Technical Architecture Document

Division	**Chanel F&B**
Project	**Beauty Plan**

Classification:
☐ Public document (C1)
☒ Internal document (C2)
☐ Confidential document (C3)
☐ Highly confidential document (C4)

Document update history

This document was created and updated by the following people: The version management system is the one described in the standard configuration management process, in the document management section. The update rules (revisions) are described in the introduction of this document.

Version	Version date	Nature of the modification	Editor
1.0	2023/01/23	Document creation	s.loison
V2	2023/02/03		
V3	2023/02/08	Add APIM to intermediate APIs	s.loison
V4	2023/02/13	Update JWT handling sequences	s.loison

Proofreading and validations

This document has been reviewed and validated by the following individuals (peer reviews or re-reads):

Version	Proofreader	Responsibility	Date of validation

Approvals

Version	Approver	Responsibility	Date of validation

Distribution list

This document will be distributed to the following people:

Recipient	Responsibility	Distribution date

Summary

1. Referenced documents

This section lists all the documents referenced in the architecture file.

Name	Title	Owner
Backlog	230207_BP MVP_Scoping_Stream backlog_V4.4.pptx (without last requirements received post February 3rd,2023)	27/01 by Anne-Laure F. (Niji)
Development Guidelines	CODE+GUIDELINES	By Chanel
Content Partner Guidelines	RFP Appendix 8 - Chanel.com Content Partner Guidelines	By Chanel
Performance Guidelines - editorial content	RFP Appendix 10 - Chanel.com Performance Guidelines - editorial content	By Chanel
Wishlist SDK	ONEECO-EcommercejavascriptSDK-160123-1609	By Chanel

2. Introduction

2.1. The objectives of this document

This document presents the functional, application and technical architectures of the Beauty Plan project.

This document is not intended to replace the following documents:

- Expression of Requirement documents,
- Other documents related to the General Design Phase such as:
  - General Functional Specifications (GFS),
- Other documents related to the Delivery Phase such as:
  - Installation procedure,
  - The operating file.

2.2. The revisions

2.2.1. When the document should be revised?

This document must be revised at each technical evolution and major functional evolution of the project.

2.2.2. How to proceed to revise the document

The validation and approval procedures are the same for a major modification as for the initial architectural file.

1. Archive the previous version [version -1].

2. Make changes in the document [version +1] in consultation with the concerned stakeholders to obtain their commitment. (Don't forget to update in the document [version +1] the history of changes at the beginning of the document, and to update the footer).
3. Conduct a review of the changes
4. Have the changes approved

2.2.3. What types of changes have an impact on the document?

Any event that impacts the architecture will trigger a revision of this document.

The following events will trigger a revision of the architecture document:

- Addition of a new functional module (Internal or from a partnership)
- Modification of a technical brick (DBMS, Web, PHP, Javascript, Java ...)
- Addition of a new data flow

2.3. Terminology and acronyms

Term / Acronym	Definition
DCT	**Data Capture Tool**
APIM	**Api Manager**
CTO	**Chanel Try On service**

2.4. Versioning

Version	Date	Scope
V1.0		
V2.0	2023/02/02	Integration of APIM in between BP Back for front and DCT Use Gigya Token for internal Beauty Plan authentication
V3.0	2023/02/08	Use APIM to request Salesforce and Amazon S3 Use Gigya Token from Gigya API

3. Input elements structuring the target architecture

3.1. Context, stakes, and objectives

3.2. Guiding principles

This section lists all the high-level requirements that the target system must meet.

Type	Principe	Description	Source
SI	Performance	The application must have state of the art performance (see Performance Guidelines - editorial content)	Client

SI	Architecture	The architecture is backend for frontend. The Chanel IS is not directly accessible from the client browser.	Client

3.3. Non-functional requirements

ID	Category	Description
EXNF_001	Technical Stack	The technical stack is based on javascript: NodeJs / ReactJS
EXNF_002	DevOps	The continuous integration and deployment chain is based on Azure DevOps

3.4. Constraints

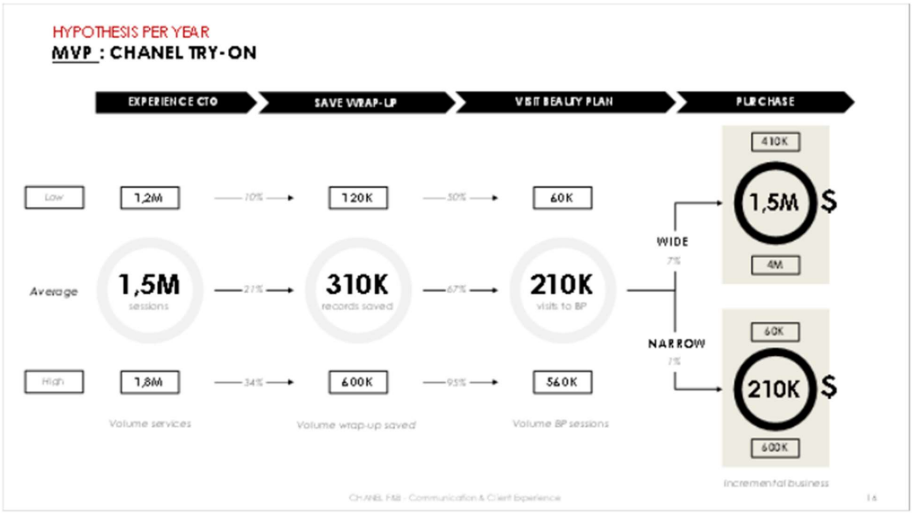
This section describes all the technical constraints on the solution.

