives rejetées
- Conséquences : impacts, dette technique
- Lien vers un document extérieur si pertinent (ticket, issue, etc)

ADR : Démo



**Grimoire Club**
Architecture knowledge base



Search...

Decision log

- 
- June 2025
 -  ADR-0002 : Choix de JDBI pour l'accès aux données
ACCEPTED
 -  ADR-0001 : Choix de Javalin comme framework web
ACCEPTED
- 

ADR-0001 : Choix de Javalin comme framework web

 Jun 10, 2025 **ACCEPTED**
 Mathieu Barberot

Contexte

Dans le cadre du projet de cours, il fallait de choisir un framework web pour exposer une API REST. Plusieurs options étaient disponibles : des solutions complètes comme Spring Boot ou Jakarta EE ou des frameworks plus légers.

Décision

Après étude des différentes possibilités, le framework retenu est **Javalin**.

Justification

- **Facile à installer** : Le framework est léger, sans configuration complexe.
- **Prise en main facile** : Le framework repose sur des concepts simples (routes, handlers, middleware) qui sont compréhensibles même sans connaissance préalable des frameworks Java traditionnels.

Table of contents

- [Contexte](#)
- [Décision](#)
- [Justification](#)
- [Conséquences](#)

Dans le code source

- Par les commentaires
- Par les commentaires de documentation
- Par le nommage du code
- Par l'outillage
- Par les outils de versionnement

Par les commentaires

3 types de commentaires :

- Comment ?
- Pourquoi ?
- Invitation à l'action (TODO / FIXME)

Exemple

```
public String generateRandomPassword() {
    /* TODO: keep this in sync with the security requirements */
    String UPPER = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    String LOWER = "abcdefghijklmnopqrstuvwxyz";
    String DIGITS = "0123456789";
    String SYMBOLS = "!@#$%&*";
    String ALL_CHARS = UPPER + LOWER + DIGITS + SYMBOLS;

    /* Generate the password char by char while ensuring every
       char category is represented */
    StringBuilder password = new StringBuilder(32);
    /* Using a SecureRandom is required to avoid attacker
       guessing the generated password */
    SecureRandom random = new SecureRandom();
    for (int i = 0; i < 32; i++) {
        int idx = random.nextInt(ALL_CHARS.length());
        password.append(ALL_CHARS.charAt(idx));
    }
    return password.toString();
}
```

Exemple

```
public String generateRandomPassword() {
    /* ✓ TODO: keep this in sync with the security requirements */
    String UPPER = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    String LOWER = "abcdefghijklmnopqrstuvwxyz";
    String DIGITS = "0123456789";
    String SYMBOLS = "!@#$%&*";
    String ALL_CHARS = UPPER + LOWER + DIGITS + SYMBOLS;

    /* ✗💀 Generate the password char by char while ensuring every
       char category is represented */
    StringBuilder password = new StringBuilder(32);
    /* ✓ Using a SecureRandom is required to avoid attacker
       guessing the generated password */
    SecureRandom random = new SecureRandom();
    for (int i = 0; i < 32; i++) {
        int idx = random.nextInt(ALL_CHARS.length());
        password.append(ALL_CHARS.charAt(idx));
    }
    return password.toString();
}
```

Example 2

```
@SuppressWarnings("unused") // Fix false positive by Sonar (used by Spring)
@Configuration
public class DatabaseConfiguration {
    @Bean
    public DataSource datasource() {
        /* ... */
    }

    private DatabaseConfiguration() {
        // This class should not be instantiated manually,
        // it is managed by Spring
    }
}
```

Faire de bons commentaires

- Répondre à la question "pourquoi ?"
 - Phrases claires
 - Expliquer les choix ou les éléments implicites
- Laisser des indications (TODO / FIXME)

Par les commentaires de documentation

- Permet de générer de la documentation pour les classes
- Intéressant lorsque le code doit être publié
 - SDK
 - Framework interne

