

Bbriccs

gematik GmbH

Version 0.1.9, 2024-09-12

Inhaltsverzeichnis

1. Einleitung	1
1.1. Acronym	1
1.2. Motivation	1
2. Bricks-Module	3
2.1. Configuration-Bricks	3
2.2. FHIR-Bricks	4
2.2.1. FHIR-Validator	4
2.3. Perimeter-Bricks	5
2.3.1. Smartcards-Brick	5
2.3.2. Konnektor	8
2.4. RESTful-Bricks	9
2.5. CLI-Bricks	10
2.6. Utility-Bricks	11
Appendix A: Release Notes	12

1. Einleitung

B²ric²s (**briks**) offers reusable building bricks for flexible and maintainable test suites based on clean code principles.

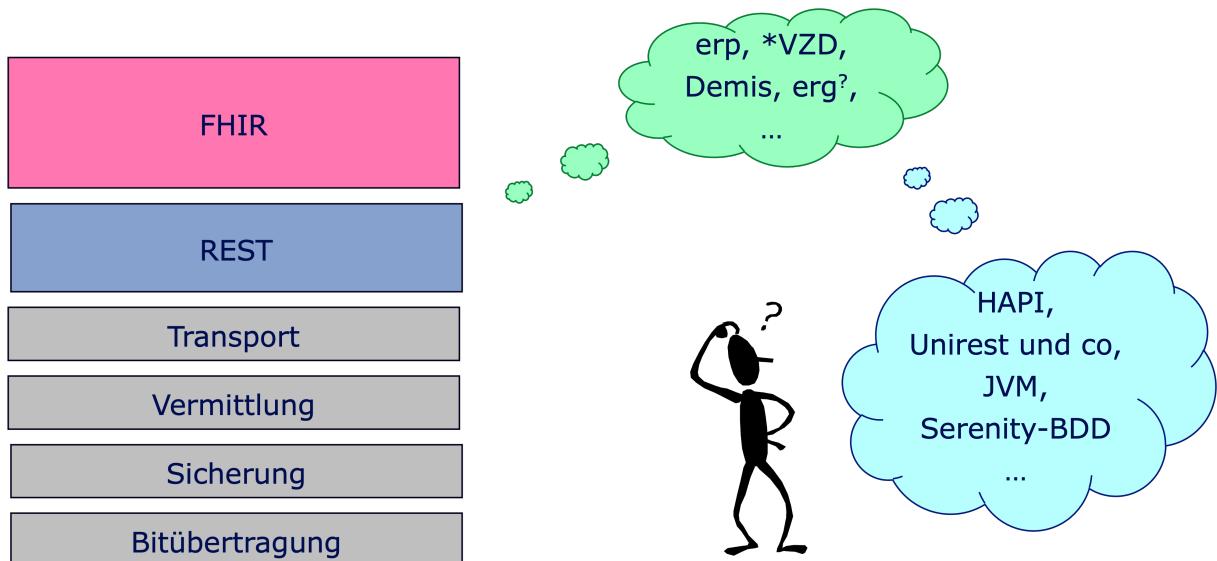
1.1. Acronym

While B²ric²s is merely the word mark and is rather unwieldy in everyday use, the following uses are recommended depending on the situation:

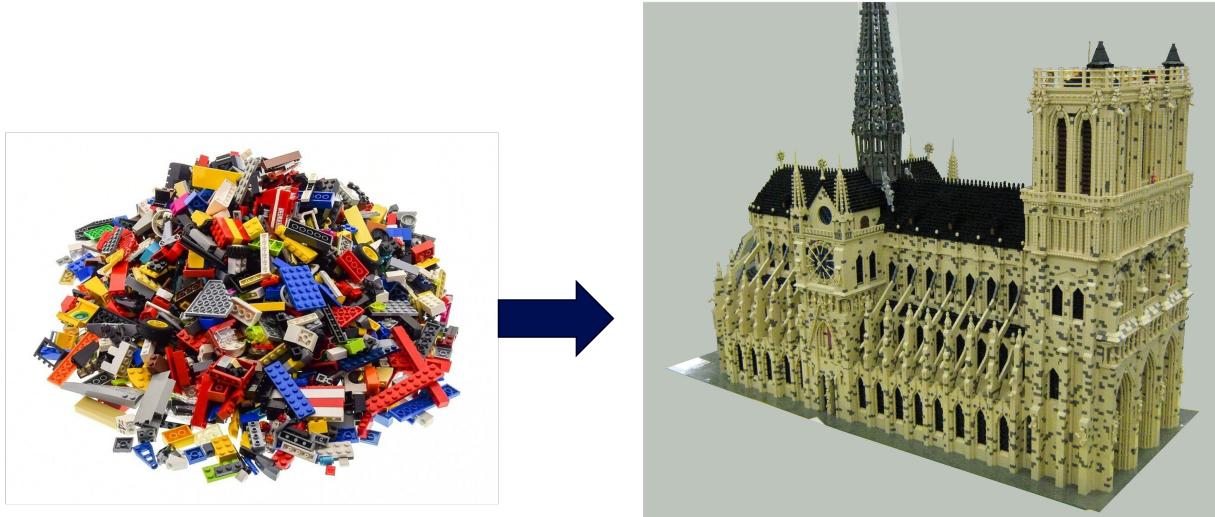
- Pronunciation: simply **briks** based on the idea clamping blocks
- Spelling: in the simplest case just **bricks** or **bbriccs** as a mnemonic for the word mark
- Code: the spelling **bbriccs** is only to be used for package names and namespaces, while **brick** (or **bricks** for plural) shall be used for variable, class or module names

