FLIP+ Library

Tech Guide

Version 1 - 20251120

Services for the FLIP+ Platform

Lot 1 : IT development, maintenance and hosting
Lot 2 : Project coordination, management and monitoring

Summary

1 Project Architecture

The FLIP+ Library application is based on a monolithic architecture using the Symfony framework (PHP).

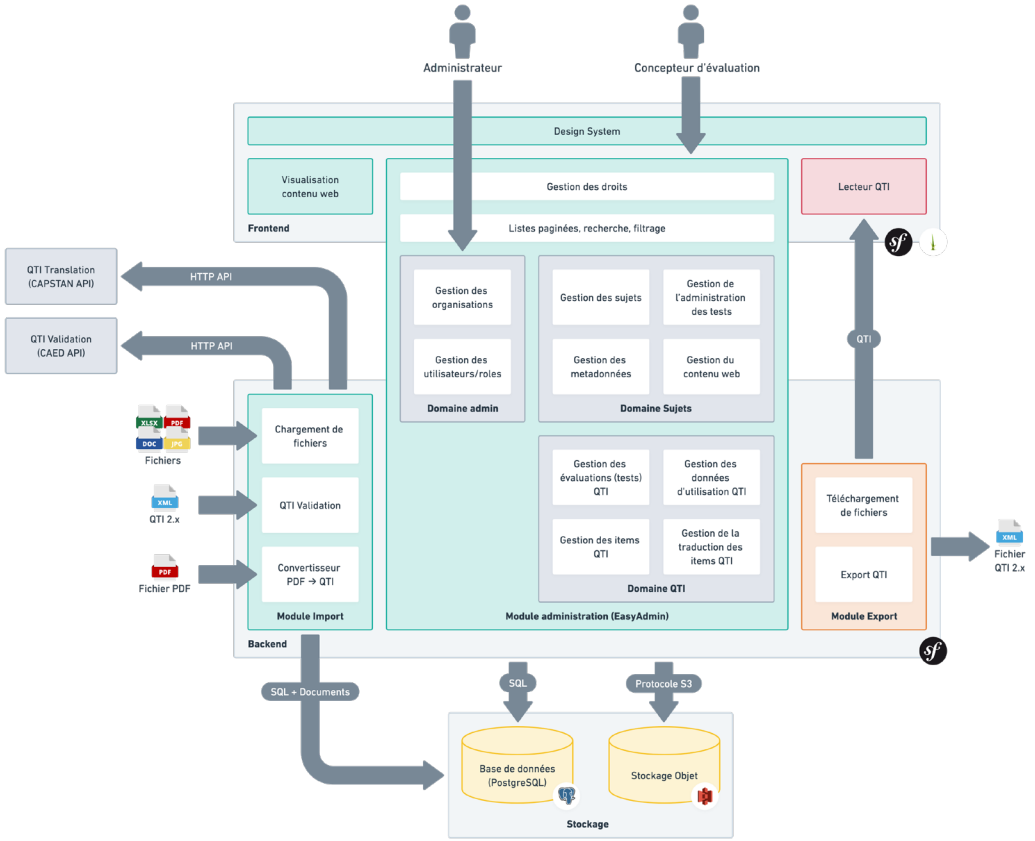
The user interface (Frontend) relies on a common design system and is implemented via Symfony's templating language (Twig). The javascript framework HTMX is used to enhance the user experience by adding interactivity to certain components.

The application is divided into three functional domains:

