



RAPPORT DE STAGE DE FIN D'ÉTUDE

# Patron de sécurité pour la défense d'une application web

*Henri Stoven*

*CI 2020 SPID*

supervisé par

Joël CHAMPEAU

Sylvain GUÉRIN

25 février 2021

# 1 Remerciements

Je tiens à remercier en premier lieu Joël Champeau et Sylvain Guérin pour leur aide constante et infiniment précieuse pendant toute la durée du stage.

Mes remerciements particuliers à Luka Leroux, Ciprian Teodorov, Bastien Druot, Jean-Charles Roger qui m'ont consacré du temps pour me conseiller et m'aider à accomplir mon travail dans les meilleures conditions.

Je voudrai aussi dire merci à l'Ensta Bretagne qui, par le biais de Mr Champeau, m'a permis de compléter mes études d'ingénieur malgré les difficultés.

Enfin, une reconnaissance affectueuse et toute particulière à mes parents pour leurs encouragements à poursuivre et mener à terme mes études. Un grand merci à eux ainsi qu'à toute ma famille et tous mes amis qui m'ont soutenu moralement tout au long de cette période.

# Table des matières

<b>1</b>	<b>Remerciements</b>	<b>1</b>
<b>2</b>	<b>Resume</b>	<b>4</b>
<b>3</b>	<b>Abstract</b>	<b>4</b>
<b>4</b>	<b>Introduction</b>	<b>5</b>
4.1	Présentation de l'entreprise . . . . .	5
4.1.1	Un bref historique de l'ENSTA Bretagne . . . . .	5
4.1.2	La recherche à l'ENSTA Bretagne . . . . .	6
4.2	Organisation du travail lors du stage . . . . .	7
4.3	Contextualisation et problématique . . . . .	9
4.3.1	La sécurité des logiciels . . . . .	9
4.3.2	Le concept du "Design by Contract" . . . . .	10
4.3.3	Enjeu de la recherche . . . . .	11
4.3.4	Problématique et sujet du stage . . . . .	12
<b>5</b>	<b>Pamela et les patrons de sécurité</b>	<b>13</b>
5.1	Le framework Pamela . . . . .	13
5.2	Les patrons de sécurité . . . . .	15
5.3	L'implémentation des patrons de sécurité avec Pamela . . . . .	17
<b>6</b>	<b>Les outils pour le développement web</b>	<b>20</b>
6.1	Eclipse et le langage Java . . . . .	20
6.2	Le framework Spring . . . . .	21
6.2.1	Fonctionnement de Spring MVC . . . . .	22

6.2.2	Fonctionnement de Spring Security . . . . .	23
<b>7</b>	<b>Construction de l'application Spring</b>	<b>24</b>
7.1	Architecture du site web . . . . .	24
7.2	L'authentification . . . . .	26
7.3	Le build.gradle . . . . .	27
7.4	Les ressources . . . . .	28
<b>8</b>	<b>Insertion du patron de sécurité d'authentification dans l'application Spring</b>	<b>28</b>
8.1	Le choix d'un patron de sécurité pour le web . . . . .	28
8.2	Application du patron d'authentification préexistant . . . . .	30
8.2.1	Aspect fonctionnel . . . . .	30
8.2.2	Aspect conception . . . . .	32
8.3	Implémentation d'une nouvelle propriété pour parer les attaques de force brute . . . . .	34
8.4	Bilan de l'expérimentation . . . . .	36
8.4.1	Concernant les propriétés du patron d'authentification préexistant (implémentation décrite en 8.2) . . . . .	36
8.4.2	Concernant la nouvelle propriété implémentée (décrite en 8.3) . . . . .	37
<b>9</b>	<b>Conclusion</b>	<b>38</b>
9.1	Réponse à la problématique . . . . .	38
9.2	Résumé global . . . . .	39
9.3	Apport personnel . . . . .	39
9.4	Ouverture . . . . .	40

## 2 Resume

De nos jours, les cyberattaques se multiplient. Avec l'avènement de l'informatique et notamment d'internet au 21ème siècle, les organisations sont de plus en plus dépendantes des nouvelles technologies. Dans ces domaines, la concurrence se fait de plus en plus rude et toutes les failles des concurrents vont chercher à être exploitées. Il est donc important que les réseaux et les applications liées au web soient bien protégés. Le marché mondial de la sécurité informatique devrait représenter 170,4 milliards d'ici 2022[1]. Dans ce contexte, le lab-STICC de l'Ensta Bretagne, lié au ministère des Armées via la Direction Générale de l'Armement, s'intéresse à l'application d'un de ses outils, le framework Pamela, pour sécuriser des logiciels programmés en Java. C'est dans cet optique que furent implémentés les modules permettant à Pamela d'ajouter des patrons de conception de sécurité qui vont permettre de faciliter la démarche de sécurisation d'une application lors de sa conception. Ma mission lors du stage est de développer une application web peu sécurisée et d'y appliquer le patron d'authentification de Pamela afin de tester celui-ci sur un exemple concret en le mettant en relation avec les technologies web et notamment le framework Spring.

## 3 Abstract

These days, cyber attacks are on the increase. With the advent of computers and especially the Internet in the 21st century, organizations are increasingly dependent on new technologies. In these areas, competition is getting tougher and all the weaknesses of the competitors will seek to be exploited. It is therefore important that networks and web-related applications are well protected. The global computer security market is expected to represent 170.4 billion by 2022 [1]. In this context, the lab-STICC of Ensta Bretagne, linked to the French Ministry of Armed Forces via the Direction Générale de l'Armement (General Directorate of Armament), is interested in the application of one of its tools, the Pamela framework, to secure software

programmed in Java. It is with this in mind that the modules were implemented allowing Pamela to add security design patterns that will facilitate the process of securing an application during its design. My mission during the internship is to develop an insecure web application and apply Pamela's authentication pattern to it in order to test it on a concrete example by relating it to web technologies and in particular the Spring framework.

## 4 Introduction

### 4.1 Présentation de l'entreprise

#### 4.1.1 Un bref historique de l'ENSTA Bretagne

L'ENSTA Bretagne est une grande école d'ingénieurs française située à Brest dans le Finistère. L'établissement fut à l'origine créé il y a deux siècles dans le but de former des cadres pour le domaine de la construction navale. En 1967, l'école s'engage dans la formation d'ingénieurs des études et techniques de l'armement. Projet piloté par la Direction Générale de l'Armement, l'école deviendra 4 ans plus tard l'École Nationale Supérieure des Ingénieurs des Études et Techniques d'Armement et verra diplomer sa première promotion d'ingénieurs militaires.

L'école va ensuite se diversifier progressivement au cours de années qui suivirent, s'ouvrant aux élèves civils en 1990 et proposant un enseignement pluridisciplinaire couvrant de nombreuses spécialités comme l'hydrographie, l'architecture navale, la pyrotechnie ou encore l'informatique.

En 1992 l'école se lance dans les activités de recherche. Activités qui seront renforcées en 2010 avec l'école qui est alors rebaptisée École Nationale Supérieure des Techniques Avancées (ENSTA Bretagne).

#### 4.1.2 La recherche à l'ENSTA Bretagne

L'ENSTA Bretagne comprend trois laboratoires de recherche rassemblant 228 personnels et ayant permis la soutenance de 239 thèses depuis 2003. La recherche à l'ENSTA se fait en majeure partie en relation avec l'industrie qui est impliquée dans 85% des contrats de recherche[2].

Au sein de l'école, j'ai travaillé avec le Lab-STICC (Laboratoire pour la recherche en Sciences et Technologies de l'Information, de la Communication et de la Connaissance). Le laboratoire conduit des recherches dans des domaines divers et variés tels que l'intelligence artificielle ou la cyber-sécurité. La présence de nombreux doctorant et chercheurs dans ces domaines fut d'une aide précieuse dans la conduite de mon projet.

Le lab-STICC est divisé en trois branches elles-mêmes divisées en 11 sous-divisions. J'ai travaillé pour la sous-division MOCS (*Methods, tOols, Circuits / Systems*) qui fait partie de la branche CACS (Communications, Architectures, Circuits et Systèmes). Dans celle-ci, j'ai intégré l'équipe P4S (*Processes for Safe and Secure Software and Systems*) qui travaille sur les outils pour décrire et modéliser les logiciels et les systèmes afin de les évaluer et de garantir leur sécurité. L'équipe est composée de 12 membres permanents, 3 post-doctorants et ingénieurs associés ainsi que 6 doctorants. L'équipe est dirigée par Fabien Dagnat de l'IMT Atlantique. Joël Champeau supervise l'équipe au sein de l'Ensta Bretagne.

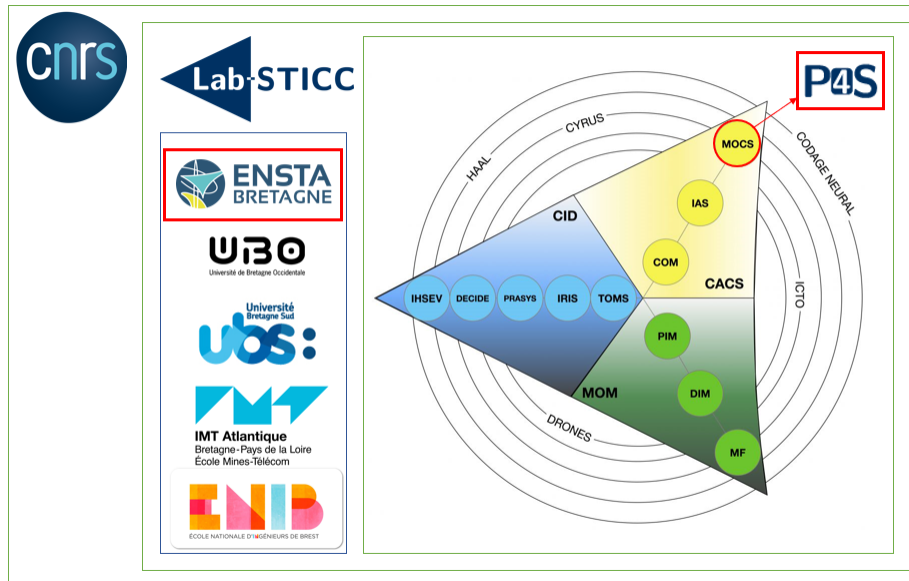


FIGURE 1 – Organisation des institutions de recherche (localisation de mon travail encadrée en rouge)

## 4.2 Organisation du travail lors du stage

Mon travail concernant le modèle de sécurité pour la défense d’une application web a débuté en Août 2020 pour se terminer en Janvier 2021. La conjoncture de l’année 2020 avec notamment la crise du Covid-19 ont fait de l’organisation du travail une problématique complexe lors de la durée de ce stage.

Les premières semaines se sont faites en présentiel sur le site de l’école. Des entrevues régulières avec mes maitres de stage m’ont permis de bien prendre en compte les différentes parties du sujet.