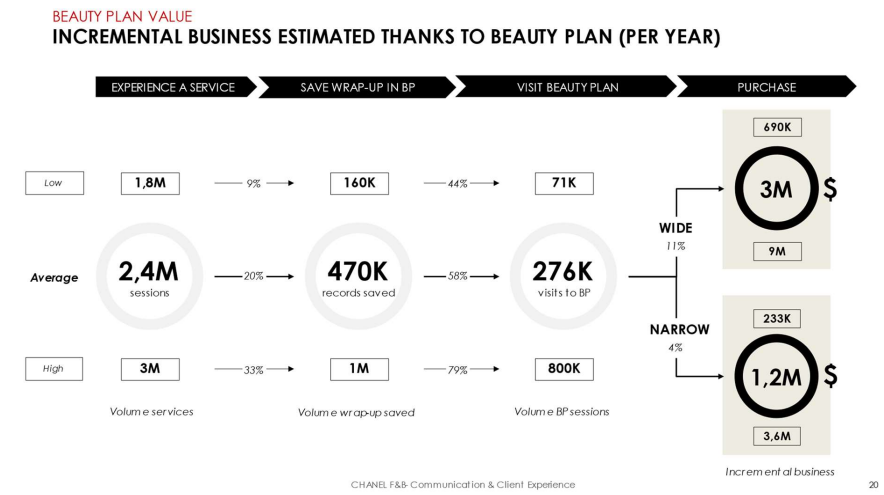
ID	Category	Description

3.5. Assumptions

This section describes the technical assumptions made during the development of this architecture package to address a necessary technical solution element not covered by a non-functional requirement.

ID	Category	Description
VOL	Volumetry	Estimated volumetry is presented below. Architecture has been designed to handle these requirements.





4. Functional architecture

4.1. Actors

A Chanel Client: user of the platform (B2C).

4.2. Use cases

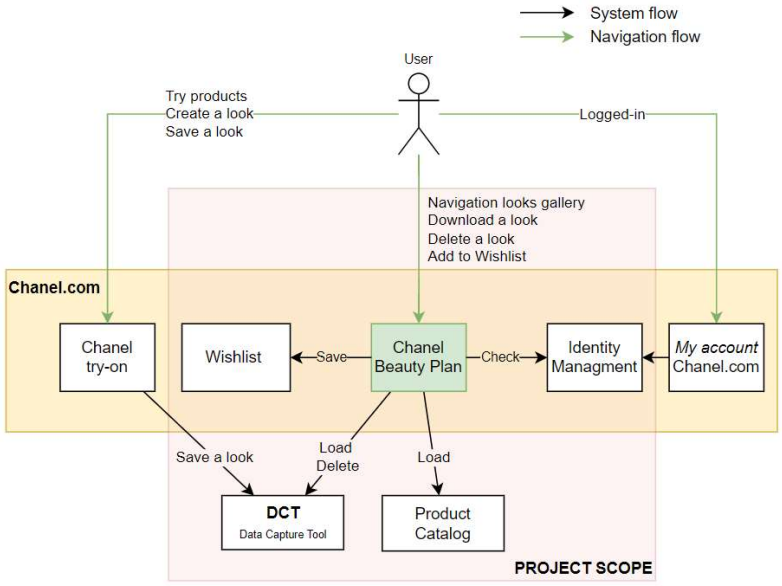
Prerequisites:

- Chanel.com: Login into the client account on chanel.com,
- Try-on:
  - Product testing at MVP:
    - Lipstick,
    - Eyeshadow,
    - Eyeliner.
  - Creation of a look containing a minimum of 1 product and a maximum of 3,
  - Saving the look:
    - Photo of the look, title, date,
    - Products.

The use cases of the Beauty Plan application for the Chanel customer:

- Browse the gallery of looks,
- Downloading the photo of the look,
- Delete look,
- Add to Wishlist.

4.3. Functional architecture scheme



- DCT / Data Capture Tool: allows you to record and display customer data related to the creation of their Look (Customer, Service, Product).
- Product catalog: repository of Chanel products that can be included in a customer's look.
- Wishlist management: service allowing users to save products before purchasing.

- Identity management: service allowing users to authenticate themselves on the chanel.com website

4.4. Production of data

The Beauty Plan platform produces "looks" representing a makeup session: a photo, a list of products.

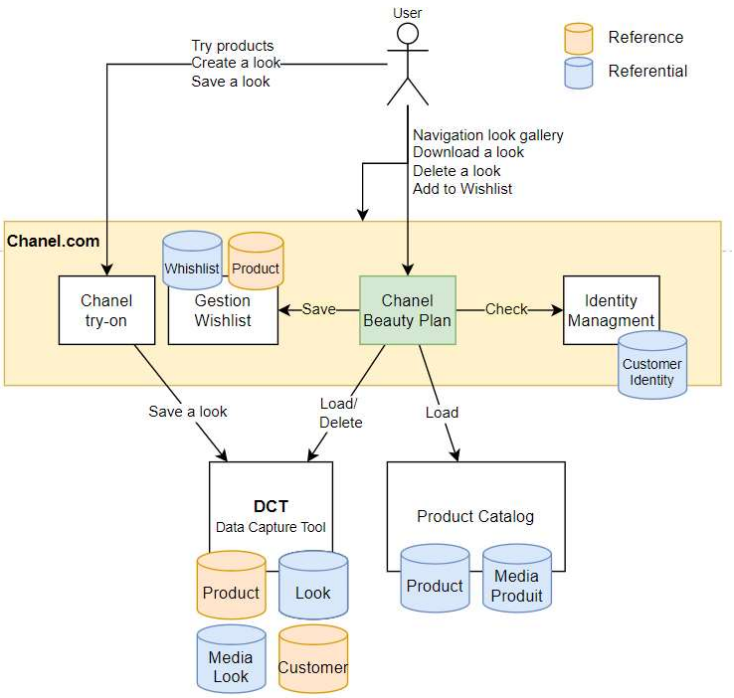
Technically, the platform produces activity logs.

