



dottech

# Manual do Desenvolvedor

## BrainLight

### **Equipa LGP 5A** **BrainLight**

#### **Developers**

André Pinheiro  
David Azevedo  
João Monteiro  
José Lima  
Luís Natividade  
Luís Pinto

#### **MM Delegates**

Nerea Castro  
Simão Pereira

#### **Designers**

Diana Magalhães  
Mariana Almeida

#### **Cliente**

INOVA+



## ÍNDICE

1. Introdução.....	3
BrainLight.....	3
2. Pré-requisitos.....	3
3. Software relevante .....	4
4. API .....	4
5. Processo de criação de módulos .....	4
5.1 Adicionar um dispositivo.....	4
5.2 Criar novas análises .....	5
5.3 Implementar novas funcionalidades .....	6
6. Processos de desenvolvimento incompletos .....	6
7. Problemas conhecidos .....	7



## 1. Introdução

A **BrainLight** é uma *framework* desenvolvida para computadores Windows. Este manual destina-se a programadores com conhecimentos de Java e, idealmente, familiaridade com os SDK dos dispositivos implementados ou de outros que pretendam implementar.

### BrainLight

A **BrainLight** é uma *framework* que lê os dados fornecidos pelos dispositivos EEG compatíveis (numa fase inicial, apenas o são o NeuroSky Mindset e Emotiv Epoc) e os disponibiliza num formato unificado através de uma API que pode ser utilizada por outras aplicações. Para além disso, também inclui uma interface gráfica para a visualização dessas informações, bem como funcionalidades adicionais como análises às ondas e gravação e leitura de um histórico.

## 2. Pré-requisitos

A **BrainLight** foi desenvolvida com a versão 8 para 32 bits do Java SE Development Kit (JDK8)<sup>1</sup>. Para o desenvolvimento do projeto é necessário incluir todos os módulos que a constituem: “Logic” (módulo de gestão e lógica), “Interface” (interface gráfica) e “Devices” (SDKs dos dispositivos). O módulo “Logic” encontra-se subdividido em “Analysis” (cálculos de análises às ondas), “History” (funcionalidades de histórico), “MainModule” (conector de todos os outros), “Neurosky” e “Emotiv” (que comunicam com os respetivos SDKs). As dependências entre módulos e *libraries* são as seguintes:

- Os módulos "Logic/Analysis" e "Logic/History" dependem do ficheiro "junit-4.12.jar" na pasta "Libs";
- O módulo "Logic/History" depende também dos ficheiros presentes na pasta "poi-3.14", também na pasta "Libs";
- Os módulos "Logic/Emotiv" e "Logic/Neurosky" precisam, respetivamente, dos SDKs presentes em "Devices/EmotivSDK" e "Devices/NeuroskySDK".
- Os módulos "Devices/EmotivSDK" e "Devices/NeuroskySDK" dependem dos ficheiros .dll e .jar que se encontram nas respetivas pastas.
- O módulo "Logic/MainModule" depende de todos os outros módulos presentes na pasta "Logic".

---

<sup>1</sup> Pode efetuar o *download* aqui: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

- O módulo "Interface" depende do "Logic/MainModule".
- Poderá ser exigido que sejam incluídas outras *libraries* comuns em alguns ficheiros (como `javafx`, `java.util.*`, etc); no entanto como são referidos nos imports de cada ficheiro, o IDE deve criar notificações e corrigir tudo automaticamente.

O desenvolvimento foi feito recorrendo aos IDEs IntelliJ IDEA e Eclipse em Windows e em Linux.

### 3. Software relevante

Para o desenvolvimento da **BrainLight** foram necessários os SDK dos dispositivos NeuroSky Mindset e Emotiv Epoc. Estes encontram-se no módulo "Devices", e são necessários para comunicar e ler a informação dos dispositivos.

### 4. API

A API encontra-se no ficheiro "API.jar", e deve ser incluída como dependência na aplicação externa que for criada. Aí, poderá chamar as funções que foram criadas para a **BrainLight**.


### 5. Processo de criação de módulos

#### 5.1 Adicionar um dispositivo

Comece por colocar na pasta "Devices" o conjunto dos ficheiros fornecidos pelo SDK do dispositivo. Depois, na pasta "Logic", crie um novo módulo com o nome do seu dispositivo.

Nesse módulo que criou terá de criar funções que vão depois comunicar com o "MainModule.java" por nós criado; se quiser pode consultar como exemplo os ficheiros criados para o Emotiv Epoc e Neurosky Mindset.

O MainModule neste momento recebe a informação de diferentes formas para cada dispositivo, uma vez que ambos têm especificidades incompatíveis. A função "initMerge" recebe



um `HashMap<String, Object>` para o Emotiv e um `HashMap<String, HashMap<String, Object>>` para o NeuroSky. Dependendo do dispositivo que estiver a tentar incluir, poderá usar uma dessas opções ou criar uma nova. Depois disso, o Emotiv e o NeuroSky utilizam as funções “initGetRaw” e “initGetWaves” respetivamente, de modo a ler cada envio de dados desses dispositivos.

O seu novo dispositivo terá de ser incluído também na interface! Para isso, terá de ir ao módulo “Interface”. Na classe Main, deverá incluir uma nova função de “launch”, igual às já construídas, e incluir também o nome do dispositivo no elemento ComboBox. Depois disso, aconselha-se que use o controlador principal, “MenuController.java”, na Interface, onde já está desenvolvida uma interface que pode ser usada por qualquer dispositivo. Se precisar de incluir especificidades que não se encontram nesse controlador e na vista correspondente, aconselha-se a criação de controladores e vistas específicos (veja como exemplos aqueles cujo nome contém “Neurosky” ou “Emotiv”).