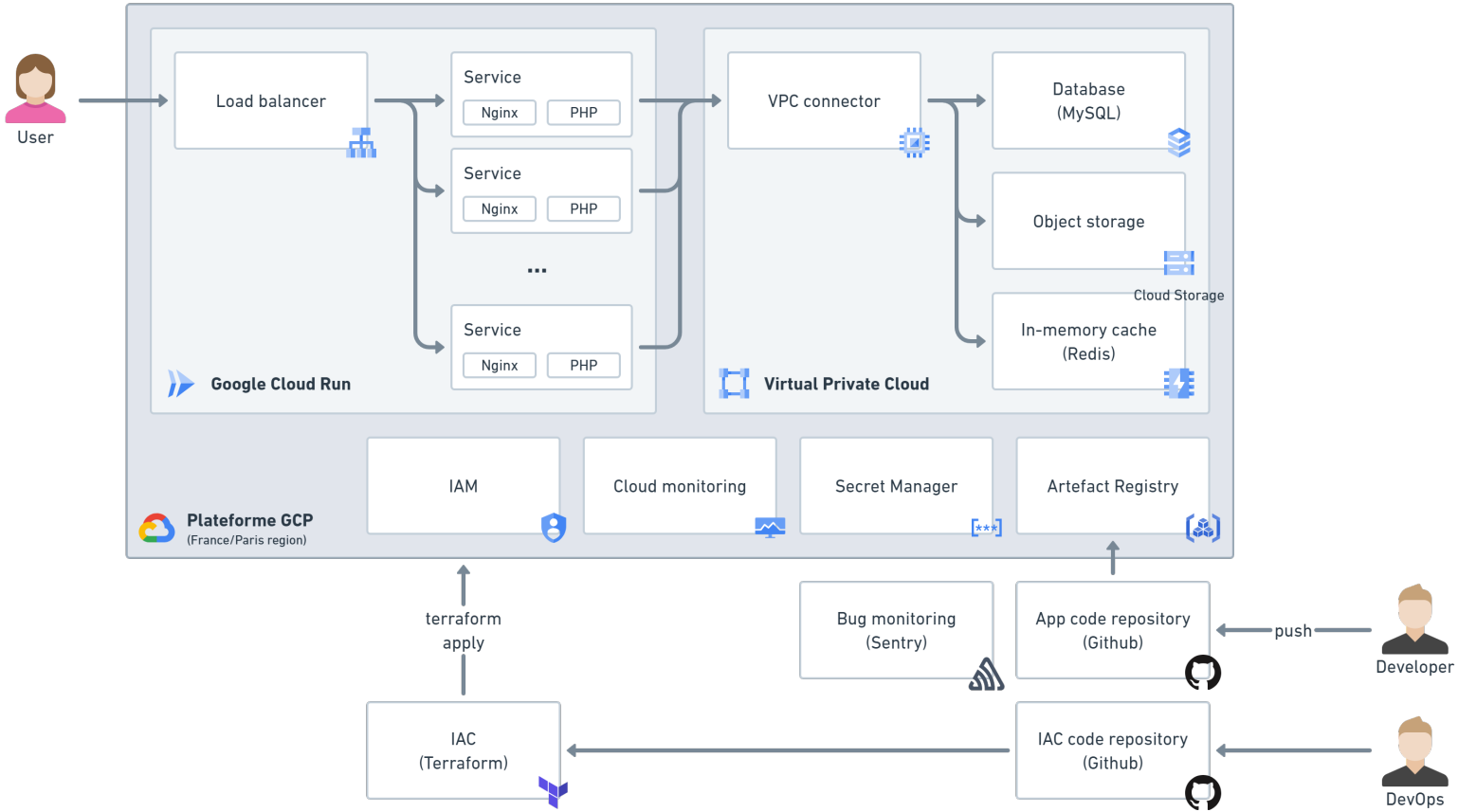
- **Admin domain** (management of users, roles and organisations)
- **Subjects domain** (management of subjects, metadata, web content and test administration)
- **QTI domain** (management of assessments, QTI items, usage data and translations)

The qtsim library is used to enable the import and export of QTI packages.
The data is then stored either in a PostgreSQL database (SQL data) or in object storage via the S3 protocol (files).

2 external APIs are used : CapStan API for QTI translation, and CAED API for efficient QTI format validation.



1 Infrastructure Architecture



2 Key features documentation

The FLIP+ Library application features include :

1. **Item Management** (Browse, search, filter and view assessment items with QTI content rendering, edit metadata, version tracking)
2. **Test Management** (Browse and search assessments containing multiple items, view test structure and items, edit metadata, PDF viewer for paper-based tests, version tracking)
3. **QTI Import** (Three-step wizard for importing IMS Content Packages, upload QTI ZIP packages with optional CSV metadata, automatic QTI v2 → v3 conversion, asynchronous background processing,)
4. **Paper-Based Import** (Manual import workflow for PDF assessments, PDF upload with visual item delimitation interface and coordinate-based item positioning, manual metadata entry)
5. **Translation Integration** (Automated multi-language translation via Capstan API)
6. **QTI Export** (Export individual items or complete tests as IMS Content Packages
7. **Admin Panel** (User and Organizations management, domain/subdomain taxonomy management , browse and restore archived content)

The **full documentation** for each of these features (Including detailed description, UML and sequence diagrams, and description of key components) is available in the code repository in the `docs/features/` directory (link)

3 Technologies used

Languages

- PHP
- HTML
- CSS
- Javascript

Frameworks

- Symfony
- Tailwind
- HTMX
- AlpineJS

Librarires

- Twig
- Qtsim
- citolab/qti-components
- EasyAdmin

Databases

- PostgreSQL
- Redis

Standards

- QTI (2.x & 3.x)

Infrastructure

(Cloud provider + managed services)

- GCP
- Cloud Run
- Cloud SQL
- Memory Store
- Artifact registry
- Cloud Monitoring
- IAM

CI/CD

- Github
- Github actions

4 ADR : Frontend technology

Problem :
We need to find a way to define a standard way to handle client side interaction in our frontend components. We're using HTMX, Twig and Symfony and we need guidelines in how to deal with client side interactions.
We're developing custom code in vanillaJS as if we were reinventing the wheel everytime. The code is not concise and does not follow HTMX guidelines (hypermedia scripting friendly).