4.5. Data repositories

The Beauty Plan platform is not a repository of any data.

It uses the referential data of the following external applications:

- Identity management: Client identity,
- Data Capture Tool (DCT):
  - Customer (Customer Reference),
  - Service (Look),
  - Product (Product Reference),
  - Media associated with the Look (Photo),
- Product Catalog:
  - Product,
  - Product media,
- Wishlist management
  - Product (Reference),
  - Wishlist.



4.6. Mass treatments

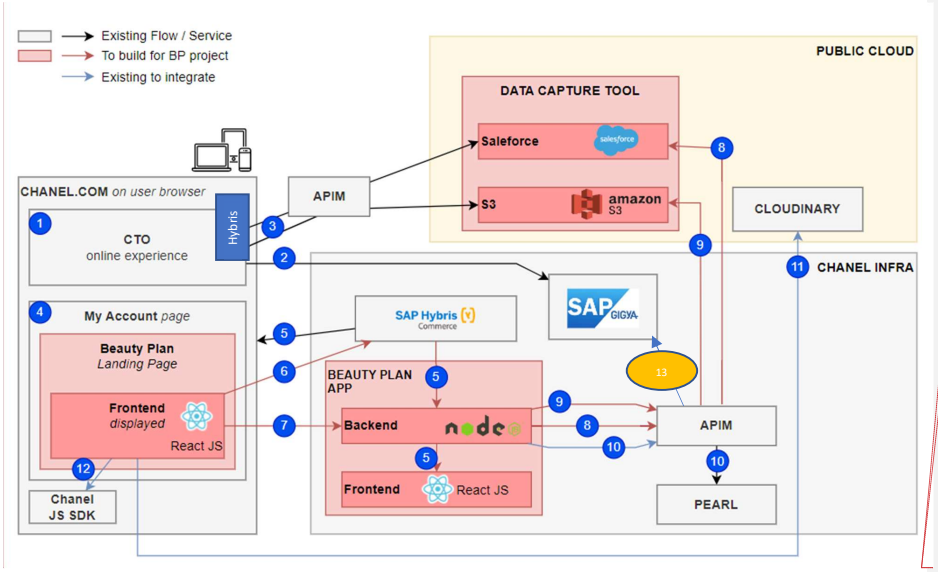
This section describes the significant mass treatments of the solution.

è As of today, no mass treatment is planned.

Name of the task	Manipulated volumes and expected results	Frequency and duration	Prerequisites	Impacts on the transactional

5. Software architecture

5.1. Logical Architecture Diagram



Commenté [CDM(1]: Flow #13 : APIM calls Gigya to verify the JWT sent by systems calling the DCT. This verification is not done for the calls done by Hybriss (a specific policy will be implemented in APIM to allow this)

Commenté [CDM(2]: Flows # 3 are calls from Hybris, not from the customer browser

Use Case 1: Try and save a look

1 Sign-in on chanel.com and use CTO,

2 Authenticate the user to chanel.com,

3.1 Post of the customer ID in Salesforce through APIM,

3.2 Post of the service ID in Salesforce through APIM,

3.3 Post of the product ID in Salesforce through APIM,

3.4 Post of customer picture in Amazon S3 through APIM.

Use Case 2: Retrieve a look

4 Access Beauty Plan Landing page (as a logged in user of chanel.com),

5 Display BP content using an HTML connector,

6 Retrieve Gigya ID through Hybris,

7 Intermediate for the front (from 8 to 10 flows),

8 Get look details from Salesforce,

9 Get look image from Amazon S3,

10 Get products details from PEARL,

11 Get products image from Cloudinary,

12 Interact with Wishlist.

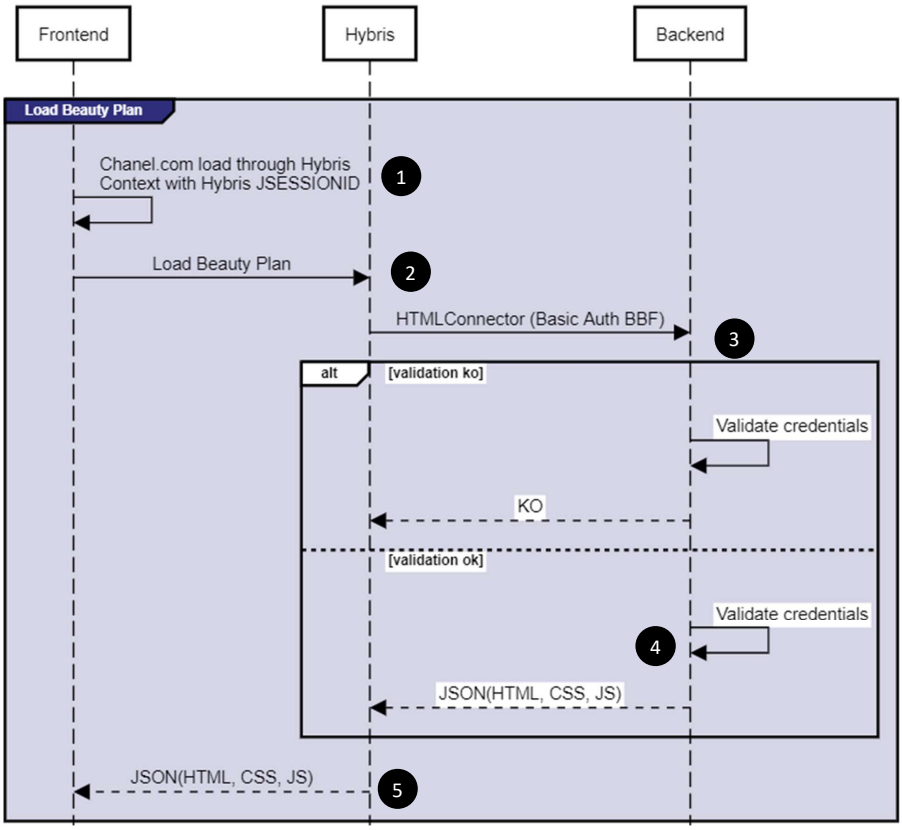
5.2 Description of the logic bricks

5.1.1. Frontend ReactJS

The Beauty Plan frontend application carries the screens, graphical elements and kinematics allowing the Beauty Plan journey to unfold.

