UNIVERSIDADE DE AVEIRO

Departamento de Eletrónica, Telecomunicações e Informática

InFlow

Payment & Billing System

Project Report

Engenharia de Software

Carolina Sofia Pereira da Silva • 113475

Luana Carolina Cunha Reis • 131193

Francisco Ribeiro Arada • 131117

Matilde Moital Portugal Sampaio Teixeira • 108193

Miguel Marques Vieira • 85095 (desistiu)

2025/2026

Contents

Introduction **5**
- **Overview of the project** . . . 5

Product concept and requirements **5**
- **Vision statement** . . . 5
- **Personas** . . . 6
- **Supported Scenarios (user stories)** . . . 11
  - Epic 1 - Invoice . . . 11
  - Epic 2 - Payments . . . 15
  - Epic 3 - Notifications . . . 17
  - Epic 4 - Audit . . . 19
  - Epic 5 - Sales Representative Functions . . . 21
  - Epic 6 - Authentication . . . 22
  - Without Epic . . . 26

Architecture notebook **30**
- **Key quality requirements** . . . 30
  - Functional Requirements . . . 30
  - Non-Functional Requirements . . . 32

System architecture **35**
- Components . . . 35
- Technology Stack . . . 36
- Interactions . . . 37
  - Communication Protocols . . . 39
  - Event-Driven Notification Policies . . . 39
- Dependencies . . . 40
- Critical flows . . . 41
  - Functional flows (User Stories) . . . 42
  - Architectural flows . . . 43

Information Model **46**

Production Configuration and Deployment Pipeline **48**
- Overview of the Production Architecture . . . 48
- Environment Variables . . . 48
- Deployment Pipeline (GitHub Actions) . . . 49

CI/CD Workflow Structure . . . 49
Environment Configuration . . . 51
Health Checks and Validation . . . 51
Terraform Provisioning . . . 52
Provider and Network . . . 52
Image Builds . . . 53
Replicas . . . 53
Kong Gateway Routing . . . 54
Observability Stack . . . 54
Java CI Workflow . . . 55
End-to-End Testing Workflow and Allure Reporting . . . 55
Testing and Reporting Workflow . . . 55
Workflow Definition . . . 56
Summary . . . 60
Conclusion . . . 60

Installation Guide **60**
Purpose and Scope . . . 60
High-Level Architecture . . . 60
System Preparation . . . 61
Base System Updates . . . 61
Container Runtime Installation . . . 62
Docker Engine . . . 62
Docker Compose . . . 63
Development Runtime Installation . . . 63
Java Development Kit . . . 63
Node.js Runtime . . . 64
Source Code Repository . . . 64
Environment Configuration . . . 64
System Deployment . . . 68
Service Initialization . . . 68
Verification . . . 69
Service Access Points . . . 69
Backend Build Configuration . . . 70
Frontend Build Configuration . . . 70
Kong Gateway Configuration . . . 70
Observability Infrastructure . . . 70
Weaviate Configuration . . . 70

Ollama Configuration 71

Conclusion **72**
Key Achievements 72
Project Impact 73
Final Remarks 73

References and Resources **75**

Introduction

Overview of the project

This project is to be developed as part of the Software Engineering (ES) course. The main goal is to design an accessible system, following a micro-service approach, so that it can easily integrate and communicate with the services created by other groups within the course.

The expected product is a **Payment and Billing system**, which aims to automate and manage the entire billing and payment lifecycle. Its purpose is to invoice customers, process payments and track financial transactions, serving customers, finance staff, sales representatives, and auditors.

InFlow serves different user types:

- **Billing Administrator/ Finance Staff**, responsible for generating and sending invoices, applying payments, and managing accounts.
- **Customer/Client,** able to view invoices, make payments, and access history through a dedicated platform;
- **Sales representatives,** who can review a customer's account status, payment history and receipts;
- **Auditors/Compliance Officers,** with access to financial records and audit trails;

With features like invoice generation, payment processing, recurrent billing, reporting dashboards, and audit trails, the **InFlow** ensures a centralized and efficient solution for financial management, whilst providing usability and compliance.

Product concept and requirements

Vision statement

The **InFlow** platform delivers a seamless and centralized solution for managing the complete invoicing and payment experience. Connects customers and finance staff through a unified digital ecosystem that simplifies invoice creation, payment application, online viewing, and online payment processing.

The platform addresses key challenges in billing and finance, including incorrect payment allocation, delayed invoice access, and inefficient manual processes. It ensures that critical financial operations like issuing invoices, applying payments correctly, viewing or settling invoices online are performed accurately and reliably.

For **Billing Administrators and Finance Staff**, the platform provides an intuitive interface to issue invoices to customers (US2) and automatically apply payments to the correct invoices (US4), reducing manual errors and increasing operational efficiency.

For **Customers**, the platform offers fast and secure access to their invoices (US13), allowing them to review their payment history and pay invoices online (US14) with convenience and transparency.

Core features include:

- Invoice creation and management with an Automated Payment Application
- Online invoice viewing with complete payment history
- Secure online payment processing
- User-friendly dashboards for finance staff and customers

By aligning the needs of finance teams and customers, **InFlow** creates a reliable, efficient, and user-friendly invoicing and payment experience, ensuring accuracy, convenience, and operational excellence.

Personas

Billing Administrator/Finance Staff

Name: Maria Pereira
Age: 50
Gender: Female
Occupation: Billing Administrator
Location: Lisbon, Portugal
Salary: Medium
Technology Familiarity: Comfortable with basic accounting software, but not very technical Maria handles the billing for an SME (Small and Medium-sized Enterprise) and needs to keep track of hundreds of active invoices simultaneously. She is meticulous and organized,

likes to have control over the company's activities and
financial status. She spends most of her day in the billing system, generating invoices, validating payments, and handling bank reconciliations.

Goals:

1. Issue and send invoices to clients quickly and accurately
2. Monitor the status of invoices (paid, overdue, pending)
3. Apply payments
4. Manage client accounts
5. Reconcile payments with bank statements
6. Handle payment disputes or refunds

Frustrations:

1. Time-consuming manual processes
2. Human errors in invoice processing that require end-of-month corrections
3. Difficulty tracking real-time payment statuses from clients
4. Lack of automatic alerts for overdue invoices

Behaviors:

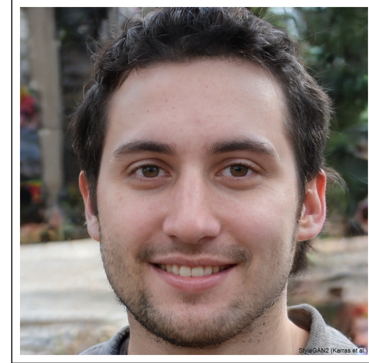
1. Checks the KPI dashboard first thing in the morning
2. Accesses the billing system multiple times a day to monitor invoice statuses
3. Prefers automated processes but likes to manually verify higher-value invoices
4. Works standard office hours, managing multiple clients simultaneously
5. Performs bank reconciliations at the end of the day

Needs:

1. A quick way to issue single or recurring invoices from subscriptions or contracts
2. Automatic alerts for overdue or failed payments
3. Dashboard with key indicators (issued, pending, paid, overdue invoices, etc.)
4. Functionality to reconcile payments with bank statements

5. Audit trail of changes to invoices and client accounts

Customer/Client



Name: Guilherme Manuel
Age: 33
Gender: Male
Occupation: Financial Manager
Location: Porto, Portugal
Salary: High
Technology Familiarity: Uses various management software in his daily work but often questions their complexity Guilherme is responsible for ensuring that his company's accounts payable are settled efficiently and on time. On a daily basis, he receives multiple invoices that he must process by verifying and paying them, while keeping a record of every action taken.

Goals:

1. View pending invoices
2. Review his payment history, both in summary and detailed form
3. Make payments quickly and intuitively
4. Manage and update his billing information

Frustrations:

1. The need to communicate directly with administrative services to resolve billing issues, which is very time-consuming
2. Difficulty monitoring the status of an invoice
3. The need to use payment methods not supported by many platforms
4. Security risks associated with using certain platforms

Behaviors:

1. Makes online payments
2. Checks payment history frequently
3. Primarily uses web applications on a desktop

Needs:

1. A page where he can view all payment-related information at a glance
2. A secure way to use the various payment methods he previously used on other platforms
3. Receive email notifications related to account activity
4. Access to his invoice and platform activity history, with the option to export it
5. Ability to manage billing information automatically

Sales Representative

Name: Carolina Miranda
Age: 28
Gender: Female
Occupation: Account / Sales Manager
Location: Lisbon, Portugal
Salary: High
Technology Familiarity: Very comfortable with mobile applications and CRMs (Customer Relationship Management) Carolina manages multiple client accounts and needs visibility into each client's financial status before offering new service or product proposals. She often works on the go and requires quick responses, especially during meetings.

Goals:

1. Check if a client has overdue invoices
2. Avoid selling to clients with a history of non-payment
3. Access information within seconds, even outside the office

Frustrations:

1. Having to request information from the finance team
2. Waiting for responses while with a client
3. Systems that do not function properly on mobile devices

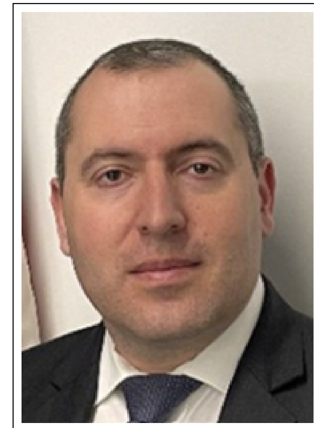
Behaviors:

1. Uses laptop or mobile phone to check data
2. Performs a quick review before each meeting or proposal
3. Accesses information in read-only mode (does not edit data)

Needs:

1. Quick access to clients' financial status
2. Simple visualization, e.g., green/red indicators for regular/irregular status
3. Integration with CRM (Customer Relationship Management) to centralize and analyze contact information, interactions, and purchase history, enabling optimization of sales, marketing, and customer support processes
4. Mobile-friendly interface
5. Real-time data updates

Auditor/Compliance Officer



Name: Ricardo Almeida
Age: 42
Gender: Male
Occupation: Digital Compliance and Risk Auditor
Location: Lisbon, Portugal
Salary: Medium
Technology Familiarity: High; uses digital e-commerce platforms and audit dashboards daily, with regular use of financial and transaction data analysis tools

Goals:

1. Clearly identify payment inconsistencies
2. Verify compliance with financial, tax, and regulatory standards
3. Gather evidence of fraud
4. Analyze tax reports

Frustrations:

1. Incomplete, unclear, or outdated data

2. Excess of irrelevant notifications and lack of critical alerts
3. Limited fraud analysis tools
4. Non-intuitive payment lifecycle processes
5. Difficulty reconciling invoices/payments with bank statements

Behaviors:

1. Regularly accesses the PBS to analyze financial trails, checking timestamped logs and monitoring risk alerts such as failed payments or potential fraud
2. Generates periodic reports on compliance, reconciliations, and payment statistics
3. Reviews invoices, refunds, and disputes to verify tax compliance

Needs:

1. Detailed and configurable reports to quickly verify compliance
2. Ability to analyze individual payments or aggregated data over time
3. Separation of invoices by different stages of the payment process
4. Easy export of reports in PDF/CSV format
5. Communication channel with the finance department to clarify questions about specific invoices or reports, or to flag tax non-compliance
6. Receive intelligent, high-priority alerts about failed payments, discrepancies, or potential fraud

Supported Scenarios (user stories)

In order to measure the effort required to complete a user story, the team used the Fibonacci sequence [1, 2, 3, 5, 8, 13]. The value 1 was the least amount of effort, whereas the value 13 was the most. Each member independently assigned one of these numbers to a user story and then the team discussed it until reaching a consensus.

