Architecture DreamScape - Documentation Complète

> **Architecture Hybride Big Pods** - 4 repositories de développement → 3 Big Pods de déploiement

Table des matières

---

0. Prerequis

Installation Docker

Docker Engine (requis)

```
# Ubuntu/Debian
sudo apt update && sudo apt install docker.io docker-compose-plugin

# Vérification
docker --version
docker compose version
```

Permissions utilisateur

PROF

```
sudo usermod -aG docker $USER
# Redémarrer la session
```

Configuration recommandée

- Docker Engine 24.0+
- Docker Compose V2
- 8GB RAM minimum
- 50GB espace disque libre

Récupération des Repositories

Clone des repositories de développement

```
# Créer le dossier de travail
mkdir -p ~/dreamscape-project && cd ~/dreamscape-project

# Cloner les 4 repositories
git clone https://github.com/dreamscape/dreamscape-infra.git
git clone https://github.com/dreamscape/dreamscape-services.git
git clone https://github.com/dreamscape/dreamscape-frontend.git
git clone https://github.com/dreamscape/dreamscape-test.git
```

Structure finale attendue

```
~/dreamscape-project/
├── dreamscape-infra/          # Infrastructure & Docker configs
├── dreamscape-services/       # Backend services (AI, Auth, Payment,
User, Voyage)
├── dreamscape-frontend/       # Frontend apps (Web-client, Panorama)
└── dreamscape-test/         # Tests & validation
```

Vérification

```
# Vérifier la structure
ls -la ~/dreamscape-project/

# Vérifier les branches principales
cd dreamscape-infra && git branch -a
cd ../dreamscape-services && git branch -a
cd ../dreamscape-frontend && git branch -a
cd ../dreamscape-test && git branch -a
```

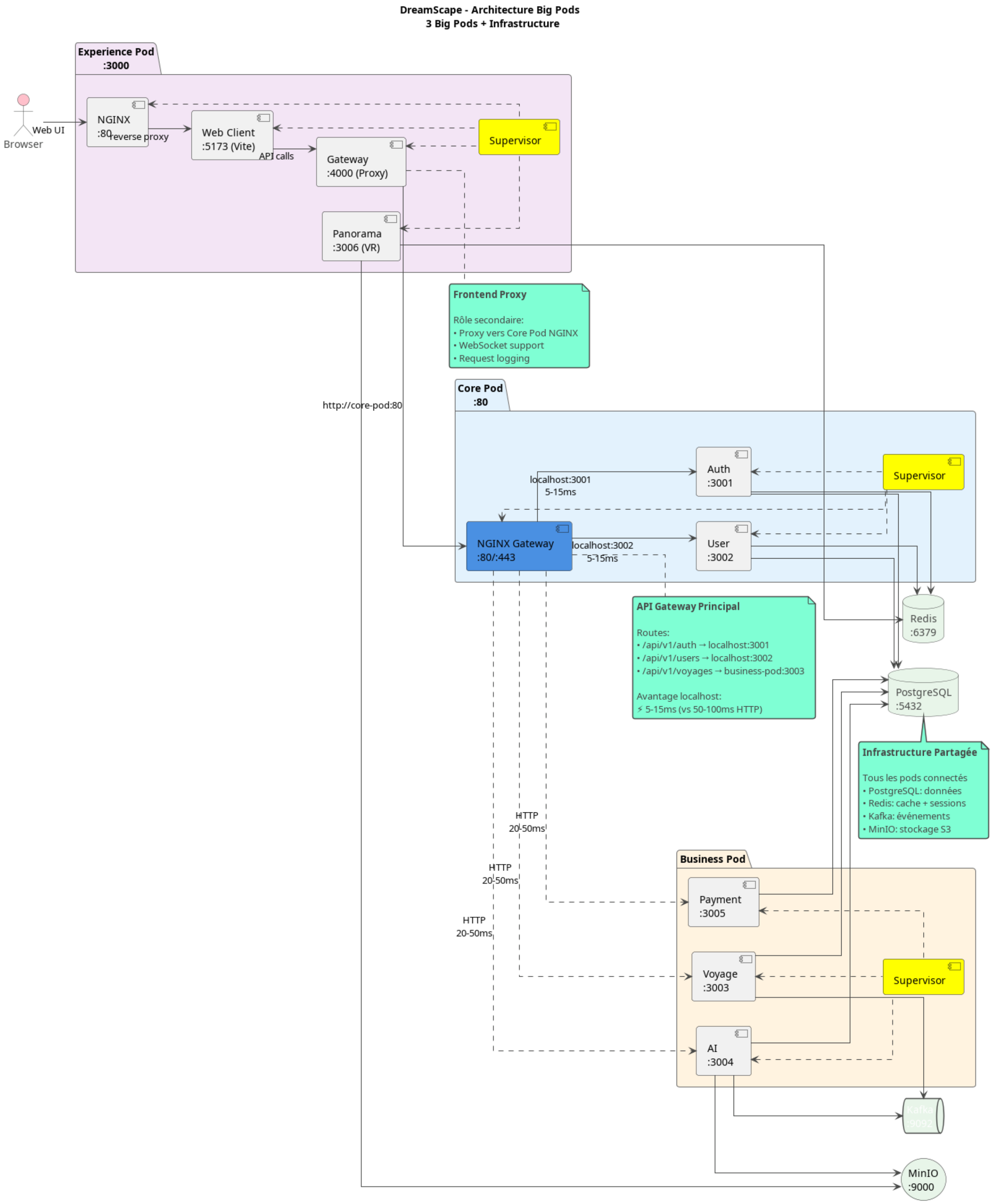
PROF

1. Vue d'ensemble

DreamScape utilise une **architecture hybride** qui combine :

- **4 repositories** pour le développement
  - **dreamscape-infra**: repo de l'infrasctuture
  - **dreamscape-services** repo des différents services (ai, auth, payment, user, voyage)
  - **dreamscape-frontend** repo du frontend (panomrama, web-client)
  - **dreamscape-test**: repo des tests
- **3 Big Pods** pour le déploiement (efficacité opérationnelle)
  - **experience-pod**: web-client, panorama
  - **business-pod**: voyage-service, payment-serice, ai-service
  - **core-pod**: auth-service, user-service

Diagrame App



Frontend Proxy

Rôle secondaire:
- Proxy vers Core Pod NGINX
- WebSocket support
- Request logging

API Gateway Principal

Routes:
- /api/v1/auth → localhost:3001
- /api/v1/users → localhost:3002
- /api/v1/voyages → business-pod:3003

Avantage localhost:
⚡ 5-15ms (vs 50-100ms HTTP)

Infrastructure Partagée

Tous les pods connectés
- PostgreSQL: données
- Redis: cache + sessions
- Kafka: événements
- MinIO: stockage S3

PROF

2. Composant détaillés

Experience pod - Interface & Expérience Utilisateur

Rôle : Gestion de l'interface utilisateur, expériences immersives VR et proxy API frontend

Services Inclus

- **Web Client (:5173)**
  Stack technique : React 18 + TypeScript + Vite + TailwindCSS

  Responsabilités :
  ✅ Interface utilisateur principale (recherche vols, réservation)
  ✅ Gestion de l'état (Redux Toolkit / Zustand)
  ✅ Authentification JWT (stockage tokens, refresh)
  ✅ Routing client-side (React Router)
  ✅ Hot Module Replacement (HMR) pour développement

  Fonctionnalités clés :

  - Moteur de recherche de vols
  - Calendrier de disponibilités
  - Formulaire de paiement
  - Profil utilisateur
  - Carte interactive des destinations

- **Panorama (:3006)**
  Stack technique : Node.js + Express + Three.js + WebGL

  Responsabilités :

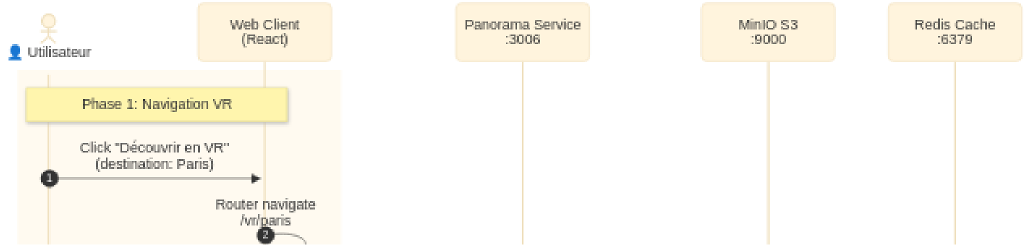
```
✅ Streaming vidéos 360° (destinations VR)
✅ Gestion des assets VR (stockage MinIO)
✅ API de prévisualisation immersive
✅ Optimisation des textures (compression, CDN)
✅ Support WebXR (casques VR)
```

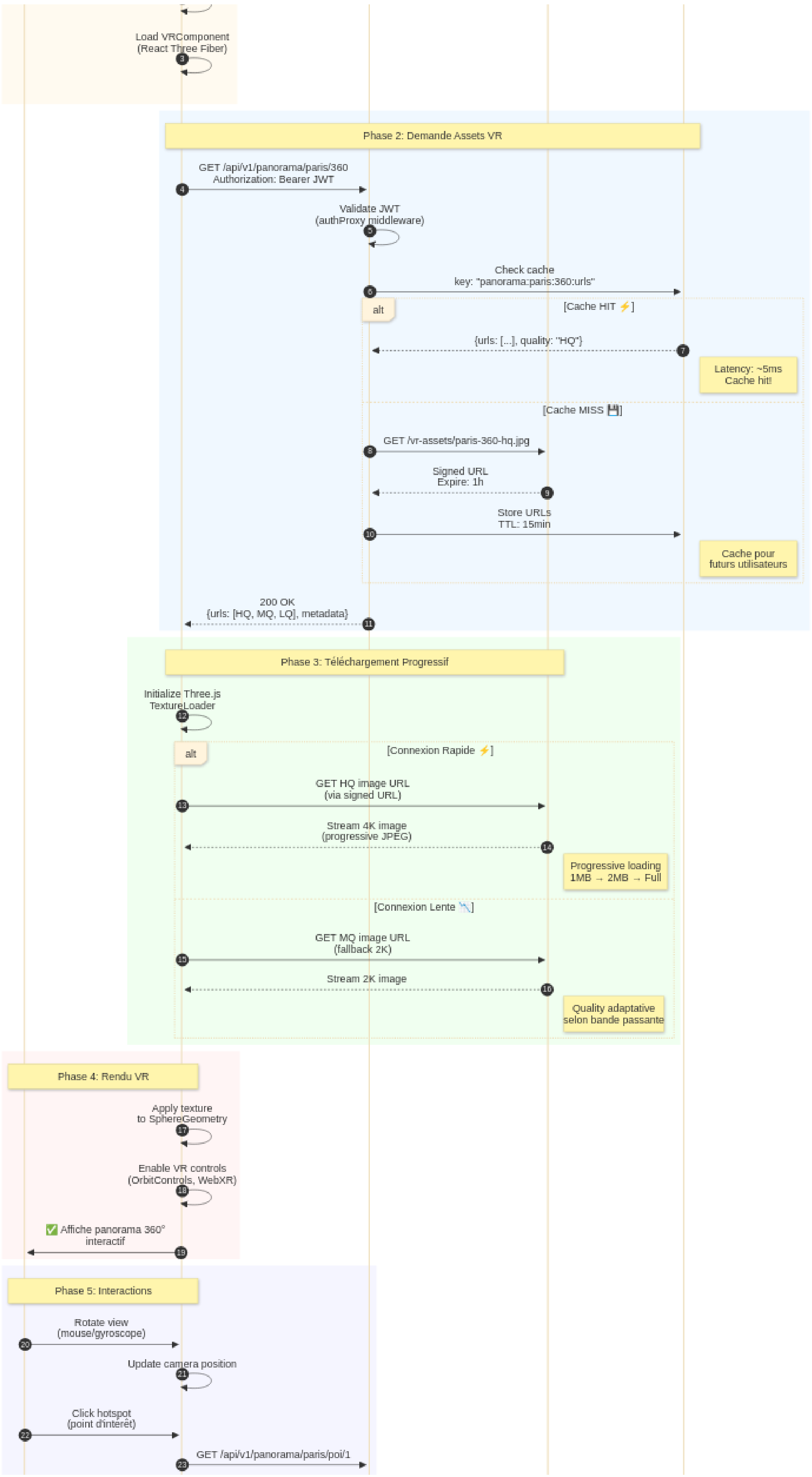
PROF

  Fonctionnalités clés :