1.2. Motivation

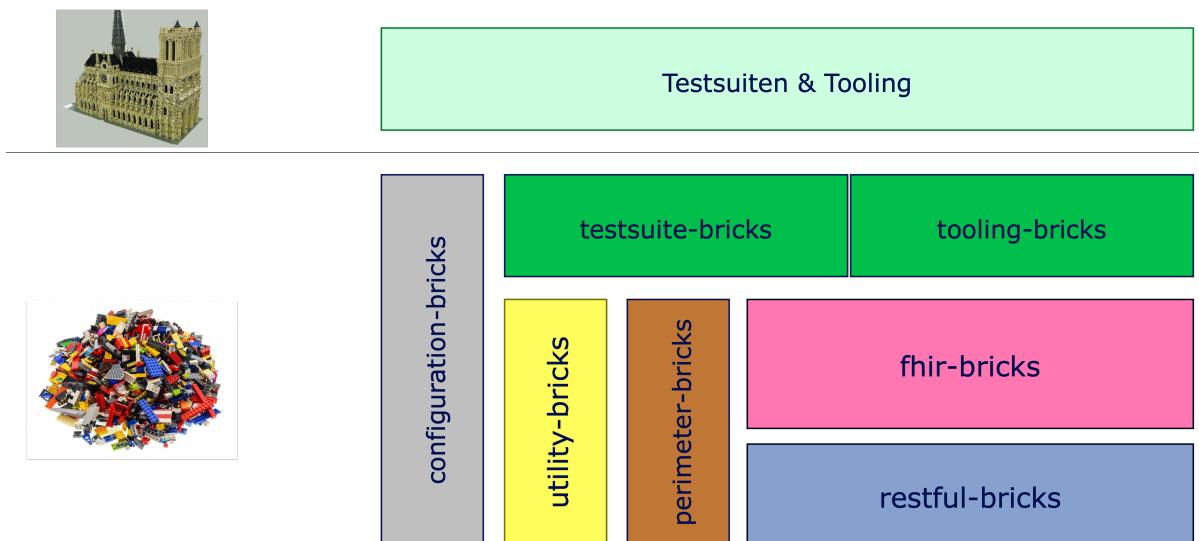
Within gematik GmbH, many of the products developed have a very similar technology stack.



B²ric²s provides a modular construction kit for the implementation of test suites and test tools, which simplifies the reuse of test code and the creation of test suites.



Thanks to the modular design, test suites can be assembled from existing modules and adapted to the respective requirements.



2. Bricks-Module

Nach dem Baukastenprinzip ist B²ric²s in mehrere separate Teile gruppiert...

2.1. Configuration-Bricks

TODO: Beschreibung what are Configuration-Bricks

2.2. FHIR-Bricks

Mit HL7 FHIR

TODO: Beschreibung what is FHIR-Bricks

```
<div class="imageblock"><div class="content"></div></div>
```

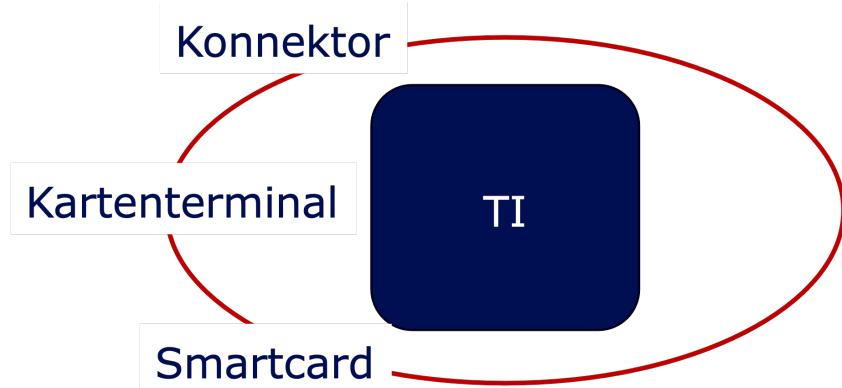
2.2.1. FHIR-Validator

Was ist der FHIR-Validator und wie funktionieren die Implementierungen...
TODO

```
<div class="imageblock"><div class="content"></div></div>
```

2.3. Perimeter-Bricks

Was sind die TI-Perimeter-Bricks... TODO



2.3.1. Smartcards-Brick

Smartcards mittels [smartcard-brick](#).... TODO

```
<div class="imageblock"><div class="content"></div></div>
```