A partir du 29 Octobre, le 2ème confinement a commencé. Ma présence à l’école n’étant pas indispensable, j’ai préféré travailler en distancielle, ce qui était préconisé pour des mesures sanitaires.

Afin de ne pas perdre le fil du projet, un appel en visioconférence journalier avec



Mr Guérin m’a permis de faire un retour régulier sur mon travail et de prendre des décisions sur la suite du travail à effectuer.

Les appels pouvaient durer de 10 minutes à plus d’une heure selon les besoins et ont permis une avancée du travail efficace sans que je reste bloqué sur des questions techniques ou décisionnelles auxquelles Mr Guérin a toujours su me répondre.

Ces retours en continue constituent la base de la méthode Agile qui est la référence en matière de développement logiciel. Des retours réguliers permettent d’adapter le projet aux différents problèmes rencontrés et de mieux rendre compte de son travail auprès des encadrants.

Comme pour l’ensemble des travaux de recherche, mon travail sur le projet s’est divisé en trois parties majeures :

- Maîtrise du sujet et état de l’art
- Mise en place de l’expérimentation
- Interprétation des résultats

La maîtrise du sujet fut une étape primordiale de mon projet. La complexité des outils à prendre en main ainsi que la compréhension de la tâche à effectuer ont eu pour conséquence que cette partie est toujours resté au fond de ma pensée tout au long du travail.

La mise en place de l’expérimentation découle directement de la maîtrise du sujet. En effet, le choix de l’expérience à effectuer va se concrétiser au fur et à mesure que la compréhension du sujet évolue.

Enfin, l’interprétation des résultats est le but ultime d’un processus de recherche. C’est par celle-ci que va découler la réponse au problème dont découle le sujet de la recherche effectuée.

Le diagramme de Gantt en annexe (Annexe 1) donne plus de détail sur le déroulement du projet.

## 4.3 Contextualisation et problématique

### 4.3.1 La sécurité des logiciels

Pour les personnes malintentionnées, les possibilités d'attaques sur des applications informatiques sont nombreuses. L'approche de sécuriser une application après sa conception est souvent onéreuse et moins efficace. La sécurité d'un système doit donc être prise en compte dès la phase de développement[3]. On parle alors de *Secure by Design*. Dans cette approche, la sécurité est intégrée au système dans son ensemble et pendant toute sa phase de conception. Cela commence par la construction d'une architecture robuste. Les décisions de conception architecturale de sécurité sont souvent basées sur des tactiques de sécurité bien connues et des modèles définis comme des techniques réutilisables pour répondre à des problèmes spécifiques. Les modèles de sécurité fournissent des solutions pour appliquer les exigences d'authentification, d'autorisation, de confidentialité, d'intégrité des données, de confidentialité, de responsabilité, de disponibilité, de sécurité et de non-répudiation, même lorsque le système est attaqué. Afin d'assurer la sécurité d'un système logiciel, non seulement il est important de concevoir une architecture de sécurité robuste mais il est aussi nécessaire de préserver cette architecture lors de l'évolution du logiciel. Les pratiques malveillantes sont considérées comme acquises et des précautions sont prises pour minimiser les dégâts lorsqu'une vulnérabilité de sécurité est découverte ou sur des entrée utilisateurs non valides [4].

Le projet s'inscrit dans une logique de recherche mettant en lien différents domaines. Les trois grands axes de recherche sont la sécurité, le *Design By Contract*[5] et les patrons de conceptions (voir figure 3). La sécurité des logiciels requiert une certaine rigueur au niveau de la conception, celle-ci peut être définie par les contrats qui doivent être respectés pour chaque processus de l'application.

### 4.3.2 Le concept du "Design by Contract"

La notion de contrat est apparue à la fin des années 1980. C'est en travaillant sur le langage Eiffel que Bertrand Meyer a introduit cette notion. Elle s'inspire d'un contrat réel dans lequel chacun des acteurs du contrat serait lié par des obligations et garantie certains bénéfices en contrepartie des obligations des autres acteurs. L'application d'un contrat dans le domaine de l'ingénierie logicielle se fait entre une classe (nommée "caller" car elle appelle) et une "routine appelée"(called routine) et se fait par le biais d'une assertion qui impose à la routine des conditions sur ce qu'elle va traiter et sur ce qu'elle va retourner[5]. Le contrat spécifie aussi généralement ce que l'on appelle les invariants de classe et qui permettent de vérifier que certaines propriétés sont préservées.

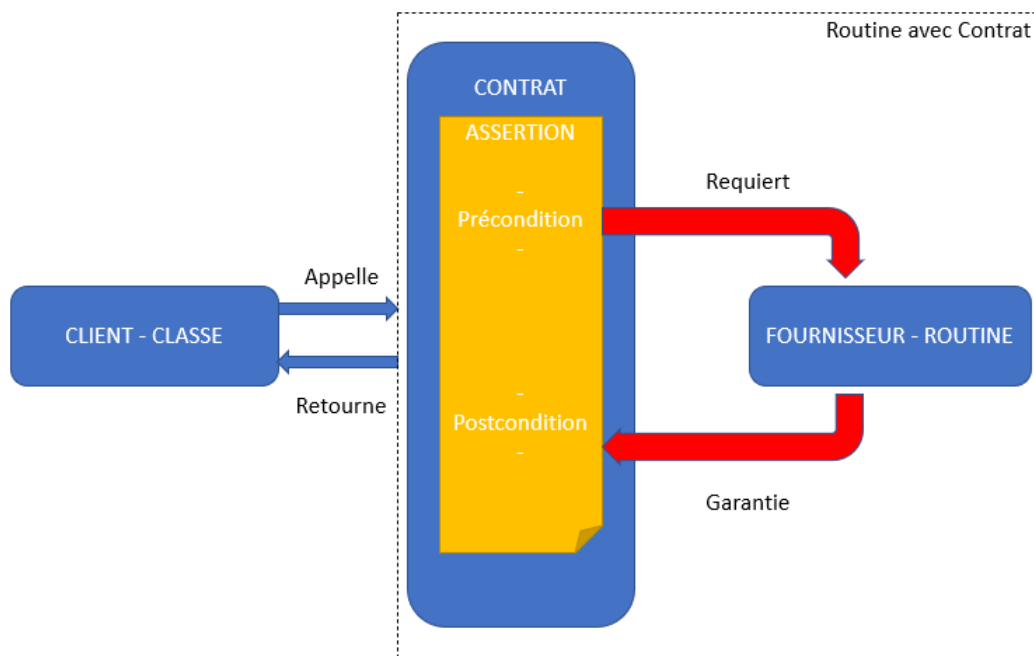


FIGURE 2 – Fonctionnement du "Design by Contract"

Si le contrat est respecté alors si les invariants de classe sont vrais et les préconditions sont remplies avant que la routine soit appelée, alors les invariants de

classe et les postconditions seront vrais quand la routine est finie. De ce fait, quand on fait appel au fournisseur/routine, le logiciel doit s'assurer de ne pas violer les préconditions du fournisseur. Ainsi le contrat va permettre d'assurer la fiabilité des systèmes logiciels. La robustesse de ces systèmes est inhérente à sa sécurité. De plus, un contrat clair et bien documenté facilite sa réutilisation pour un développement ultérieur concernant un problème similaire, ce qui est par exemple le cas pour [6].

### 4.3.3 Enjeu de la recherche

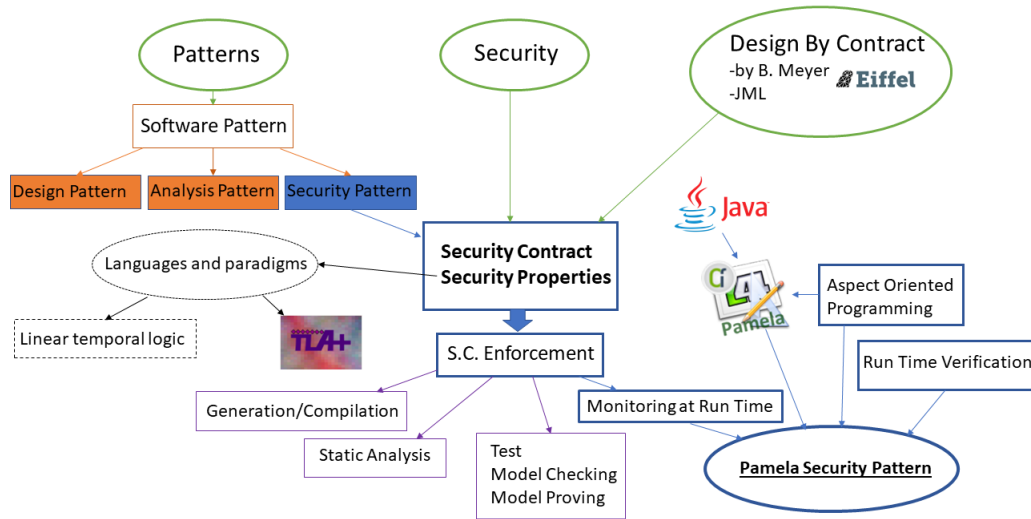


FIGURE 3 – Positionnement de l'étude menée (en bleu)

Le cadre des travaux est focalisé sur le "Security by Design" qui préconise la mise en place des outils pour la sécurité (comme les motifs de sécurité) dès la phase de conception. La synthèse des trois grands domaines de recherche dans lesquels s'inscrivent notre sujet (qui sont la sécurité, le "Design by Contract" et les patrons de sécurité) peut se résumer à la notion de "Security by Contract".

De plus, on va intégrer d'autres principes comme l'AOP (programmation orientée

aspect dont le but est d'augmenter la modularité), la surveillance/vérification à l'exécution ("Monitoring at runtime"). Toutes ces notions sont à la base de la conception et de l'application des patrons de sécurité avec Pamela.

#### 4.3.4 Problématique et sujet du stage

Le paradigme actuel concernant la sécurité rend l'utilisation de Pamela pour assister le développement d'un logiciel sécurisé plus pertinente que jamais. En effet, les nombreuses fonctionnalités de Pamela tels que la programmation par contrat fournissent un cadre idéal pour produire des applications sécurisées. C'est dans cet optique que des modules de sécurité permettant à Pamela de vérifier la sécurisation d'un système lors de son développement lui ont été ajoutés.

Mon but dans ce projet est d'appliquer, grâce aux modules préconçus à cet effet, des patrons de sécurité dans le cadre du développement d'une application web afin de les tester.

Après avoir testé les modules préconçus, il s'agira d'implémenter de nouvelles propriétés pour les patrons de sécurité qui permettent de répondre aux besoins spécifiques du web.

Il s'agit en parallèle de voir la compatibilité du framework Pamela avec le framework Spring, qui est le framework le plus utilisé pour la création d'application web en Java.

En plus de cette expérience, ce rapport englobe toutes les notions essentielles qui permettent de comprendre l'étendue des domaines de recherche informatique concernés.