It is developed in ReactJS and built as a Single Page Application (SPA).

The initial loading of the application is done through the Hybris application which integrates the Beauty Plan frontend in a template (header/footer) chanel.com.



1. Browser load Chanel.com site through Hybris application. Hybris session is open and correlate user authentication with its JSESSIONID
2. Browser call hybris to load the front-end of Beauty Plan

3. Hybris triggers the retrieval of static elements via an HTTP API request to the Backend BP.
4. Backend BP constitutes a response in the form of a JSON document consisting of three elements containing:
   - The Beauty Plan HTML body
   - A URL to the Beauty Plan application CSS
   - A URL to the Beauty Plan JS
5. Hybris includes these elements in the Chanel.com page and sends the page back to the browser.

The Beauty Plan frontend application is written in Typescript and uses a component-oriented approach.

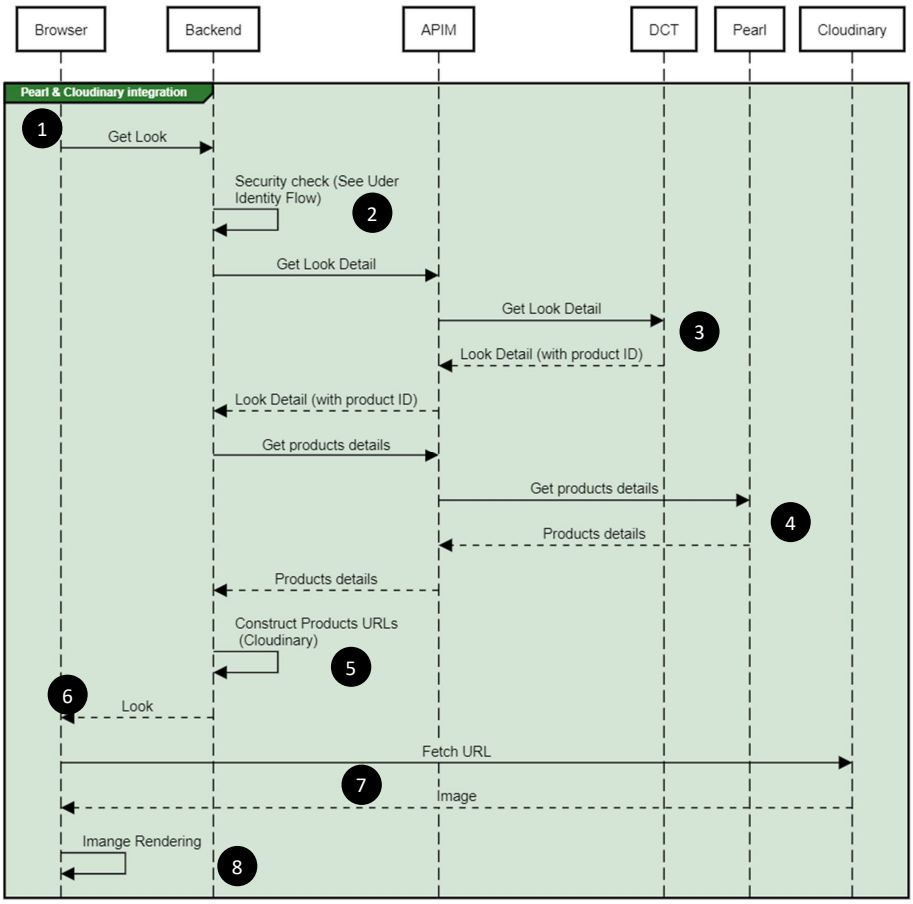
Lazy load is used to better balance experience & technical performance.

The registration of products in the chanel.com user Wishlist is done by calling a JavaScript SDK natively present in the context of the Beauty Plan frontend application. This existing mechanism is outside the scope of the project.

The Cloudinary integration is done via URLs. Those URLs are built by Backend application with information provided by Pearl. The security of these URLs is native see § 5.5.5.2.2 Cloudinary.

For instance, during the Get Look sequence:

1. The front requests a look to the Backend of Beauty Plan,
2. Backend validates the user authentication through the JWT,
3. The request and response flow between the Backend and the DCT (through APIM) to retrieve a look details,
4. The request and response flow between the Backend and Pearl (through APIM) to retrieve products details,
5. Build the products images URL based on Cloudinary,
6. Display the look and associated products details,
7. The browser fetches the products images from Cloudinary,
8. The front renders the products images.



5.1.1.1. URLs pattern and SEO

Business rules to consider:

- URLs to a specific look may be linked or bookmarked,
- Back button must be functional in the browser,
- SEO: URL should exclude dynamic parameters,
- Sustainability: limit unnecessary storage of pages & resources.

Regarding all these constraints, the Beauty Plan frontend will use URL fragment (https://datatracker.ietf.org/doc/html/rfc3986#section-3.5).

Each specific look will have a specific URI eg: https://chanel.com/beautyplan#Look1

It covers:

- The possibility to link or bookmark as the URI present in browser is specific to a look
- SEO: URL should exclude dynamic parameters

Beauty Plan frontend is a single page application, it will use a fragment (hash) routing mechanism https://reactrouter.com/en/main/router-components/hash-router

"Limit unnecessary storage of pages & resources" is covered by the fact that we have only one technical URL which is provided to load frontend SPA application. JS and CSS are loaded only once.