Exemple

```
/**
 * Connecteur de base de données de la suite d'outils Imaginary
 * @see <a href="https://libs.imaginary.com">Imaginary Website</a>
 */
public class ImaginaryDB implements AutoCloseable {

    /**
     * Chemin vers le fichier de configuration par défaut
     */
    public static final String DEFAULT_CONFIGURATION_FILE =
        "src/main/resources/imaginary-db.properties";
}
```

Exemple

```
/**
 * Crée une nouvelle instance automatiquement configurée
 * via le fichier par défaut
 *
 * @return La nouvelle instance
 * @throws java.lang.RuntimeException Si le fichier par défaut est absent
 * {@link ImaginaryDB#DEFAULT_CONFIGURATION_FILE}
 */
public static ImaginaryDB getInstance() {
    // ...
}
```

Exemple

```
/**
 * Exécute une requête modification
 * <p>
 * La requête peut comporter des paramètres, via l'utilisation
 * du caractère <code>?</code> :
 * <pre>execute("UPDATE my_table SET foo = '?'", "bar")</pre>
 * </p>
 *
 * @param sql      La requête SQL
 * @param params   Les paramètres de la requête, si nécessaire
 * @return         Le nombre de lignes impactées
 * @throws         SQLException En cas d'erreur lors de l'exécution de la requête
 */
public int execute(String sql, Object... params) throws SQLException {
    // ...
}
```

Rendu dans l'IDE

Au survol de l'élément documenté :

```
© iut.libs.ImaginaryDB  
public int execute(  
    String sql,  
    Object... params  
)  
throws SQLException
```

Exécute une requête modification La requêtes peut comporter des paramètres, via l'utilisation du caractère ? :

```
execute("UPDATE my_table SET foo = '?'", "bar")
```

Params: `sql` – La requête SQL
`params` – Les paramètres de la requête, si nécessaire

Returns: Le nombre de lignes impactées

Throws: `SQLException` – En cas d'erreur lors de l'exécution de la requête

exercices-java

Rendu HTML

Au format HTML, via un génération de site statique :

Représente une bibliothèque d'interfaçage avec une base de données

Field Summary

Fields

Modifier and Type	Field	Description
static final String [Ⓔ]	DEFAULT_CONFIGURATION_FILE	Chemin vers le fichier de configuration par défaut

Constructor Summary

Constructors

Constructor	Description
ImaginaryDB(String [Ⓔ] url, int port, String [Ⓔ] user, String [Ⓔ] password)	Crée une nouvelle instance

Method Summary

All Methods Static Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
void	close()	
int	execute(String [Ⓔ] sql, Object [Ⓔ] ... params)	Exécute une requête modification La requêtes peut comporter des paramètres, via l'utilisation du caractère <i>point d'interrogation</i> :
static ImaginaryDB	getInstance()	Crée une nouvelle instance automatiquement configurée via le fichier par défaut
ImaginaryQueryResult	query(String [Ⓔ] sql, Object [Ⓔ] ... params)	Exécute une requête de sélection

Rendu HTML

execute

```
public int execute(String sql,  
                  Object... params)  
    throws SQLException
```

Exécute une requête modification La requêtes peut comporter des paramètres, via l'utilisation du caractère ? :

```
execute("UPDATE my_table SET foo = '?'", "bar")
```

Parameters:

sql - La requête SQL

params - Les paramètres de la requête, si nécessaire

Returns:

Le nombre de lignes impactées

Throws:

SQLException - En cas d'erreur lors de l'exécution de la requête

Cas des langages faiblement typé

Avec un langage comme javascript :

- On peut ajouter le typage des paramètres dans la JSDoc
- L'IDE va alors pouvoir l'utiliser
 - Pour ajouter des warnings si le type passé dans un appel ne correspond pas
 - Proposer une meilleure autocomplétion

Par le nommage du code

- Bien nommer son code
 - Eviter les commentaires de type *Comment ?*
- Utiliser les termes métier
 - Permet de parler le même langage
 - \Rightarrow Domain Driven Design