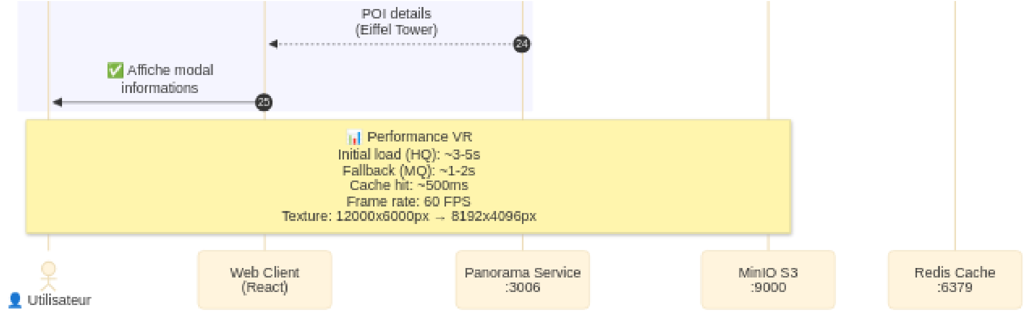
  - Visites virtuelles de destinations
  - Prévisualisation hôtels en VR
  - Navigation interactive (gyroscope, contrôleurs)
  - Galeries photos 360°
  - Vidéos immersives lieux touristiques

Exemple de flux VR :




Load VRComponent
(React Three Fiber)
3
Phase 2: Demande Assets VR
GET /api/v1/panorama/paris/360
Authorization: Bearer JWT
4
Validate JWT
(authProxy middleware)
5
Check cache
key: "panorama:paris:360:urls"
6
alt
[Cache HIT ⚡]
{urls: [...], quality: "HQ"}
7
Latency: ~5ms
Cache hit!
[Cache MISS 💾]
GET /vr-assets/paris-360-hq.jpg
8
Signed URL
Expire: 1h
9
Store URLs
TTL: 15min
10
Cache pour
futurs utilisateurs
200 OK
{urls: [HQ, MQ, LQ], metadata}
11
Phase 3: Téléchargement Progressif
Initialize Three.js
TextureLoader
12
alt
[Connexion Rapide ⚡]
GET HQ image URL
(via signed URL)
13
Stream 4K image
(progressive JPEG)
14
Progressive loading
1MB → 2MB → Full
[Connexion Lente 📉]
GET MQ image URL
(fallback 2K)
15
Stream 2K image
16
Quality adaptative
selon bande passante
Phase 4: Rendu VR
Apply texture
to SphereGeometry
17
Enable VR controls
(OrbitControls, WebXR)
18
✅ Affiche panorama 360°
interactif
19
Phase 5: Interactions
Rotate view
(mouse/gyroscope)
20
Update camera position
21
Click hotspot
(point d'intérêt)
22
GET /api/v1/panorama/paris/poi/1
23

PROF



- **Gateway (:4000)**
  Stack technique : Node.js + Express + http-proxy-middleware

  Responsabilités :

  ✅ Proxy API vers Core Pod NGINX (:80)
  ✅ WebSocket upgrade (chat, notifications temps réel)
  ✅ Gestion CORS pour frontend
  ✅ Logging des requêtes (monitoring)
  ✅ Retry logic (en cas d'échec temporaire)

  Architecture du proxy :

  Pourquoi un Gateway séparé ?

  ✅ Découplage frontend/backend (isolation)
  ✅ Transformation des requêtes si nécessaire
  ✅ Rate limiting côté client
  ✅ Logs centralisés des appels API frontend

PROF

- **Nginx (:80)**
  Rôle : Reverse proxy local pour le pod Experience

  Configuration :

  Avantage : Un seul point d'entrée :3000 pour tout le frontend

Métriques Experience Pod

Métrique	Valeur Cible	Réel (Dev)
Cold Start	< 30s	~25s
Hot Reload	< 2s	~1.5s (Vite HMR)
API Latency	< 100ms	~80ms (via Gateway)
VR Load Time	< 5s	~3.5s (vidéo 360°)

Métrique	Valeur Cible	Réel (Dev)
Memory	< 512MB	~400MB

Core pod - Authentification & Services Critiques

Rôle : Gateway API principal, authentification JWT, gestion utilisateurs, cache Redis

Services Inclus

- **Nginx Gateway (:80/:443)**
  Rôle : API Gateway central de DreamScape

  Responsabilités :

  ✅ Reverse Proxy : Route toutes les API vers les bons services
  ✅ Rate Limiting : Protection DDoS (10 req/s auth, 50 req/s API)
  ✅ Load Balancing : Distribution de charge (future échelle)
  ✅ SSL/TLS Termination : HTTPS (certificats Let's Encrypt)
  ✅ CORS Management : Headers sécurisés
  ✅ Health Checks : Monitoring santé

  Configuration clé :

  Avantage localhost :
  ⚡ 5-15ms latency (vs 50-100ms HTTP externe)
  📈 10x plus rapide pour Auth/User
  🔒 Pas d'exposition réseau interne

- **Auth Service (:3001)**
  Stack technique : Node.js + Express + Passport + JWT + bcrypt

  PROF

  Responsabilités :

  ✅ Inscription utilisateurs (email + password)
  ✅ Login avec JWT (access token + refresh token)
  ✅ Validation tokens (middleware pour toutes les APIs)
  ✅ Gestion sessions (stockage Redis)
  ✅ OAuth2 (Google, Facebook - futur)
  ✅ Réinitialisation mot de passe (email)

  API Endpoints : htpp://localhost:3001/api/v1/

  Auth Service - Complete Route List
  Authentication Routes
  Base: /api/v1/auth

User Registration & Login

Method	Endpoint	Description	Auth Required	Rate Limited	Parameters
POST	/register	Register a new user	× No	✅ Yes	email, password, firstName (optional), lastName (optional), username (optional)
POST	/login	Authenticate user and get token	× No	✅ Yes	email, password, rememberMe (optional)

Profile Management

Method	Endpoint	Description	Auth Required	Parameters
GET	/profile	Get current user profile	✅ Yes	None (uses auth token)
PUT	/profile	Update user profile	✅ Yes	email (optional), firstName (optional), lastName (optional), username (optional)

Password Management

Method	Endpoint	Description	Auth Required	Parameters
POST	/change-password	Change user password	✅ Yes	currentPassword, newPassword

Token Management

Method	Endpoint	Description	Auth Required	Rate Limited	Parameters
POST	/refresh	Refresh access token	× No (refresh token)	✅ Yes	refreshToken (cookie or body)
POST	/verify-token	Verify if token is valid	✅ Yes × No None (validates auth token)		

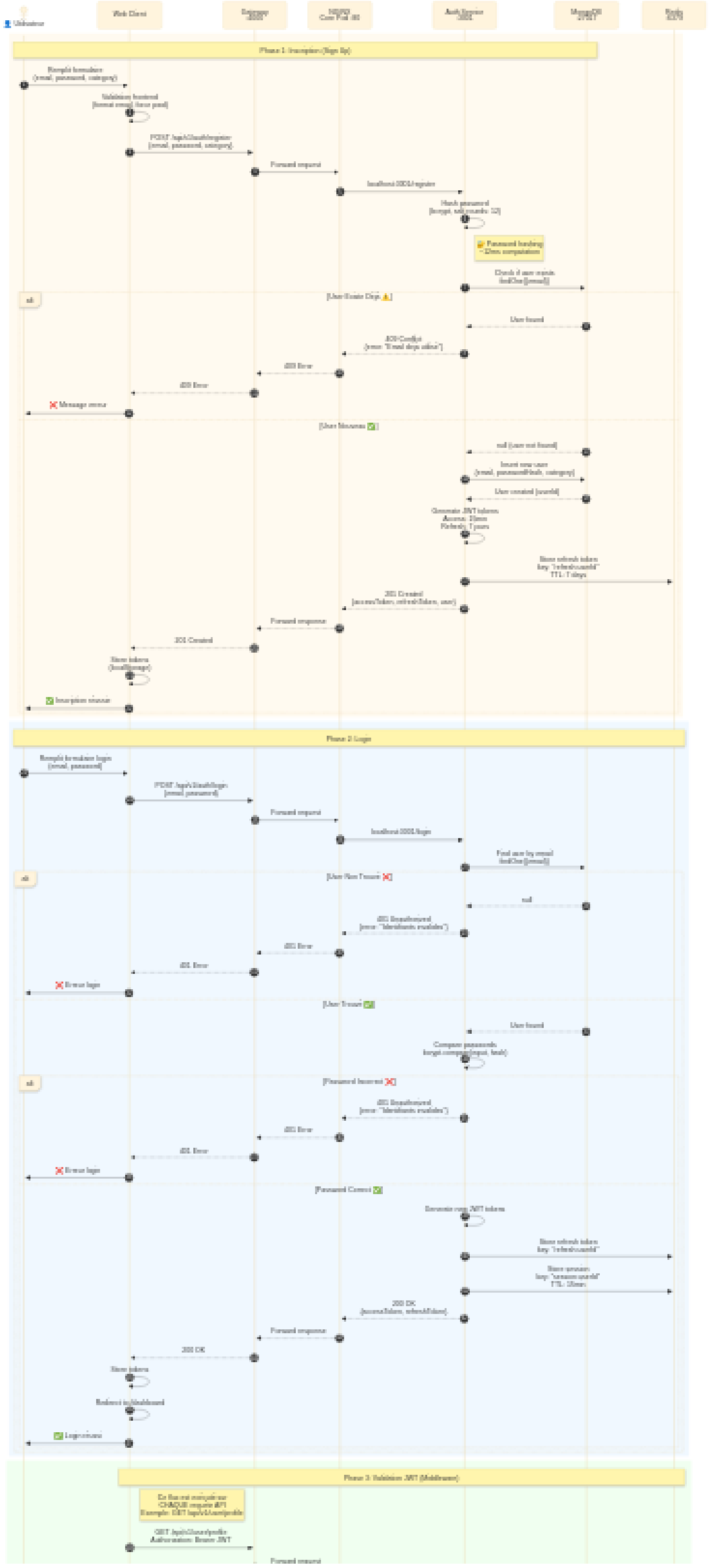
Session Management

Method	Endpoint	Description	Auth Required	Parameters

PROF

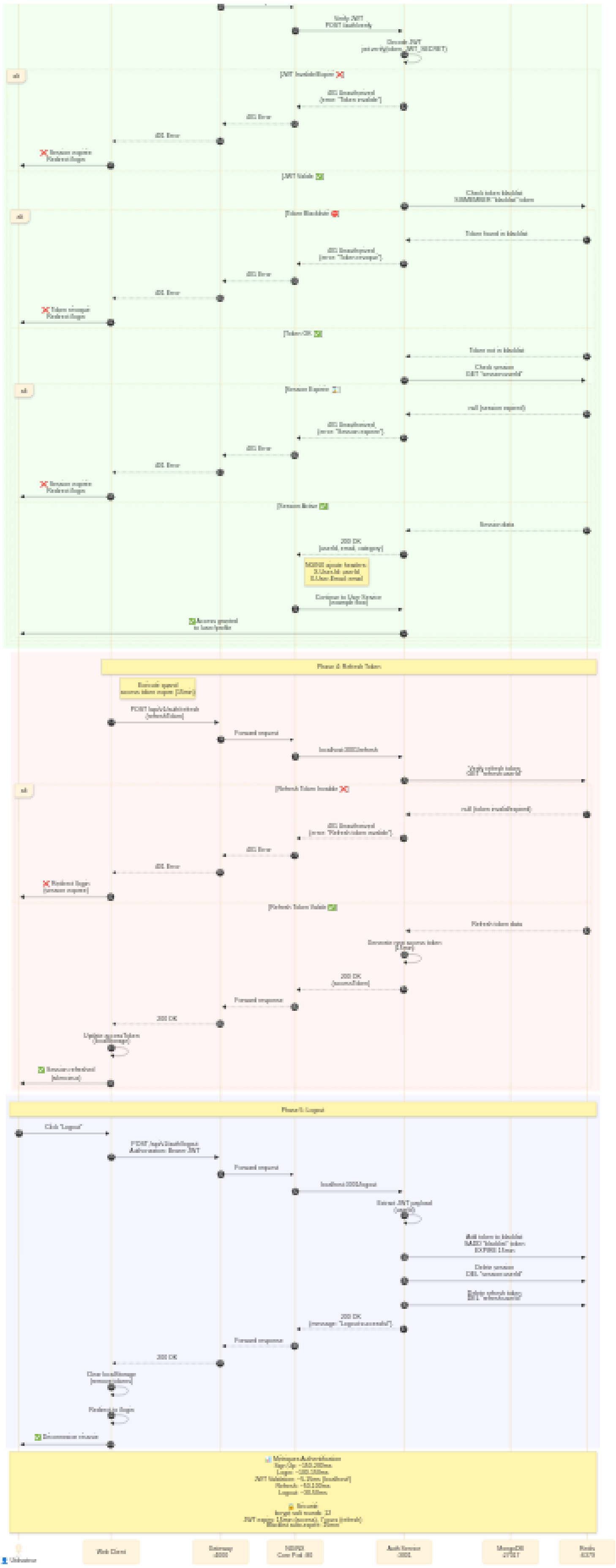
Method	Endpoint	Description	Auth Required	Parameters
POST	/logout	Logout user and revoke refresh token	✕ No	None (uses cookies)
POST	/logout-all	Logout user from all devices	✅ Yes	None (uses auth token)

Flux d'authentification :