## 5.2 Criar novas análises


As análises encontram-se no ficheiro “Calculations.java”, no módulo “Analysis”. Nesse módulo encontram-se dois outros ficheiros, “Tests.java” e “TypesOfCalculations.java”.

Para acrescentar uma análise é necessário começar por acrescentar um novo elemento ao enum “TypesOfCalculations”, no ficheiro homónimo, com um nome diferente dos usados até esse momento e que descreva a análise que está a ser implementada.

Depois disso, aconselha-se vivamente a criação de testes unitários no ficheiro “Tests.java”. Para isso podem-se seguir os exemplos dados: nos arrays “values” (para valores únicos) e “valuesXY” (para tuplos de valores) encontram-se já alguns exemplos de dados que poderão ser usados. Se for necessário poder-se-ão acrescentar novos elementos a esses arrays ou criar novos. No array TypesOfCalculations será preciso adicionar pelo menos um array no final com o novo elemento criado no enum anteriormente (mas poderão ser criados arrays com mais do que um tipo de cálculo).

Para fazer os testes, é preciso chamar o construtor da classe Calculations. Esse construtor encontra-se “overloaded”: há duas opções de o invocar. O primeiro argumento pode ser um array de floats (para a situação de haver apenas uma dimensão de variáveis) ou um array de arrays de floats (no caso de as variáveis serem bidimensionais). O segundo argumento é sempre um array de enums “TypesOfCalculations”. O programa vai calcular apenas as análises especificadas nesse(s) enum(s). Após invocar o construtor, basta chamar a função “getResult()” para obter uma HashMap de enums (iguais aos enviados) e o valor correspondente a cada um (em float).

Para adicionar a nova análise no ficheiro “Calculations”, basta acrescentar o novo enum no switch presente na função “calc()” dessa classe, bem como uma função nova que calcule essa nova análise e que seja incluída nesse switch. A média é calculada sempre automaticamente devido a ser usada extensivamente em grande parte das análises e o seu valor encontra-se na variável global “mean”.



De lembrar que as análises terão de ser acrescentadas também à interface. Para isso, é preciso alterar o ficheiro que termine em “\*View.fxml” que se pretende e o controlador (“\*Controller.java”) correspondente.

### 5.3 Implementar novas funcionalidades


Sugerimos que novas funcionalidades que queira implementar na aplicação sejam desenvolvidas dentro do módulo “Logic”, paralelamente ao histórico e análises já desenvolvidos. Depois de implementar uma nova funcionalidade deverá criar funções no MainModule que a utilizem, de modo a poder depois chamar essas funções na Interface.

## 6. Processos de desenvolvimento incompletos

Para o Emotiv, o *pane* de ações ficou por desenvolver. Estas informações estão a ser recebidas pelo módulo, mas não estão a ser transmitidas para a interface. Esta já se encontra também completa e pronta para mostrar essas informações. A razão para este ponto não ter sido implementado prende-se com a necessidade de criar um modo de calibração antes da sua leitura, o que pode ser desenvolvido utilizando funções fornecidas pelo SDK do Emotiv mas que o grupo não teve possibilidade de investigar e implementar.

O gráfico radar também não foi implementado. Para o desenvolver aconselha-se a consulta da componente do Paciente da **BrainStream**, uma vez que esta já contém todo o código necessário para desenvolver um modo de comunicação entre a aplicação Java e um servidor Firebase. Após isso basta modificar no ficheiro “MenuController.java”, na Interface, o URL que a WebView vai consultar para disponibilizar a sua informação. Para isto, basta consultar o ficheiro “BrainStream/page-patients/widgets/patient-chart.html”. Nesta página são implementados dois *widgets*: “page-charts-emotiv” para o Emotiv e “patient-chart” para o Neurosky. Terá de se criar um *widget* numa nova solução, podendo para isso ser utilizada uma cópia do ficheiro “page-patients.html”, bastando copiar o *widget* “page-charts-emotiv” e, na função “setupChart()”, alterar o campo “type” para ‘radar’.

As análises encontram-se praticamente implementadas, mas não tivemos tempo para as concluir. O módulo já está a efetuar todos os cálculos necessários na função “calculate” no “MainModule.java”, que recebe num array de arrays cada um dos dados necessários. Para testar basta alterar o parâmetro “finalWavesArray” para o array a ser testado (tal como descrito na linha 270 do ficheiro referido). No ficheiro de testes deve-se chamar a função “calculate” com um `int[][]`, cujos valores dos três arrays representam respectivamente: quais os sensores a ser lidos, quais os tipos de cálculo a ser realizados (cada número tem uma correspondência no “switch case” da linha 280), e finalmente quanto tempo o array vai ser lido e acumulado antes de ser enviado.



Poderiam ter sido implementadas outras funcionalidades, como a especificação da pasta em que se deseja guardar o histórico. Para isto basta alterar o caminho definido no código e torná-lo numa variável. Isso pode ser modificado no “MenuController.java”, função “fileChooser”.

## 7. Problemas conhecidos

Na fase final de desenvolvimento não é conhecido nenhum *bug* na **BrainLight**. Existem pequenas peculiaridades, referidas na resolução de problemas do manual de utilizador, mas nenhuma que se qualifique como *bug*.