## 5 Pamela et les patrons de sécurité

### 5.1 Le framework Pamela

OpenFlexo est une organisation à but non lucratif qui regroupe des chercheurs dont l'objectif est de créer des outils innovants permettant de répondre à certaines problématiques rencontrées dans le monde informatique actuel.

Le principal axe de recherche d'OpenFlexo concerne la fédération de modèle, c'est à dire l'exploration des moyens pour faciliter l'interopérabilité et le regroupement de différents modèles utilisés lors du développement d'un même système. Par ailleurs, OpenFlexo propose une multitude de composants comme des frameworks ou des TechnologyAdapter ayant chacun leurs spécificités[7].

Pamela est un framework de modélisation Java basé sur des annotations développé par OpenFlexo.

Le but de Pamela est de proposer un couplage fort entre le code source et le modèle d'un système. Cela permettrait une conception plus efficace du système en permettant au programmeur d'être en même temps modélisateur du système. Cela se fait par l'identification des patrons ou des abstractions et en y ajoutant des annotations pour permettre à l'interpréteur Pamela de mettre à jour le modèle pendant l'exécution (on parle alors de monitoring).

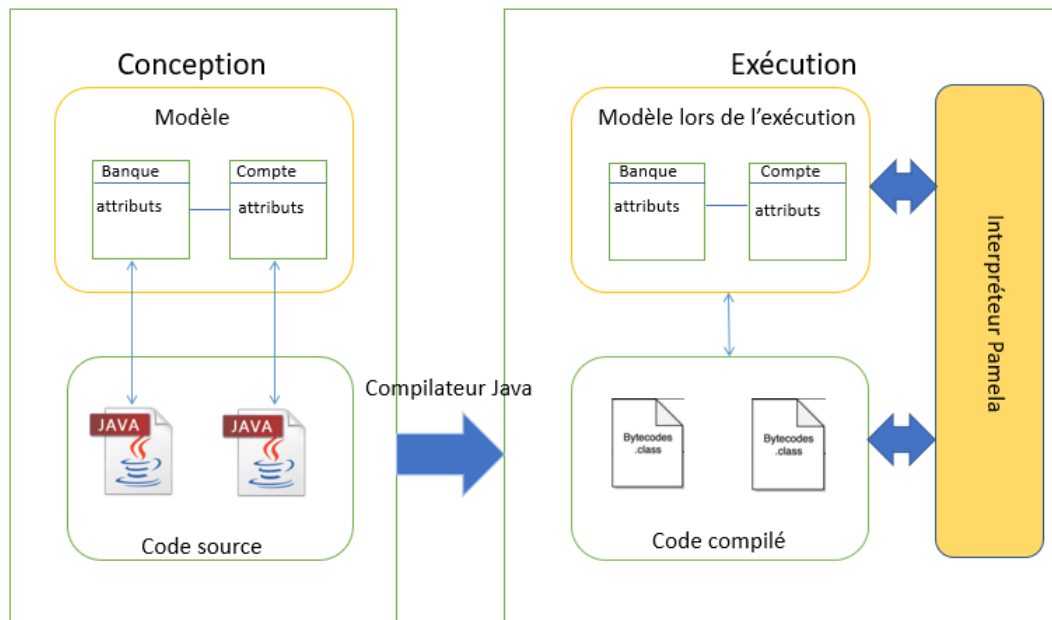


FIGURE 4 – Approche Pamela pour le couplage entre modèle et code

L'approche Pamela concernant les patrons (ou modèles) de conception (design pattern en anglais) consiste en 3 classes qui sont *PatternFactory*, *PatternDefinition* et *PatternInstance*. La première est chargée d'identifier, au moment de l'exécution, les modèles déclarés dans le bytecode Java. La deuxième représente une occurrence du modèle dans le bytecode fourni. Elle a la responsabilité de maintenir les liens avec toutes les classes et méthodes impliquées dans le patron, ainsi que de gérer le cycle de vie de ses *PatternInstances*. Cette dernière représente l'instance d'un modèle au moment de l'exécution. Elle est chargée de maintenir l'état du patron et de fournir le comportement du modèle et l'exécution du contrat. Un exemple est donné en partie 6.4.

## 5.2 Les patrons de sécurité

Dans le domaine de l'ingénierie logicielle. Un patron (pattern en anglais) décrit un problème récurrent particulier de conception qui apparaît dans un contexte de conception spécifique, et présente une solution générale efficace pour le résoudre. La solution consiste en un ensemble de rôles interagissant entre eux qui peuvent être arrangés pour former de multiple structure de conception et ce avec le processus de création de structure particulière[8]. Lorsque le problème décrit par le patron concerne la sécurité, on parle alors de patron de sécurité (security pattern).

Les patrons de sécurité ont déjà prouvé leur efficacité et présentent des avantages indéniables. Ils permettent notamment de codifier la connaissance de manière pratique et structurée, ce qui permet une meilleure transmission du savoir.

En prenant l'exemple du patron d'authentification, un patron de sécurité doit contenir :

- **Le nom.** ex : Le patron d'authentification
- **Un résumé court.** ex : Le but est de vérifier qu'un utilisateur soit authentifié lorsqu'il accède à du contenu sécurisé
- **Un exemple.** ex : Pour un site web d'achat et de vente en ligne comme Amazon, certaines informations comme les coordonnées bancaires doivent être protégées.
- **Le contexte.** ex : Le patron s'applique dans les cas où une partie d'une application web est réservée à un utilisateur authentifié.
- **Le problème.** ex : Vous devez vérifier que chaque demande provient d'une entité authentifiée, le choix des mécanismes d'authentification pour l'utilisateur nécessitent souvent des modifications en fonction de l'évolution des exigences du client, des caractéristiques intrinsèques de l'application et infrastructures de sécurité sous-jacentes.
- **La solution.** ex : Créer une application d'authentification centralisée qui effectue l'authentification des utilisateurs et encapsule les détails du mécanisme d'authentification..



- **La structure.** ex : La structure est souvent illustrée par un diagramme de classe UML.

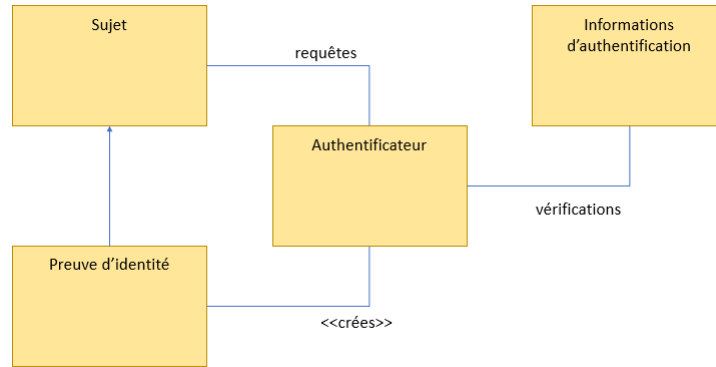


FIGURE 5 – Diagramme de classe du patron d'authentification

- **Les Dynamiques.** ex : Les dynamiques sont illustrées par un diagramme de séquence.

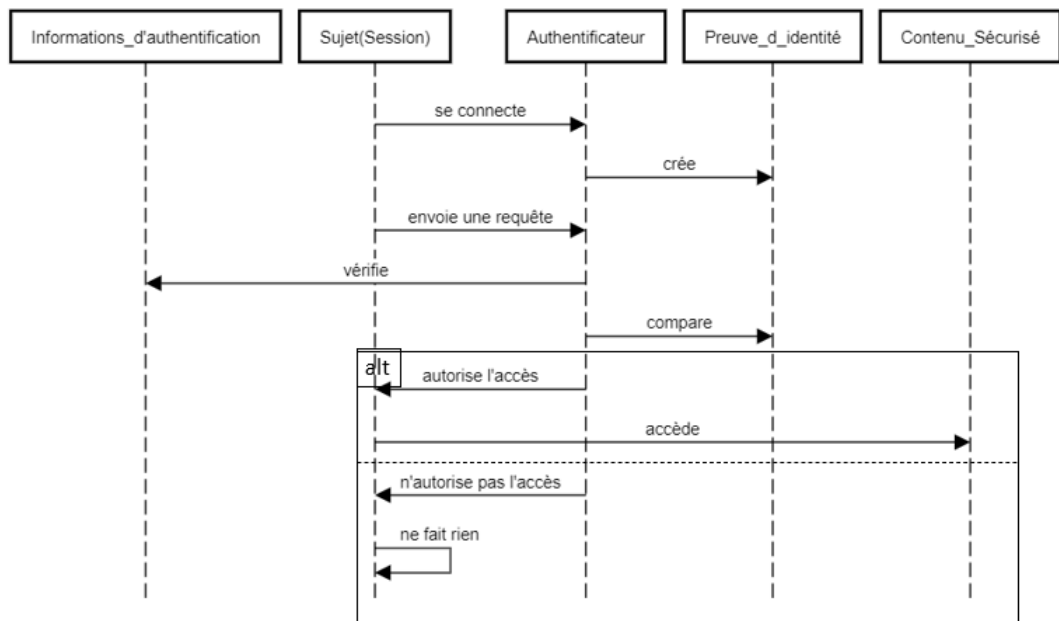


FIGURE 6 – Diagramme de séquence pour l'authentification

- **L’implémentation.** ex : voir la partie 8
- **Exemple résolu.** ex : voir les résultats à la partie 8.4
- **Les variantes.** Cette partie peut contenir une brève description des variantes ou des spécialisations pour le patron.
- **Les utilisation connues.** Cette partie contient des exemples de système existant contenant des exemple d’utilisation du patron.
- **Les conséquences.** ex : En utilisant le patron de sécurité, les utilisateurs peuvent accéder à un environnement sécurisé
- **Voir aussi.** Dans cette partie on peut faire référence à des patrons qui peuvent aider à peaufiner le patron que nous sommes actuellement en train de décrire.

Il existe diverses formes pour documenter un patron, la structure ci-dessus provient de [9] et le contenu de l’exemple pour chaque partie est très condensé par rapport à un patron normal qui prendrait plus d’une dizaine de pages comprenant tous les détails du problème. Une autre façon de le documenter se fait dans cet article [6], cela nous montre que la manière d’organiser et de mettre en page les patrons est diverse et variée mais restera toujours claire et pertinente dans son contenu.