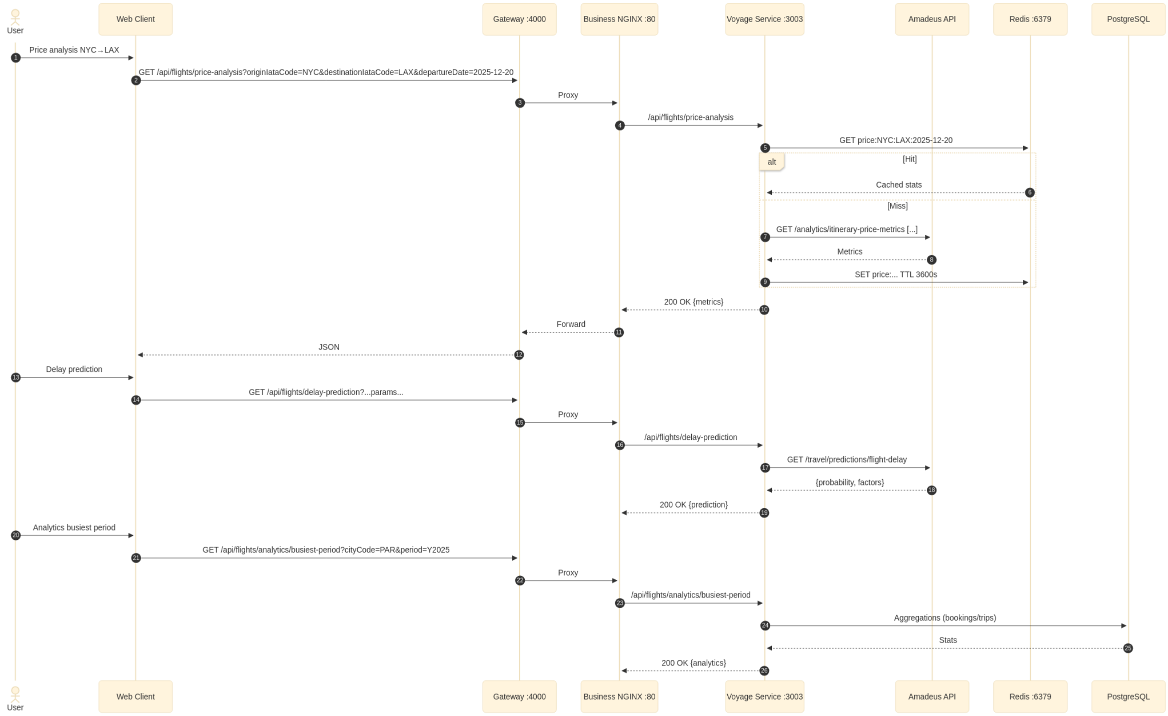
Utilisateur
Web Client
Phase 1: Inscription (Sign Up)
Remplit formulaire (email, password, category)
Validation frontend (format email, force pwd)
POST /api/v1/auth/register (email, password, category)
Forward request
localhost:3001/register
Hash password (bcrypt, salt rounds: 12)
Password hashing ~12ms computation
Check if user exists findOne({email})
alt
[User Existe Déjà ⚠️]
User found
409 Conflict {error: "Email déjà utilisé"}
409 Error
409 Error
❌ Message erreur
[User Nouveau ✅]
null (user not found)
Insert new user (email, passwordHash, category)
User created {userId}
Generate JWT tokens Access: 15min Refresh: 7 jours
Store refresh token key: "refresh:userId" TTL: 7 days
201 Created {accessToken, refreshToken, user}
Forward response
201 Created
Store tokens (localStorage)
✅ Inscription réussie
Phase 2: Login
Remplit formulaire login (email, password)
POST /api/v1/auth/login (email, password)
Forward request
localhost:3001/login
Find user by email findOne({email})
alt
[User Non Trouvé ❌]
null
401 Unauthorized {error: "Identifiants invalides"}
401 Error
401 Error
❌ Erreur login
[User Trouvé ✅]
User found
Compare passwords bcrypt.compare(input, hash)
alt
[Password Incorrect ❌]
401 Unauthorized {error: "Identifiants invalides"}
401 Error
401 Error
❌ Erreur login
[Password Correct ✅]
Generate new JWT tokens
Store refresh token key: "refresh:userId"
Store session key: "session:userId" TTL: 15min
200 OK {accessToken, refreshToken}
Forward response
200 OK
Store tokens
Redirect to /dashboard
✅ Login réussi
Phase 3: Validation JWT (Middleware)
Ce flux est exécuté sur CHAQUE requête API Exemple: GET /api/v1/user/profile
GET /api/v1/user/profile Authorization: Bearer JWT
Forward request

PROF

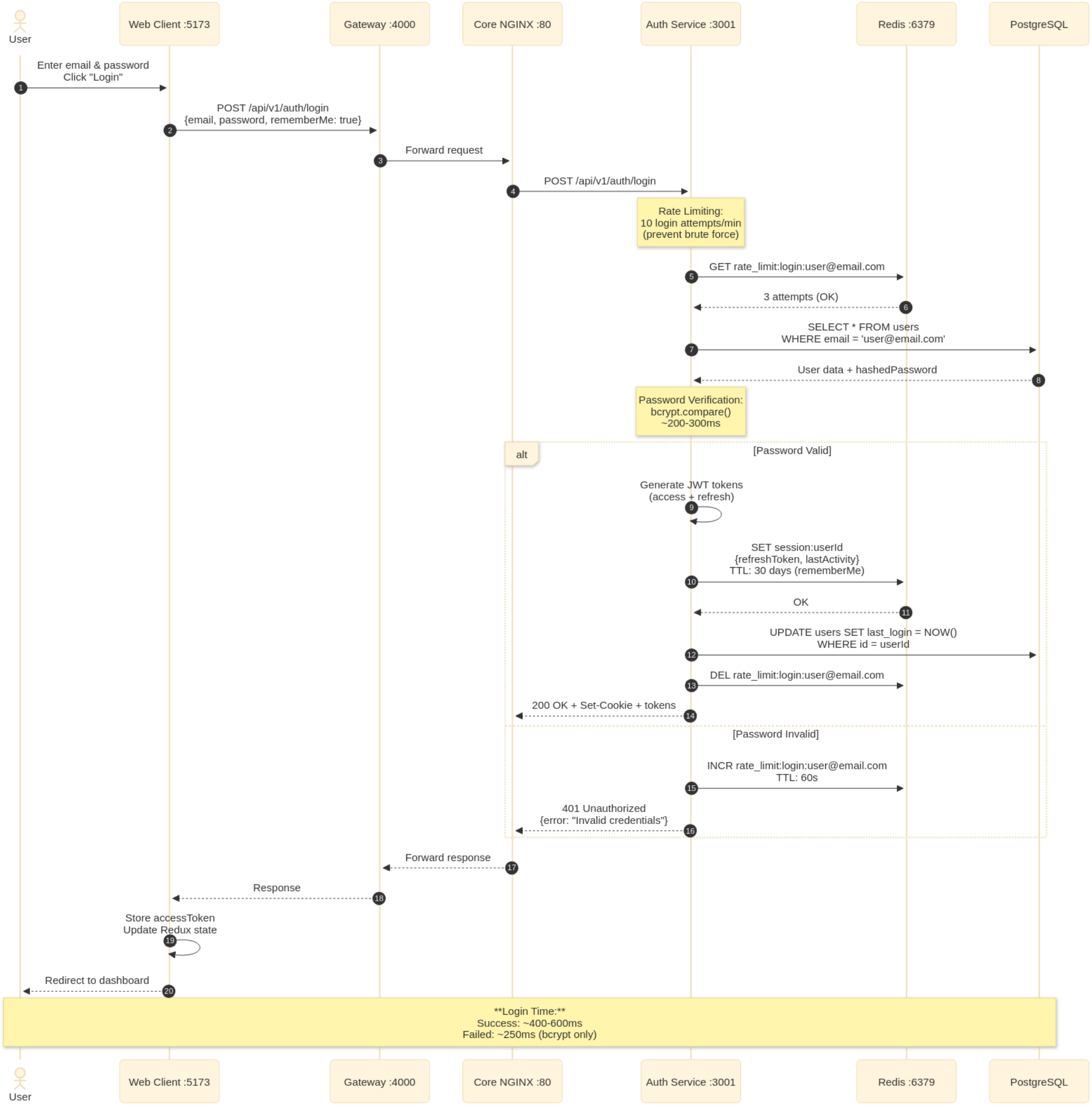

Verify JWT
POST /auth/verify
Decode JWT
jwt.verify(token, JWT_SECRET)
alt
[JWT Invalid/Expired ❌]
401 Unauthorized
{error: "Token invalid"}
401 Error
401 Error
❌ Session expired
Redirect /login
[JWT Valid ✅]
Check token blacklist
SISMEMBER "blacklist" token
alt
[Token Blacklisted 🚫]
Token found in blacklist
401 Unauthorized
{error: "Token revoked"}
401 Error
401 Error
❌ Token revoked
Redirect /login
[Token OK ✅]
Token not in blacklist
Check session
GET "session:userId"
alt
[Session Expired ⏳]
null (session expired)
401 Unauthorized
{error: "Session expired"}
401 Error
401 Error
❌ Session expired
Redirect /login
[Session Active ✅]
Session data
200 OK
{userId, email, category}
NGINX injects headers:
X-User-Id: userId
X-User-Email: email
Continue to User Service
(example flow)
✅ Access granted
to /user/profile
Phase 4: Refresh Token
Beforehand: expired
access token expires (15min)
POST /api/v1/auth/refresh
{refreshToken}
Forward request
localhost:3001/refresh
Verify refresh token
GET "refresh:userId"
alt
[Refresh Token Invalid ❌]
null (token invalid/expired)
401 Unauthorized
{error: "Refresh token invalid"}
401 Error
401 Error
❌ Redirect /login
(session expired)
[Refresh Token Valid ✅]
Refresh token data
Generate new access token
(15min)
200 OK
{accessToken}
Forward response
200 OK
Update accessToken
(localStorage)
✅ Session refreshed
(silent auth)
Phase 5: Logout
Click "Logout"
POST /api/v1/auth/logout
Authorization: Bearer JWT
Forward request
localhost:3001/logout
Extract JWT payload
(userId)
Add token to blacklist
SADD "blacklist" token
EXPIRE 15min
Delete session
DEL "session:userId"
Delete refresh token
DEL "refresh:userId"
200 OK
{message: "Logout successful"}
Forward response
200 OK
Clear localStorage
(remove tokens)
Redirect to /login
✅ Disconnection success
📊 Métriques Authentification
Sign Up: ~150-200ms
Login: ~100-150ms
JWT Validation: ~5-15ms (localhost)
Refresh: ~50-100ms
Logout: ~30-50ms
🔒 Sécurité
bcrypt salt rounds: 12
JWT expiry: 15min (access), 7 jours (refresh)
Blacklist auto-expire: 15min
👤 Utilisateur
Web Client
Gateway
:8000
NGINX
Core Pod :80
Auth Service
:3001
MongoDB
:27017
Redis
:6379