Epic 1 - Invoice

Enable invoice generation, including one-time and recurring, automated delivery to customers, online access via the customer portal, and lifecycle management with real-time status tracking (Paid, Pending, Overdue).

US1: Invoice customers [SP: 8]

As a Billing Administrator/Finance Staff,

I want to generate and send invoice to customers,

So that customers are billed correctly.

Acceptance Criteria:

Given the customer has their billing data (name, tax id, address, email) on the system

When an invoice is generated

Then the system generates an invoice with a unique number, issue date, due date, items, total and QR code, sends the invoice by email to the customer, makes it visible in the customer's portal, and saves the invoice in the system with a status "Pending" until payment is made.

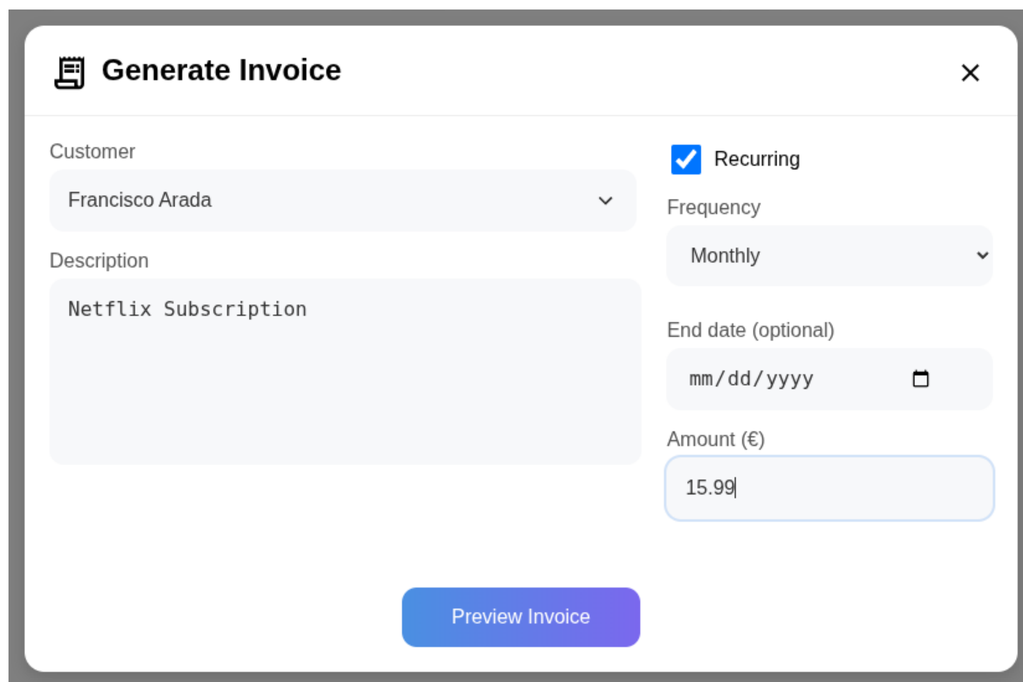


Figure 1: Interface for generating and sending invoices (US1)

InFlow
Billing & Invoicing
23 Cornelia Street
West Village, Manhattan
New York, NY 10014 (USA)
EIN: 12-3456789

Invoice / Receipt
Doc Nr.: preview-temp
Date: 10/12/2025
Copy: Original

Bill To:
—

Invoice Nr.: INV-preview-temp

Date	Description	Amount
10/12/2025	Netflix Subscription	15.99 EUR

Total: 15.99 EUR

Processed by certified program n.º1713/AT

Figure 2: Example of a generated invoice PDF

US2: View Invoices Online [SP: 2]

As a Customer/Client,

I want to be able to see my invoices online,

So that I can know what payments I have made.

Acceptance Criteria:

Given a customer is signed up and logged into the platform

When they access their invoices

Then the system displays a list of all invoices associated with the customer's account, where paid invoices show the corresponding payment date and overdue invoices are highlighted in red.

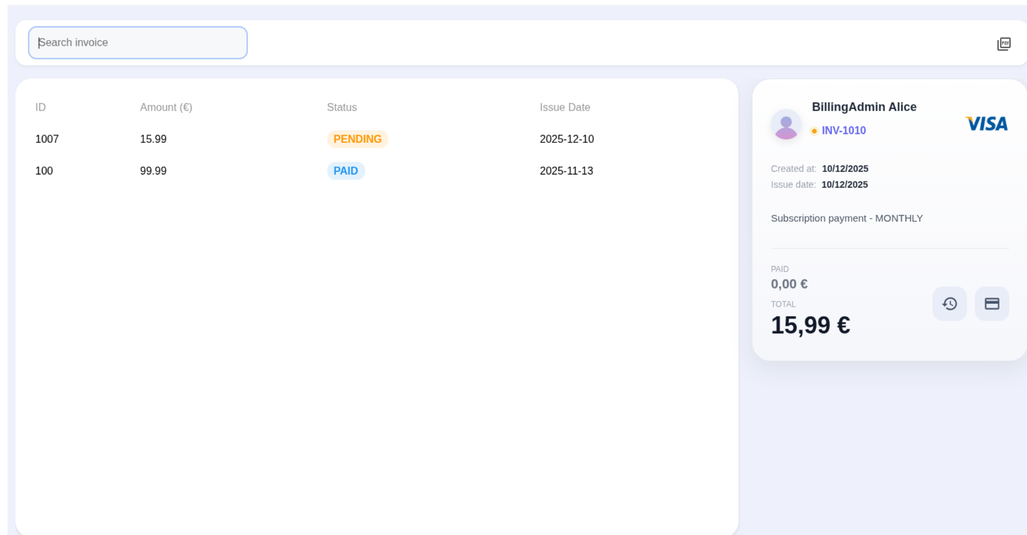


Figure 3: Customer portal showing the list of invoices (US2)

US3: Create recurring invoices [SP: 5]

As a Billing Administrator/Finance Staff,

I want to configure recurring invoices for contract customers (generating invoices, charging

customers on a set schedule, and managing subscription changes),

So that I reduce manual work and ensure consistent billing.

Acceptance Criteria:
Given the billing administrator/finance staff accesses contract customer settings

When they configure recurrence (monthly, quarterly, yearly)

Then the system automatically generates and sends invoices in each cycle and the customer receives a notification each time a new invoice is issued.

US4: Track invoice status [SP: 3]

extbfAs a Billing Administrator/Finance Staff,
extbfI want to see if an invoice is paid, pending, or overdue,
extbfSo that I can act quickly on overdue accounts.

Acceptance Criteria:
Given billing administrator/finance staff view an invoice or dashboard

When they query invoice status

Then the system displays the status as "Paid", "Pending" or "Overdue" and shows an aggregated view of invoice statuses by customer.

Epic 2 - Payments

Enable multi-channel payment processing (cards, bank transfer, digital wallets) with automated reconciliation, refund management, and integration with an external payment gateway.

US5: Pay invoice online [SP: 8]

As a Customer/Client,
I want to pay my invoice using a credit card, bank transfer, or digital wallet
So that I have flexibility and convenience in payments

Acceptance Criteria:
Given the customer accesses the portal

When they choose to pay an invoice

Then the portal shows invoice details with amount and due date, allows payment via credit/debit card, bank transfer or digital wallets (e.g., Apple Pay, Google Pay), and once processed, the payment is automatically applied to the invoice.

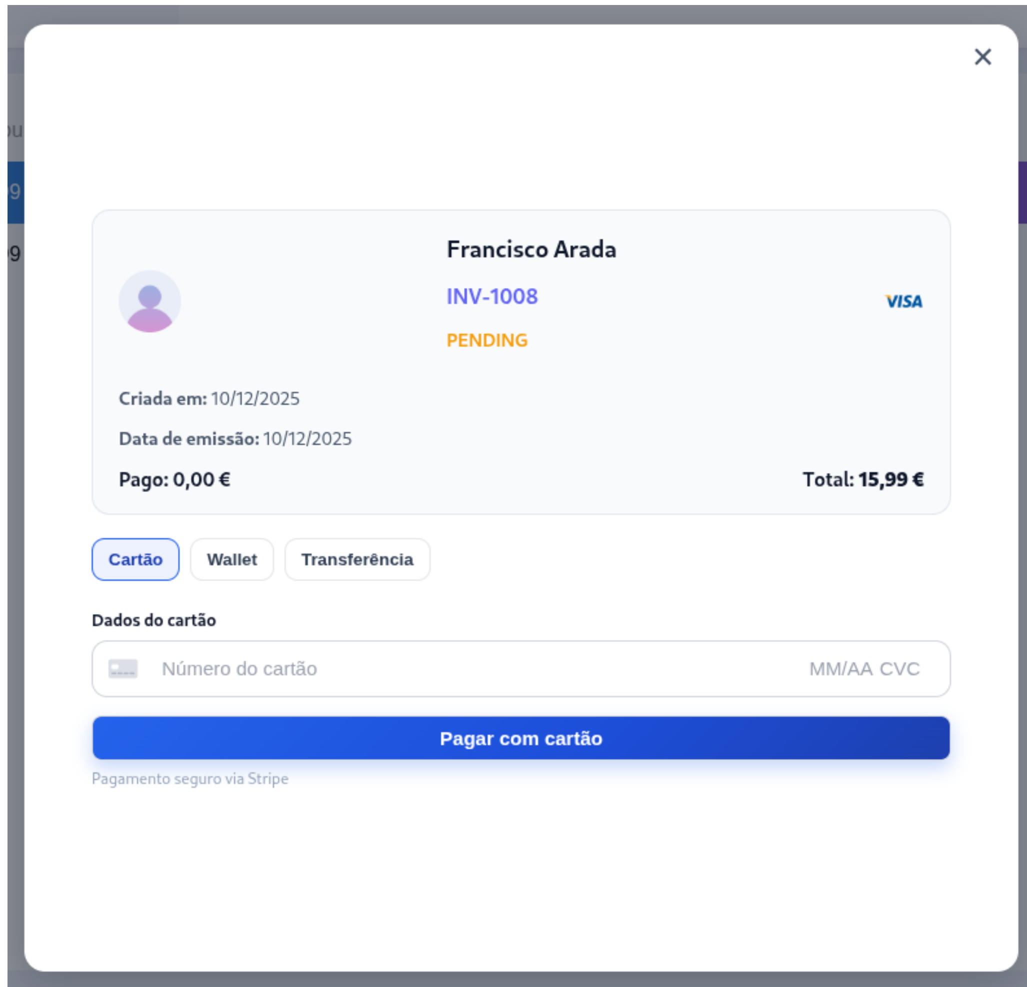


Figure 4: Online payment interface with multiple payment methods (US5)

US6: Apply payments to correct invoice [SP: 5]

As a Billing Administrator/Finance Staff,
I want payments to be automatically reconciled,
So that invoices are always updated correctly.