### 5.3 L’implémentation des patrons de sécurité avec Pamela

Le framework Pamela permet d’implémenter quatre patrons de sécurité différents : le patron d’authentification (Authentication pattern), le patron d’autorisation (Autorisation pattern), le patron de point d’accès unique (Single access point) et le patron de propriété (Owner pattern). Ainsi Pamela permet au développeur de surveiller la sécurité de son application pendant sa conception en utilisant ce que nous avons décrit dans la partie 6.1. Pour déclarer un patron sur du code existant, les éléments de patron tels que les *Pattern Stakeholders* ainsi que les méthodes doivent être annotés avec des annotations spécifiques fournies. Ces annotations seront détectées au moment de l’exécution par *PatternFactory* et stockées dans les attributs de *PatternDefinition*.

```

@Component
@Scope(value = SCOPE_SESSION, proxyMode = TARGET_CLASS)
@Entity
@ImplementationClass(SessionInfo.SessionInfoImpl.class)
@AuthenticatorSubject(patternID = SessionInfo.PATTERN_ID)
public interface SessionInfo {

    String SESSION_INFO = "SESSION_INFO";
    String PATTERN_ID = "AuthenticatorPattern";
    String USER_NAME = "username";
    String IP_ADRESS = "ipAdress";
    String AUTHENTICATION_PROVIDER = "authenticationProvider";
    String ID_PROOF = "idProof";

    @Getter(value = USER_NAME)
    @AuthenticationInformation(patternID = PATTERN_ID, paramID = CustomAuthenticationProvider.USER_NAME)
    String getUserName();
}

```

FIGURE 7 – Annotations dans le code source

C'est le patron d'authentification que nous avons testé sur notre application web. Ce patron consiste en quatre entités (les *ModelEntity*) : *Subject*, *Authenticator*, *Authentication Information* et *Proof of Identity*, le diagramme de classe a été exposé dans la partie précédente pour exemplifier un patron de sécurité. On peut voir ci-dessus un exemple du code source où l'on définit le sujet du modèle (avec `@ModelEntity` et `@AuthenticatorSubject` juste devant le nom de l'interface (idem pour une classe)). On peut aussi remarquer que les annotations de Pamela sont compatibles avec celles d'autres framework (le `@Component` est une annotation de Spring par exemple). En soi, les entités qui constituent le modèle seront liées à des classes correspondant respectivement au sujet, à l'authentificateur, à l'information d'authentification et à la preuve d'identité. Cela permet de les faire reconnaître par Pamela, grâce aux annotations *@ModelEntity* ainsi que l'annotation correspondant au nom de la classe (*@AuthenticatorSubject* pour le sujet par exemple). La définition d'un ID de patron (patternID) dans les annotations permet d'implémenter plusieurs patrons en parallèle sur un même système.

De nombreuses annotations sont disponibles pour définir les accesseurs des variables contenues dans les classes mais aussi la méthode *request* utilisée lorsqu'un sujet essaye de s'authentifier ainsi que la méthode *authenticate* qui enclenche le processus d'authentification. Lors de son appel, une nouvelle preuve d'identité est générée avec la méthode *request*.

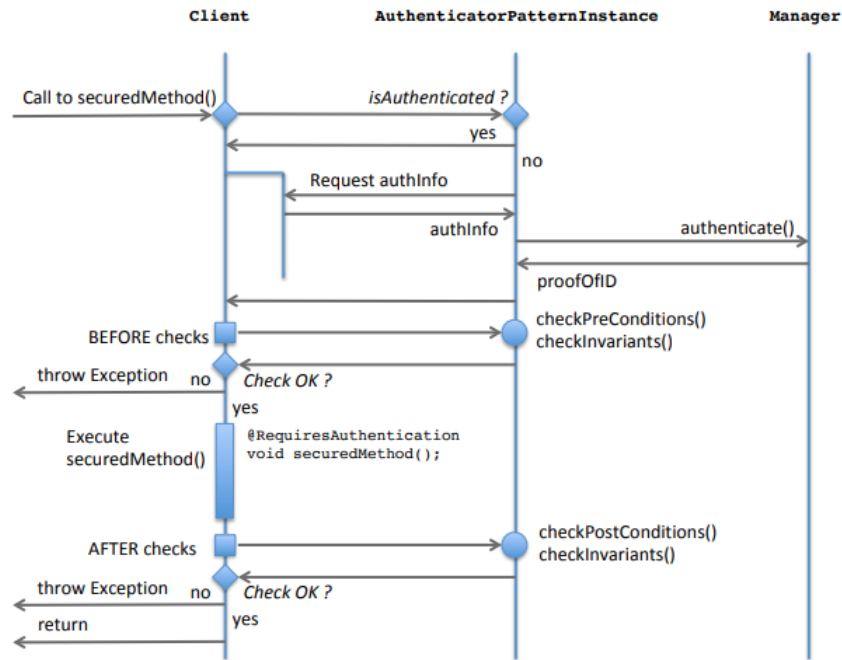


FIGURE 8 – Flux de contrôle pour l’authentification avec *Client* comme sujet et *Manager* comme authentificateur[6]

De plus le monitoring de Pamela permet de vérifier que la preuve d’identité de chaque sujet soit toujours valide et que les informations d’authentifications ainsi que l’authentificateur liés à un sujet soient finaux. Ce qui implique que leurs valeurs ne devraient jamais être modifiées. Le framework vérifie aussi que chaque sujet ait des informations d’authentification différentes.

Le but est de tester cet outil dans le cadre du développement d’une application web qui doit être sécurisée.

## 6 Les outils pour le développement web

Pour le développement de l'application, j'ai créé les templates web en html et en CSS (le CSS est importé depuis *nicepage.com*). Mon intention première était de créer une application au design élégant pouvant potentiellement servir par la suite comme une plate forme de transaction commerciale pour le pays de Brest, avec une volonté de travailler sur un projet innovant mêlant utilité pour la science (avec l'activité de recherche menée par *Openflexo Pamela*) et facultativement une utilité sociale, avec un site favorisant les commerces de proximité.

J'ai vite abandonné l'idée d'une application à des fins autres que scientifique en me rendant compte que je n'avais ni le temps ni l'expertise pour mener à bien cette partie du projet.

Je me suis donc concentré sur la partie importante du projet c'est à dire l'implémentation d'un site sécurisé pour permettre l'expérimentation du framework *Pamela* pour la sécurité.

### 6.1 Eclipse et le langage Java

Lors du développement, j'ai principalement utilisé le langage de programmation Java, dont est issu *Pamela* et qui est compatible avec le framework. *Pamela* est spécifiquement conçu pour aider au développement en Java.

Java est aujourd'hui le deuxième langage de programmation le plus utilisé avec 12.56% de part de marché selon l'index TIOBE. Sur le site *developpez.com*, Java fait partie des compétences requises pour un quart des 20 000 offres d'emplois disponibles en 2019. C'est donc un langage d'actualité. De sa popularité découle une documentation très riche et variée qui n'a cessé d'évoluer pour que le langage Java devienne plus performant et permette de nouvelles fonctionnalités. Java est aussi compatible avec de nombreuses plateformes, à condition de pouvoir y installer un JRE.

Eclipse est une IDE faite en Java dont le but principal est de développer

en Java. Eclipse permet l'ajout de nombreux plugins pour pouvoir développer dans d'autres langages ou encore pour intégrer JavaFX ou d'autres composants.

## 6.2 Le framework Spring

Pour le développement d'application web en Java, Spring est indéniablement le framework à utiliser. Spring est un framework permettant de faciliter le développement d'application en langage Java à l'aide de librairie prédéfinie comprenant des classes réutilisables. La variété des classes est à la fois un avantage pour offrir au développeur de nombreuses possibilités mais aussi un inconvénient qui implique que Spring est très étendue et long à assimiler au premier abord. Il est surtout reconnu pour son injection de dépendance et pour l'inversion de contrôle. Spring comporte un module web qui supporte l'API Servlet (interface de programmation applicative pour les classes Java qui permettent de créer dynamiquement des données au sein d'un serveur HTTP) qui s'appelle *Spring MVC* pour Model View Controller. De plus Spring comprend de nombreux projets dont *Spring Security*[10].

### 6.2.1 Fonctionnement de Spring MVC

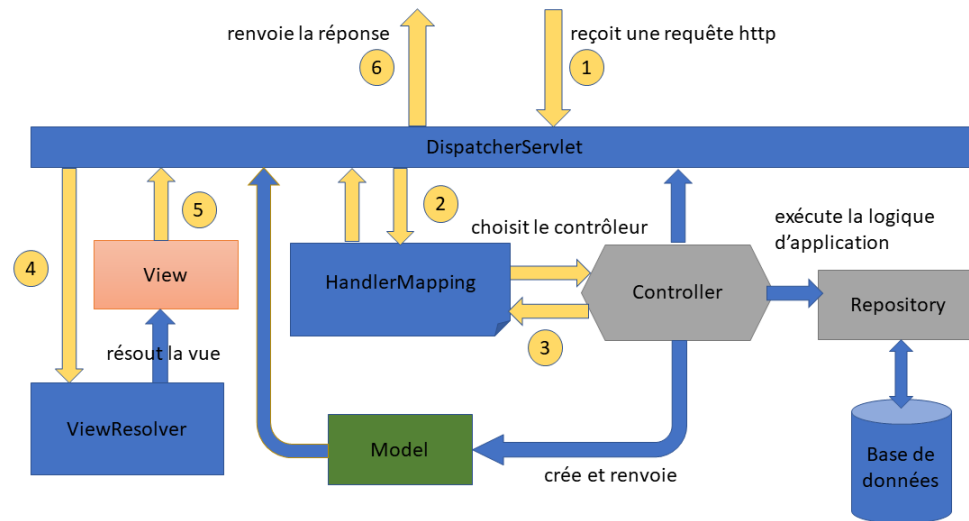


FIGURE 9 – Architecture de SpringMVC

Le DispatcherServlet de Spring gère toutes les requêtes reçues par l'application ①, il va d'abord solliciter le *HandlerMapping* ② (Mappage de gestionnaire) qui va faire le lien avec un bean contrôleur enregistré annoté par *@Controller*. Celui-ci va ensuite exécuter la logique d'application fournie par le *HandlerAdapter* qui va résulter en un modèle. Il va aussi retourner un nom logique de vue au *HandlerAdapter* ③. Après cela, le DispatcherServlet va solliciter le *ViewResolver* ④ qui va résoudre la *View* grâce au nom logique ⑤. Le *DispatcherServlet* envoie ensuite le processus de rendu à la vue renvoyée qui va constituer la réponse à la requête initialement reçue ⑥.

## 6.2.2 Fonctionnement de Spring Security

*Spring Security* est un framework permettant la gestion de l'authentification et le contrôle d'accès. Comme le reste des projets Spring, il est facilement personnalisable. Son utilisation va permettre de faciliter l'utilisation du patron de sécurité avec Pamela grâce à des classes relativement faciles à identifier. Parmi les fonctionnalités de Spring Security, celle qui va nous intéresser pour ce projet est l'authentification.