PROF

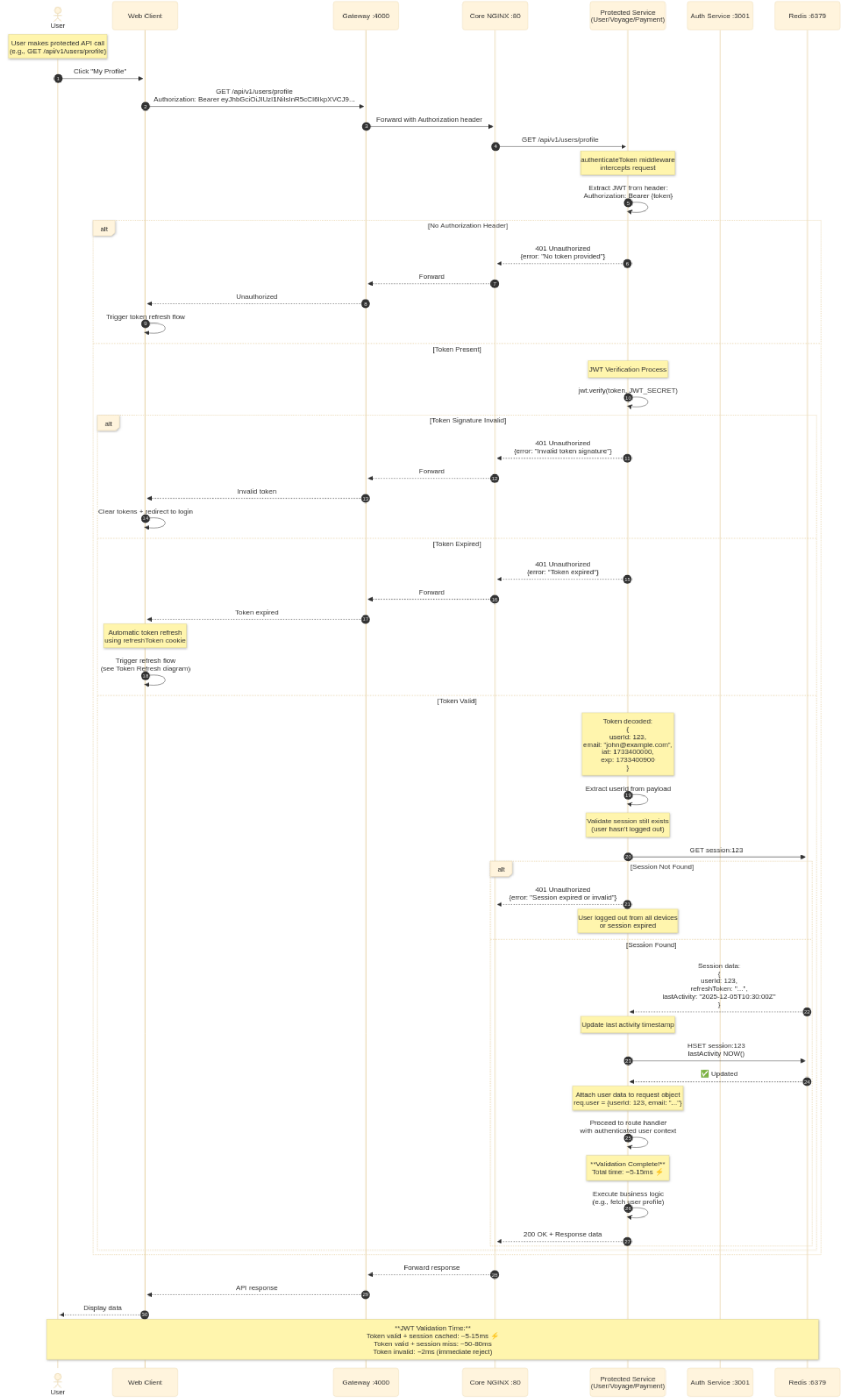
Flux de logout



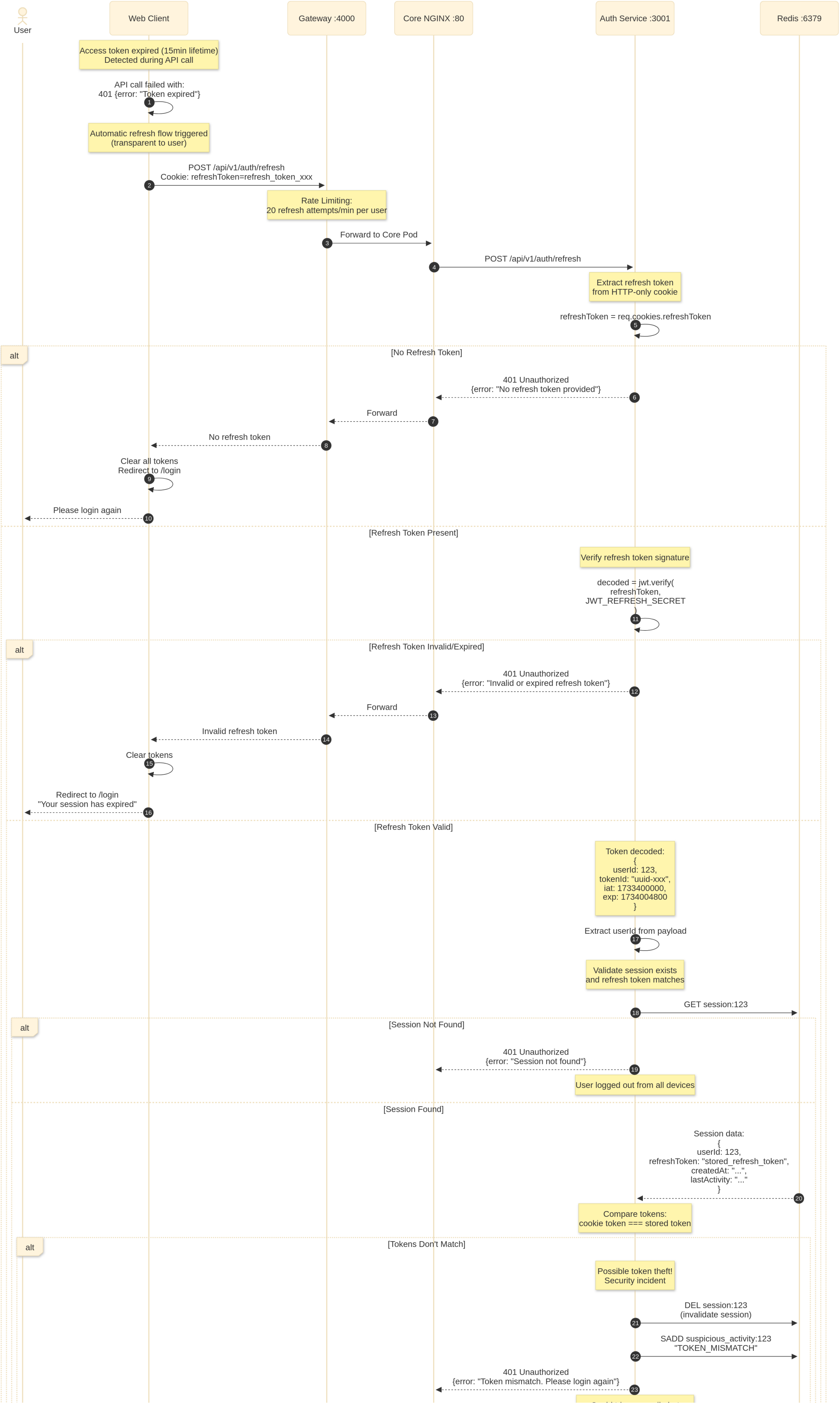
Flux de login



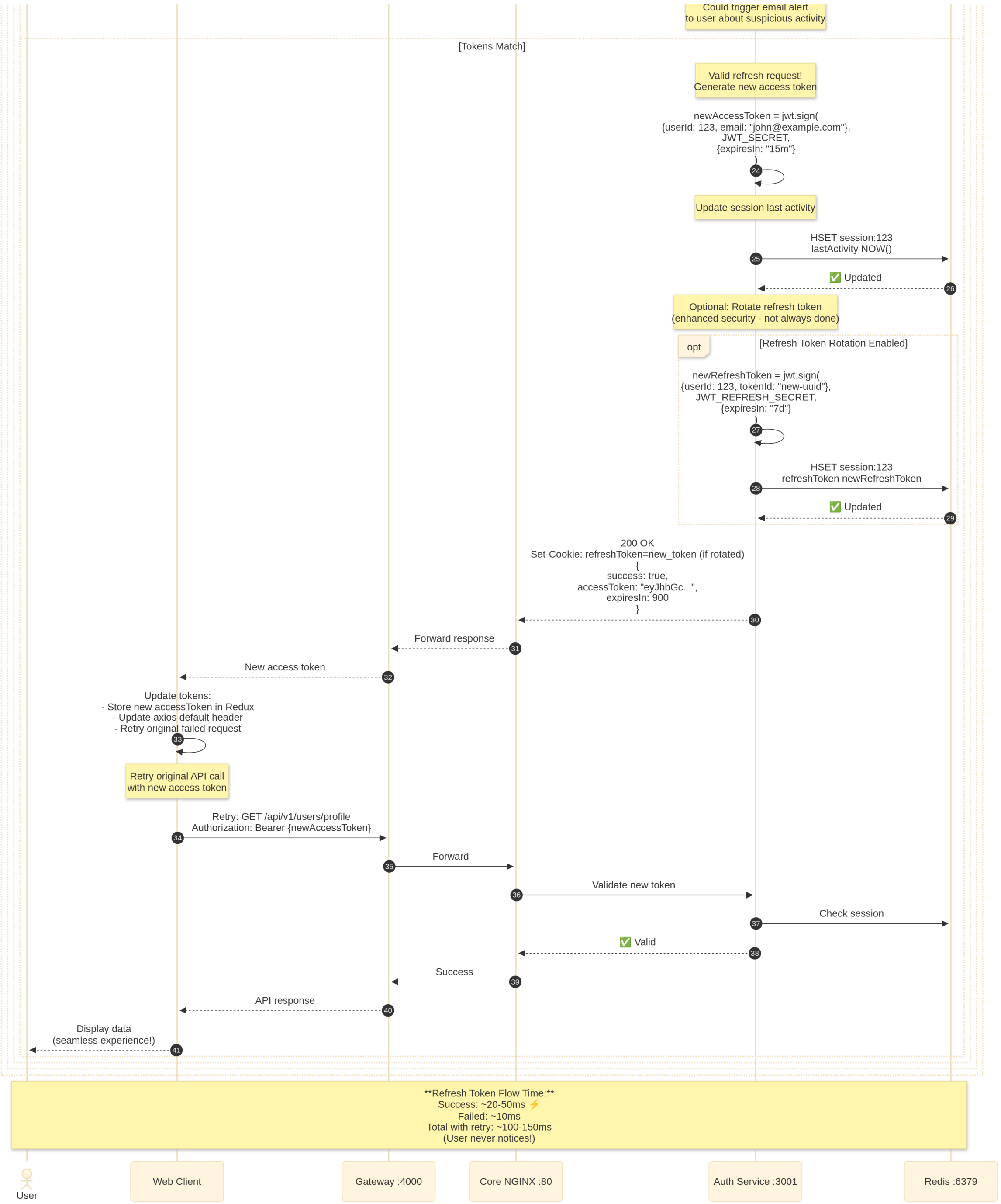
Flux JWT Token Validation


User
Web Client
Gateway :4000
Core NGINX :80
Protected Service (User/Voyage/Payment)
Auth Service :3001
Redis :6379
User makes protected API call (e.g., GET /api/v1/users/profile)
1 Click "My Profile"
2 GET /api/v1/users/profile Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
3 Forward with Authorization header
4 GET /api/v1/users/profile
authenticateToken middleware intercepts request
5 Extract JWT from header: Authorization: Bearer {token}
alt
[No Authorization Header]
6 401 Unauthorized {error: "No token provided"}
7 Forward
8 Unauthorized
9 Trigger token refresh flow
[Token Present]
JWT Verification Process
10 jwt.verify(token, JWT_SECRET)
alt
[Token Signature Invalid]
11 401 Unauthorized {error: "Invalid token signature"}
12 Forward
13 Invalid token
14 Clear tokens + redirect to login
[Token Expired]
15 401 Unauthorized {error: "Token expired"}
16 Forward
17 Token expired
Automatic token refresh using refreshToken cookie
18 Trigger refresh flow (see Token Refresh diagram)
[Token Valid]
Token decoded: { userId: 123, email: "john@example.com", iat: 1733400000, exp: 1733400900 }
19 Extract userId from payload
Validate session still exists (user hasn't logged out)
20 GET session:123
alt
[Session Not Found]
21 401 Unauthorized {error: "Session expired or invalid"}
User logged out from all devices or session expired
[Session Found]
22 Session data: { userId: 123, refreshToken: "...", lastActivity: "2025-12-05T10:30:00Z" }
Update last activity timestamp
23 HSET session:123 lastActivity NOW()
24 ✅ Updated
Attach user data to request object req.user = {userId: 123, email: "..."}
25 Proceed to route handler with authenticated user context
Validation Complete! Total time: ~5-15ms ⚡
26 Execute business logic (e.g., fetch user profile)
27 200 OK + Response data
28 Forward response
29 API response
30 Display data
JWT Validation Time:
Token valid + session cached: ~5-15ms ⚡
Token valid + session miss: ~50-80ms
Token invalid: ~2ms (immediate reject)

Flux JWT Token Refresh



PROF


Could trigger email alert
to user about suspicious activity
[Tokens Match]
Valid refresh request!
Generate new access token
newAccessToken = jwt.sign(
{userId: 123, email: "john@example.com"},
JWT_SECRET,
{expiresIn: "15m"}
)
24
Update session last activity
HSET session:123
lastActivity NOW()
25
✅ Updated
26
Optional: Rotate refresh token
(enhanced security - not always done)
opt
[Refresh Token Rotation Enabled]
newRefreshToken = jwt.sign(
{userId: 123, tokenId: "new-uuid"},
JWT_REFRESH_SECRET,
{expiresIn: "7d"}
)
27
HSET session:123
refreshToken newRefreshToken
28
✅ Updated
29
200 OK
Set-Cookie: refreshToken=new_token (if rotated)
{
success: true,
accessToken: "eyJhbGc...",
expiresIn: 900
}
30
Forward response
31
New access token
32
Update tokens:
- Store new accessToken in Redux
- Update axios default header
- Retry original failed request
33
Retry original API call
with new access token
Retry: GET /api/v1/users/profile
Authorization: Bearer {newAccessToken}
34
Forward
35
Validate new token
36
Check session
37
✅ Valid
38
Success
39
API response
40
Display data
(seamless experience!)
41
Refresh Token Flow Time:
Success: ~20-50ms ⚡
Failed: ~10ms
Total with retry: ~100-150ms
(User never notices!)
User
Web Client
Gateway :4000
Core NGINX :80
Auth Service :3001
Redis :6379

PROF

Passwords: bcrypt (salt rounds: 12)

JWT expiration: 15min (access), 7 jours (refresh)

Secrets: Variables environnement (JWT_SECRET)