Alternatives : Vanilla javascript, Jquery, Alpine, Hyperscript

Key points to consider

- Readiness: is the tool ready for production ? Does it have a stable version released ? Is it reliable ?
- Lightweight: what's the size of the minified bundle ?
- supports for native events & custom events (explicit)
- inline scripting: see hypermedia scripting (Consider inline scripting)
- Conciseness / Verbosity: how many lines of code do we have to write to develop a dropdown for example ?
- Community: is there any resources available ? is there many questions/answers on stackoverflow ? is there any reddit chan' ?
- Toolchain: is it difficult to integrate it as is as symfony asset ? do we need a builder or some other things ?
- Lifecycle: is it still active ? should we consider investing on it without being locked on a legacy ?
- formation : How much time should we invest in order to learn ? As a dev, am I Ok to invest some time on it ? is it dead ?

	VanillaJS	JQuery	AlpineJS	Hyperscript
Readyness (production)	✅	✅	✅	🚫
lightweight	✅	++	+++	++++
supports for native/custom events	✅	✅	++	✅
inline scripting	🚫	🚫	✅	✅
Conciseness/verbosity	🚫	++	+	🚫
Community	✅	✅	++	🚫
Toolchain	✅	✅	✅	✅
Lifecycle (not dead)	✅	🚫	✅	✅
Formation	✅	🚫	✅	?

Recommended option

4 ADR : PDF management (viewer + annotations)

Problem :
We need to store items and tests in PDF format, and find them in the library. During import, we want to add metadatas and statistics for each item on the PDF.
We need to visualize a PDF, annotate it (select a portion of it with coordinates), and when confirming the selection, open a new div on the side

Alternatives :
PDF.js
Pdfjs_express
Nutrient.io

Key points to consider
- Easy to integrate: js package to use with a script tag
- Display the PDF: we want to visualize the PDF
- Annotation: we should be able to select a portion of the PDF, either by creating a rectangle, or moving lines along the PDF
- coordinates SDK: once selected, we should be able to retrieve the coordinates of the selected portion to open a new div
- Price: not a blocker, but better if less expensive

	PDF.js	PDF.js + html events	pdfjs.express	Nutrient.io
Integration	⭐⭐⭐ pkg bundle	⭐⭐⭐ pkg bundle	⭐⭐ manual integration	⭐⭐⭐ npm compatible package
Display	⭐⭐⭐ example	⭐⭐⭐ example	⭐⭐⭐ viewer	⭐⭐⭐ viewer
Annotation	⭐ only highlight and draw	⭐⭐⭐ listen to mouse events	⭐⭐⭐ tool: Rectangle annotation	⭐⭐⭐ - hide reveal area - rectangle annotation > bounding box
Coordinates SDK	⭐ No documentation	⭐⭐⭐ js to compute scaled coordinates	⭐⭐⭐ coordinates interactivity	⭐⭐⭐ listen to created annotations
Price	⭐⭐⭐ Free	⭐⭐⭐ Free	⭐ viewer: free ; premium $600/mois	??? 30 days free trial

Recommended option

4 ADR : QTI Import performance

Problem :

Our PHP application processes hundreds of small QTI XML files and validates each against a complex XSD schema using `DOMDocument::schemaValidate()`. Each file takes ~3 seconds to validate, leading to poor performance even in background jobs. We are evaluating ways to optimize this process to reduce total wait times while maintaining reliability.

Alternatives :

- Use Local XSDs with XML_CATALOG_FILES
- Parallelize using PHP multi-threading or worker pools
- Use an external XML validation API
- Disable validation altogether

Key points to consider

- **Performance**: Time saved per validation and scalability
- **Reliability**: Consistency of results and error detection
- **Complexity**: Implementation difficulty and maintenance
- **Security**: Risk of malformed or malicious XML
- **Standards Compliance**: Ability to guarantee QTI conformance