Acceptance Criteria:
Given payments are received online or imported via bank statement

When a payment is processed by the system

Then online payments are auto-applied to invoices, bank statement imports allow manual reconciliation if needed, errors or unmatched payments are highlighted in the dashboard for user attention and the event is logged in the audit trail with timestamp, user ID/email, action performed and reason/outcome if applicable.

US7: Issue refunds [SP: 8]

As a Billing Administrator/Finance Staff,

I want to issue refunds,

So that incorrect payments are corrected.

Acceptance Criteria:
Given a finance staff member chooses to issue a refund linked to the original payment

When a refund is processed

Then the refund (full or partial) is recorded in the system with amount, date, reason, and linked invoice, the customer is notified, and the event is logged in the audit trail with timestamp, user ID/email, action performed and reason/outcome if applicable.

Epic 3 - Notifications

Enable automated notifications for billing and payment events, including failed payments, new invoices, payment confirmations, and new customer sign-ups, with delivery via email and visibility in customer and finance staff dashboards.

US9: Failed payment notification for Finance Staff [SP: 5]

As a Billing Administrator/Finance Staff,

I want to receive notifications when a payment fails,

So that I can resolve it and follow up with the customer.

Acceptance Criteria:
Given a payment attempt fails (card declined, error, insufficient funds)

When the system detects the failure

Then a notification is sent automatically to finance staff with customer details (name, id, email), invoice number, amount and failure reason, the failed payment is logged in the system as "Pending", and the event is logged in the audit trail with timestamp, user ID/email, action performed, and reason/outcome if applicable.

US10: Failed payment notifications for Clients [SP: 2]

As a Customer/Client,
I want to be notified when a payment fails
So that I can fix the issue quickly.

Acceptance Criteria:
Given a payment attempt fails

When the failure is detected

Then the customer immediately receives a notification by email and in the customer portal, the invoice status updates to "payment failed", and the customer can retry the payment from the portal.

US11: Receive invoice notifications [SP: 2]

As a Customer/Client,
I want to get alerts whenever a new invoice is issued
So that I never miss a payment deadline

Acceptance Criteria:
Given a new invoice is issued

When the system processes the invoice

Then a notification is sent by email and made visible in the portal, which includes amount, due date, and a direct payment link.

US12: Receive payment confirmation notifications [SP: 2]

As a Customer/Client,

I want to receive a notification with a confirmation that a payment has been done,

So that I can be sure that the payment was successful.

Acceptance Criteria:
Given a customer completes a payment

When the payment is validated with the payment gateway and status is updated to "Paid"

Then a notification is sent to the customer via email and/or portal with the confirmation (invoice number, payment amount, date, and payment method), the record is stored in the customer's portal and the event is logged in the audit trail with

timestamp, user ID/email, action performed, and reason/outcome if applicable.

US13: New Customer notification [SP: 2]

As a Billing Administrator/Finance Staff,

I want to receive notifications when a new customer signs up,

So that I can see a new customer has joined.

Acceptance Criteria:
Given a new customer completes the sign-up process

When the system registers the account

Then a notification is automatically sent to the finance staff with the customer's data (name, tax id, email, company), the notification is seen in the finance staff dashboard and the event is logged in the audit trail with timestamp, user ID/email, action performed, and reason/outcome if applicable.

Epic 4 - Audit

Enable auditing and compliance by providing financial report exports, immutable audit trails, and real-time dashboards for revenue, overdue invoices, payments, and taxes.

US15: Access audit trail [SP: 3]

As an Auditor,
I want to access detailed logs of all financial transactions
So that I can verify compliance and detect fraud

Acceptance Criteria:
Given an auditor accesses the audit module

When they query the transaction log

Then each log shows user, datetime, action and value, logs are immutable and digitally signed, filters are available by date range and transaction type and the event is logged in the audit trail with timestamp, user ID/email, action performed, and reason/outcome if applicable.

Audit Trail

Search...

All Users | All Actions | mm/dd/yyyy | mm/dd/yyyy | Reset

TIMESTAMP	USER	ACTION	DETAILS	SIGNATURE
12/8/2025, 9:42:49 PM	Francisco Arada		Unknown log template	d+lpbdvZBIDi+TCJkvZp+1JxHhud04ivtY/tebC...
12/10/2025, 11:05:57 PM	BillingAdmin Alice	SUBSCRIPTION_INVOICE_CREATED	Created subscription 4 for client 10 (amount=1...	sTDchiL5lhAqBX1hkFT6h0j2WFDTw+ReFlVI...
12/10/2025, 11:05:57 PM	Francisco Arada	SUBSCRIPTION_INVOICE_CREATED	Unknown log template	xUZ8S649mQKbD/uLX94SFeRVwfj11IuWNu4...
12/10/2025, 11:05:59 PM	BillingAdmin Alice	SUBSCRIPTION_INVOICE_CREATED	Created subscription 5 for client 10 (amount=1...	tAjJoLji6YQqm+aTkprHoWAQWuJvNDrXwQb...
12/10/2025, 11:05:59 PM	Francisco Arada	SUBSCRIPTION_INVOICE_CREATED	Unknown log template	r+1Exa/exw0RcTEYhZ3lCCS065q5mznLp5Nl...
12/10/2025, 11:05:59 PM	BillingAdmin Alice	SUBSCRIPTION_INVOICE_CREATED	Created subscription 6 for client 10 (amount=1...	f8Q5HJKFHid2mCVncLQv6/9b7Fn6DSeVBw1...
12/10/2025, 11:05:59 PM	Francisco Arada	SUBSCRIPTION_INVOICE_CREATED	Unknown log template	DpN9EV2vHJy2/nuy6OalOHe7rR19sLcjzLW5...
12/10/2025, 11:06:00 PM	BillingAdmin Alice	SUBSCRIPTION_INVOICE_CREATED	Created subscription 7 for client 10 (amount=1...	sXD6h9v5YvEoCM4rznwh5L/yzFPV335p3b50...
12/10/2025, 11:06:00 PM	Francisco Arada	SUBSCRIPTION_INVOICE_CREATED	Unknown log template	5pD/QGBqEyYunGlBW20CenKSaXUOnr9wq...
12/10/2025, 11:16:34 PM	BillingAdmin Alice	SUBSCRIPTION_UPDATED	Upgraded subscription 7 from 15.99 to 20.0	n/8VRu1PWqYpBpnTKGtZQQJJaj+KDDiYrQ1...
12/10/2025, 11:16:59 PM	BillingAdmin Alice	SUBSCRIPTION_UPDATED	Amount unchanged subscription 7 from 20.0 t...	jmTBc4wRA7jLFWu74vPm0ebEb5c91RDhqh...

Figure 5: Audit trail interface showing transaction logs (US15)

US16: Use Reporting Features [SP: 3]

As an Auditor,

I want to use reporting features to verify billing accuracy,

So that I can ensure compliance with financial regulations.

Acceptance Criteria:

Given an Auditor is logged into the platform

When they access the reporting dashboard from their account

Then the system must display reports which update in real-time that include revenue by customer, overdue invoices, payment trends, and sales tax and the event must be logged in the audit trail with the timestamp, user ID/email, action performed, and reason/outcome if applicable.

US17: Export financial reports [SP: 3]

As an Auditor,
I want to export reports in Excel/PDF
So that I can analyze financial data outside the system.

Acceptance Criteria:

Given an Auditor is logged into the platform

When they select the option to export financial reports

Then the system must generate a report that includes revenue, taxes, and overdue accounts and the Auditor must be able to select the export format as either PDF or Excel. The generated file must include a timestamp and a generation log. Finally, the export event must be logged in the audit trail with the timestamp, user ID/email, action performed, and reason/outcome if applicable.

Epic 5 - Sales Representative Functions

Enable sales representatives to view customer payment history and account status in read-only mode.

US21: View Customer's Payment History [SP: 2]

As a Sales Representative,

I want to view a customer's payment history,

So that I can understand their past payment behavior.

Acceptance Criteria:

Given a Sales Representative is logged into the platform with that role

When they view a customer's payment history

Then the system must display a list of past payments including the Invoice numbers, Payment dates, Amounts, and Status (paid, overdue, disputed, refunded) and the information displayed must be consistent with the finance staff records. Furthermore, the Sales Representative must have read-only access (unable to edit, delete or add records).

US22: View Account Status [SP: 2]

As a Sales Representative,

I want to view a customer's account status,

So that I can decide whether to offer a new service or product.

Acceptance Criteria:

Given a Sales Representative is logged into the platform with that role

When they select a customer's account

Then the system must display the customer's account summary, which includes the total overdue balance, number of overdue invoices, and current subscription status (if applicable) and the information displayed must be consistent with the finance staff records, also the Sales Representative must have read-only access (can't edit, delete or add records).

Epic 6 - Authentication

US23: Customer Sign Up [SP: 13]

As a Customer/Client,

I want to sign up in the platform,

So that I can access the platform's service**.**

Acceptance Criteria:

Given a potential Customer/Client is on the sign-up page

When they submit the registration form with valid credentials (email and password),

Then the system must create the account and a notification is automatically sent to the finance staff in order to activate the account and the sign-up event must be logged in the audit trail with the timestamp, user ID/email, action performed, and reason/outcome if applicable.

Figure 6: Customer sign-up page (US23)

US24: Customer Log In [SP: 8]

As a Customer/Client,

I want to log into the platform,

So that I can make payments online and access my history**.**

Acceptance Criteria:

Given a user is signed up in the system with the "Customer" role

When they attempt to log in by providing valid credentials (email and password)

Then the system must grant access and redirect them to their respective dashboard but if the provided credentials are invalid the system must show an error message and deny access and, in all cases, the system must log the login attempt in the audit trail with the timestamp, user ID/email, outcome (success/failure), and reason if failed.