Par l'outillage

- On utilise un outil externe
- On documente au niveau du code pour cet outil

Exemple : OpenAPI

Exemple

- avec des annotations

```
@OpenApi(  
    tags = {"Grimoire Club"},  
    summary = "Add a sorcerer into the club",  
    path = "/clubs/{id}/members",  
    methods = POST,  
    pathParams = {  
        @OpenApiParam(name = "id", required = true, description = "Club ID"),  
    },  
    requestBody = @OpenApiRequestBody(  
        content = @OpenApiContent(from = AddMemberDto.class)  
    )  
    // ...  
)  
public void addMember(Context ctx) { /* ... */ }
```

Exemple

- avec des commentaires

```
/**
 * @api [post] /clubs/{id}/members
 * tags:
 *   - Grimoire Club
 * summary: Add a sorcerer into the club
 * parameters:
 *   - name: id
 *     in: path
 *     description: Club ID
 *     required: true
 *     schema:
 *       type: string
 * requestBody:
 *   ...
 */
public void addMember(Context ctx) { /* ... */ }
```

Par les outils de versionnement

- Reverser de l'information dans les commits
 - Le *pourquoi*
 - Le contexte, le numéro du ticket traité
 - Les choix et justifications

Pensez à vous dans 4 ans, qui allez devoir fouiller dans l'historique pour comprendre ce qui s'est passé !

Conventional Commit

- Une convention de nommage des commits
- Permet d'unifier le format des commits
- Historique plus clair

Site officiel

Conventional Commit : Modèle

```
type(optional scope): description  
  
optional body  
can be multiline !
```

- `type` : feat | fix | refactor | chore
- `scope` : ce sur quoi vous avez travaillé (ie: le module, la thématique)
- `description` : ce que vous avez fait en une phrase
- `body` : les détails, le *pourquoi* ?

Conventional Commit

- Exemple : **Le repository du cours**
- Bonus : on peut générer un changelog automatiquement !

Documentation fonctionnelle

- Format généralement imposé (confluence, wikis, word...)
- Quelques possibilités pour les développeurs

Spécifications vivantes avec BDD

- Specifications dans un format lisible (Gherkin, Markdown)
- Lisible par des non-techniques
- Executables sous forme de tests
- Permet de fusionner specification et tests

BDD avec Gherkin

```
Feature: Authentification
```

```
  Scenario: Connexion avec un mot de passe valide
```

```
    Given John et son mot de passe '5RziLh2w'
```

```
    When il saisit ses identifiants
```

```
    Then il est redirigé vers son tableau de bord
```

Exemple du test avec Cucumber

```
public class AuthentificationSteps {
    private String login;
    private String password;
    private Response response;

    @Given("{word} et son mot de passe '{word}'")
    public void existing_user_with_password(String login, String password) {
        this.login = login;
        this.password = password;
    }

    @When("il saisit ses identifiants")
    public void il_saisit_ses_identifiants() {
        this.reponse = authenticationService.authenticate(login, password);
    }

    @Then("il est redirigé vers son tableau de bord")
    public void il_est_redirige_vers_son_tableau_de_bord() {
        assertThat(response.getRedirectPath()).isEqualTo("/dashboard");
    }
}
```

BDD avec Markdown

```
# Authentication
## Connexion avec un mot de passe valide

Lorsque [John](- "#login") utilise son mot de passe '[5RziLh2w](- "#password")'
[pour se connecter](- "#response = authenticate(#login, #password)"),
il est [redirigé vers son tableau de bord](- "c:assertTrue=isDashboard(#response)
```

Exemple du test avec Conordion

```
public class AuthenticationFixture {  
  
    private String login;  
    private String password;  
  
    public void setLogin(String login) {  
        this.login = login;  
    }  
  
    public void setPassword(String password) {  
        this.password = password;  
    }  
  
    public Response authenticate(String login, String password) {  
        return authenticationService.authenticate(login, password);  
    }  
  
    public boolean isDashboard(Response response) {  
        return "/dashboard".equals(response.getRedirectPath());  
    }  
}
```

Merci de votre attention