iscte
INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

Módulo 6: Princípios de Engenharia de Software

Aula 2 - Testes de Software, Documentação e Análise de Código

UPskill

O que são **Testes de Software?**

- São um processo que faz parte do desenvolvimento de um software;
- Principal objectivo é detectar falhas/bugs/erros para que sejam corrigidos até que o produto final atinja a qualidade desejada;
- São geralmente a última etapa no desenvolvimento de um programa, ou numa metodologia ágil, a última etapa de cada *sprint*;
- As pessoas que fazem este tipo de testes têm o nome de *testers;*


iscte
INSTITUTO UNIVERSITÁRIO DE LISBOA
emprego digital

O que precisa de ser **testado?**

A norma ISO 9126, que trata da qualidade de produtos de software, designa 6 atributos que devem ser avaliados:

- **Funcionalidade:** avalia se a aplicação faz, de facto, aquilo para o qual foi desenhada;
- **Confiabilidade:** O sistema consegue manter o padrão de desempenho ideal quando é utilizado dentro das funções previstas?
- **Usabilidade:** tem a ver se o utilizador consegue entender e fazer uso da aplicação de uma forma mais simples;

O que precisa de ser **testado?**

- **Eficiência:** a análise da eficiência num sistema engloba testes sobre o tempo de processamento ou resposta da aplicação;
- **Manutenibilidade:** facilidade com a qual o software pode passar por mudanças;
- **Portabilidade:** a capacidade de o sistema de ser transferido para ambientes diferentes daquele para o qual ele foi planeado inicialmente;

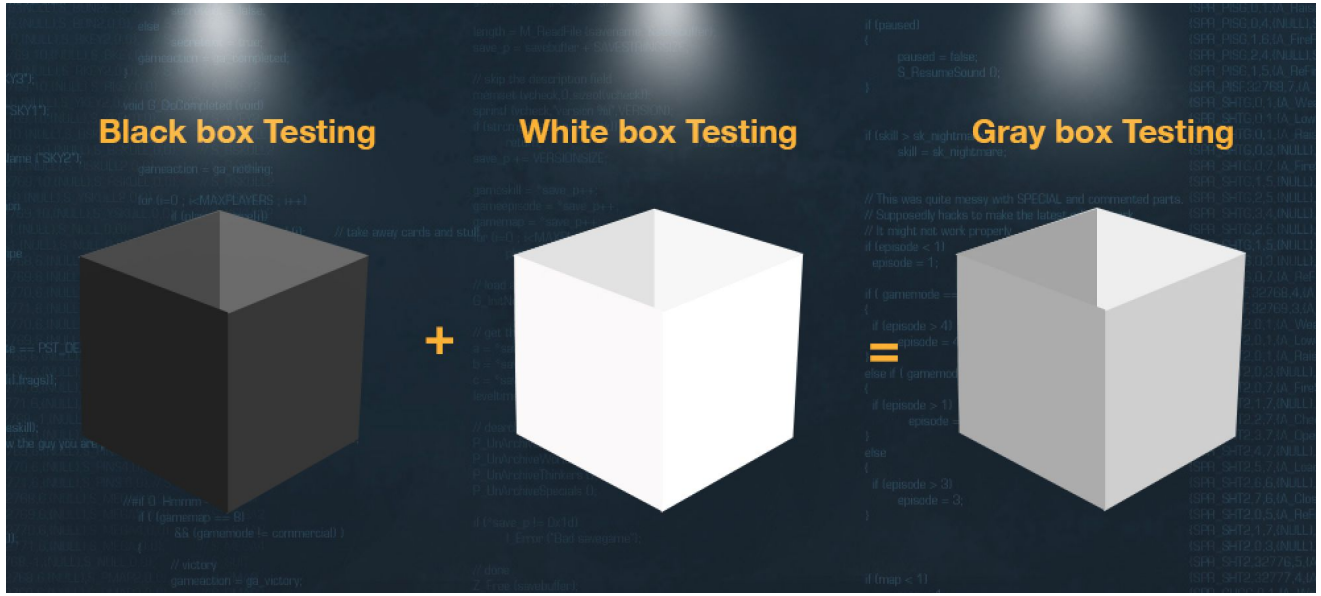
Tipos de **Testes de Software**

Existem diferentes tipos de testes que podem ser utilizados num software para identificar as suas principais falhas, tais como:

- **Testes *White-Box*:** servem para testar o código-fonte com o objetivo de avaliar o fluxo dos dados, os caminhos lógicos e ciclos, entre outros aspectos internos do software. É chamado de teste estrutural ou orientado à lógica.
- **Testes *Black-Box*:** avalia as funções externas que vão ser realizadas pelo sistema. Verifica se ao colocar determinados dados na aplicação, se obtêm as respostas esperadas. Não é analisado o código em si.