Rate limiting: 10 tentatives login/min

- **User Service (:3002)**

  Stack technique : Node.js + Express + Prisma ORM

  Responsabilités :

  ✅ Gestion profils utilisateurs (CRUD)
  ✅ Préférences voyages (catégories: LEISURE, BUSINESS, etc.)
  ✅ Historique réservations

✅ Wishlist destinations
✅ Notifications préférences

API Endpoints : htpp://localhost:3002/

User Service - Complete Route List
Profile Routes
Base: /api/v1/users/profile

Method	**Endpoint**	**Description**	**Auth Required**	**Parameters**
GET	/	Get user profile and settings	✅ Yes	None (uses auth token)
POST	/:userId	Create user profile	× No	userId (param), firstName, lastName (body)
PUT	/	Update user profile and settings	✅ Yes	Profile data (body)
POST	/:userId/avatar	Upload avatar	× No	userId (param), avatar (file)
DELETE	/:userId	Delete user profile	× No	userId (param)

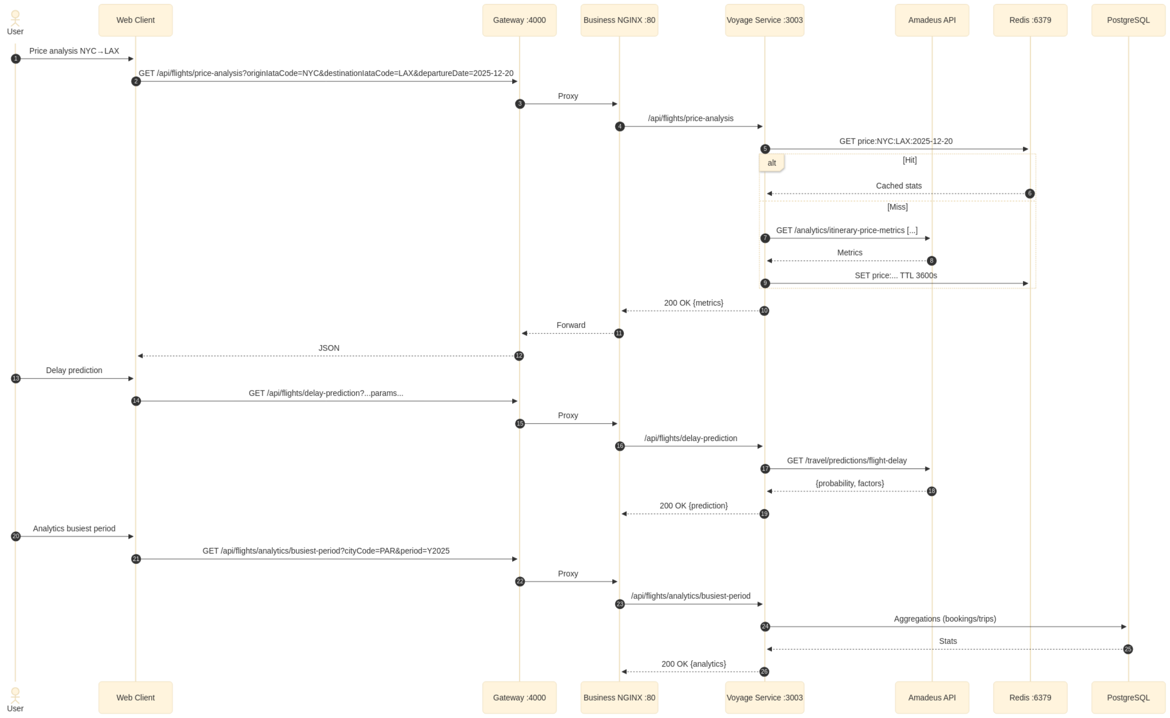
Activities Routes
Base: /api/v1/activities (Currently disabled - AmadeusService import issue)

Method	**Endpoint**	**Description**	**Required Parameters**
GET	/search	Search activities	(latitude + longitude) OR (north + west + south + east)
GET	/details/:activityId	Get activity details	activityId (param)
GET	/:activityId	Get activity by ID (alternative)	activityId (param)

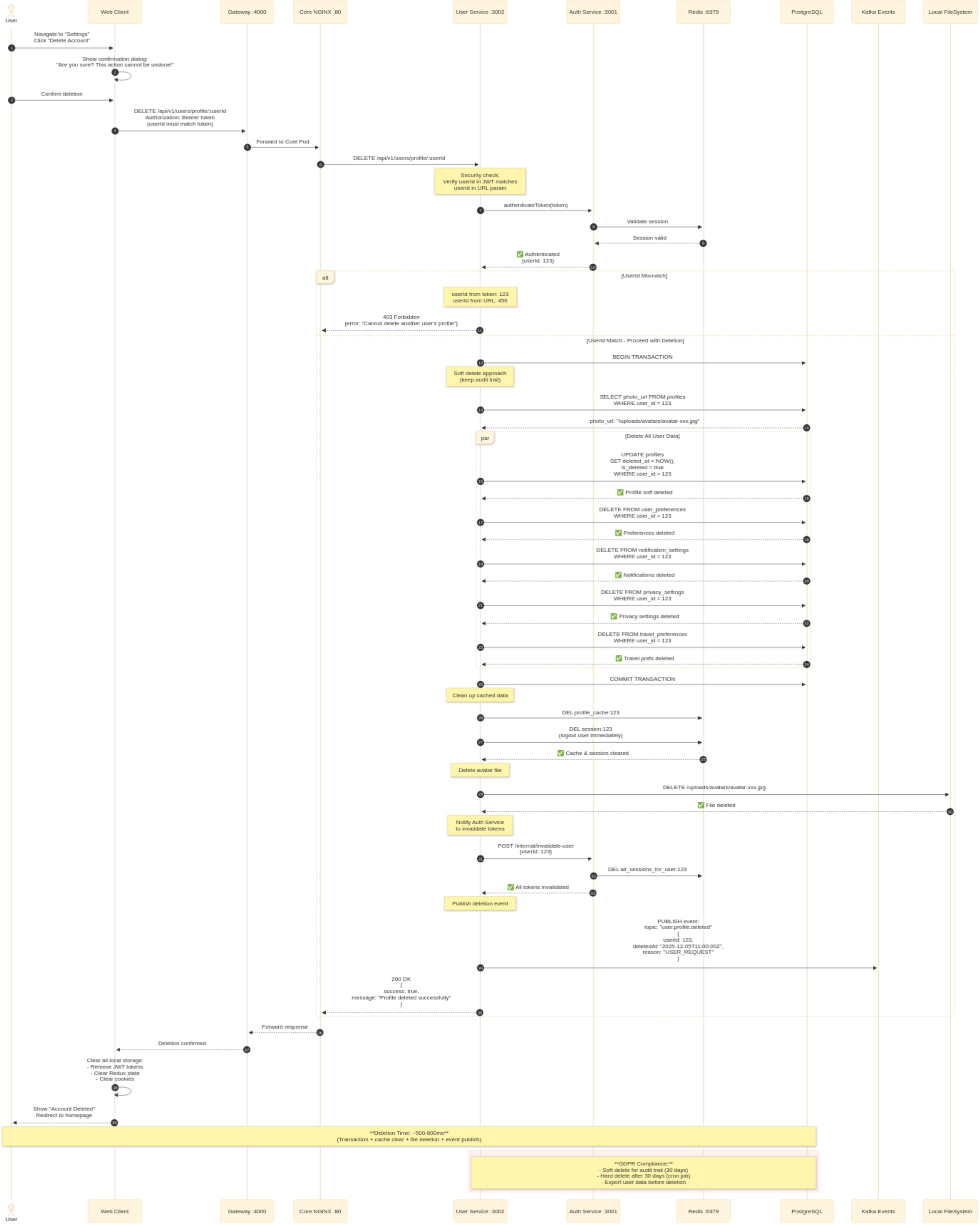
Health Route

Method	**Endpoint**	**Description**
GET	/health	Health check endpoint

Flux User

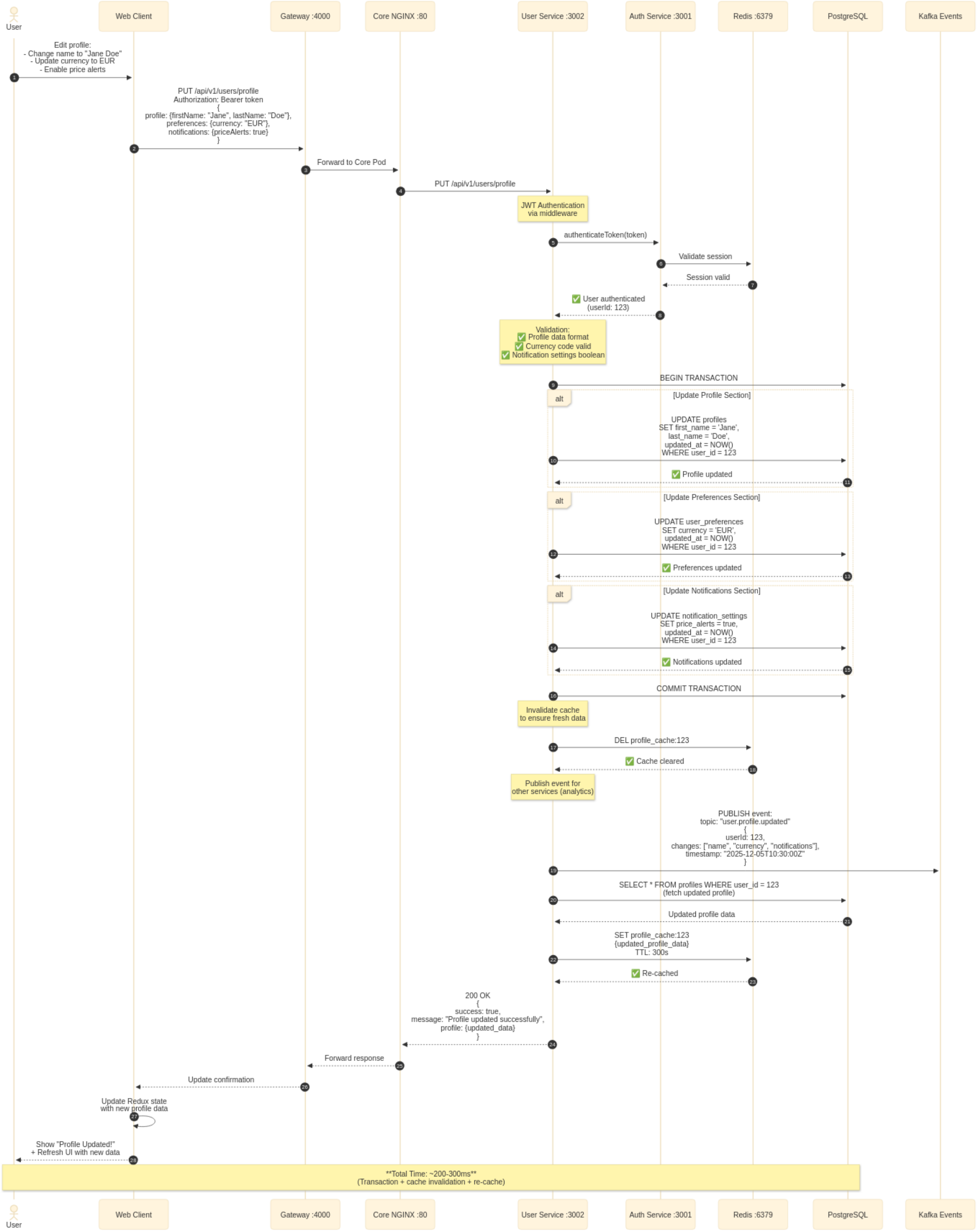

User
Web Client
Gateway :4000
Business NGINX :80
Voyage Service :3003
Amadeus API
Redis :6379
PostgreSQL
1 Price analysis NYC→LAX
2 GET /api/flights/price-analysis?originIataCode=NYC&destinationIataCode=LAX&departureDate=2025-12-20
3 Proxy
4 /api/flights/price-analysis
5 GET price:NYC:LAX:2025-12-20
alt
[Hit]
6 Cached stats
[Miss]
7 GET /analytics/itinerary-price-metrics [...]
8 Metrics
9 SET price:... TTL 3600s
10 200 OK {metrics}
11 Forward
12 JSON
13 Delay prediction
14 GET /api/flights/delay-prediction?...params...
15 Proxy
16 /api/flights/delay-prediction
17 GET /travel/predictions/flight-delay
18 {probability, factors}
19 200 OK {prediction}
20 Analytics busiest period
21 GET /api/flights/analytics/busiest-period?cityCode=PAR&period=Y2025
22 Proxy
23 /api/flights/analytics/busiest-period
24 Aggregations (bookings/trips)
25 Stats
26 200 OK {analytics}
User
Web Client
Gateway :4000
Business NGINX :80
Voyage Service :3003
Amadeus API
Redis :6379
PostgreSQL