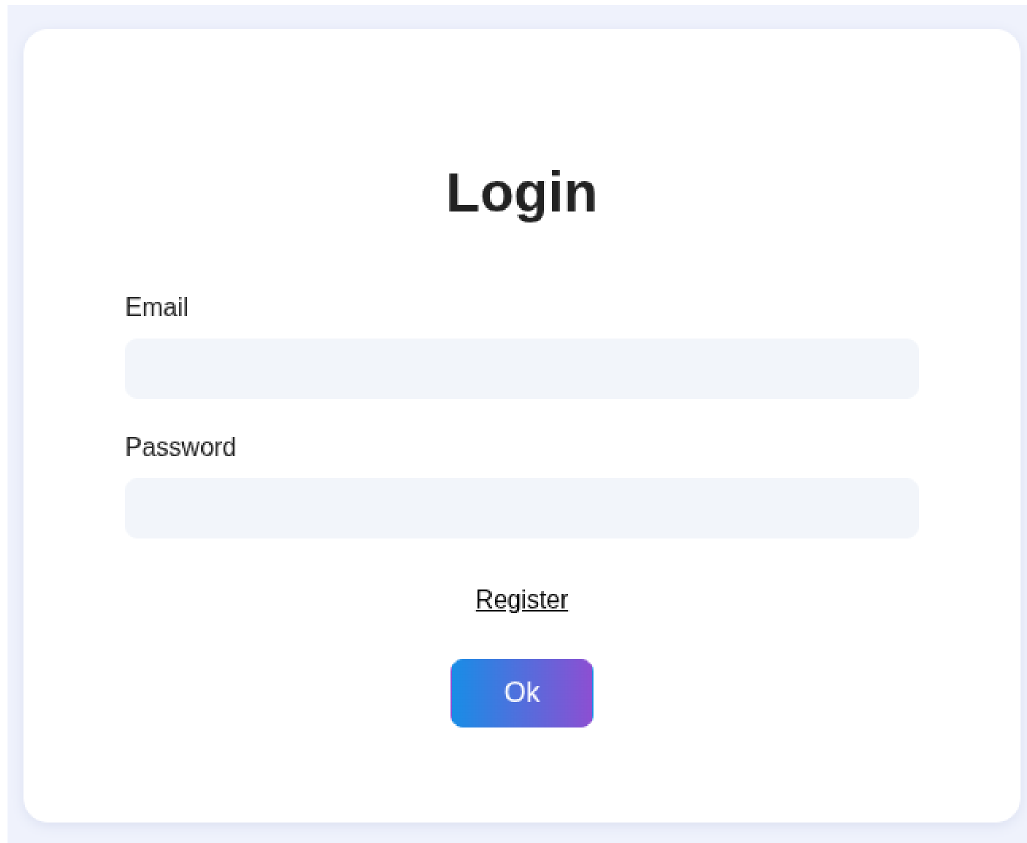


Figure 7: Customer login page (US24)

US25: Billing Administrator/Finance Staff Log In [SP: 8]

As a Billing Administrator/Finance Staff,

I want to log into the platform,

So that I can generate and send invoices to customers.

Acceptance Criteria:

Given a user is signed up in the system with the "Billing Administrator/Finance Staff" role

When they attempt to log in by providing valid credentials (email and password)

Then the system must grant access and redirect them to their respective dashboard

but if the provided credentials are invalid the system must show an error message and deny access and, in all cases, the system must log the event must be logged in the audit trail with: timestamp, user ID/email, action performed, and reason/outcome if applicable.

US26: Auditor Log In [SP: 5]

As an Auditor,

I want to log into the platform,

So that I can review financial records and audit trails.

Acceptance Criteria:

Given a user is signed up in the system with the “Auditor” role

When they attempt to log in by providing valid credentials (email and password)

Then the system must grant access and redirect them to their respective dashboard but if the provided credentials are invalid the system must show an error message and deny access and, in all cases, the system must log the login attempt in the audit trail with the timestamp, user ID/email, outcome (success/failure), and reason if failed.

US27: Sales Representative Log In [SP: 5]

As a Sales Representative,

I want to log into the platform,

So that I can access the system to view a customer's payment history and account status.

Acceptance Criteria:

Given a user is signed up in the system with the “Sales Representative” role

When they attempt to log in by providing valid credentials (email and password)

Then the system must grant access and redirect them to their respective dashboard but if the provided credentials are invalid the system must show an error message and deny access and, in all cases, the system must log the login attempt in the audit trail with the timestamp, user ID/email, outcome (success/failure), and reason if failed.

Without Epic

US8: Handle payment disputes [SP: 5]

As a Billing Administrator/Finance Staff,

I want to handle payment disputes,

So that billing errors are tracked until resolved.

Acceptance Criteria:

Given a user with the role "Billing Administrator/Finance Staff" is viewing a payment in the system

When they mark the payment as disputed

Then the system requires them to provide a reason for the dispute (e.g., duplicate charge, incorrect amount), and the disputed amount is flagged in the system and not taken into consideration in financial totals until the dispute is resolved and the system must log the event in the audit trail with the timestamp, user ID/email, action performed, and reason/outcome if applicable.

US14: Update billing information [SP: 3]

As a Customer/Client,
I want to update my billing details (address, tax number)
So that my invoices always reflect accurate information.

Acceptance Criteria:

Given a Customer/Client is signed up and logged into the platform

When they access their billing details and update the information using a secure form with editable fields

Then the system must save the new details (address, tax number) and these updates must apply only to future invoices, ensuring past invoices remain unchanged for audit purposes.

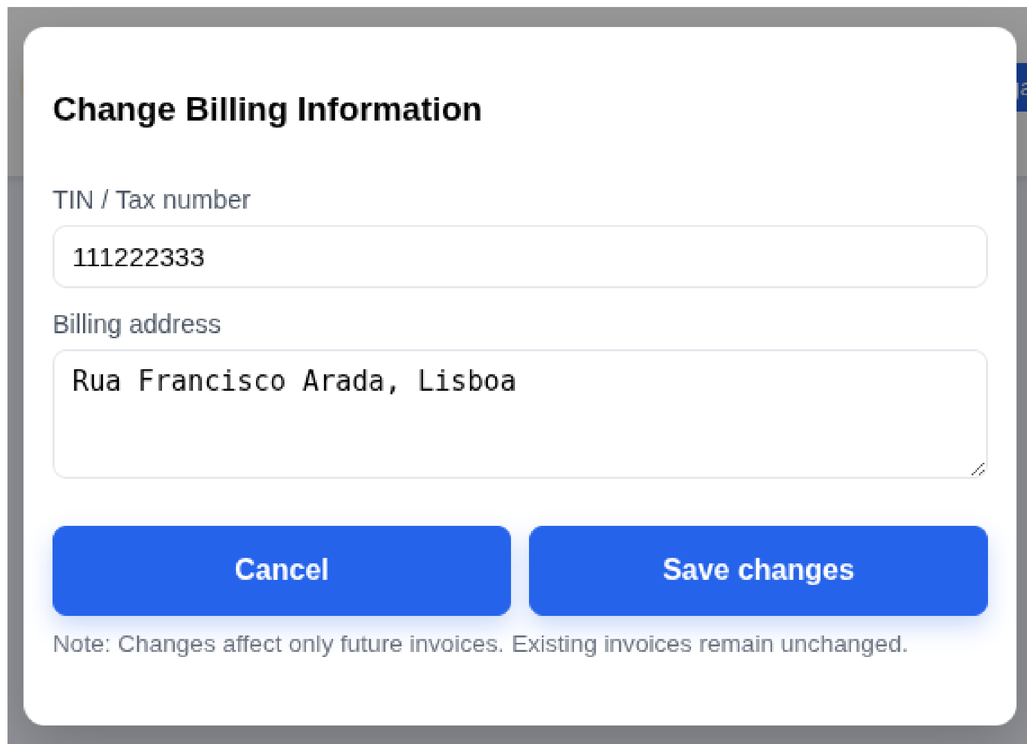


Figure 8: Interface for updating customer billing information (US14)

US18: Manage Customer Accounts [SP: 5]

As a Billing Administrator/Finance Staff,

I want to manage customer accounts (upgrade, downgrade, cancel),

So that I can keep billing information accurate.

Acceptance Criteria:

Given a user with the role "Billing Administrator/Finance Staff" is logged into the platform

When they navigate to the customer account management section

Then the system must display customer accounts in a searchable list and the user can choose to upgrade, downgrade, or cancel a customer's account and the system processes the change and logs the event in the audit trail with the timestamp, user ID/email, action performed, and reason/outcome if applicable.

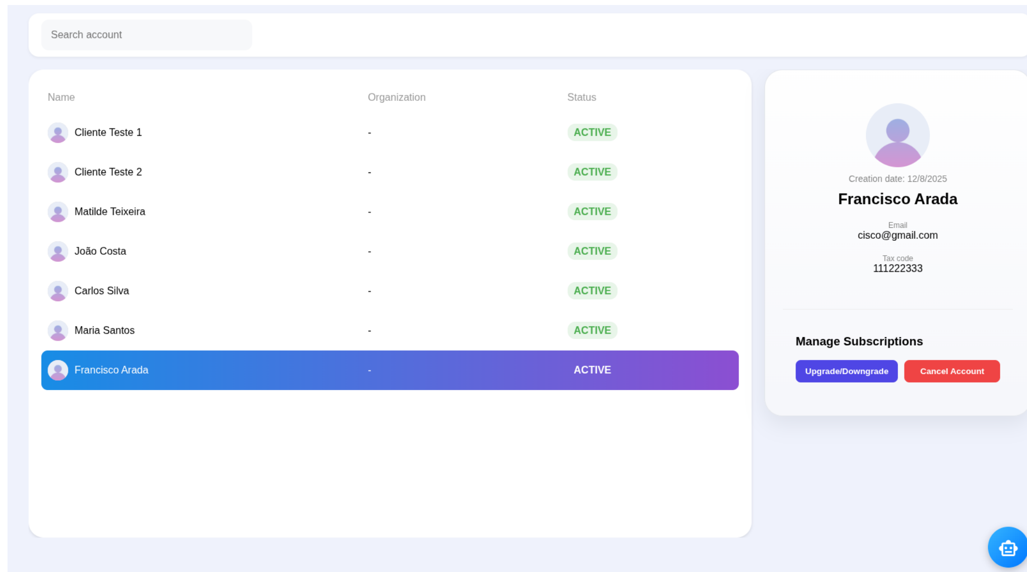


Figure 9: Interface for managing customer accounts (US18)

US19: Revenue and payment dashboard [SP: 3]

As a Billing Administrator/Finance Staff,
I want to see metrics on revenue, overdue invoices, and payment trends,
So that I have real-time visibility into cash flow.

Acceptance Criteria:

Given a user with the role "Billing Administrator/Finance Staff" is logged into the platform

When they access the reporting dashboard

Then the dashboard must display graphs showing revenue by customer, overdue invoices, and payment trends and the reports must update in real-time.