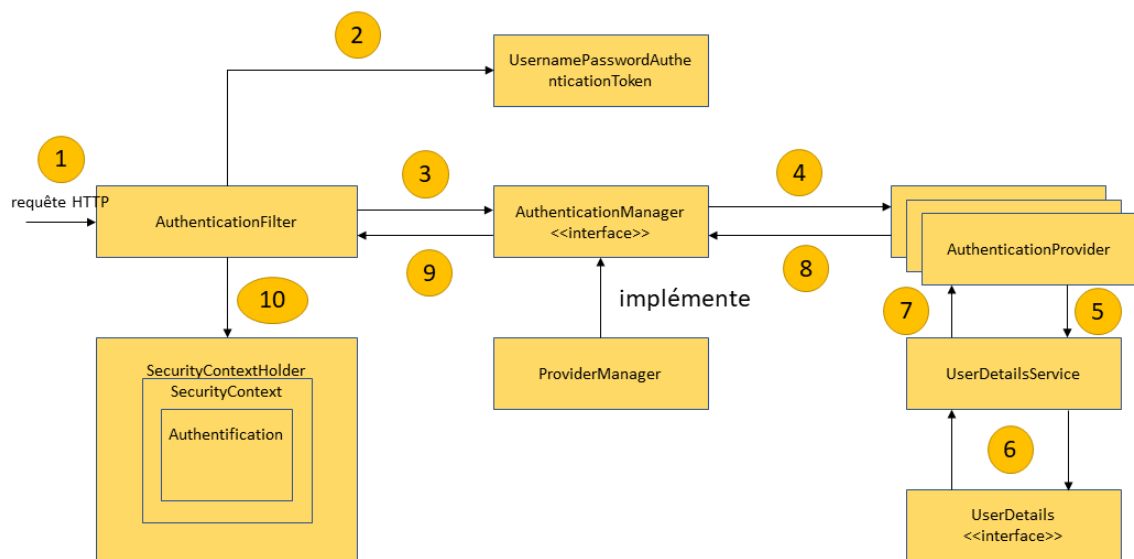


FIGURE 10 – Architecture de l'authentification avec Spring Security

Lorsque l'application développée avec *Spring Security* reçoit une requête, la requête passe à travers une chaîne de filtre ①. Quand la requête contient une demande d'authentification, l'*AuthenticationFilter* va extraire les informations d'identification de l'utilisateur (généralement son nom d'utilisateur et son mot de passe) et créer un



objet *Authentication*. Si les informations reçues contiennent un nom d'utilisateur et un mot de passe, un *UsernamePasswordAuthenticationToken* sera créé contenant le nom d'utilisateur et le mot de passe ②.

Ce token sera utilisé pour invoquer la méthode *authenticate* de l'*AuthenticationManager* qui est implémenté par *ProviderManager* ③. Il existe plusieurs *AuthenticationProvider* déjà configurés et listés dans *ProviderManager*. Celui que nous utilisons dans le projet est le *DaoAuthenticationManager*. DAO correspond à "data access object", c'est un modèle qui fournit une interface abstraite à un type de base de données. En mappant les appels d'application à la couche de persistance, le DAO fournit certaines opérations de données spécifiques sans exposer les détails de la base de données. Le *DaoAuthenticationManager* utilise *UserDetailsService* ⑤ pour récupérer les données de l'utilisateur en fonction de son nom d'utilisateur ⑥, ⑦, ⑧, ⑨. Si l'authentification ⑩ réussit alors l'objet *Authentication* complet (avec "authenticated = True", la liste des autorités et le nom d'utilisateur) est renvoyé. Sinon une exception *AuthenticationException* sera levée. Enfin, *AuthenticationManager* renvoie l'objet *Authentication* à l'*AuthenticationFilter*, l'authentification a réussi et l'objet est stocké dans le *SecurityContext*.

## 7 Construction de l'application Spring

### 7.1 Architecture du site web

Pour commencer une application Spring, il faut tout d'abord, depuis le site <https://start.spring.io/>, créer un *SpringInitializr* qui permet de télécharger un projet Spring vide qui contient les dépendances Spring souhaitées ainsi qu'un wrapper Gradle ou Maven. Pour notre projet, nous avons pris les dépendances Spring Security et Spring web, puis nous en avons rajouté par la suite en modifiant le fichier "build.gradle" qui gère les dépendances. Ce projet, au départ, ne va contenir que la classe qui permet de lancer l'application avec le main. Il faut ensuite créer de

nouvelles classes ou étendre celles qui sont disponibles avec le framework.

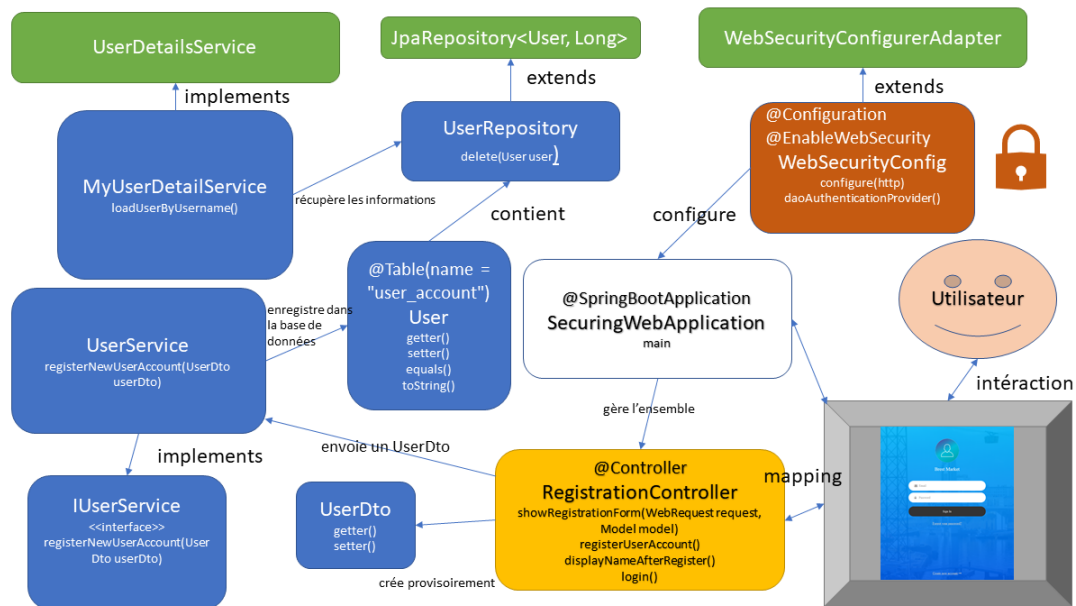


FIGURE 11 – Architecture de l'application web

Les classes/interfaces bleues correspondent aux classes principales de la partie modèle de l'application que nous avons développés, la classe orange est la classe principale pour la sécurité, la classe jaune correspond au controleur, l'interface et la vue pour l'utilisateur sont dans le cadre gris, les classes/interfaces vertes sont fournies par Spring et sont les classes que je vais étendre. Les vues de l'utilisateur se trouvent en Annexe 3.

Mis à part les classes, une application Spring se divise en 3 composantes principales sur lesquelles le programmeur va travailler :

- Le code Java
- Le build.gradle
- Les ressources

L'ensemble des classes se trouve en Annexe 2.

Concernant notre code, c'est la classe **SecuringWebApplication** qui contient le *main*, celui-ci permet de lancer l'application Spring.

La configuration de notre application commence avec la classe **WebSecurityConfig**, les annotations *@Configuration* et *@EnableWebSecurity* indiquent que cette classe va contenir les éléments de configuration de l'application et va utiliser la fonctionnalité WebSecurity de Spring.

Cette fonctionnalité permet notamment d'implémenter une authentification pour notre application web.

## 7.2 L'authentification

Dans la méthode *Configure* de la classe **WebSecurityConfig**, on peut voir la liste des requêtes http qui sont disponibles sans authentification :

- les ressources (javascript, CSS, etc...)
- la page d'accueil (home)
- la page d'identification (login)
- la page d'inscription (register)

Pour accéder au reste du site, il va falloir s'inscrire puis s'identifier.

Pour cela, on va tout d'abord créer une classe **UserDto** pour Data Transfer Object qui permettra d'instancier les objets contenant les informations de l'utilisateur (Nom, Prénom, Email, Mot de Passe). Ces objets seront vérifiés puis transférés à la base de donnée des utilisateurs.

La récupération des données et les transferts vers la base de donnée sont réalisés avec le **RegistrationController** défini par l'annotation *@Controller* du framework Spring.

Dans cette classe, on va tout d'abord lier (bind en anglais) la page register.html avec un objet UserDto grâce à l'annotation *@GetMapping* et la méthode *showRegistrationForm*.

Il faut ensuite récupérer les informations entrées par l'utilisateur dans le formulaire (form) auquel on attribue une action *doRegister*. Enfin, le post-mapping et la méthode *registerUserAccount* permettent de recevoir ces données.

Pour traiter ces données et les envoyer vers la base de donnée. Il faut faire appel à la classe **UserService** prévue pour cet effet.

Cette classe va tout d'abord instancier un **UserRepository** qui est un répertoire JPA(Java persistence API) conçu par la plate-forme Spring spécialement pour la gestion de base de données avec Java. Elle va ensuite vérifier que le contenu de l'objet instance de **UserDto** soit bien conforme.

Pour que les données fournies par l'utilisateur soient conformes, elles doivent vérifier si :

- l'adresse email n'a pas encore été vérifiée
- l'adresse email est de la bonne forme
- le mot de passe a bien été confirmé

Si toutes ces conditions sont vérifiées, la méthode *registerNewUserAccount* de **UserService** va alors créer un nouvel objet instance de **User** dans lequel seront transposé les information du **UserDto** et enregistré celui-ci dans l'instance de **UserRepository**.

Il est à noter qu'une partie de la vérification se fait par l'ajout de *@Valid* devant l'instanciation du **UserDto** dans le **RegistrationController**. *@Valid* va vérifier que les annotations *@NotNull*, *@NotEmpty* et *@ValidEmail* contenues dans l'écriture de la classe **UserDto** soient respectés.

## 7.3 Le build.gradle

Notre projet Spring est munie d'un moteur de construction gradle qui permet de contrôler le processus de développement pour la compilation et les tests.

Le **buid.gradle** va notamment permettre l'implémentation de toutes les dépendances dont notre projet aura besoin comme par exemple les starter thymeleaf, web

et security qui sont indispensables.

## 7.4 Les ressources

Les ressources vont comprendre tout ce qui n'a pas trait au code Java. Le répertoire *template* va contenir tout le code html permettant l'implémentation des pages web.

La personnalisation de ces pages web se fait via la classe **MvcConfig**. Cette partie ne nous intéressant pas, nous nous en sommes limité avec le strict minimum.

Le répertoire *static* comprend tous ce qui a trait à la personnalisation des pages web (couleur, image, etc). Il va donc contenir le code CSS et le Javascript.

Enfin, les ressources vont aussi prendre en compte le fichier **application.properties** qui va définir les caractéristiques de la base de données, ainsi que de Javamail qui permet à notre application de communiquer avec les utilisateurs (et éventuellement envoyer un mail de vérification pour la création du compte).