Tipos de **Testes de Software**

- **Testes *Gray-Box*:** é a união das duas técnicas de teste descritas anteriormente. Este teste avalia as funções externas do sistema ao mesmo tempo que testa a parte lógica, interna.



Tipos de **Testes de Software**

- ***Unit Tests* (testes unitários):** testam-se unidades menores do software, de modo isolado, para ver se todas funcionam adequadamente. Com este tipo de testes, é possível descobrir pequenas falhas, ainda que o software como um todo pareça funcionar correctamente.
- **Existem várias frameworks para auxiliar testes em Javascript:**
  - Mocha.js
  - Puppeteer
  - Jasmine

Exemplo Mocha.js

- É uma das mais populares frameworks de *testing* em Javascript.
- Podemos fazer asserções, verificando se o programa devolve ou não o que esperamos.
- Exemplo:

```
it("returns status 200", function(done) {
     request(url, function(error, response, body) {
          expect(response.statusCode).to.equal(200);
          done();
     });
});
```

Tipos de **Testes de Software**

- **Testes de integração:** depois das unidades testadas, devemos testar se elas funcionam juntas, integradas. Podem descobrir-se incompatibilidades ao funcionarem em conjunto, mesmo depois de cada uma ter passado o seu teste unitário.
- **Testes de carga:** é feito para avaliar os casos limite da utilização do software, como ele suporta um grande número de utilizadores em simultâneo, volume de tráfego, etc, sem que apresente erros.

Tipos de **Testes de Software**

- **Testes de usabilidade:** são feitos por um pequeno grupo de utilizadores (não técnicos e habitualmente de fora do projeto) para averiguar se o software satisfaz as suas necessidades e se é fácil e intuitivo de usar. Neste teste, analisa-se a forma como o utilizador usa o sistema, verificando onde ele tem mais dificuldade e as suas impressões.

Processo de Testes (Ciclo de Vida)

1. **Análise de Requisitos:** identificar os requisitos do que deve ser testado.

2. **Fase de Planeamento:** são definidos todos os planos de teste. É calculado o esforço e o custo estimados para o trabalho dos testes.