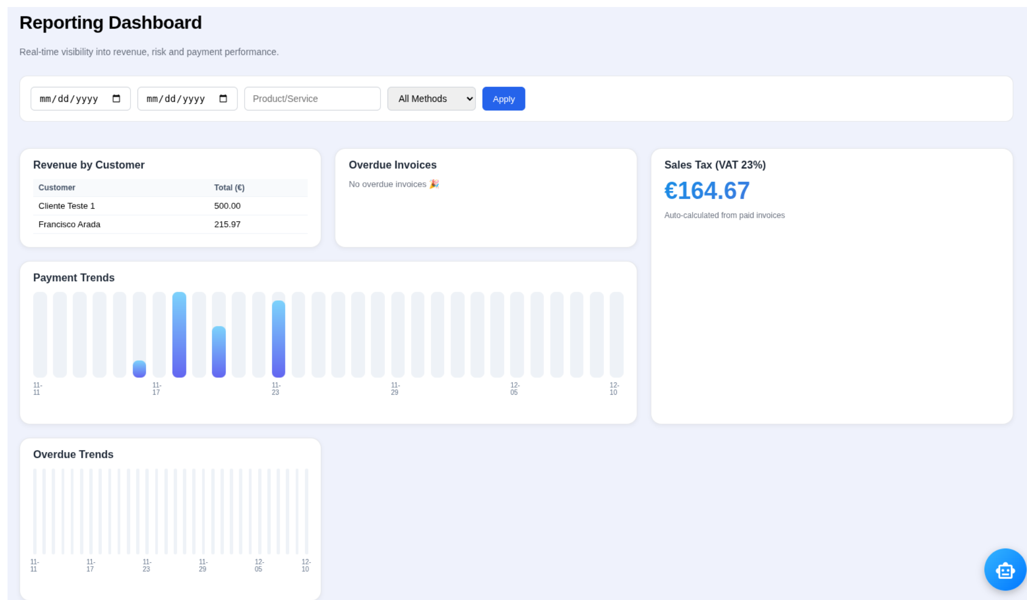


Figure 10: Revenue and payment dashboard with real-time metrics (US19)

US20: View payment history [SP: 2]

As a Customer/Client,
I want to see all my past invoices and receipts,
So that I can track my spending and prepare accounting records.

Acceptance Criteria:

Given a customer is logged into the platform

When they navigate to their payment history page

Then the system displays a list of all past invoices and receipts ordered by date and each entry in the list must display the amount, date, status, and provide the corresponding receipt in PDF format and the system must provide an option to download invoices and receipts in bulk.

Architecture notebook

Key quality requirements

The architecture of the **InFlow** system is the result of an in-depth analysis of both functional needs and broader architectural characteristics. These requirements guide the selection of technologies and the organization of system components to ensure performance, maintainability, scalability, and adaptability within the context of this course.

This section identifies key issues, constraints and the proposed architecture.

Functional Requirements

These are the primary capabilities the system must provide to end users in order to fulfill the business and operational goals defined in the case study:

1. **Multi-role user management**:
   The system must support different user types (Billing Administrator/Finance Staff, Customer/Client, Sales Representative, Auditor/Compliance Officer) with tailored access and functionality, ensuring that each role only has permissions appropriate to their responsibilities.

2. **User authentication and authorization**:
   Users must be able to securely log in using valid credentials. Authentication must support role-based access control, with invalid login attempts denied and logged in the audit trail.

3. **Invoice generation and management**:
   Billing Administrators/Finance Staff must be able to create, edit, and send invoices to customers, including one-time and recurring invoices. Invoices must include details such as invoice number, items, amounts, issue and due dates, and payment status.

4. **Payment processing and reconciliation**:
   Customers must be able to pay invoices via multiple methods (credit/debit card, bank transfer, digital wallets), with payments automatically applied to the correct invoices. Finance staff must also be able to reconcile payments manually when needed, with all actions logged.

5. **Invoice tracking and status monitoring**:

The system must display invoice status (Paid, Pending, Overdue) on dashboards and lists for finance staff, sales representatives, and customers. Alerts and notifications must be sent for overdue or failed payments.

6. **Customer account management**:
   Billing Administrators/Finance Staff must be able to manage customer accounts, including upgrading, downgrading, or cancelling services, updating billing information, and viewing the customer's financial activity.

7. **Notifications and alerts**:
   The system must send notifications to relevant users for key events, including new invoices, payment confirmations, failed payments, new customer sign-ups, and payment disputes. Notifications must be available via email and in-platform dashboards.

8. **Reporting and dashboards**:
   Finance staff and auditors must have access to configurable dashboards and reports showing revenue, payment trends, overdue invoices, and other key metrics. Reports must be exportable to PDF and Excel and include timestamps for auditing.

9. **Payment disputes and refund handling**:
   Finance staff must be able to flag disputed payments, resolve discrepancies, and issue refunds (full or partial), with all actions linked to the original payment and logged in the audit trail.

10. **Audit trail and compliance support**:
    The system must maintain a detailed, immutable audit trail of all financial transactions and user actions. Auditors must be able to filter logs by date, transaction type, and user, and export them for compliance verification.

11. **Mobile and web interface accessibility**:
    The system must provide responsive interfaces for desktop and mobile devices, enabling sales representatives to quickly access client account status and financial data on-the-go, and customers to manage payments and billing information.

12. **Data security and integrity**:
    All financial and personal data must be securely stored and transmitted, with encryption and access controls. The system must ensure data integrity for invoices, payments, and account information, preventing unauthorized modifications.

Non-Functional Requirements

The following non-functional requirements ensure the overall quality of the system and alignment with the project's software engineering practices:

1. **Performance and Scalability:**

   The system must guarantee high performance and efficient scalability. All critical operations, such as user authentication, invoice generation, and payment processing, should respond in less than two seconds under normal load. The architecture must support horizontal scalability, allowing new service instances to be deployed in containers without code modification, ensuring that the platform can handle growth in demand. It should sustain at least one thousand concurrent users while maintaining an acceptable level of responsiveness and stability.

2. **Availability and Reliability:**

   High availability is a requirement, with the system expected to remain operational at least 99% of the time, excluding scheduled maintenance windows. Reliability must be ensured through fault-tolerant microservice design, so that the failure of a single service does not compromise the whole platform. Regular automated backups must be implemented, enabling recovery with a maximum recovery point objective (RPO) of 24 hours and a recovery time objective (RTO) not exceeding four hours.

3. **Security:**

   Security is essential across all components. The system must implement centralized authentication and authorization using Keycloak, avoiding the management of credentials in custom database tables. Role-Based Access Control (RBAC) will be used to ensure that users have only the permissions appropriate to their roles. All passwords and authentication tokens are managed by Keycloak, and all communication between client and server must use TLS 1.3 to ensure data confidentiality.

   The system must also comply with GDPR for all personal and financial data, and maintain a tamper-proof audit trail with cryptographically signed timestamps to ensure traceability and accountability of user actions.

4. **Maintainability and Extensibility:**

The solution must be easy to maintain and extend. It should adopt a modular microservices architecture, enabling individual components to be updated or replaced independently. All code must be versioned in Git, following a feature-branch workflow and pull request process to ensure collaborative development and traceability. The API must be fully documented using OpenAPI/Swagger, and user and developer documentation must be kept up to date in the repository. Unit and integration tests should provide at least 80% coverage of critical functionality, supporting continuous integration and reducing regression risk.

5. **Usability and Accessibility:**

   The system must provide a smooth and intuitive user experience. Its interface must be responsive, working seamlessly across desktop and mobile devices, and must provide clear feedback for user actions such as successful operations, errors, and loading states. Accessibility is also a priority, and the interface should conform as closely as possible to WCAG 2.1 AA guidelines to ensure it is usable by people with disabilities.

6. **Portability and Deployment:**

   Portability must be guaranteed by using containerization. Each service must run in its own Docker container, respecting the principle of separation of concerns, and the deployment process must support running on any environment compatible with Docker and Kubernetes (or similar orchestration tools). A continuous integration and continuous delivery (CI/CD) pipeline must be established to automate builds, run tests, and deploy to staging environments, reducing the risk of human error during releases.

7. **Observability and Monitoring:**

   The system must be observable and easy to monitor. All microservices should generate structured, centralized logs to support troubleshooting and auditing. Key performance indicators, such as average response time and daily invoice generation counts, must be exposed through dedicated monitoring endpoints. Automated alerting should be configured to notify the team of critical failures, anomalies, or performance degradation, ensuring rapid response and minimal downtime.

8. **Interoperability:**

The platform must easily integrate with external systems, including payment gateways, CRM platforms, and potential third-party accounting tools. All APIs must follow a consistent, RESTful design and use standard data formats such as JSON. This ensures that other enterprise systems can consume or push data without custom adapters or proprietary integrations.

9. **Compliance and Audit Readiness:**

   Because the system deals with financial data, it must support compliance with financial regulations and audit requirements. All transactions, user actions, and configuration changes must be logged immutably and be exportable on demand. Reports must include timestamps, user identifiers, and digital signatures where appropriate, to guarantee authenticity.

System architecture

The current system architecture follows a microservice-based approach that separates responsibilities into independent components, each exposing its own interfaces while collaborating through synchronous and asynchronous communication channels. The system is composed of six main architectural layers: authentication, frontend, API gateway, microservices, message queue, and observability.

Components

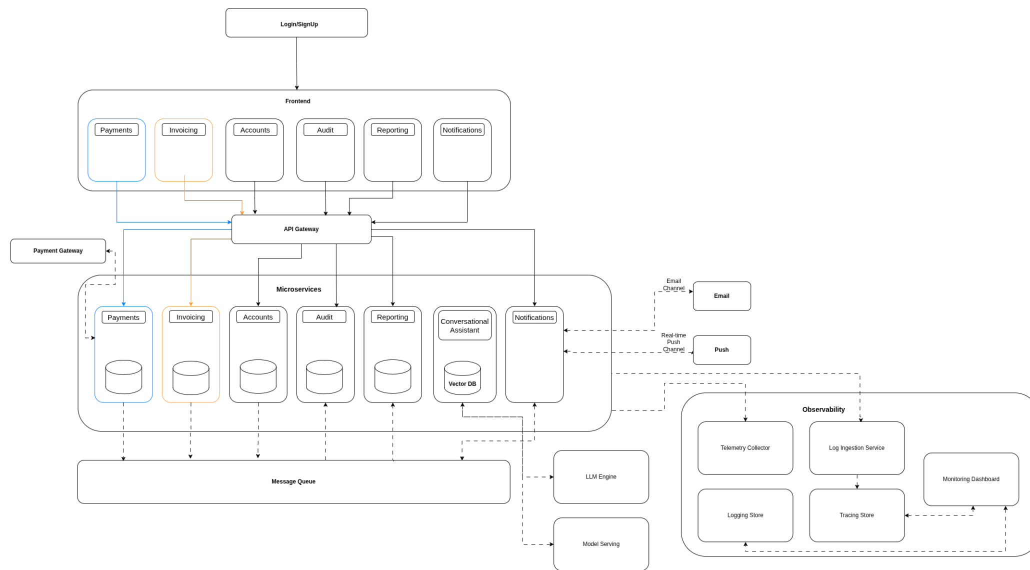


Figure 11: Current system architecture

Figure 11 illustrates the current deployed architecture of the InFlow system.

The **Authentication** component (Login/SignUp) manages user identity and access control, issuing tokens consumed by the remaining components of the system.

The **Frontend** is responsible for presenting information to the different user roles. It communicates exclusively with the backend through the API Gateway and offers modules for Payments, Invoicing, Accounts, Audit, Reporting, Notifications and the Conversational Assistant.

The **API Gateway** acts as the single entry point for all backend requests, routing them to the appropriate microservice and enforcing a uniform interface.

The **Microservices layer** contains all domain services:

- Payments,
- Invoicing,
- Accounts,
- Audit,
- Reporting,
- Notifications,
- Conversational Assistant.

Each service maintains its own data store and operates independently. The Conversational Assistant integrates a vector-based store and model serving components to support retrieval-augmented generation (RAG).