# 8 Insertion du patron de sécurité d'authentification dans l'application Spring

## 8.1 Le choix d'un patron de sécurité pour le web

L'article [6] explicite un patron de sécurité particulier : le patron d'authentification. Il définit ainsi 6 propriétés qui doivent être vérifiées afin de satisfaire un contrat, permettant ainsi de mieux sécuriser l'application Java concernée. Ces propriétés concernent l'unicité des informations d'authentification, l'invariance des informations d'authentifications, l'invariance de l'authentificateur, la validité de la preuve d'identité, l'exactitude de la méthode d'authentification (qui correspond à la méthode

authenticate pour Pamela) et l'exactitude de la méthode de requête (qui correspond à la méthode request pour Pamela).

Ces propriétés correspondent à des besoins primaires qui vont former le coeur de toute authentification fonctionnelle.

Cependant, lorsque l'application a une dimension web (cela pourrait être aussi le cas sur d'autres réseaux), de nouvelles vulnérabilités apparaissent, nécessitant de nouvelles contraintes de sécurité.

- Lorsqu'une personne malveillante cherche à s'authentifier avec les identifiants d'une autre personne sans connaître ceux-ci au préalable, on parle d'**attaque par force brute**. Une solution serait alors de limiter les tentatives d'essai d'authentification successifs pour une même personne, il s'agira alors soit de bloquer le compte associé au nom d'utilisateur et potentiellement envoyer un mail à son adresse afin qu'il puisse changer son mot de passe ou qu'il soit averti qu'une personne essaye de se connecter à son compte. Une autre possibilité est de limiter les tentatives en bloquant l'adresse IP avec laquelle la personne tente de se connecter si trop de tentatives ont lieu dans un court laps de temps. Par exemple, si une personne essaye de se connecter 5 fois en moins de 2 minutes et qu'elle échoue, elle ne pourra pas réessayer dans les 10 minutes qui suivent. Cela permet de rendre une attaque par force brute obsolète.
- Un autre problème concerne la difficulté d'un mot de passe. Lié à l'attaque par force brute. Un mot de passe faible pourrait être facilement retrouvé et les malfaiteurs pourraient accéder aux données de l'utilisateur. Il faut donc exiger un mot de passe plus complexe lors de l'enregistrement d'un nouvel utilisateur.
- Pour aller plus loin, de nombreuses applications web vont proposer à l'utilisateur de pouvoir *Rester connecté*. Cette fonctionnalité se fait par le biais d'un cookie qui sera créé lorsque l'utilisateur l'actionne. Ce cookie peut contenir le nom d'utilisateur, souvent haché avec le mot de passe et une clé prédéfini. Si une personne malveillante met la main sur ce cookie, il y a donc un risque qu'elle puisse accéder aux identifiants. La création du cookie doit donc être manipulée avec précaution.

Afin d'exemplifier l'utilisation de Pamela pour la mise en place des patrons de sécurité, nous allons implémenter le patron d'authentification (*AuthenticatorPattern*) déjà existant dans le framework Pamela et qui comprend certaines propriétés définies par [6]. En effet, l'authentification est cruciale pour une application à dimension web et dont l'accès serait disponible pour toute personne ayant une confection internet. Le choix de ce patron est donc une évidence.

Dans un deuxième temps, nous avons décidé de tester de tester la capacité d'adaptation de Pamela, c'est à dire la facilité avec laquelle nous pourrions lui rajouter la possibilité de vérifier des propriétés (selon le principe du "Design By Contract") via des extensions pour les *PatternInstance*. Nous allons donc ensuite étendre le patron d'authentification pour qu'il puisse aussi permettre d'assurer la sécurité dans le cas d'une attaque de force brute.

## 8.2 Application du patron d'authentification préexistant

### 8.2.1 Aspect fonctionnel

La partie 5.2 décrit les fonctionnalités du patron d'authentification en suivant le formalisme exposé dans [8], les diagrammes de classes et de séquence sont respectivement les figures 5 et 6.

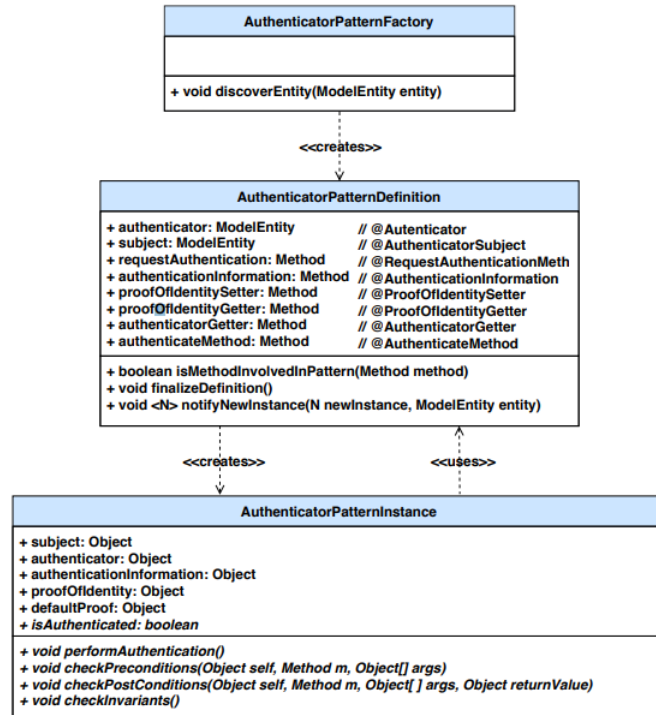


FIGURE 12 – Les annotations du patron d’authentification

Le but étant d’implémenter un patron Authentificateur, il faut identifier les classes qui correspondent à ce que nous avons défini dans la partie 6.4.

Ainsi, il va falloir étendre les classes correspondantes pour y ajouter les annotations de Pamela sur les bonnes méthodes et les modifier pour que cela fonctionne correctement (voir Figure 7).

L’authentificateur de Spring correspond à la classe *AuthenticationProvider*. Nous créons donc une extension *CustomAuthenticationProvider* avec les annotations Pamela correspondante (*@ModelEntity*, *@Authenticator*).

Concernant le sujet, on peut identifier l’utilisateur qui se connecte et qui utilise l’application comme une session. En effet, les sessions HTTP sont une fonctionnalité standard qui permettent aux serveurs Web de conserver l’identité de l’utilisateur et de stocker des données spécifiques à l’utilisateur lors de multiples interactions de



demande / réponse entre une application cliente et une application Web. Nous allons donc créer une session personnalisée avec les annotations de Pamela (*@ModelEntity*, *@AuthenticatorSubject*) en récupérant les informations de la session Spring à l'aide de *@Scope*). Cette classe personnalisée est l'implémentation *SessionInfoImpl* de l'interface nommée *SessionInfo*. Cette implémentation se fait par le biais d'une annotation *@ImplementationClass* fournie par Pamela, il en est de même pour *CustomAuthenticationProvider*.

C'est la session qui va définir le *Pattern\_ID*, c'est à dire que Pamela va implémenter une instance de patron de sécurité pour chaque session créée. Ainsi, on retrouvera cet ID dans les annotations concernant les patrons devant chaque classe faisant partie du modèle de Pamela.

Comme nous l'avons vu à la partie 6.1, Pamela met en place un *PatternFactory*, ce type de patron fournit l'un des meilleurs moyens de créer un objet. Dans le *PatternFactory*, nous créons un objet sans exposer la logique de création au client et faisons référence à un objet nouvellement créé à l'aide d'une interface commune. Les factorys pour notre application se trouvent dans *AuthManagerService*.

Les informations d'authentification sont récupérées dans avec le *CustomAuthenticationProvider*, c'est grâce à *@RequestAuthentication* associée à la méthode *request* et qui retourne l'identifiant unique de l'instance de modèle d'authentification associée.

Ainsi toute ces annotations vont lier les composantes Pamela aux entités correspondantes de l'application. Nous avons ainsi pu tester une des composantes du patron de sécurité, l'unicité des informations d'authentification.

### 8.2.2 Aspect conception

**WebSecurityConfig** : cette classe va permettre de configurer la sécurité de l'application en indiquant quels ressources seront accessibles ou nécessiteront une authentification. Elle va aussi définir la page de login, le type de fournisseur d'authentification, la manière dont l'application va gérer les erreurs, la manière de gérer les sessions ou encore comment les mots de passe seront cryptés.

**AuthManagerService** : cette classe va être définie comme un service Spring avec l'annotation *@Service*. C'est cette classe qui fait le pont entre notre application et Pamela car celle-ci va définir les factory et le contexte du modèle de notre application. Elle va donc permettre de créer les éléments nécessaires au patron comme la session qui correspond au sujet et l'authenticatorProvider qui va correspondre à l'authentificateur.

**CustomAuthenticationProvider** : cette classe permet de personnaliser l' *AuthenticationProvider* de Spring afin de pouvoir y insérer les annotations du framework Pamela. En effet c'est cette classe qui va définir la façon d'authentifier en implémentant la méthode *authenticate* qui va procéder à l'authentification. De plus c'est dans cette classe que nous allons imposer un nouveau contrat à la méthode *authenticate* par le biais de l'annotation *@Requires*.

**MyHttpSessionEventPublisher** : cette classe qui hérite de *HttpSessionEventPublisher* nous permet de garder la main sur la création et la destruction de session avec un affichage en temps réel dans la console.

**MyUserDetailsService** : cette classe va permettre de retirer les informations des utilisateurs enregistrés dans la base de donnée.

**SessionInfo** : nous avons défini la session comme étant le sujet du patron d'authentification pour notre application web. Cette interface va donc permettre à Pamela, par le biais d'annotation de localiser et de récupérer les informations pour le modèle par le biais de setter/getter.

**CustomAuthenticatorPatternDefinition, CustomAuthenticatorPatternFactory, CustomAuthenticatorPatternInstance, CustomPatternsLibrary, TooManyLoginAttemptsException** : ces classes permettent de personnaliser les classes Pamela correspondantes du patron préexistant pour y ajouter de nouvelles clauses au contrat.

### 8.3 Implémentation d'une nouvelle propriété pour parer les attaques de force brute

Il s'agit ici de permettre au framework Pamela de pouvoir s'adapter aux différentes spécificités des domaines dans lesquels il sera utilisé.

Les attaques de force brut sont par exemple des types failles moins généraux que les propriétés déjà définies pour le patron d'authentification de Pamela mais sont tout de même assez importantes pour que l'intervention de Pamela pour assurer l'impossibilité soit pertinente.

Il a donc fallu modifier le noyau de Pamela pour que le framework puisse permettre la mise en place de précondition, postcondition et d'invariant de classe (voir 4.3.2 sur le "Design By Contract") personnalisables.