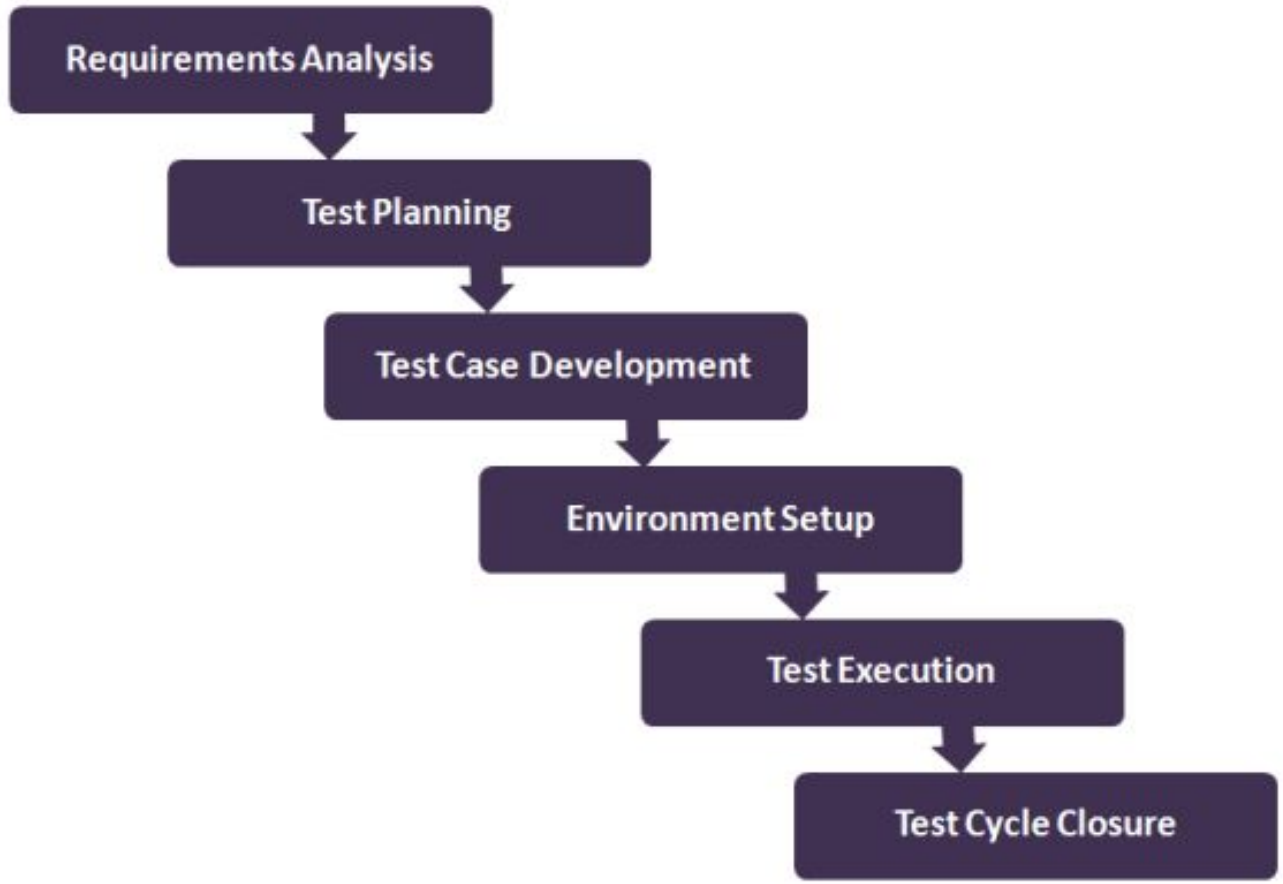
3. **Desenvolvimento de Casos de Teste:** os testes e scripts são elaborados, analisados e também são preparados os dados necessários para os testes.

Processo de Testes (Ciclo de Vida)

4. **Configuração do Ambiente de Teste:** decide as condições em que o software é testado. É uma atividade independente e pode ser iniciada junto com o desenvolvimento do caso de testes.

5. **Execução do Teste:** desenvolvidos os casos de teste e configurado o ambiente de teste, esta fase é iniciada e os testes são executados.

6. **Fecho de Teste:** o resultado dos testes é analisado.



A importância dos **Testes de Software**

Num projecto de desenvolvimento de software, os problemas podem surgir em qualquer uma das etapas do ciclo de vida. É quase garantido que o código final contenha erros, tanto nos requisitos, quanto de design ou funcionalidade.

Para identificar estas falhas antes que elas ocorram num ambiente crítico, é necessário testar muito bem o software e cumprir todas as etapas do processo de testes.

A importância dos **Testes de Software**

Alguns motivos que justificam a importância deste processo:

- Assegura que o software é confiável e garante a satisfação do cliente
- Garante a qualidade do produto, o que acaba por fidelizar o cliente
- Reduz os custos de manutenção
- É muito caro corrigir erros em fases mais avançadas de desenvolvimento de software

Test Driven Development

- Este é um paradigma de desenvolvimento de software no qual **os testes são escritos antes do desenvolvimento da aplicação**.

- Desta forma, as funcionalidades desenvolvidas precisam de "passar no teste" para serem validadas. Assim, **é a aplicação é que se vai adaptar ao que é esperado dela**.

- É um paradigma que apesar de já não ser tão utilizado como antigamente, continua a ser um exemplo interessante e que demonstra a grande importância dos testes.

Lint? Linter? O que é?

- Lint é um programa que executa uma verificação de código.
- Tem como objetivo procurar erros de programação e de boas práticas (estilo).

- É usado para:
  - Unificar o estilo de escrita num projeto.
  - Obrigar o uso de boas práticas
  - Melhorar a interpretação do código
  - Aumentar a performance do projeto

Lint? Linter? O que é?

- Exemplos de erros de **programação**:
  - Falta de “;” quando obrigatório
  - Variáveis declaradas e não usadas
  - Uso de variáveis antes de declaradas
  - Código depois de um *return*
  - Uso de imports descontinuados ou não seguros
  - Ciclos infinitos

Lint? Linter? O que é?