The **Message Queue** enables asynchronous communication between microservices, allowing events to be published and consumed without direct coupling.

The **Observability layer** provides end-to-end monitoring of logs, metrics, and traces. A telemetry collector receives instrumentation data from the microservices; logs are shipped to a log ingestion service and stored in a logging store; traces follow a parallel path into a tracing store; and all information is visualized through a unified monitoring dashboard.

Technology Stack

Backend Services

- Spring Boot 3.x (Java 17)
- PostgreSQL 15 (Payments, Invoicing, Accounts datastores)
- Apache Kafka 3.x (Message Queue)

Frontend

- React 18.x

Authentication

- Keycloak

API Gateway

- Kong

Observability

- OpenTelemetry (telemetry collector)
- Elasticsearch (logging store)
- Tempo (tracing store)
- Grafana (monitoring dashboard)

Conversational Assistant

- Vector Store: Weaviate
- Model Serving: HuggingFace

External Integrations

- Payment Gateway: Stripe
- Email: SMTP
- Push Notifications: Websocket

Interactions

The interactions within the InFlow system follow a layered communication pattern in which the Frontend, API Gateway, Microservices, external channels, and observability components exchange information through synchronous and asynchronous mechanisms. The main interaction model can be described as follows.

User-facing interactions. All user actions originate in the Frontend, which communicates exclusively with the API Gateway. Requests for Payments, Invoicing, Accounts, Audit, Reporting, Notifications and the Conversational Assistant are routed by the gateway to their corresponding microservice. Responses follow the same path in reverse, ensuring a single, unified access point for the system.

Synchronous service interactions. Most domain operations follow a request–response model. The API Gateway forwards HTTP requests to the appropriate microservice, which performs its internal logic, accesses its private data store, and returns the result to the gateway. This pattern enforces service isolation and guarantees that each service remains responsible for its own data and behaviour.

Asynchronous interactions via the message queue. Certain interactions between microservices rely on asynchronous event-based communication. Domain events (e.g., payment processed, invoice generated, notification triggered) may be published to the message queue, allowing other services to react without requiring direct coupling. This pattern improves system decoupling and enables background processing without blocking the user-facing workflow.

External channel interactions. The Notifications service communicates with external delivery channels. Email notifications are sent through an external email channel, while real-time push updates are delivered through a dedicated push channel. Similarly, the Payments service interacts with an external payment gateway for processing online transactions initiated by users.

Conversational Assistant interactions. The Conversational Assistant integrates multiple components to provide retrieval-augmented generation. The microservice receives user input via the API Gateway, queries its vector store to retrieve semantically relevant information, and interacts with a model-serving component to produce a contextualised response. The result is then returned to the gateway and, consequently, to the Frontend.

Observability interactions. All microservices emit logs, metrics, and traces that are collected by the telemetry collector. Logs are forwarded to a log ingestion service and stored in a logging backend, while traces follow a parallel ingestion path into a tracing backend. The monitoring dashboard queries these stores to provide a unified view of the system's operational state.

Communication Protocols

Synchronous Communication

- **Protocol:** HTTP/1.1 and HTTP/2 (REST)
- **Data Format:** JSON
- **Authentication:** Bearer tokens (JWT via Keycloak)
- **Request Timeout:** 30 seconds

Asynchronous Communication

- **Protocol:** Apache Kafka
- **Message Format:** JSON with event metadata
- **Topics:** `payments-events`, `invoicing-events`, `notifications-events`
- **Partitioning Strategy:** By `customer_id` for ordering guarantees
- **Consumer Groups:** One per microservice

Event-Driven Notification Policies

The Notifications service implements a policy-based routing mechanism that determines which users receive notifications and through which channels.

Policy Structure

Each domain event type is mapped to a notification policy that defines:

- **Recipients:** User roles that should be notified (customer, billingadministrator)
- **Channels:** Delivery mechanisms (email, push, or both)

Current Notification Policies

Policy Processing

1. Domain service publishes event to Kafka topic.
2. Notifications service consumes event from topic.

Event Type	Trigger Service	Recipients	Channels
`invoice_generated`	Invoicing	Customer	Email, Push
`recurrent_invoice`	Invoicing	Customer	Email, Push
`refund_event`	Payments	Customer	Email, Push
`payment_failed`	Payments	Customer, BillingAdministrator	Email, Push
`payment_confirmed`	Payments	Customer	Email, Push

3. Policy engine resolves recipients based on event type.
4. Notifications are dispatched to configured channels (SMTP for email, WebSocket for push).

Policy Configuration

Policies are stored as JSON and can be updated without code changes:

```
{
  "payment_failed": {
    "destinatarios": {
      "customer": ["email", "push"],
      "billingadministrator": ["email", "push"]
    }
  }
}
```

Dependencies

The InFlow system is composed of multiple layers and microservices that depend on each other to deliver cohesive functionality. These dependencies arise from data access requirements, business logic coordination, external integrations, and system-wide operational concerns. The main forms of dependency in the architecture are described below.

Frontend dependencies. The Frontend depends on the Authentication component to issue valid access tokens and relies on the API Gateway as the exclusive communication interface with the backend. All functional modules (Payments, Invoicing, Accounts, Audit, Reporting, Notifications and the Conversational Assistant) depend on the gateway to reach their corresponding microservices.

API Gateway dependencies. The API Gateway depends on the availability of all backend microservices to correctly route requests. It also interacts with the authentication layer to validate access tokens and enforce routing and access-control rules before forwarding any request.

Microservice dependencies. Each microservice depends primarily on its own private data store, ensuring strict ownership of domain data. Some microservices also depend on each other to complete multi-step operations. Payments and Invoicing are interdependent through the message queue for payment confirmation and invoice generation. Notifications depends on domain events emitted by various services to trigger delivery actions. The Conversational Assistant depends on its vector store and on the model-serving component to retrieve context and generate responses.

Message queue dependencies. The message queue is a central asynchronous dependency for microservices that publish or consume domain events. Services performing business actions that span multiple domains rely on the message queue to coordinate updates without direct coupling.

External system dependencies. Several microservices depend on external channels or systems. The Payments service depends on an external payment gateway to process online transactions. The Notifications service depends on the email and push delivery channels to distribute system alerts. The Conversational Assistant depends on a model-serving component to produce contextual responses.

Observability dependencies. All backend services depend on the observability layer to record logs, metrics and traces. The telemetry collector depends on microservices for instrumentation data, while the log ingestion and tracing stores depend on the collector to receive structured information. The monitoring dashboard depends on both stores to provide a unified operational view of the system.

Critical flows

To illustrate how the architecture supports both functional and architectural behaviour, this section presents the system's critical end-to-end flows. These flows combine the components, interactions and dependencies previously described and are aligned with the core user stories implemented in the system.

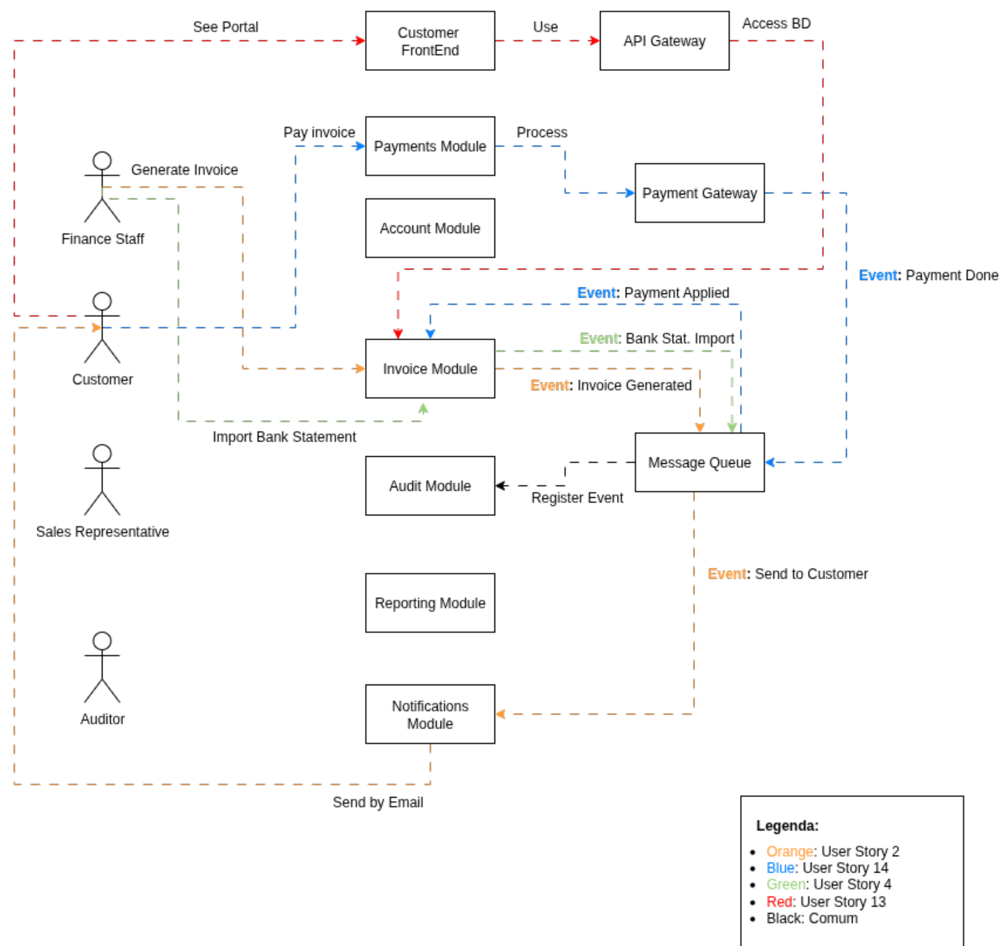


Figure 12: Critical Flows

Functional flows (User Stories)

Figure 12 highlights the main functional flows triggered by the different user roles. Each coloured path corresponds to a user story and demonstrates how responsibilities are distributed across modules and microservices.

User Story 2 – Generate Invoice (orange). A Finance Staff member requests the creation of an invoice through the Frontend. The request is routed via the API Gateway to the Invoicing microservice, which generates the invoice and stores it in its private data store. An event is then emitted to notify other services, allowing the Notifications module to inform the customer if required.

User Story 14 – Online Payment (blue). A Customer initiates a payment in the Payments module. The operation is forwarded through the API Gateway to the Payments microservice, which interacts with the external payment gateway. Once the transaction is confirmed, a "Payment Applied" event is published to the message queue. The Invoicing service consumes this event, updates the invoice status and emits an "Invoice Generated" event, enabling subsequent customer notifications.

User Story 4 – Import Bank Statement (green). A Sales Representative uploads a bank statement in the Frontend. The request reaches the Invoicing module, which performs reconciliation against existing invoices. A "Bank Statement Imported" event is emitted, and the Audit module records the operation for traceability.

User Story 13 – View Invoices (red). A Customer accesses the portal and requests to view invoice history. The Frontend forwards the request to the API Gateway, which queries the Invoicing microservice. The results are returned through the gateway and rendered in the portal.

Architectural flows

In addition to functional user stories, three architectural flows are essential to the system's operation and are represented in Figure 11.