"Back button" is covered by internal ReactJs router features.

5.1.2. Backend NodeJS

The backend NodeJS Beauty Plan is responsible for:

- Providing the Beauty Plan frontend to the Hybris application (see § 5.2.1 ReactJS frontend),
- Exposing APIs to the Beauty Plan frontend application,
- Integrate with third party applications:
  - Gigya,
  - Pearl via APIM,
  - DCT (Salesforce / S3).

The backend exposes REST/JSON APIs via a Swagger/OpenAPI interface and API documentation.

It is developed in NodeJS / Typescript and offers a layered implementation of hexagonal architecture.

5.1.3. APIM Integration

The API Manager is used to secure Chanel IS applications. see § 5.5.5.2.1 Pearl

It authorizes the Beauty Plan backend application to access Chanel IS applications.

5.1.4. Gigya Integration via APIM

The Gigya application is the user identity provider.

Its integration is carried out during the user identity recovery process, see § 5.5.3 Authentication management

5.1.5. Pearl Integration via APIM

The Pearl application is part of the Chanel product catalog. It contains the information (name, reference, color, etc..) of the products that can be chosen during the creation of a "Look".

It offers REST/JSON APIs allowing the Beauty Plan backend to retrieve the products needed to build the customer's "Look".

It is the Pearl application which provided the product image reference through the Cloudinary repository.

The access authorization is done by the API Manager see § 5.2.2.3 Pearl integration via APIM.

5.1.6. Cloudinary Integration

The SaaS application Cloudinary https://cloudinary.com/ is part of the Chanel product catalogue.

The retrieval of product image references is done by Pearl. Backend will then create the final image URL by adding a based URL prefix.

5.1.7. Salesforce Integration via APIM

The Salesforce SaaS application is part of the DCT and allows the saving of "looks" (Service) attached to a user (Customer) and consisting of one to three products (Product).

The integration is carried out via see § 5.5.5.2.3 DCT (Salesforce

5.1.8. Integration S3 via APIM

The SaaS S3 service is part of the DCT and allows to store the images related to the user's "Look"

The integration is carried out via see § **Erreur ! Source du renvoi introuvable. Erreur ! Source du renvoi introuvable.**

5.3. Third-party technologies

This section describes the set of technologies used to build the software architecture of the solution.

5.3.1. Description of third-party components

The table below describes the technologies implemented in the conception.

Technology	Version	Licence	Description	Website
NodeJS	18.X.Y	MIT License	Runtime JS	https://nodejs.org

Fastify	4.X.Y	MIT License	Web framework	https://www.fastify.io/
Jest	29.3	MIT License	Test framework	https://jestjs.io/

5.3.2. Third Party Technology Update Policy

This section describes the policy for updating third-party technologies during the solution conception phase. This section should indicate the strategy to be implemented for each component for each release type:

- Major version,
- Minor version,
- Security hotfix.

Example:

The versions of the different libraries will be frozen and changed via a voluntary action of the development team. The availability of new versions of the libraries will be checked regularly by the team and updates will be made according to the version type available:

- Major version: Upgrade to the new version to be studied according to the impacts on the solution,
- Minor version: Systematic update to the new version,
- Security hotfix: Systematic and fast update to the new version.

5.4. Volume of stored data

The Beauty Plan application is not a repository of any data of its own.

5.5. Application of the security

This section describes the technical solution elements implemented to meet the security requirements of the application.

5.5.1. User management

The end user is known to the Beauty Plan application through his Gigya token. There is no local persistence of user data in the application.

5.5.2. Identity management

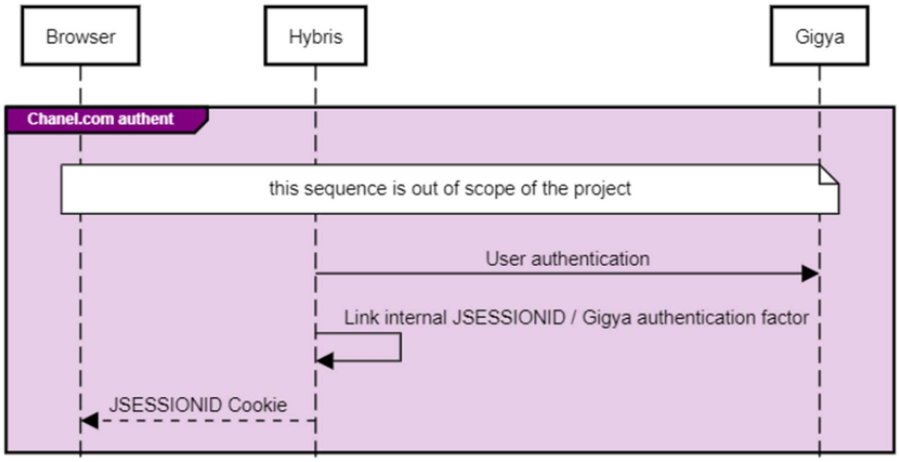
The identities are managed by the Gigya identity provider.

The Beauty Plan application uses the Gigya ID as a user ID. The following paragraph describes the kinematics of the recovery of this identifier.