- Exemplos de erros de **estilo**:
  - Espaçamento indevido
  - Falta de identação
  - Ultrapassagem de limites de caracteres por linha
  - Uso misto de tipo de aspas (“ e ‘)
  - Detecção de classes muito longas

SonarLint

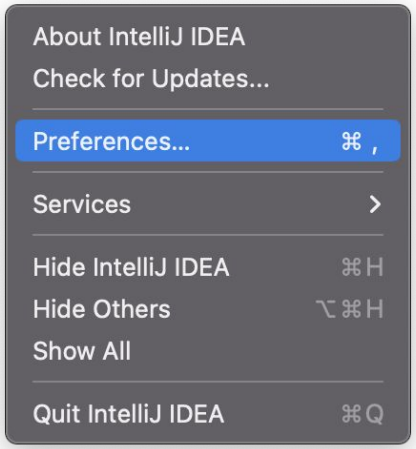
- É um plugin que contém uma ferramenta de Lint
- É usada para identificar erros no código durante o desenvolvimento

- O *SonarLint* é um projeto *openSource* desenvolvido pela *SonarSource*
- Está disponível para IDE's como o **IntelliJ**, Eclipse e VSCode

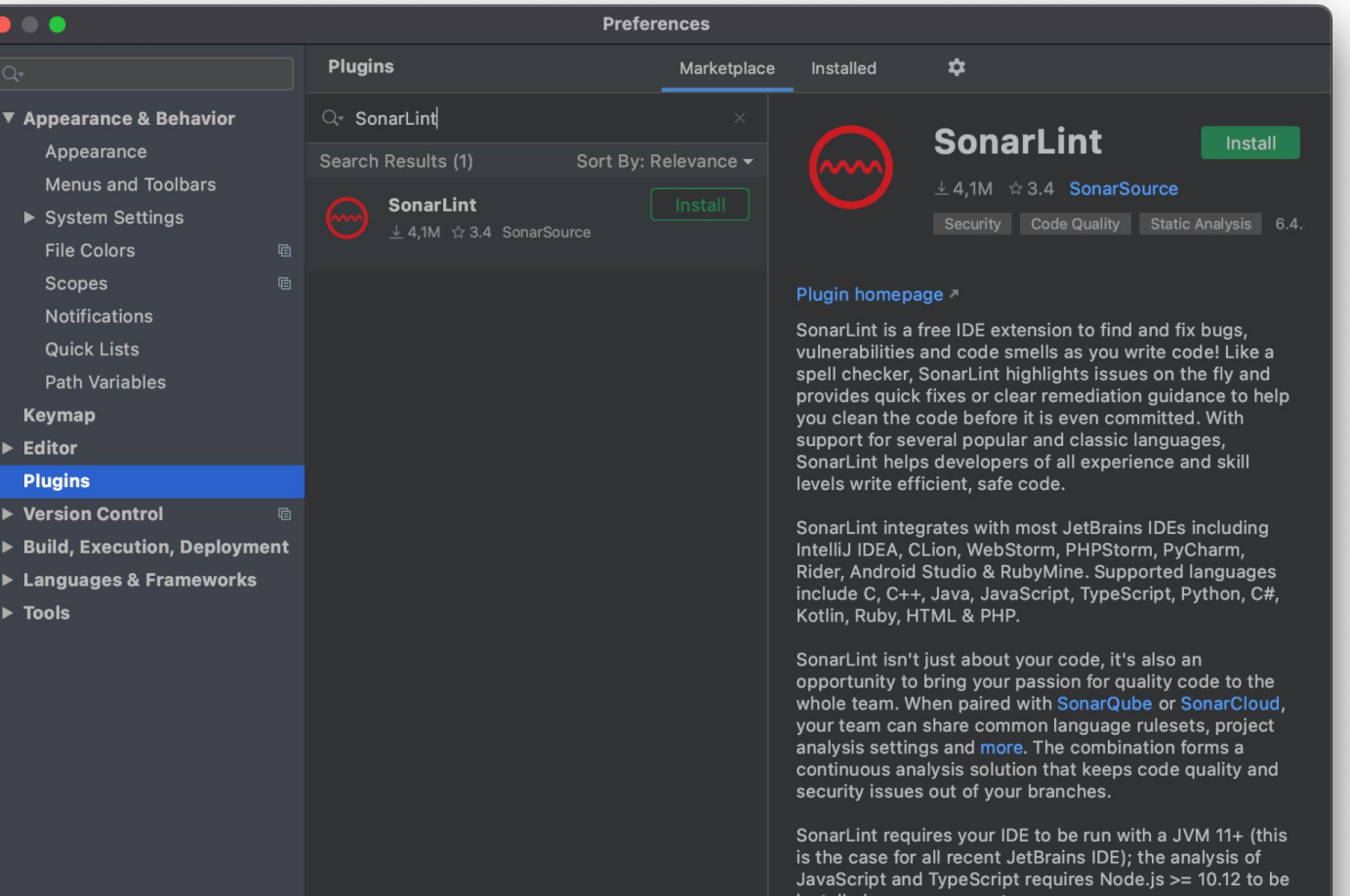