Il va donc falloir créer une classe héritant de *AuthenticatorPatternInstance* et qui va compléter la méthode *invokePrecondition* de la classe mère avec de nouvelles contraintes (ou *property*) sur les préconditions. Une nouvelle annotation `@Requires` contenant en paramètre le nom du nouveau patron avec le nom de la propriété à vérifier et qui sera placé avant la méthode concernée par le contrat (*authenticate* pour le patron d'authentification). Si la propriété n'est pas vérifiée, la méthode renvoie une exception elle aussi personnalisable.

```
@Override
@Requires(
    patternID = SessionInfo.PATTERN_ID,
    type = PropertyParadigmType.TemporalLogic,
    property = "assert always auth_fail[*3] & time_limit<3min @ (auth_fail)",
    exceptionWhenViolated = TooManyLoginAttemptsException.class)

public Authentication authenticate(Authentication authentication) throws AuthenticationException;
```

FIGURE 13 – Ajout de l'annotation `@Requires` avant la méthode *authenticate()*

Dans le cadre de notre étude, nous avons créer la propriété qui contraint l'utilisateur à ne pouvoir tenter que trois fois de s'authentifier dans les trois minutes qui suivent une première tentative d'authentification. Cette contrainte étant temporelle, nous l'avons formalisé en langage PSL (pour *Property Specification Language*). Le

PSL est un langage pour la logique temporelle standardisé par une commission de l'IEEE qui permet de réaliser une spécification matérielle à l'aide de propriétés et d'assertions. Il est très utilisé pour la vérification de modèles dans l'industrie et est compatible avec le VHDL et le Verilog. En PSL la propriété s'écrirait donc "assert always auth\_fail[\*3] & time\_limit<3min @ (auth\_fail)". Concernant la terminologie, *auth\_fail* est un évènement correspondant à une tentative d'authentification qui a échoué. *time\_limit* est un compteur temporel (chronomètre) qui s'initialise lors d'un échec d'authentification (d'où le @(auth\_fail) qui correspond au "clocking event"). Enfin, le [\*3] correspond à la limite de trois essais consécutifs pour *auth\_fail*. La volonté de chercher un formalisme efficace pour décrire les propriétés vient de la nécessité de simplifier l'expression de la propriété pour faciliter la compréhension de relecture du programme tout en restant assez compact.

Pour implémenter la propriété, on initialise un cache lors du première appel de la précondition. Ce cache est défini pour se réinitialisé automatiquement au bout de 3 minutes. A chaque appel de la précondition, la valeur dans le cache augmente de 1. Lorsque cette valeur dépasse 3, une exception *TooManyLoginAttemptsException* est renvoyée et l'authentification n'est plus possible avant que les trois minutes soient écoulées.

## 8.4 Bilan de l'expérimentation

### 8.4.1 Concernant les propriétés du patron d'authentification préexistant (implémentation décrite en 8.2)

```
Registering SessionInfoImpl created on 2021-01-15T17:58:34.2116448 as Subject for pattern instance org.openflexo.pamela.securitypatterns.authent:
    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)
    at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
    at java.base/java.lang.Thread.run(Thread.java:830)
Registering com.example.securingsweb.authentication.CustomAuthenticationProvider$CustomAuthenticationProviderImpl_$$jvst103_0@4641fb6 as Authent:
Tiens j'ai trouve des AuthInfo identiques
currentAuthInfo=auth_info1
oppositeAuthInfo=auth_info1
2021-01-15 17:58:34.224 ERROR 5624 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Session event listener threw exception
org.openflexo.pamela.exceptions.ModelExecutionException: Subject Invariant Violation: Authentication information are not unique
```

FIGURE 14 – Résultats lors de l'enregistrement d'un deuxième utilisateur avec des informations identiques au premier

Nous pouvons voir qu'il y a bien un suivi du modèle réalisé par Pamela et que le test effectué a bien fonctionné.

L'utilisateur peut s'enregistrer et se login sans soucis s'il respecte les conditions mais l'application détecte bien si deux utilisateurs essayent de se connecter avec les mêmes identifications et bloque la deuxième session qui va essayer de se connecter avec les même informations d'authentications. On remarquera aussi que le modèle créé par Pamela est bien disponible dans la console.

### 8.4.2 Concernant la nouvelle propriété implémentée (décrite en 8.3)

```
1ère tentative:
checkAuthentication() !
Invoking preconditions for public org.springframework.security.core.Authentication com.example.securingweb.authentic
Invoking precondition assert always auth_fail[*3] & time_limit<3min @ (auth_fail)
J'appelle la methode public org.springframework.security.core.Authentication com.example.securingweb.authentication.
LA CLE VAUT 0:0:0:0:0:0:1
La tentative numero 1 a echoue
L'authentification a echoue, la valeur dans le cache augmente de 1
On utilise bien le processMethodBeforeInvoke
On utilise bien le CustomAuthenticationProvider pour org.springframework.security.authentication.UsernamePasswordAuthenticationToken@000

2ème tentative:
LA CLE VAUT 0:0:0:0:0:0:1
La tentative numero 2 a echoue
L'authentification a echoue, la valeur dans le cache augmente de 1

3ème tentative:
LA CLE VAUT 0:0:0:0:0:0:1
La tentative numero 3 a echoue
L'authentification a echoue, la valeur dans le cache augmente de 1

Nouvelle tentative pendant les 3 prochaines minutes:
LA CLE VAUT 0:0:0:0:0:0:1
Too many login attempts !!!!!!! Violated property assert always auth_fail[*3] & time_limit<3min @ (auth_fail)
```

FIGURE 15 – Résultats lorsqu'un attaquant échoue plusieurs fois à trouver le mot de passe

Nous pouvons voir qu'au moment de l'authentification, le framework Pamela va bien chercher à vérifier si les préconditions sont remplies. Il va donc bien vérifier si l'utilisateur n'a pas déjà essayé de s'authentifier plusieurs fois dans un court laps de temps.

Lorsque l'application est lancée, nous avons bien un affichage en temps réel des exceptions retournées lorsque des contraintes pour l'authentification ne sont pas respectées. On peut voir sur les figures 14 et 15 les résultats retournés sur la console avec le monitoring de Pamela en action. La vue de l'utilisateur est montrée en Annexe 4.

Tous ces résultats nous montrent que Pamela est bien compatible avec le développement d'application Spring pour le web. En matière d'authentification, le framework permet de renforcer la sécurité dès la phase de développement. Ces résultats encourageants permettent de conforter l'idée que le framework Pamela s'avère efficace dans le domaine des patrons de sécurité.

## 9 Conclusion

### 9.1 Réponse à la problématique

Concernant la compatibilité entre Spring et Pamela, la principale difficulté réside dans l'identification des bonnes classes Spring pour l'application du patron. En effet, le framework Spring a pour objectif de limiter la quantité de code à fournir par le développeur et s'occupe automatiquement de la plupart des fonctionnalités. Il n'est donc pas fait à l'origine pour une personnalisation intégrale de l'application même s'il le permet. C'est pourquoi l'implémentation du patron a demandé une recherche profonde à l'intérieur du framework pour ressortir les bonnes classes et pouvoir les personnaliser en rajoutant les annotations. L'utilisation de Pamela pour implémenter des patrons de sécurité sur une application Spring est donc possible mais délicate. Il est à noter cependant que le fait de l'avoir fait une fois lors de ce projet rend la tâche plus facile pour des implémentations ultérieures.

Comme nous avons pu le voir dans la partie précédente, le framework Pamela permet bien de rajouter une couche de sécurité lors du développement de l'application. Il permet donc efficacement de surveiller que les propriétés de sécurité soient bien vérifiées.

Lors du projet, nous avons rajouté une nouvelle fonctionnalité pour Pamela permettant de rajouter des clauses au contrat en étendant certaines classes du framework. Leurs mises en place est relativement facile et fonctionne de manière efficace.

Accompagné de mon maître de stage, nous avons donc bien répondu à l'ensemble des problèmes auxquels nous devons répondre en plus d'avoir permis au framework Pamela de s'enrichir afin de répondre à de nouveaux besoins.

## 9.2 Résumé global

Pour conclure, j'ai effectué mon stage de fin d'études pour mon diplôme d'ingénieur en tant que stagiaire en recherche pour la cybersécurité au sein du lab-STICC à l'Ensta Bretagne. Lors de ce stage de 6 mois, j'ai pu mettre en pratique mes connaissances théoriques acquises durant ma formation en génie logicielle et sécurité et je me suis confronté aux difficultés du monde du travail et de la recherche dans le secteur de la cybersécurité.

Plus particulièrement j'ai utilisé l'IDE Eclipse pour développer en langage Java une application web utilisant le framework Spring. Spring m'a permis de mettre en place une authentification simple sur laquelle j'ai greffé le patron de sécurité d'authentification du framework Pamela, régit par un mécanisme de contrat, afin de tester celui-ci.

Tout cela dans le but concret d'exemplifier l'utilisation de Pamela dans la sécurisation d'application web.

## 9.3 Apport personnel

Lors de ce stage, j'ai pu découvrir les défis qui découlent de l'activité de recherche. En effet, dans des secteurs très pointus comme celui dans lequel j'ai travaillé, j'ai dû accaparer rapidement un grand nombre de connaissance pour pouvoir maîtriser le sujet. Ainsi, la découverte de notions et de toute une partie du monde informatique (patron de conception, le "Design by Contract", etc...) fut à la fois passionnante et à la fois très exigeante en terme de capacité de compréhension.

Plus que jamais ce stage m'a conforté dans l'idée que la communication est la clé du bon fonctionnement de tout projet. Lors du début du stage, j'ai été d'abord réticent à faire des point régulièrement et j'avais du mal à admettre que je ne pouvais pas réussir à tout faire par moi-même. J'ai peu à peu réprimé mes craintes initiales et progressé concernant les interactions avec mes encadrants de projet. Mon manque



de confiance en soi persistant montre que cela reste quand même un point sur lequel j'ai du progrès à faire.

De même, confinement aidant, ce stage a impliqué de moi une plus grande autonomie par rapport à ce à quoi j'ai pu être confronté précédemment dans ma vie. Additionner à cela une difficulté à avancer sur un sujet que l'on ne maîtrise pas encore, l'utilisation des visioconférences pour faire des points réguliers avec les encadrants fut vital pour s'accrocher au projet. Ainsi j'ai réalisé qu'autonomie n'est pas synonyme d'isolation. Bien au contraire, savoir quand demander de l'aide lorsque l'on est bloqué est crucial dans des projets tel que celui-ci.

Enfin, j'ai acquis au cours de ce projet de nombreuses facultés techniques. Que ce soit pour la programmation en Java avec l'utilisation du framework Spring et des annotations, pour la découverte et le fonctionnement des wrappers comme Gradle, des outils de communication à distance et aussi comment faire des points rapides avec PowerPoint qui soient efficaces et pertinents.