Online payment and invoicing flow. This flow combines synchronous and asynchronous steps to implement a core business process:

1. The user initiates a payment in the Frontend.
2. The API Gateway routes the request to the Payments microservice.
3. The service verifies the operation and interacts with the external payment gateway.
4. Upon confirmation, a payment event is published to the message queue.
5. The Invoicing service consumes the event, generates or updates the invoice, and persists it.
6. A notification event may be emitted to inform the customer of the completed operation.

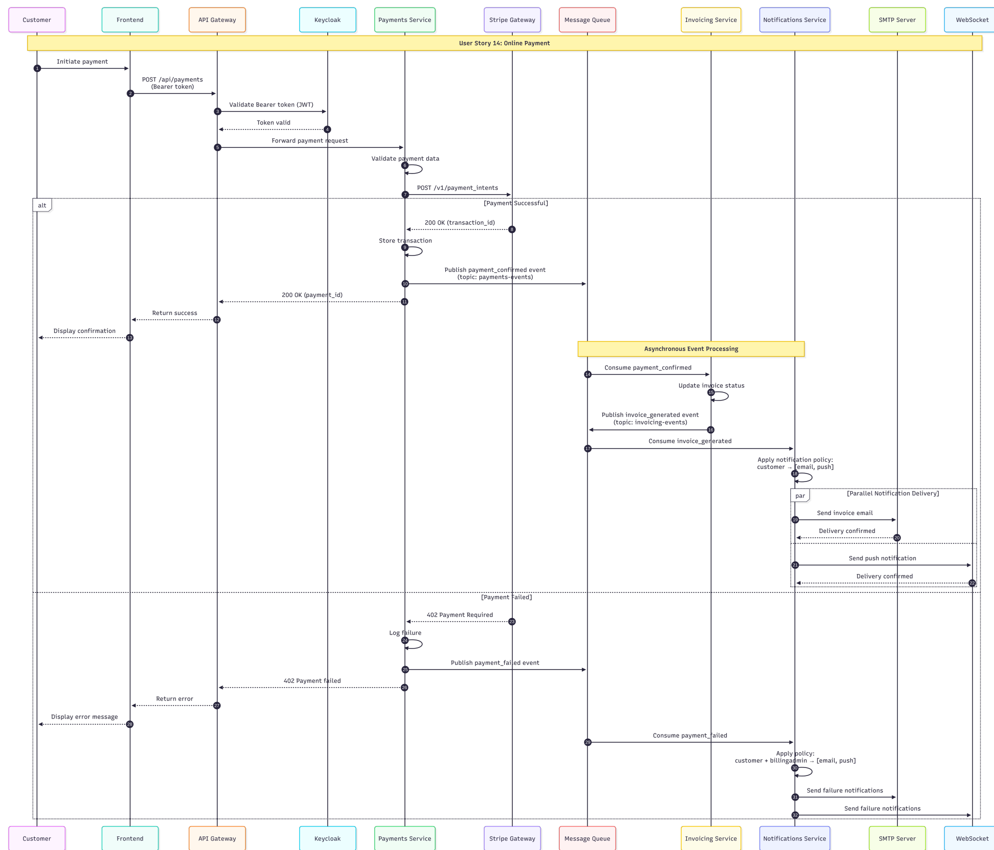


Figure 13: Sequence Diagram - Online Payment Flow (US14)

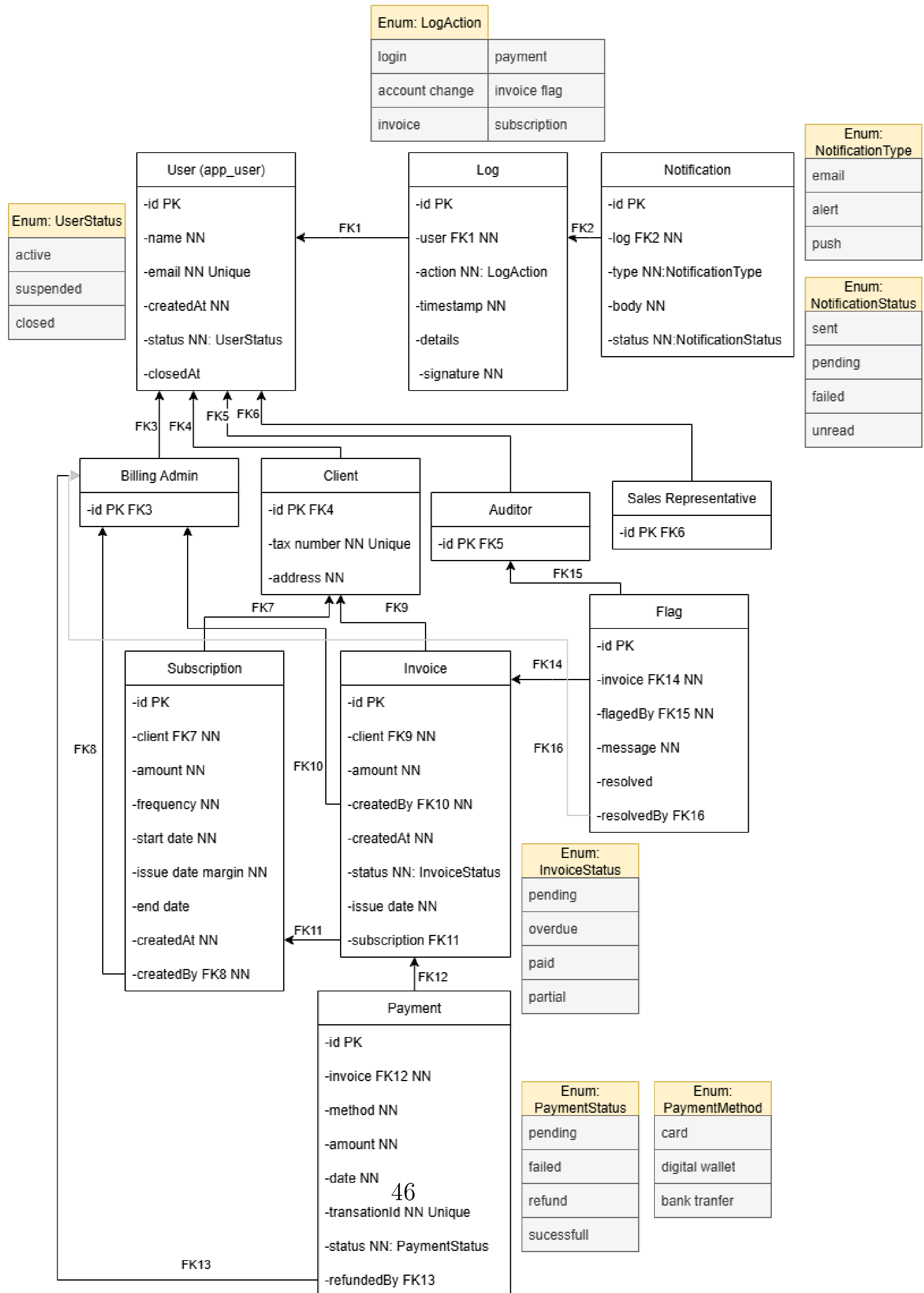
Conversational assistant flow. The Conversational Assistant provides retrieval-augmented responses by orchestrating multiple components:

1. The user submits a query through the Frontend.
2. The request is routed to the Conversational Assistant microservice.
3. The microservice transforms the query and retrieves relevant information from the vector store.
4. Context and query are forwarded to the model-serving component, which generates a response.
5. The answer is returned through the API Gateway to the Frontend.

Observability and incident analysis flow. This flow ensures system reliability and visibility:

1. Microservices and the API Gateway emit logs, metrics and traces during execution.
2. The telemetry collector receives and processes this instrumentation data.
3. Logs are forwarded to the log ingestion service and stored in the logging backend, while traces are persisted in the tracing backend.
4. The monitoring dashboard queries these stores to provide real-time insights into system health, performance and error conditions.

Information Model



The domain model for **InFlow** organizes the essential business entities required to manage invoicing, payments, subscriptions, notifications, and auditing in a financial operations context.

At the core, the **User** entity represents all system actors, including **Clients**, **Billing Administrators**, **Auditors**, and **Sales Representatives**. Users are assigned a **status** (active, suspended, or closed) and their actions are tracked through the **Log** entity, which records events such as logins, invoice creation, payments, and subscription changes.

Clients hold unique tax identifiers and addresses, and may subscribe to services through the **Subscription** entity, which defines billing cycles, amounts, and validity periods. Subscriptions generate **Invoices**, which track financial obligations, amounts, creation dates, and statuses (e.g., pending, overdue, paid, or partial).

Invoices are settled through **Payments**, which store transaction details, methods, and outcomes (successful, failed, refunded). Failed or suspicious invoices can be marked with a **Flag**, created by Auditors or Administrators, and later resolved by authorized users.

To ensure transparency and compliance, every significant action is recorded in the **Log** entity, while users are kept informed through **Notifications**, which may be delivered via email, push, or alert, and carry statuses such as sent, pending, or failed.

The relationships between these entities ensure that subscriptions automatically generate invoices, invoices trigger payments, and all critical actions are logged and auditable. This interconnected structure supports **InFlow's** goals of reliability, accountability, and user-centric financial management.

Production Configuration and Deployment Pipeline

This section describes the complete production configuration of the InFlow system, including runtime configuration, CI/CD pipelines, Terraform provisioning, container orchestration and monitoring stack.

Overview of the Production Architecture

The production environment runs on a Google Cloud VM and follows a hybrid deployment approach combining Docker Compose and Terraform:

- A unified Docker network (`inflow_net`) for all services.
- Base infrastructure (DBs, Keycloak, Kafka, Weaviate, monitoring) via Docker Compose.
- Backend and frontend replicas built and deployed through Terraform.
- GitHub Actions pipeline deploying automatically on the `develop` branch.
- Secure routing, SSL termination and API orchestration through Kong Gateway.
- Complete, Grafana, Loki, Promtail, Tempo and OpenTelemetry.

Environment Variables

Environment variables are injected dynamically during deployment. They are written to:

- `.env` — used by Docker Compose.
- `terraform.tfvars` — used by Terraform.

Table 1 summarises the main variable categories.

Category	Variables
Databases	POSTGRES_USER, POSTGRES_PASSWORD, DB_HOST, MONGO_DB
Kafka/Zookeeper	KAFKA_PORT, ZOOKEEPER_CLIENT_PORT, KAFKA_BROKER_ID
Keycloak	KEYCLOAK_ADMIN, KEYCLOAK_ADMIN_PASSWORD, KEYCLOAK_AUTH_SERVER_URL
Frontend build vars	VITE_BACKEND_URL, VITE_REDIRECT_URI, VITE_STRIPE_PUBLISHABLE_KEY
Stripe	STRIPE_SECRET_KEY, STRIPE_WEBHOOK_SECRET
Email SMTP	SMTP_HOST, SMTP_USER, SMTP_PASSWORD
HuggingFace	HUGGINGFACE_API_KEY, HF_API_URL, HF_MODEL
Observability	ELASTIC_APM_SECRET_TOKEN, OTEL_ENDPOINT

Table 1: Primary environment variable groups used in production.