sonarlint

Instalar o SonarLint no IntelliJ

1. Abrir o IntelliJ
2. File > Settings (ou IntelliJ > Preferences no mac)



Instalar o SonarLint no IntelliJ

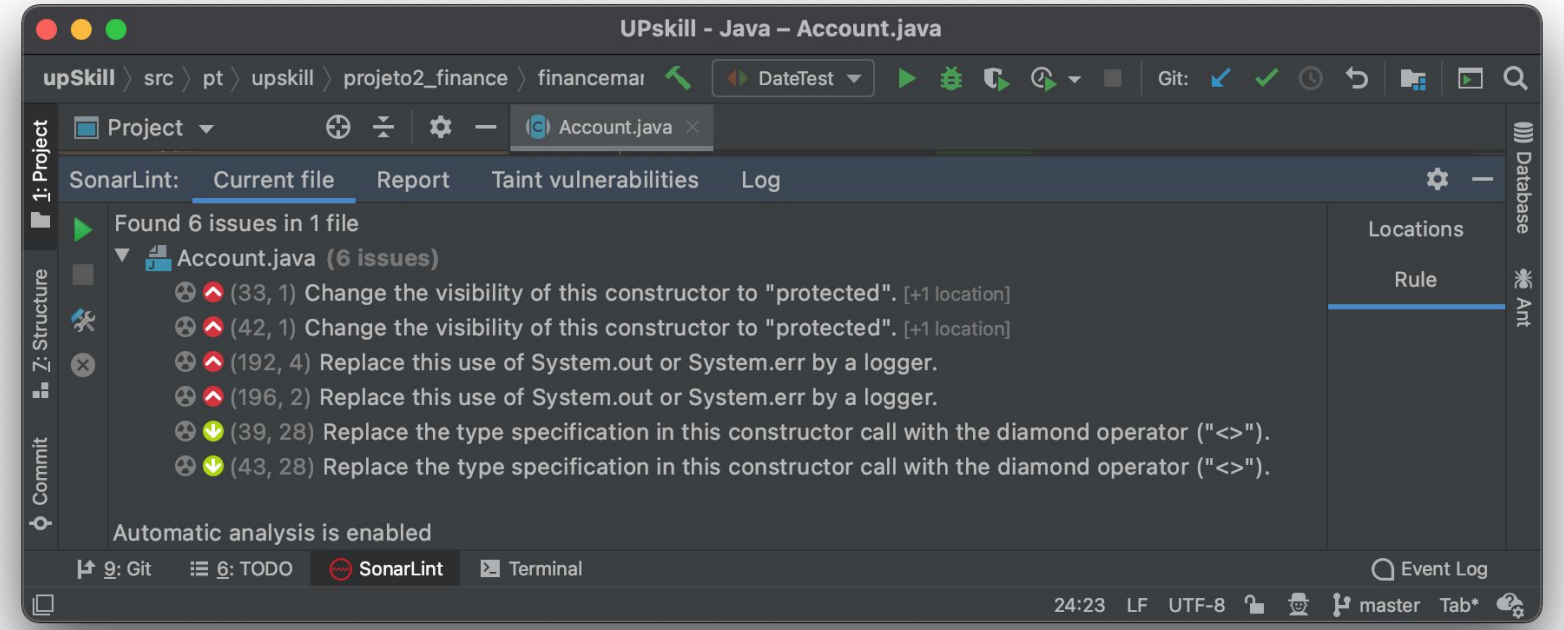


1. Selecionar Plugins
2. Selecionar Marketplace
3. Pesquisar por “SonarLint”
4. Instalar o SonarLint
5. Reiniciar o IntelliJ

iscte INSTITUTO UNIVERSITÁRIO DE LISBOA
emprego digital

Análise de código com **SonarLint**

1. Abrir o ficheiro a analisar
2. Clicar em *SonarLint*, na aba em baixo
3. Será gerado um relatório detalhado da análise, com sugestões de correcção.