Flux Delete User



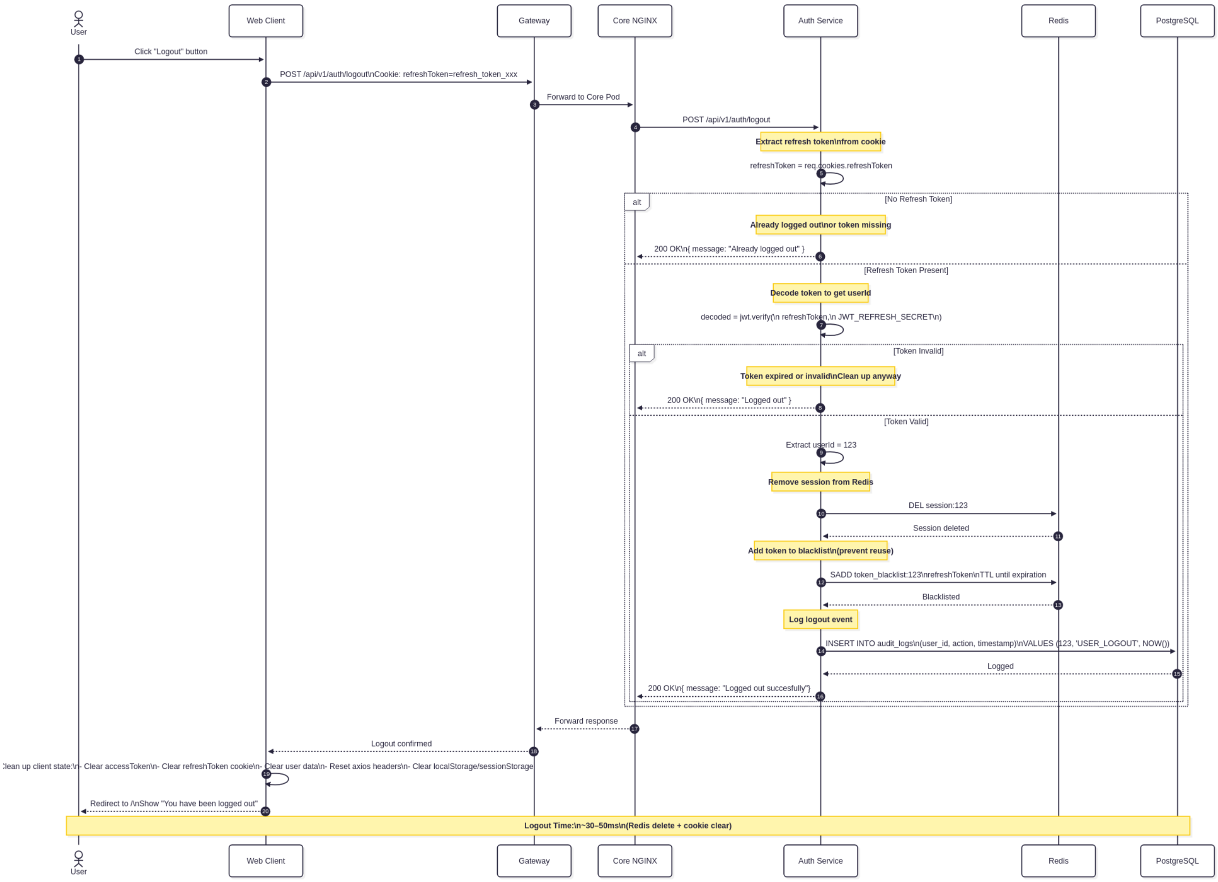
PROF

Flux Update User

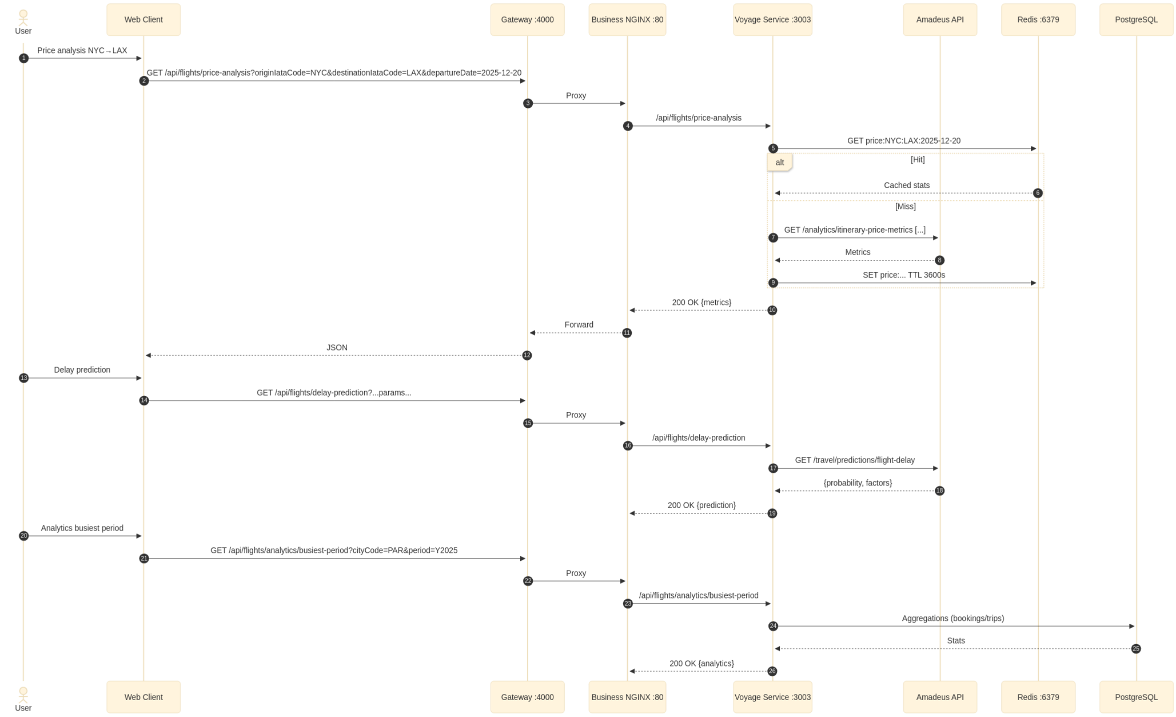


PROF

Flux Update Avatar User

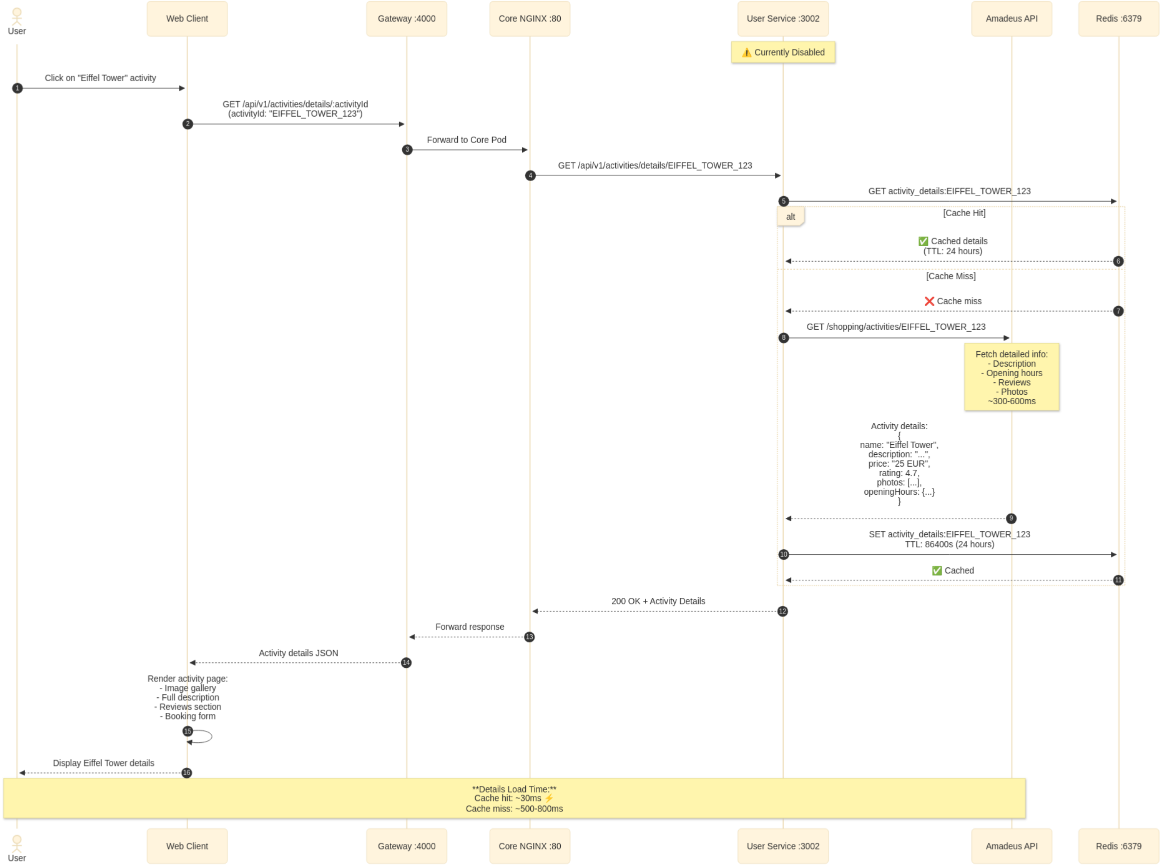

User
Web Client
Gateway
Core NGINX
Auth Service
Redis
PostgreSQL
1 Click "Logout" button
2 POST /api/v1/auth/logout\nCookie: refreshToken=refresh_token_xxx
3 Forward to Core Pod
4 POST /api/v1/auth/logout
Extract refresh token\nfrom cookie
5 refreshToken = req.cookies.refreshToken
alt
[No Refresh Token]
Already logged out\nor token missing
6 200 OK\n{ message: "Already logged out" }
[Refresh Token Present]
Decode token to get userId
7 decoded = jwt.verify(\n refreshToken,\n JWT_REFRESH_SECRET\n)
alt
[Token Invalid]
Token expired or invalid\nClean up anyway
8 200 OK\n{ message: "Logged out" }
[Token Valid]
9 Extract userId = 123
Remove session from Redis
10 DEL session:123
11 Session deleted
Add token to blacklist\n(prevent reuse)
12 SADD token_blacklist:123\nrefreshToken\nTTL until expiration
13 Blacklisted
Log logout event
14 INSERT INTO audit_logs\n(user_id, action, timestamp)\nVALUES (123, 'USER_LOGOUT', NOW())
15 Logged
16 200 OK\n{ message: "Logged out succesfully"}
17 Forward response
18 Logout confirmed
19 :lean up client state:\n- Clear accessToken\n- Clear refreshToken cookie\n- Clear user data\n- Reset axios headers\n- Clear localStorage/sessionStorage
20 Redirect to /\nShow "You have been logged out"
Logout Time:\n~30–50ms\n(Redis delete + cookie clear)
User
Web Client
Gateway
Core NGINX
Auth Service
Redis
PostgreSQL

Flux User Activities


User
Web Client
Gateway :4000
Business NGINX :80
Voyage Service :3003
Amadeus API
Redis :6379
PostgreSQL
1 Price analysis NYC→LAX
2 GET /api/flights/price-analysis?originIataCode=NYC&destinationIataCode=LAX&departureDate=2025-12-20
3 Proxy
4 /api/flights/price-analysis
5 GET price:NYC:LAX:2025-12-20
alt
[Hit]
6 Cached stats
[Miss]
7 GET /analytics/itinerary-price-metrics [...]
8 Metrics
9 SET price:... TTL 3600s
10 200 OK {metrics}
11 Forward
12 JSON
13 Delay prediction
14 GET /api/flights/delay-prediction?...params...
15 Proxy
16 /api/flights/delay-prediction
17 GET /travel/predictions/flight-delay
18 {probability, factors}
19 200 OK {prediction}
20 Analytics busiest period
21 GET /api/flights/analytics/busiest-period?cityCode=PAR&period=Y2025
22 Proxy
23 /api/flights/analytics/busiest-period
24 Aggregations (bookings/trips)
25 Stats
26 200 OK {analytics}
User
Web Client
Gateway :4000
Business NGINX :80
Voyage Service :3003
Amadeus API
Redis :6379
PostgreSQL

PROF

Flux User activities Details



Métriques Core Pod

Métrique	Valeur Cible	Réel (Dev)
Auth Latency	< 20ms	~8ms (localhost!) ⚡
JWT Validation	< 10ms	~5ms (Redis cache)
User Profile Load	< 50ms	~30ms (cache hit)
Throughput	1000 req/s	~800 req/s
Memory	< 1GB	~750MB

PROF

Business pod - Logique Métier & Intégrations

Rôle : Services métier (voyages, IA, paiements), intégrations API externes, traitement événements

Services Inclus

- **Voyage Service (:3003)**
  Stack technique : Node.js + Express + Amadeus API + Prisma

  Responsabilités :

✅ Recherche de vols (intégration Amadeus Flight Offers Search)
✅ Prix en temps réel (cache Redis 30min)
✅ Gestion réservations (statuts: PENDING, CONFIRMED, CANCELLED)
✅ Historique voyages (par utilisateur)
✅ Événements Kafka (voyage.booked, voyage.cancelled)