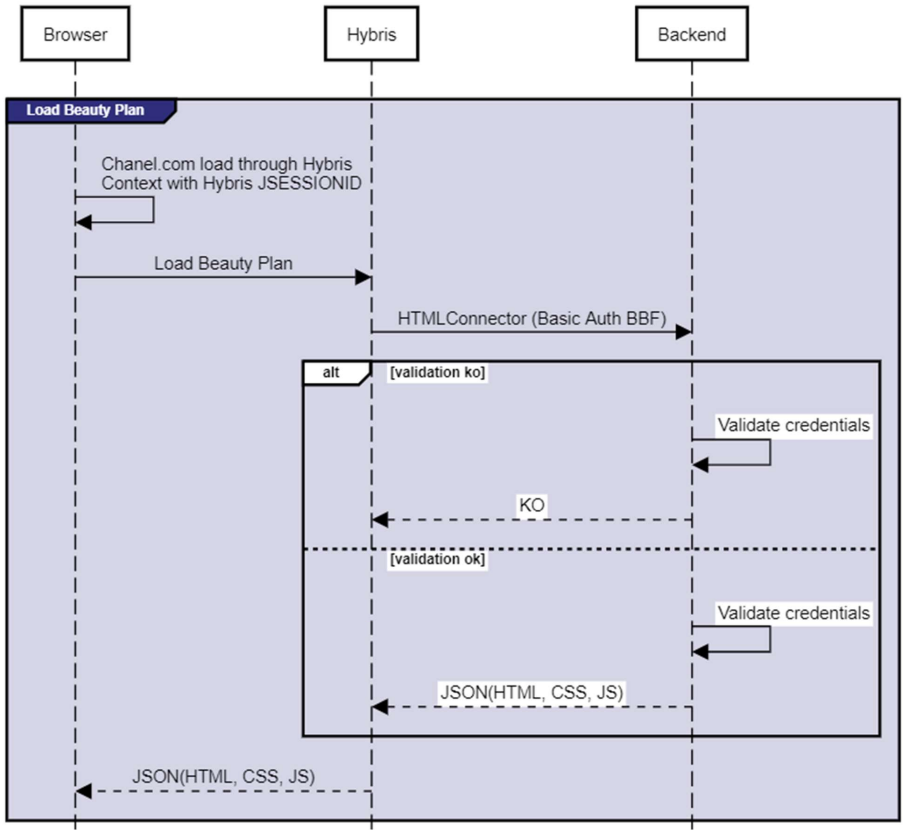
5.5.3. Authentication management

The user authentication phase is outside the scope of the project and carried out by Chanel.com. At the end of this phase:

- a JWT Gigya token has been created by Gigya,
- a target UID has been sent to browser context, this target UID is correlated to user identity.

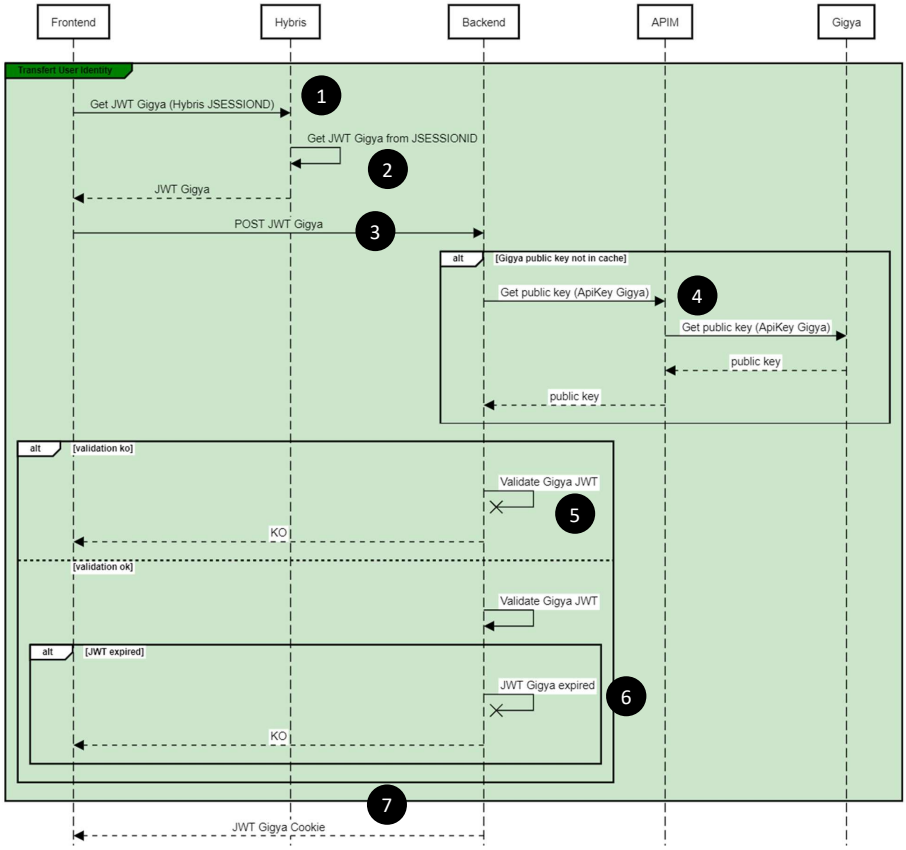


When the Beauty Plan application is loaded, Hybris calls the Backend BP (see § 5.2.1 Frontend ReactJS) to build the BP frontend page.