Smartcards

Die Smartcards sind folgendermaßen aufgebaut:

```
<div class="imageblock"><div class="content"></div></div>
```

Smartcard-Archiv

Das [SmartcardArchive](#) ist ein Archiv, in dem die Smartcards verwaltet werden.

```
<div class="imageblock"><div class="content"></div></div>
```

Bevor man das [SmartcardArchive](#) nutzen kann, muss dieses zunächst mit den gewünschten Smartcards initialisiert werden. Hierfür hat man die Möglichkeit eigene Smartcards aus dem Dateisystem zu laden:

```
var smartcardsImage = new File("your/path/to/smartcards.json");
var sca = SmartcardArchive.from(smartcardsImage);
```

Alternativ ist es auch möglich, die Smartcard Images als Java-Resources (`resources/smartcards/smartcards.json`) zu speichern und diese dann folgendermaßen in das `SmartcardArchive` zu laden:

```
var sca = SmartcardArchive.fromResources();
```

Anschließend können die Smartcards aus dem `SmartcardArchive` anhand unterschiedlicher Kriterien geladen werden:

```
var egk0 = sca.getEgk(0);
var egk1 = sca.getEgkByICCSN("80276883110000113311");
var egk2 = sca.getEgkByVnNr("X110407071");

var hba0 = sca.getHba(0);
var hba1 = sca.getHbaByICCSN("80276001011699901501");

var smcb0 = sca.getSmcB(0);
var smcb1 = sca.getSmcBByICCSN("80276001011699900861");
```

Das `SmartcardArchive` bietet darüber hinaus auch die Möglichkeit eine Smartcard anhand des konkreten Typs und der ICCSN zu laden:

```
Egk egk1      = sca.getByICCSN(Egk.class, "80276883110000113311");
Smartcard egk2 = sca.getSmartcardByICCSN(SmartcardType.EGK, "80276883110000113311");

Hba hba1      = sca.getByICCSN(Hba.class, "80276001011699901501");
Smartcard hba2 = sca.getSmartcardByICCSN(SmartcardType.HBA, "80276001011699901501");

SmcB smcb0    = sca.getByICCSN(SmcB.class, "80276001011699900861");
Smartcard hba2 = sca.getSmartcardByICCSN(SmartcardType.SMC_B, "80276001011699900861");
```



Bei dem Aufruf über `getSmartcardByICCSN` wird die Smartcard als Basis-Klasse `Smartcard` zurückgegeben, weil hier der konkrete Typ der Smartcard syntaktisch nicht bekannt ist. Es sollen, wenn immer möglich, die konkreten und typisierten Methoden verwendet werden um die Typsicherheit zu bewahren.

Das `SmartcardArchive` kann natürlich aber nur Smartcards laden und herausgeben, die konfiguriert sind. Das bedeutet, dass die folgenden beiden Aufrufe zur Laufzeit zu Fehlern führen:

```
// exceeding index
assertThrows(IndexOutOfBoundsException.class, () -> sca.getEgk(100));

// unknown ICCSN
assertThrows(SmartcardNotFoundException.class, () -> sca.getEgkByICCSN("123"));
```



Grundsätzlich wird dringend davon abgeraten, das Laden der Smartcards über den Index als Standardmethode zu verwenden.

Beide Fehlerfälle können zur Laufzeit in etwa folgendermaßen behandelt werden:

```
// gives you the amount of known EGK smartcards
int idx = 100;
int amountEgks = sca.getConfigsFor(SmartcardType.EGK).size();
if (amountEgks <= idx) {
    // do something about it
}

// gives you an empty optional because ICCSN "123" is unknown
Optional<Smartcard> opt = sca.getByICCSN("123");
if (opt.isEmpty()) {
    // do something about it
}
```

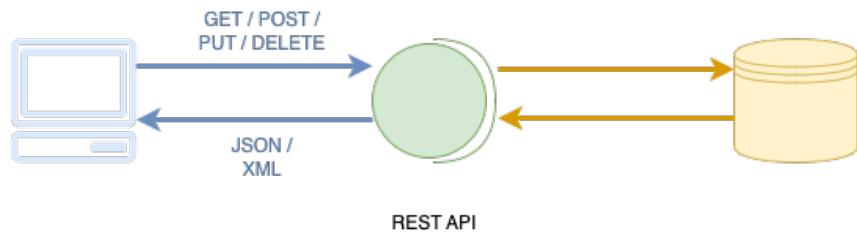
2.3.2. Konnektor

Der Konnektor-Client ist über mehrere Module aufgebaut... TODO

```
<div class="imageblock"><div class="content"></div></div>
```

2.4. RESTful-Bricks

Was sind die RESTful-Bricks?



```
<div class="imageblock"><div class="content"></div></div>
```

2.5. CLI-Bricks

TODO: Beschreibung what are CLI-Bricks

2.6. Utility-Bricks

TODO: Beschreibung what are Utility-Bricks

Appendix A: Release Notes

Release 0.1.8

- initial implementation of the core framework