API Endpoints : http://localhost:3003/

Complete Route List for Voyage Service
Health & Monitoring Routes
Base: /api/health

Method	EndpEndpointoint	Description
GET	/	Full health check with all services
GET	/ready	Readiness probe
GET	/live	Liveness probe
GET	/cache	Cache statistics and status

Flight Routes
Base: /api/flights

Search & Discovery

Method	Endpoint	Description	Required Parameters
GET	/search	Search flights with mapped DTOs	originLocationCode, destinationLocationCode, departureDate
GET	/destinations Search	flight destinations (inspiration)	origin
GET	/price-analysis	Flight price analysis	originIataCode, destinationIataCode, departureDate
GET	/inspiration	Flight inspiration search	origin
GET	/cheapest-dates	Flight cheapest date search	origin, destination

PROF

Flight Information

Method	Endpoint	Description	Required Parameters
GET	/status	Flight status	carrierCode, flightNumber, scheduledDepartureDate

Method	Endpoint	Description	Required Parameters
GET	/delay-prediction	Flight delay prediction	Multiple (see code)
GET	/checkin-links	Flight check-in links	airlineCode
GET	/seatmap	Seat map display	flightOfferId
GET	/branded-fares	Branded fares upsell	flightOfferId

Analytics

Method	Endpoint	Description	Required Parameters
GET	/analytics/most-traveled	Most traveled destinations	originCityCode, period
GET	/analytics/most-booked	Most booked destinations	originCityCode, period
GET	/analytics/busiest-period	Busiest traveling period	cityCode, period

Booking & Orders

Method	Endpoint	Description	Required Parameters
POST	/choice-prediction	Flight choice prediction	Flight offers (body)
POST	/offers/pricing	Flight offers price	Flight offers (body)
POST	/availabilities	Flight availabilities search	originDestinations, travelers, sources (body)
POST	/orders	Flight create orders	Order data (body)
GET	/orders/:orderId	Flight order management	orderId (param)

PROF

Hotel Routes
Base: /api/hotels

Method	Endpoint	Description	Cache TTL	Required Parameters
GET	/search	Search hotels	5 min	checkInDate, checkOutDate, (cityCode OR latitude+longitude)
GET	/details/:hotelId	Get hotel details	15 min	hotelId (param)

Method	Endpoint	Description	Cache TTL	Required Parameters
GET	/ratings	Hotel ratings	-	TBD
POST	/bookings	Hotel booking	-	Booking data (body)
GET	/list	Hotel list	1 hour	TBD
GET	/:hotelId/images	Get hotel images	-	hotelId (param)

Location Routes
Base: /api/locations

Method	Endpoint	Description	Required Parameters
GET	/search	Search locations (cities, airports, etc.)	keyword
GET	/airports	Search airports	keyword

Transfer Routes
Base: /api/transfers

Method	Endpoint	Description	Required Parameters
GET	/search	Transfer search	startDateTime, passengers
POST	/bookings	Transfer booking	Booking data (body)
GET	/orders/:orderId	Transfer order management	orderId (param)

Airline Routes
Base: /api/airlines

Method	Endpoint	Description	Required Parameters
GET	/lookup	Airline code lookup	None (optional: airlineCodes, IATACode, ICAOCode)
GET	/routes	Airline routes	airlineCode

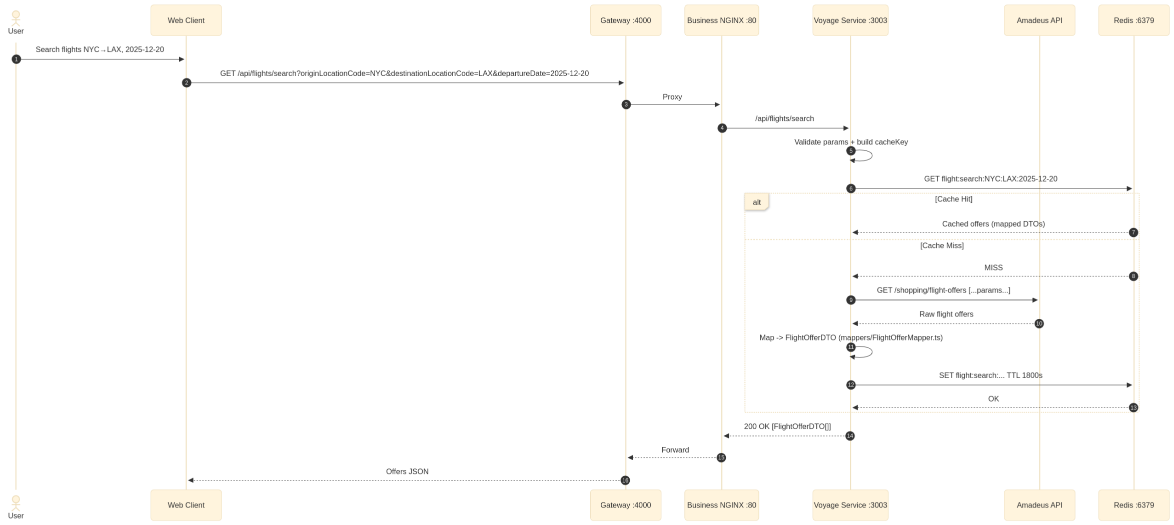
Airport Routes
Base: /api/airports

Method	Endpoint	Description	Required Parameters
GET	/on-time-performance	Airport on-time performance	airportCode, date
GET	/nearest	Airport nearest relevant	latitude, longitude

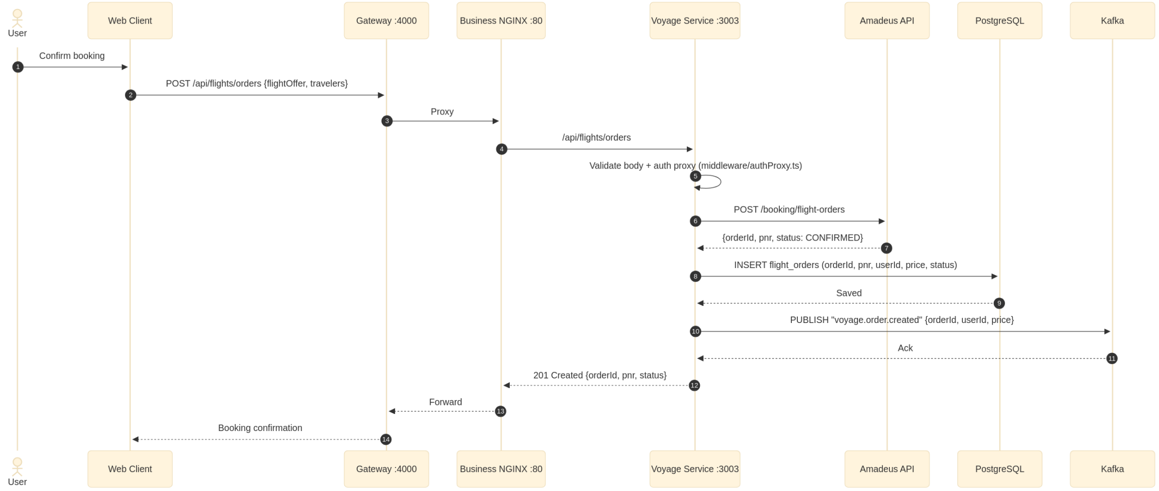
Method	Endpoint	Description	Required Parameters
GET	/routes	Airport routes	departureAirportCode

Intégration Amadeus :

Flux de recherche de vols :

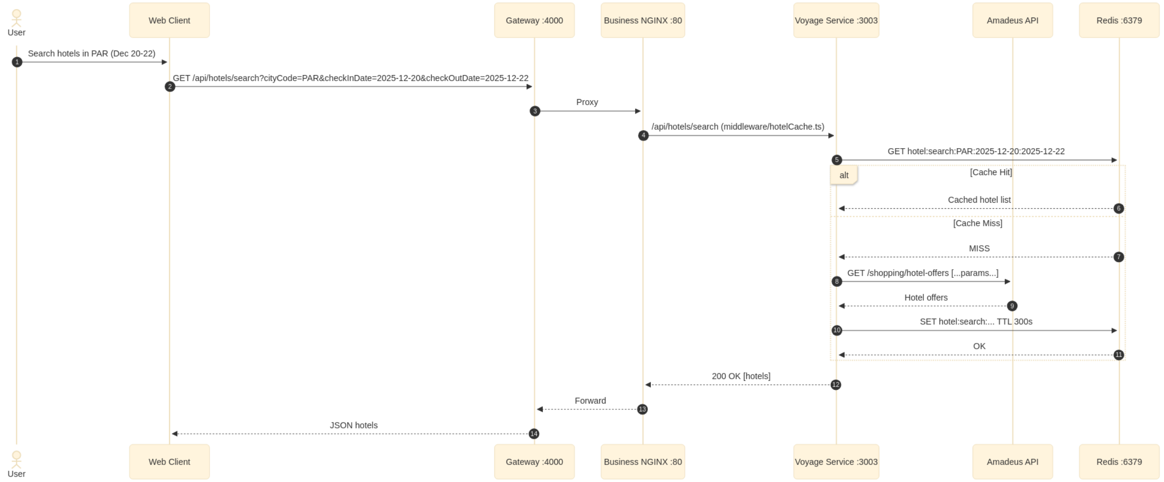

User
Web Client
Gateway :4000
Business NGINX :80
Voyage Service :3003
Amadeus API
Redis :6379
1
Search flights NYC→LAX, 2025-12-20
2
GET /api/flights/search?originLocationCode=NYC&destinationLocationCode=LAX&departureDate=2025-12-20
3
Proxy
4
/api/flights/search
5
Validate params + build cacheKey
6
GET flight:search:NYC:LAX:2025-12-20
alt
[Cache Hit]
7
Cached offers (mapped DTOs)
[Cache Miss]
8
MISS
9
GET /shopping/flight-offers [...params...]
10
Raw flight offers
11
Map -> FlightOfferDTO (mappers/FlightOfferMapper.ts)
12
SET flight:search:... TTL 1800s
13
OK
14
200 OK [FlightOfferDTO[]]
15
Forward
16
Offers JSON
User
Web Client
Gateway :4000
Business NGINX :80
Voyage Service :3003
Amadeus API
Redis :6379

Flux de reservation de vols

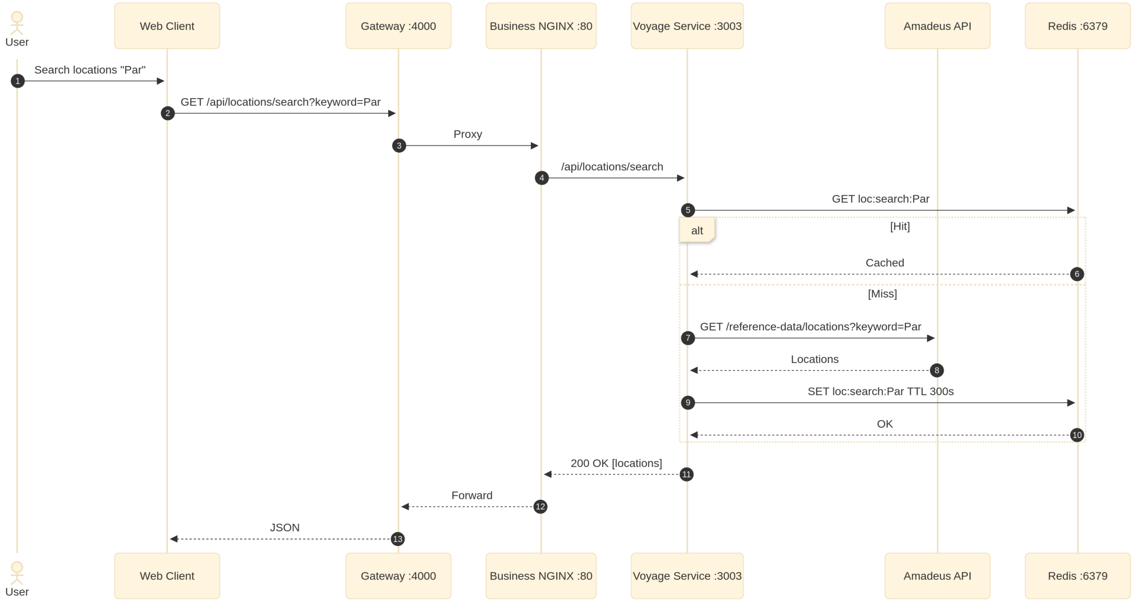

User
Web Client
Gateway :4000
Business NGINX :80
Voyage Service :3003
Amadeus API
PostgreSQL
Kafka
1
Confirm booking
2
POST /api/flights/orders {flightOffer, travelers}
3
Proxy
4
/api/flights/orders
5
Validate body + auth proxy (middleware/authProxy.ts)
6
POST /booking/flight-orders
7
{orderId, pnr, status: CONFIRMED}
8
INSERT flight_orders (orderId, pnr, userId, price, status)
9
Saved
10
PUBLISH "voyage.order.created" {orderId, userId, price}
11
Ack
12
201 Created {orderId, pnr, status}
13
Forward
14
Booking confirmation
User
Web Client
Gateway :4000
Business NGINX :80
Voyage Service :3003
Amadeus API
PostgreSQL
Kafka