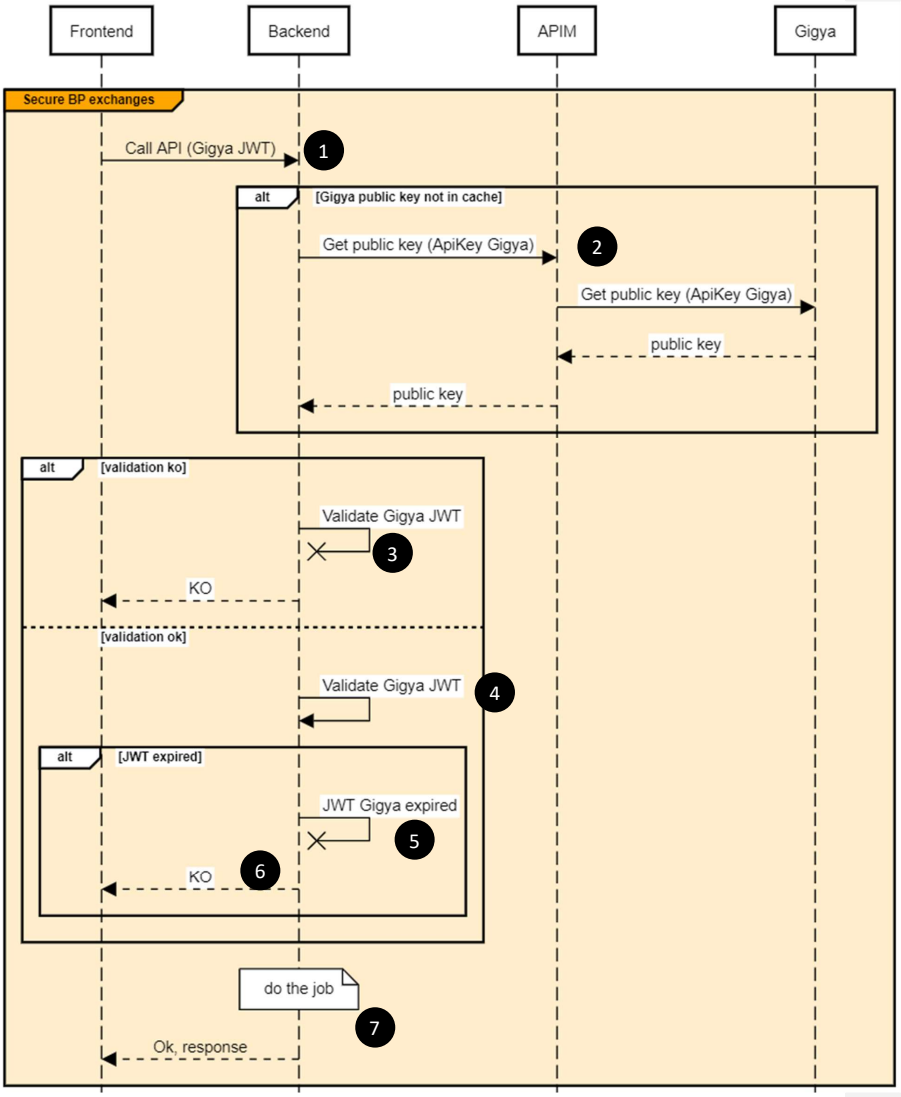
At the end the browser can display the Beauty Plan frontend.

During frontend start, it automatically a sequence in order for backend for front to retrieve the Gigya Token that will be used for BP authentication between frontend and backend.



1. Frontend call Hybris with JSESSIONID in order to get Gigya JWT Token
2. Hybris retrieve the Gigya JWT token correlated to its JSESSIONID and its Gigya ID
3. Frontend submit the JWT to Backend in order to validate it and get secured cookie used for Beauty Plan session
4. Backend retrieve the Gigya public key through APIM in order to validate the JWT
5. If JWT is not validated Backend return an error to Frontend
6. If JWT is expired Backend return an error to Frontend
7. If everything is fine, Backend returns a secured Cookie with JWT to Frontend

Hence all the BP Frontend calls the Backend are perform with the Gigya JWT token has authentication factor.



1. Gigya JWT Token Handling between the front and the back of the beauty plan,
2. If not present in cache, Gigya public key is retrieve from Gigya server through APIM,
3. If the provided Gigya token could not been validated with the Gigya public key, an error is sent back to front,
4. If the Gigya token signature is correct,
5. A check is performed on token expiration date

6. If expired, Backend send back a response of expired session,
7. Backend can realize the action linked to frontend call and send back a response.

5.5.4. Authorization management

The Beauty plan application does not carry any specific authorization. Authenticated users of the service have access to all the features.

5.5.5. Flow management

5.5.5.1. Internal

The flow between the frontend BP and the backend BP is done in HTTPS, TLS 1.2.

The certificate used is a web type provided by Chanel.

5.5.5.2. External

5.5.5.2.1. Pearl

The Pearl flow (internal application to the Chanel IS) is secured by an API Manager.

The authentication of the Backend BP with the APIM is carried out via an OAuth 2 flow of type Client Credential (server to server). The calls linked to this flow are in HTTPS TLS 1.2 public web certificate.

5.5.5.2.2. Cloudinary

The BP frontend flow to Cloudinary is naturally secured by Cloudinary which builds natively signed URLs:
https://cloudinary.com/documentation/control_access_to_media#signed_delivery_urls

5.5.5.2.3. DCT (Salesforce/S3)

The DCT flow (Salesforce/S3) is secured by an API Manager.

The authentication of the Backend BP with the APIM is secured by:

- The Gigya token which is used by APIM to control that request is performed in a user authenticated flow,
- A machine-to-machine factor (subscription key) which used by APIM to control that request is performed by a registered application.

The calls linked to this flow are in HTTPS TLS 1.2 public web certificate.

5.6. Cache management

This section describes the caches implemented and the data lifecycle strategies.

5.6.1. Backend for frontend application cache

5.6.1.1. Frontend requests

Backend application does not implement a specific cache policy for frontend requests. As these requests are all in an "authenticated/connected" context, cache management is not relevant here.

5.6.1.2. Sessions

Backend application is stateless and does not require a session.

5.6.2. Browser cache

No particular cache management policy. Browser asks static assets to Backend for Frontend.

5.7. Responsive Design Management

The application is a responsive design. The resolution thresholds known and shared with artistic direction are:

- Under 600 px: mobile.
- From 600 px to 900 px: tablet.
- After 900 px: medium desktop.