iscte INSTITUTO UNIVERSITÁRIO DE LISBOA emprego digital

Documentação

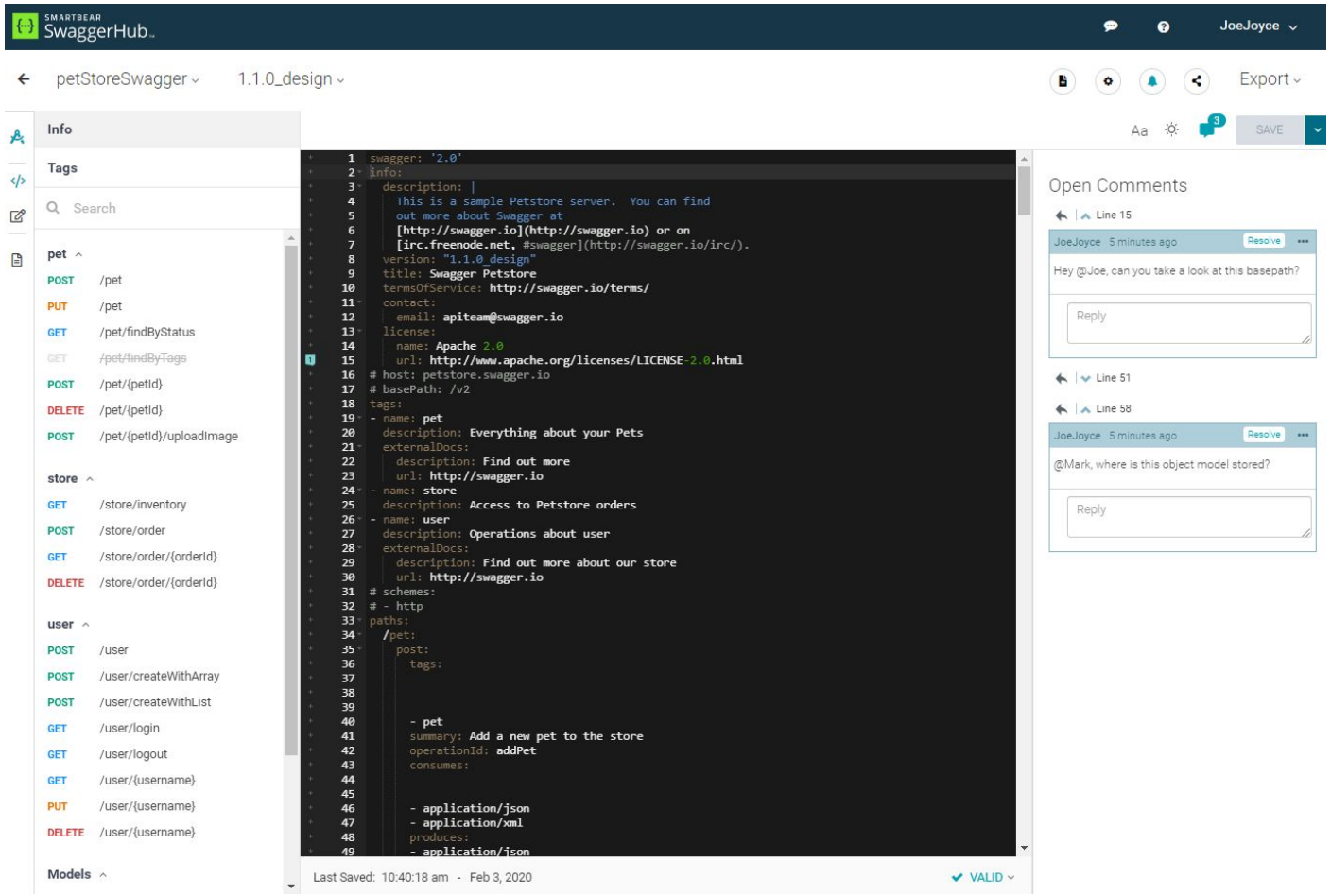
A documentação completa e correcta de um projeto de Software é fundamental para:

- Manter e melhorar o software existente
- Permitir gestão de equipas de desenvolvimento
- Partilhar código
- Grandes projetos
- Grandes equipas de desenvolvimento
- Desenvolvimento distribuído

Que documentação usamos?