PROF

Flux de recherche d'hotels



Flux de location



PROF

Connexions :

- **AI Service (:3004)**
  Stack technique : Python (FastAPI) + TensorFlow + OpenAI API + MinIO

  Responsabilités :

  ✅ Recommandations personnalisées (ML model: collaborative filtering)
  ✅ Génération d'itinéraires (OpenAI GPT-4)
  ✅ Prédiction de prix (time series forecasting)
  ✅ Analyse de sentiment (avis destinations)
  ✅ Chatbot voyage (RAG sur base de connaissances)

  API Endpoints : http:/localhost:3004/

  AI Service - Complete Route List
  Recommendations Routes
  Base: /api/v1/recommendations

Method	Endpoint	Description	Auth Required	Required Parameters
GET	/	Get travel recommendations	× No	cityCodes

Optional Parameters:
travelerCountryCode - Country code of the traveler
destinationCountryCode - Country code of the destination

Predictions Routes
Base: /api/v1/predictions

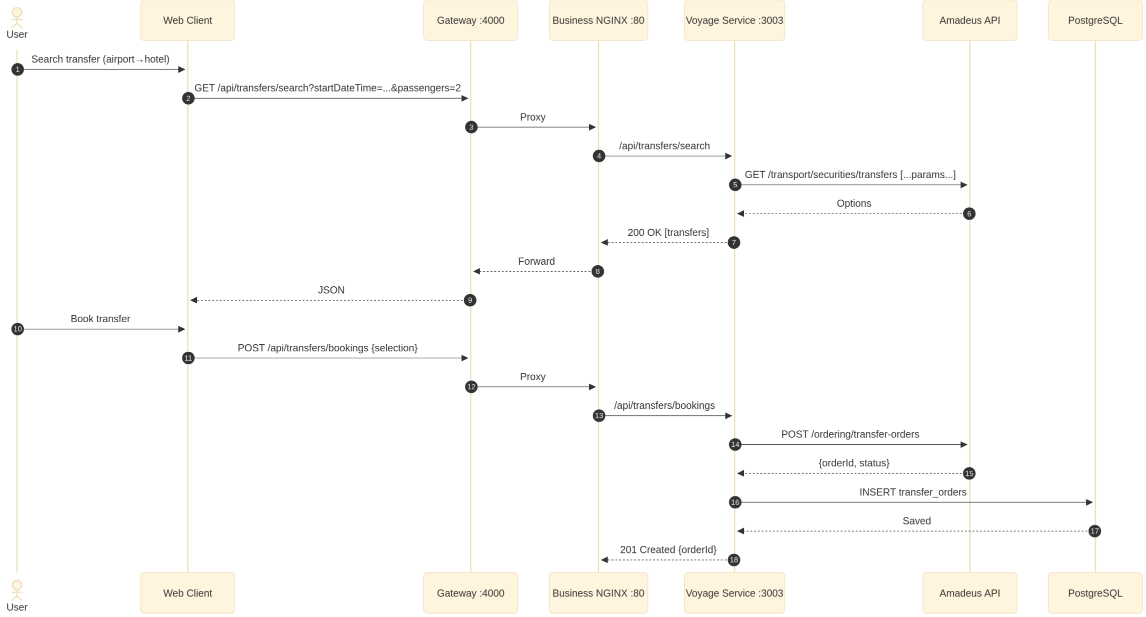
Method	Endpoint	Description	Auth Required	Required Parameters
GET	/trip-purpose	Predict trip purpose	× No	originLocationCode, destinationLocationCode, departureDate, searchDate,returnDate(optional)

PROF

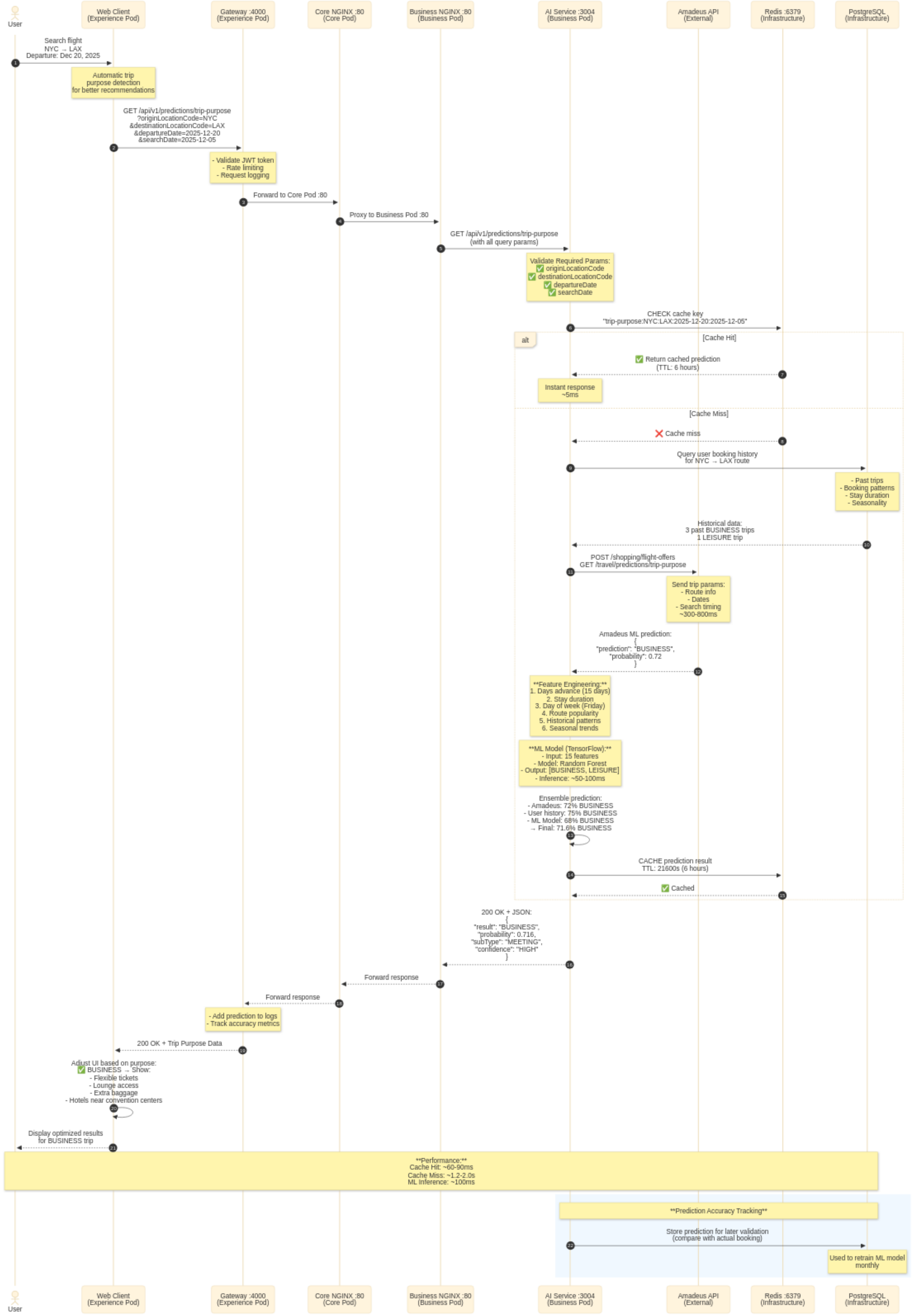
Health Route

Method	Endpoint	Description
GET	/health	Health check endpoint

Flux de recommendation



Flux de prediction


User
Web Client (Experience Pod)
Gateway :4000 (Experience Pod)
Core NGINX :80 (Core Pod)
Business NGINX :80 (Business Pod)
AI Service :3004 (Business Pod)
Amadeus API (External)
Redis :6379 (Infrastructure)
PostgreSQL (Infrastructure)
1
Search flight NYC → LAX Departure: Dec 20, 2025
Automatic trip purpose detection for better recommendations
2
GET /api/v1/predictions/trip-purpose ?originLocationCode=NYC &destinationLocationCode=LAX &departureDate=2025-12-20 &searchDate=2025-12-05
- Validate JWT token - Rate limiting - Request logging
3
Forward to Core Pod :80
4
Proxy to Business Pod :80
5
GET /api/v1/predictions/trip-purpose (with all query params)
Validate Required Params: ✅ originLocationCode ✅ destinationLocationCode ✅ departureDate ✅ searchDate
6
CHECK cache key "trip-purpose:NYC:LAX:2025-12-20:2025-12-05"
alt
[Cache Hit]
7
✅ Return cached prediction (TTL: 6 hours)
Instant response ~5ms
[Cache Miss]
8
❌ Cache miss
9
Query user booking history for NYC → LAX route
- Past trips - Booking patterns - Stay duration - Seasonality
10
Historical data: 3 past BUSINESS trips 1 LEISURE trip
11
POST /shopping/flight-offers GET /travel/predictions/trip-purpose
Send trip params: - Route info - Dates - Search timing ~300-800ms
12
Amadeus ML prediction: { "prediction": "BUSINESS", "probability": 0.72 }
Feature Engineering: 1. Days advance (15 days) 2. Stay duration 3. Day of week (Friday) 4. Route popularity 5. Historical patterns 6. Seasonal trends
ML Model (TensorFlow): - Input: 15 features - Model: Random Forest - Output: [BUSINESS, LEISURE] - Inference: ~50-100ms
13
Ensemble prediction: - Amadeus: 72% BUSINESS - User history: 75% BUSINESS - ML Model: 68% BUSINESS → Final: 71.6% BUSINESS
14
CACHE prediction result TTL: 21600s (6 hours)
15
✅ Cached
16
200 OK + JSON: { "result": "BUSINESS", "probability": 0.716, "subType": "MEETING", "confidence": "HIGH" }
17
Forward response
18
Forward response
- Add prediction to logs - Track accuracy metrics
19
200 OK + Trip Purpose Data
20
Adjust UI based on purpose: ✅ BUSINESS → Show: - Flexible tickets - Lounge access - Extra baggage - Hotels near convention centers
21
Display optimized results for BUSINESS trip
Performance: Cache Hit: ~60-90ms Cache Miss: ~1.2-2.0s ML Inference: ~100ms
Prediction Accuracy Tracking
22
Store prediction for later validation (compare with actual booking)
Used to retrain ML model monthly

PROF

- **Payment Service (:3005)**

  Stack technique : Node.js + Express + Stripe API + Prisma

Responsabilités :

✅ Création Payment Intent (Stripe)
✅ Webhook Stripe (confirmation paiements)
✅ Gestion remboursements
✅ Historique transactions
✅ Sécurité PCI-DSS (pas de stockage carte)

API Endpoints : http://localhost:3005/

Flux de paiement :
A venir non défini
Paiment

Sécurité :

🔒 PCI-DSS Compliant (Stripe gère les cartes)
🔐 Webhook signature (validation Stripe-Signature header)
🚫 Idempotency keys (évite double paiement)
📝 Audit logs (toutes transactions loggées)
Exemple code Stripe :

Métriques Business Pod

Métrique	Valeur Cible	Réel (Dev)
Voyage Search	< 2s	~1.5s (cache hit) / ~3s (Amadeus)
AI Recommendations	< 500ms	~350ms (model inference)
Payment Intent	< 1s	~800ms (Stripe API)
Throughput	500 req/s	~400 req/s
Memory	< 2GB	~1.5GB

PROF

Infratructuture - base de donnée , S3

- **Redis (:6379)**
  Rôle : Cache applicatif + stockage sessions

Utilisations :

Configuration :

Avantages :
⚡ Latency < 1ms (in-memory)
📈 Réduit charge Postgres de 60-80%

🔄 TTL automatique (sessions expirées supprimées)
🔧 Supervisor

Rôle : Process manager (PID 1 du container)

Configuration :

Pourquoi Supervisor ?

✅ Gère 3 processus simultanés (nginx + auth + user)
✅ Auto-restart on crash (container ne meurt pas)
✅ Logs centralisés (/var/log/supervisor/)
✅ Ordre de démarrage contrôlé (priorities)

Solution Big Pods :

Gain : -90% latence sur Auth/User ! 🚀

Exemple Concret : Validation JWT
Scénario : User fait une recherche de vol

Performance totale : ~600-800ms (dont 90% API Amadeus)

Si Auth était en microservice séparé :

Validation JWT: 50-100ms au lieu de 5-15ms
+100ms sur CHAQUE requête API !

---

6. Deploiement

1. Remplir toutes les conditions du prérequis.
2. Copier le fichier d'environnement
3. lancer le script de generation des secrets: dreamscape-infra/scripts/bigpods/generate-dev-secrets.sh
4. Ajouter ces clées personneles pour les différentes API
5. Lancer le script de dev dev-bigpods.sh
6. Lancer le script de deploiment deploy-bigpods.sh

PROF