Options	1. Local XSDs (XML_CATALOG_FILES)	2. PHP Parallelism	3. External API	4. Disable Validation
Performance	⭐⭐⭐⭐ Eliminates network fetches and reduces per-file time	⭐⭐⭐⭐⭐ Significant batch-time reduction via concurrency	⭐⭐⭐⭐ Offloads CPU but adds network latency	⭐⭐⭐⭐⭐ Instant processing
Reliability	⭐⭐⭐⭐⭐ Same validation engine, full fidelity	⭐⭐⭐⭐ Validation itself is unchanged	⭐⭐⭐⭐ If using a robust engine	⭐⭐ Errors might go unnoticed
Complexity	⭐⭐⭐ Requires catalog setup and testing	⭐ Requires managing threads or multiple workers	⭐⭐ Requires API integration and fallback	⭐⭐⭐⭐ Simplifies pipeline
Security	⭐⭐⭐⭐ No new surface area	⭐⭐⭐ Increased complexity needs monitoring	⭐⭐ Depends on trust and network exposure	⭐⭐ Risk of invalid/malicious input
Standards Compliance	⭐⭐⭐⭐ No new surface area	⭐⭐⭐⭐⭐ Full validation	⭐⭐⭐⭐ Assuming the external tool is QTI-compliant	⭐ No QTI compliance guarantee

5 Installation Guide (1/2)

1. Clone the repository

```
$ git clone git@github.com:Theodo-GovTech/BII.git
```

2. Start the Docker images

```
$ make start
```

3. Install the dependencies

```
$ make composer c='install'
```

4. Create a .env.local file to change the environment variable values locally

```
DATABASE_URL="postgresql://app:password@database:5432/app?serverVersion=16&charset=utf8"
REDIS_AUTH_STRING=redis://iil-redis:${REDIS_PORT}
MAILER_DSN=smtp://change:me@live.smtp.mailtrap.io:587/?encryption=ssl&auth_mode=login
CAPSTAN_API_URL="dummy"
CAPSTAN_USER_PASSWORD="dummy"
IS_TRANSLATION_MODULE_ENABLED=false
IIL_BASE_URL="https://staging.itemlibrary.org"
```

5 Installation Guide (2/2)

5 Run the migrations

```
$ make drop-migrate
```

6 Start the Tailwind worker

```
$ make tailwind-build-worker
```

7 To check that the site is running properly, try accessing it via the URL http://localhost:8000
It is also possible to run the following command to ensure everything is working correctly:

```
$ make test
```

```
<gitmoji>(type) titre du commit

Description du commit.
```

6 Contribution Guide (1/2)

To get started, please refer to the README.md file which contains detailed instructions for running the project locally.

Creating a ticket

When creating a ticket, please provide the following information:

- **Title:** A concise and descriptive title.
- **Description**: A detailed explanation of the issue, with context or screenshots if necessary.
- **Description:** A detailed explanation of the issue, with context or screenshots if necessary.
- **Expected vs actual behaviour:** Describe what you expected and what actually happened.
- **Labels:** Add relevant labels (e.g. bug, feature request, documentation).

Commit message format

All commit messages must follow this format:

```
<gitmoji>(type) titre du commit

Description du commit.
```

- <gitmoji> : Use a gitmoji that represents the purpose of the commit (e.g. ✨ for a new feature, 🔥 for a removal). Full list available ici
- (type) : Type of change (backend, frontend, CI, docker, etc.).
- titre : A short and descriptive title (*)
- **Empty line after the title**
- description : Additional details about the changes (**)

6 Contribution Guide (2/2)

Pull Requests :

Add a description explaining the purpose of the pull request to help reviewers understand the context.

Include screenshots or a short video if it helps to visualise the changes.

Don’t forget to:

- Check the commit message format
- run linters/formatters: make ecs && make rector && make phpstan
- run the tests: make test

Once the tests have passed, you can request a review from the project maintenance team.

Code Style :

Please respect the project’s code style. Use the available linting tools to ensure code cleanliness.

Tests :

All new features or fixes must be accompanied by tests. Run the test suite before pushing your changes to avoid regressions.

7 Known Limitations

- **QTI Player**

The QTI player currently lacks support for the PCI interaction types from the QTI standard (Custom interactions)
This interactions will be added in a future update.

- **Import performance**

QTI package import can be slow (3 to 5 minutes) in certain cases, especially if there are a lot of items in the package. This limitation will be addressed in a future update.

- **Localisation**

Currently, the application layout is only translated in English.
QTI packages can be imported in any languages, but only packages from these languages will be automatically translated in English :

- French
- Norwegian
- Brazilian Portuguese
- Lithuanian
- Italian

8 Additional resources and support

Need help?

If you experience any issues or have questions about how to use the platform, please contact your organisation’s administrator for support.

For general questions about the FLIP+ Library project, you can reach out to:
info@flip-plus.org

For technical questions about the FLIP+ Library platform, you can reach out to:
contact@itemlibrary.org