- Foquemo-nos na importância de documentar ***APIs***.
- As APIs permitem à nossa aplicação (neste caso o backend) comunicar com o frontend ou com outras aplicações.
- Uma API bem documentada é da máxima importância para um projeto fácil de compreender.
- Devemos documentar todos os *endpoints*, o seu tipo (GET, POST...), o que recebem e o que devolvem.
- Existem várias ferramentas para criar documentação de APIs:
  - SwaggerHub
  - Postman (também permite testar)
  - Redocly

Exemplo: SwaggerHub



iscte INSTITUTO UNIVERSITÁRIO DE LISBOA

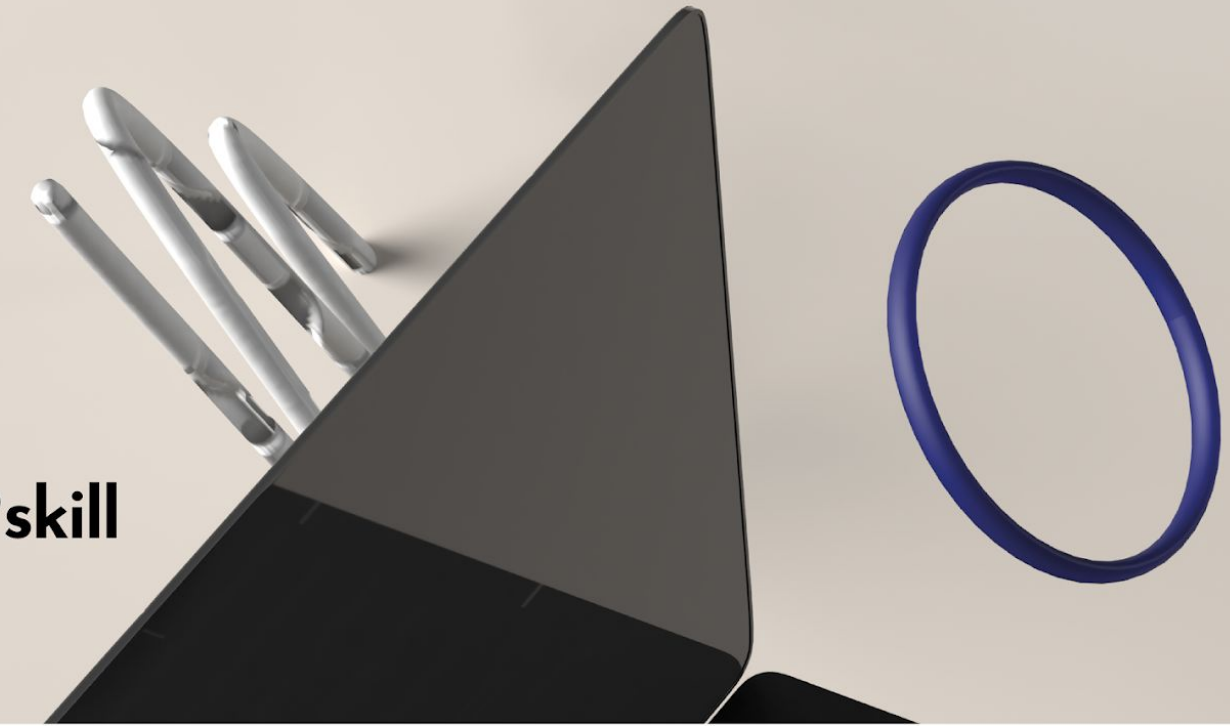
emprego digital

Exercício 1

Instale e configure o *SonarLint* no IntelliJ e corra o *Linter* no projeto do módulo 4. Observe o relatório produzido e analise as sugestões de melhoria apresentadas. São relevantes?

Não precisa de efetuar correcções ao projeto.

O futuro profissional começa aqui



iscte INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

UPskill