Design intention is mobile first.

5.8. Internationalization Management

The application supports localization and internationalization.

The market and language will be provided in the parameters of the frontend BP retrieval URL.

BP Application support 4 markets and 9 languages that which will be provided in the parameters of the frontend BP retrieval URL.

https://{URL-sous-domaine-Beauty-Plan}.chanel.com/connector/{country}/{locale}

Ex: https://{URL-sous-domaine-Beauty-Plan}.chanel.com/connector/FR/fr

6. Technical Architecture

6.1. Environments Mapping

Beauty Plan	SF	Amazon S3	Chanel .com	Pearl	APIM	Gigya
Development environments are not connected to each other						
INT	Dev sdbox2	INT	INT1	INT	INT	INT
UAT	UAT	UAT	UAT	UAT	UAT	UAT
OAT	Preprod	OAT	OAT	OAT	OAT	OAT
PROD	PROD	Prod	PROD	PROD	PROD	PROD

6.2. Non-prod Environments

6.2.1. Dev

Local environment on the developer's desktop.

This environment is connected with:

- A salesforce integration application via an integration APIM,
- An S3 integration service via an integration APIM,
- A Pearl integration application via an integration APIM.

The Hybris integration on the Chanel.com portal and the Wishlist management will be simulated.

The developments are registered in the Chanel Git source repository (Azure Devops).

The continuous integration chain is based on Azure Devops and realized by Chanel with the support of Niji.

6.2.2. Integration

Chanel environment will be used for the integration of the different components. It serves as a support for Niji's internal test.

The environment is provided by Chanel and updated by the continuous deployment chain provided by Chanel.

6.2.3. User Acceptance test

Chanel environment will be used for the Chanel business teams acceptance test.

The environment is provided by Chanel and updated by the continuous deployment chain provided by Chanel.

6.2.4. Operational Acceptance test (preprod)

Chanel environment will be used for technical tests / benchmark of the platform.

The environment is provided by Chanel and updated by the continuous deployment chain provided by Chanel.

6.3. Production Schema

The technical infrastructure is the sole responsibility of Chanel.

6.4. Description of the technical bricks

The technical infrastructure is the sole responsibility of Chanel.

6.5. Flow Matrix

The technical infrastructure is the sole responsibility of Chanel.

6.6. Exploitation

The technical infrastructure is the sole responsibility of Chanel.

7. Appendix

https://sequencediagram.org/

Sequence init

```
title Load and user identity transfert to Beauty Plan

participant Browser
participant Hybris
participant Backend
participant APIM
participant Pearl
participant Cloudinary
```

Chanel.com authent

```
group #purple Chanel.com authent #white
note over Browser,Gygia: this sequence is out of scope of the project
Gygia-->Hybris: JWT Gigya
Hybris->Hybris: Link internal JSESSIONID / JWT Gigya
end
```

Load Beauty Plan

```
group #2f2e7b Load Beauty Plan #white
Frontend->Frontend: Chanel.com load through Hybris\nContext with Hybris JSESSIONID
Frontend->Hybris: Load Beauty Plan
Hybris->Backend: HTMLConnector (Basic Auth BBF)
alt validation ko
Backend->Backend: Validate credentials
Backend-->Hybris: KO
else validation ok
Backend->Backend: Validate credentials
Backend-->Hybris: JSON(HTML, CSS, JS)
end
Hybris-->Frontend: JSON(HTML, CSS, JS)
end
```

Pearl & Cloudinary integration

```
group #2e7b2f Pearl & Cloudinary integration #white
Browser->Backend: Get Look
Backend->Backend: Security check (See Uder \nIdentity Flow)
Backend->APIM: Get Look Detail
APIM->DCT: Get Look Detail
DCT-->APIM: Look Detail (with product ID)
APIM-->Backend: Look Detail (with product ID)
Backend->APIM: Get products details
APIM->Pearl: Get products details
Pearl-->APIM: Products details
APIM-->Backend: Products details
Backend->Backend: Construct Products URLs \n (Cloudinary)
Backend-->Browser: Look
Browser->Cloudinary: Fetch URL
Cloudinary-->Browser: Image
Browser->Browser: Imange Rendering
end
```

Transfert User Identity

```
group #green Transfert User Identity

Frontend->Hybris: Get JWT Gigya (Hybris JSESSIOND)
Hybris->Hybris: Get JWT Gigya from JSESSIONID
Hybris-->Frontend: JWT Gigya


Frontend->Backend: POST JWT Gigya


alt Gigya public key not in cache
Backend->APIM: Get public key (ApiKey Gigya)
APIM->Gigya: Get public key (ApiKey Gigya)
Gigya-->APIM: public key
APIM-->Backend: public key
end

alt validation ko
Backend-xBackend: Validate Gigya JWT
Backend-->Frontend: KO
else validation ok
Backend->Backend: Validate Gigya JWT

alt JWT expired
Backend-xBackend: JWT Gigya expired
Backend-->Frontend: KO
end
end
end

Backend-->Frontend: JWT Gigya Cookie

end
```

Secure BP exchange

```
group #orange Secure BP exchanges

Frontend->Backend: Call API (Gigya JWT)

alt Gigya public key not in cache
Backend->APIM: Get public key (ApiKey Gigya)
APIM->Gigya: Get public key (ApiKey Gigya)
Gigya-->APIM: public key
APIM-->Backend: public key
end

alt validation ko
Backend-xBackend: Validate Gigya JWT
Backend-->Frontend: KO
else validation ok
Backend->Backend: Validate Gigya JWT

alt JWT expired
Backend-xBackend: JWT Gigya expired
Backend-->Frontend: KO
```

```
end
end

note over Backend:do the job
Backend-->Frontend: Ok, response
end
```