## 9.4 Ouverture

Lors de mon stage, je n'ai pas pu traiter l'ensemble des fonctionnalités que propose Pamela pour la sécurité. Cependant, j'ai conscience que mon travail constitue un tremplin qui permettrait à de futurs chercheurs de parvenir rapidement à compléter ces résultats en complexifiant l'application et en y implémentant des patrons différents. Pour finir, ce stage a constitué pour moi une première approche concrète avec la cybersécurité et j'ai réalisé à quel point ce domaine est vaste. Je peux maintenant inclure ce secteur pour mon orientation vers mon premier emploi.

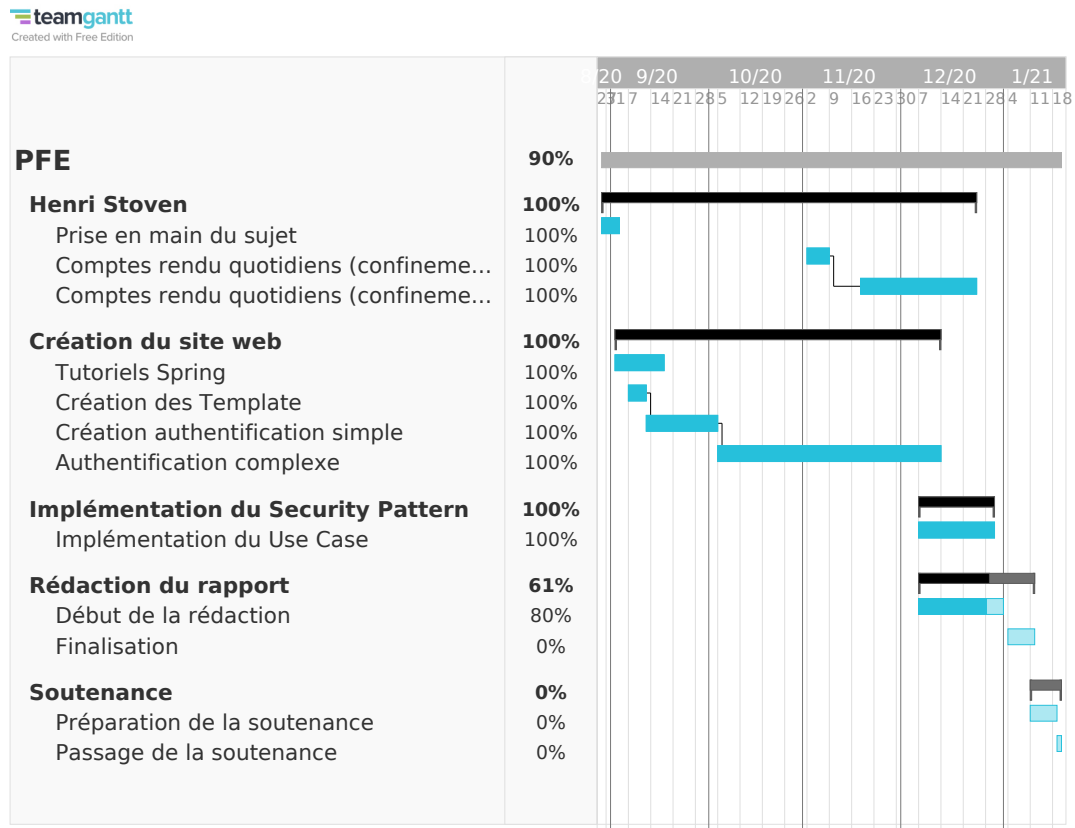
## Références bibliographiques

- [1] <https://www.varonis.com/blog/cybersecurity-statistics/>.
- [2] <https://www.ensta-bretagne.fr/fr>.
- [3] Amina SOUAG, Raúl MAZO, Camille SALINESI et Isabelle COMYN-WATTIAU. « Reusable knowledge in security requirements engineering : a systematic mapping study. » In : *Requirements Engineering* (2016).
- [4] Chad Dougherty and Kirk SAYRE, Robert C. SEACORD, David SVOBODA et Kazuya TOGASHI. *Secure Design Patterns*. Rapp. tech. 2009.
- [5] Bertrand MEYER. « Applying "Design by Contract" ». In : *Computer, IEEE* (1992).
- [6] Caine SYLVA, Sylvain GUÉRIN, Raúl MAZO et Joel CHAMPEAU. « Contract based design patterns : A Design by Contract Approach to Specify Security Patterns ». In : *ARES 2020 : The 15th International Conference on Availability, Reliability and Security Virtual Event Ireland August, 2020* (2020).
- [7] <https://www.openflexo.org>.
- [8] Markus SCHUMACHER, Eduardo FERNANDEZ-BUGLIONI, Duane HYBERTSON, Frank BUSCHMANN et Peter SOMMERLAD. *Security Patterns : Integrating Security and Systems Engineering*. 2006.
- [9] F. BUSCHMANN, R. MEUNIER, H. ROHNERT, P. SOMMERLAD et M. STAL. *Pattern Oriented Software Architecture - A System of Patterns*. John Wiley Sons, 1996.
- [10] <https://spring.io/>.

## Liste des Figures

1	Organisation des institutions de recherche (localisation de mon travail encadrée en rouge) . . . . .	7
2	Fonctionnement du "Design by Contract" . . . . .	10
3	Positionnement de l'étude menée (en bleu) . . . . .	11
4	Approche Pamela pour le couplage entre modèle et code . . . . .	14
5	Diagramme de classe du patron d'authentification . . . . .	16
6	Diagramme de séquence pour l'authentification . . . . .	16
7	Annotations dans le code source . . . . .	18
8	Flux de contrôle pour l'authentification avec <i>Client</i> comme sujet et <i>Manager</i> comme authentificateur[6] . . . . .	19
9	Architecture de SpringMVC . . . . .	22
10	Architecture de l'authentification avec Spring Security . . . . .	23
11	Architecture de l'application web . . . . .	25
12	Les annotations du patron d'authentification . . . . .	31
13	Ajout de l'annotation <i>@Requires</i> avant la méthode <i>authenticate()</i> . . . . .	34
14	Résultats lors de l'enregistrement d'un deuxième utilisateur avec des informations identiques au premier . . . . .	36
15	Résultats lorsqu'un attaquant échoue plusieurs fois à trouver le mot de passe . . . . .	37

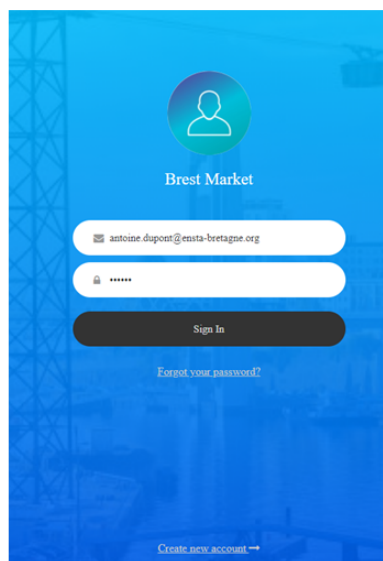
## Annexe 1 - Diagramme de Gantt généré avec *teamgantt.com*



## Annexe 2 - Ensemble des classes de l'application

```
▼ J pamela-spring-security-uc [pamela-HenriTest 1.6.HenriTest]
  ▼ src/main/java
    ▼ com.example.securingweb
      > MvcConfig.java
      > SecuringWebApplication.java
      > WebMvcConfig.java
      > WebSecurityConfig.java
    ▼ com.example.securingweb.authentication
      > AuthenticationInterceptor.java
      > AuthManagerService.java
      > CustomAuthenticationProvider.java
      > MyAuthenticationToken.java
      > MyHttpSessionEventPublisher.java
      > MyUserDetails.java
      > MyUserDetailsService.java
      > SessionInfo.java
    ▼ com.example.securingweb.exceptions
      > UserAlreadyExistException.java
    ▼ com.example.securingweb.patterns
      > CustomAuthenticatorPatternDefinition.java
      > CustomAuthenticatorPatternFactory.java
      > CustomAuthenticatorPatternInstance.java
      > CustomPatternsLibrary.java
      > TooManyLoginAttemptsException.java
    ▼ com.example.securingweb.persistence.dao
      > DeviceMetadataRepository.java
      > NewLocationTokenRepository.java
      > PasswordResetTokenRepository.java
      > PrivilegeRepository.java
      > RoleRepository.java
      > UserLocationRepository.java
      > UserRepository.java
      > VerificationTokenRepository.java
    ▼ com.example.securingweb.persistence.model
      > DeviceMetadata.java
      > NewLocationToken.java
      > PasswordResetToken.java
      > PasswordMatchesValidator.java
      > Privilege.java
      > Role.java
      > User.java
      > UserLocation.java
      > VerificationToken.java
    ▼ com.example.securingweb.registration
      > EmailValidator.java
      > IUserService.java
      > PasswordMatches.java
      > PasswordMatchesValidator.java
      > RegistrationController.java
      > UserDto.java
      > UserService.java
      > ValidEmail.java
```

### Annexe 3 - Pages d'authentification (login) et d'enregistrement(register)



The login page for Brest Market features a blue background with a faint image of a market stall. At the top, there is a circular profile icon placeholder and the text "Brest Market". Below this, there are two input fields: the first contains the email address "antoine.dupont@ensta-bretagne.org" and the second contains masked characters "\*\*\*\*\*". A dark blue "Sign In" button is positioned below the password field. A link "Forgot your password?" is located below the button. At the bottom, there is a link "Create new account" with a right-pointing arrow.

Brest Market

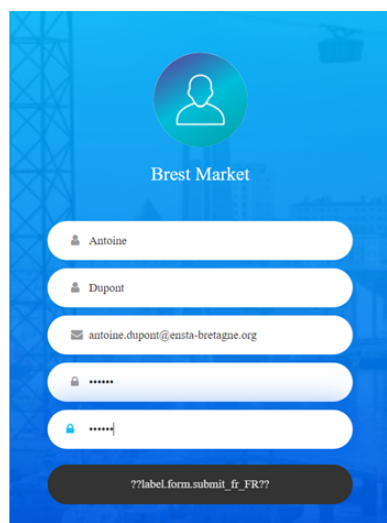
antoine.dupont@ensta-bretagne.org

\*\*\*\*\*

Sign In

[Forgot your password?](#)

[Create new account](#) →



The registration page for Brest Market has the same blue background and "Brest Market" header as the login page. It includes a circular profile icon placeholder. Below the header, there are five input fields: the first two contain the first and last names "Antoine" and "Dupont"; the third contains the email address "antoine.dupont@ensta-bretagne.org"; the fourth contains masked characters "\*\*\*\*\*"; and the fifth contains masked characters "\*\*\*\*\*" with a small blue lock icon on the left, indicating a password field. A dark blue button with the text "77label.form.submit\_fr\_FR??" is at the bottom.

Brest Market

Antoine

Dupont

antoine.dupont@ensta-bretagne.org

\*\*\*\*\*

\*\*\*\*\*

77label.form.submit\_fr\_FR??

## Annexe 4 - Vue de l'utilisateur lorsqu'une exception est levée