Deployment Pipeline (GitHub Actions)

A GitHub Actions workflow automatically deploys the application whenever new code is pushed to the `develop` branch. The workflow includes pre-deployment cleanup steps to prevent permission conflicts with Allure report history.

CI/CD Workflow Structure

Listing 1: Deploy workflow with cleanup

```
name: Deploy InFlow Project (develop)

on:
  push:
    branches: ["develop"]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Pre-clean Allure history
        run: rm -rf allure-history/.git || true

      - name: Forcefully clean previous checkout
        run: |
```

```
                    sudo rm -rf ${{ github.workspace }}/allure-history/.
git || true

                - uses: actions/checkout@v3
                  with:
                    clean: false

                - name: Configure SSH
                  run: |
                    mkdir -p ~/.ssh
                    echo "${{ secrets.GCP_VM_SSH_KEY }}" > ~/.ssh/
google_compute_engine
                    chmod 600 ~/.ssh/google_compute_engine
                    ssh-keyscan -H 34.63.253.18 >> ~/.ssh/known_hosts

                - name: Deploy to VM
                  run: |
                    ssh -i ~/.ssh/google_compute_engine user@34.63.253.18
 << 'ENDSSH'
                      set -euo pipefail
                      cd ~/group-project-es2526_302
                      git reset --hard HEAD
                      git pull origin develop
                      # Generate .env, restart Compose, run Terraform
                    ENDSSH
```

Deployment Stages The deployment process follows these sequential stages:

1. **Pre-deployment cleanup**: Remove Allure history artifacts to prevent permission conflicts.
2. **Repository synchronization**: Reset and pull latest code from `develop` branch.
3. **Environment configuration**: Generate comprehensive `.env` file with all required variables.
4. **Base infrastructure deployment**: Launch Docker Compose services including databases, Kafka, Keycloak, and monitoring stack.
5. **Network validation**: Confirm `inflow_net` network is properly created.
6. **Terraform provisioning**: Generate `terraform.tfvars` and deploy backend/frontend replicas.

7. **Container cleanup**: Remove old replica containers before deploying new versions.
8. **Gateway configuration**: Reload Kong Gateway to apply routing updates.
9. **Database initialization**: Wait for Postgres health check, then apply `data.sql`.

Environment Configuration

The deployment workflow dynamically generates a comprehensive `.env` file containing all runtime configuration. Key variable categories include:

- **Database credentials**: PostgreSQL and MongoDB connection parameters
- **Message broker**: Kafka and Zookeeper ports and replication settings
- **Authentication**: Keycloak admin credentials and proxy configuration
- **Payment processing**: Stripe API keys and webhook secrets
- **Frontend build variables**: Backend URL, Keycloak endpoints, HMR configuration
- **Email service**: SMTP server credentials for notifications
- **AI services**: HuggingFace API key and model configuration
- **Observability**: Elastic APM token and OpenTelemetry endpoint

The workflow also generates `terraform.tfvars` with build-time variables for frontend container image construction, ensuring consistent configuration across all deployment artifacts.

Health Checks and Validation

Before completing deployment, the workflow performs several validation steps:

Listing 2: Health validation steps

```
# Display running containers
docker ps --format "table {{.Names}}\t{{.Ports}}"

# Validate network connectivity
docker network inspect inflow_net \
```

```
        --format '{{range $id, $c := .Containers}}{{printf "%s\
n" $c.Name}}{{end}}' \
        | sort

      # Wait for Postgres readiness
      until [ "$(docker inspect --format='{{.State.Health.
Status}}' \
        inflow-postgres)" = "healthy" ]; do
        echo "Postgres not ready yet..."
        sleep 3
      done

      # Initialize database schema
      docker exec -i inflow-postgres psql -U admin -d inflow \
        < ./InFlow/src/main/resources/data.sql
```

These checks ensure all services are operational and properly networked before the deployment is considered complete.

Terraform Provisioning

Terraform orchestrates backend and frontend replicas, building images and deploying containers attached to `inflow_net`.

Provider and Network

Listing 3: Terraform provider

```
terraform {
  required_providers {
    docker = {
      source = "kreuzwerker/docker"
      version = "~> 3.0"
    }
  }
}

provider "docker" {
  host = "unix:///var/run/docker.sock"
}
```

```
data "docker_network" "inflow_net" {
  name = "inflow_net"
}
```

Image Builds

Listing 4: Backend image build

```
resource "docker_image" "backend" {
  name = "inflow-backend:latest"
  build {
    context = abspath("${path.cwd}/../InFlow")
    dockerfile = "Dockerfile"
  }
}
```

Listing 5: Frontend image build

```
resource "docker_image" "frontend" {
  name = "inflow-frontend:latest"
  build {
    context = abspath("${path.cwd}/../InFlow_frontend")
    dockerfile = "Dockerfile.frontend"
    build_args = {
      VITE_BACKEND_URL = var.VITE_BACKEND_URL
    }
  }
}
```

Replicas

Listing 6: Backend replicas

```
resource "docker_container" "backend" {
  count = var.backend_replicas

  name  = "inflow-backend-${count.index + 1}"
  image = docker_image.backend.name

  networks_advanced {
    name = data.docker_network.inflow_net.name
  }
```

```
    ports {
      internal = 8080
      external = 8090 + count.index
    }
  }
```

Kong Gateway Routing

Kong exposes frontend, backend and Keycloak services externally.

Listing 7: Kong configuration

```
services:
  - name: backend
    url: http://backend:8080
    routes:
      - paths: ["/api"]

  - name: frontend
    url: http://frontend:5173
    routes:
      - paths: ["/inflow"]

plugins:
  - name: cors
    config:
      origins:
        - "https://<vm-ip>"
```

Observability Stack

- **Grafana** — dashboards.
- **Loki** — log aggregation.
- **Promtail** — log scraping.
- **Tempo** — distributed traces.
- **OpenTelemetry Collector** — pipeline for exporting traces.

Java CI Workflow

Pull requests to `master` and `develop` trigger unit tests.

Listing 8: Java CI workflow

```
name: Java CI with Maven

on:
  pull_request:
    branches: ["master", "develop"]

jobs:
  build:
    runs-on: self-hosted

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-java@v4
        with:
          java-version: "21"

      - name: Run tests
        run: mvn -B clean test --file InFlow/pom.xml
```

End-to-End Testing Workflow and Allure Reporting

Besides deployment automation, the CI/CD pipeline includes a comprehensive testing workflow responsible for executing backend unit tests, frontend end-to-end tests, merging test artifacts and publishing a unified Allure report. This workflow is triggered automatically for every pull request targeting the `master` or `develop` branches.

Testing and Reporting Workflow

The workflow is composed of three sequential jobs:

- **Backend Tests**: executes Maven tests and uploads Allure results.
- **Frontend Tests**: runs Cypress E2E tests via a controlled Vite server instance.
- **Report Publisher**: merges results and publishes the final Allure report to GitHub Pages.

Workflow Definition

Listing 9: Tests & Allure Report workflow

```
name: Tests & Allure Report

on:
  pull_request:
    branches: ["master", "develop"]

jobs:
  backend-tests:
    runs-on: self-hosted
    steps:
      - name: Clean workspace
        run: |
          sudo rm -rf ${{ github.workspace }}/* || true
          sudo rm -rf ${{ github.workspace }}/.* 2>/dev/null ||
 true

      - uses: actions/checkout@v4

      - uses: actions/setup-java@v4
        with:
          java-version: '21'
          distribution: 'temurin'
          cache: 'maven'

      - name: Build and run unit tests
        env:
          JAVA_HOME: ${{ env.JAVA_HOME_21_X64 }}
          SMTP_USER: ${{ secrets.SMTP_USER }}
          SMTP_PASSWORD: ${{ secrets.SMTP_PASSWORD }}
        run: mvn -B clean test --file InFlow/pom.xml -T 1C
        continue-on-error: true

      - name: Fix Allure permissions
        if: always()
        run: |
          sudo chown -R $USER:$USER InFlow/target/allure-
results || true
          sudo chmod -R 755 InFlow/target/allure-results ||
true

      - uses: actions/upload-artifact@v4
        if: always()
        with:
```

```
                name: maven-allure-results
                path: InFlow/target/allure-results
                retention-days: 1

      frontend-tests:
        runs-on: self-hosted
        needs: [backend-tests]

        steps:
          - name: Clean workspace
            run: |
              sudo rm -rf ${{ github.workspace }}/* || true
              sudo rm -rf ${{ github.workspace }}/.* 2>/dev/null ||
 true

          - uses: actions/checkout@v4

          - uses: actions/setup-node@v4
            with:
              node-version: '20'
              cache: 'npm'
              cache-dependency-path: 'InFlow_frontend/package-lock.
json'

          - name: Cache Cypress
            uses: actions/cache@v4
            with:
              path: ~/.cache/Cypress
              key: cypress-${{ runner.os }}-${{ hashFiles('
InFlow_frontend/package-lock.json') }}

          - name: Install Cypress dependencies
            env:
              DEBIAN_FRONTEND: noninteractive
            run: |
              sudo apt-get update -qq
              sudo apt-get install -y xvfb libgtk-3-0 libnotify-dev
 \
                libgconf-2-4 libnss3 libxss1 libasound2 xauth dbus-
x11

          - name: Install Node dependencies
            working-directory: InFlow_frontend
            run: npm install

          - name: Install Cypress
```

```
            working-directory: InFlow_frontend
            run: |
              rm -rf ~/.cache/Cypress
              npx cypress install --force

          - name: Start Vite server
            working-directory: InFlow_frontend
            run: |
              npm run dev > vite.log 2>&1 &
              echo $! > vite.pid
              sleep 10

          - name: Run Cypress
            working-directory: InFlow_frontend
            env:
              CYPRESS_baseUrl: http://localhost:5173
            run: |
              xvfb-run --auto-servernum \
                npx cypress run --browser electron --headless --env
allure=true
            continue-on-error: true

          - name: Stop Vite
            if: always()
            working-directory: InFlow_frontend
            run: |
              kill $(cat vite.pid) 2>/dev/null || true

          - uses: actions/upload-artifact@v4
            if: always()
            with:
              name: cypress-allure-results
              path: InFlow_frontend/allure-results
              retention-days: 1

      publish-report:
        runs-on: self-hosted
        needs: [backend-tests, frontend-tests]
        if: always()

        steps:
          - uses: actions/checkout@v4

          - uses: actions/download-artifact@v4
            with:
              name: maven-allure-results
```

```
          path: allure-results/maven

      - uses: actions/download-artifact@v4
        with:
          name: cypress-allure-results
          path: allure-results/cypress

      - name: Merge Allure results
        run: |
          mkdir -p allure-results-merged
          cp -r allure-results/maven/* allure-results-merged/
2>/dev/null || true
          cp -r allure-results/cypress/* allure-results-merged/
 2>/dev/null || true

      - name: Fetch previous history
        uses: actions/checkout@v4
        with:
          ref: gh-pages
          path: gh-pages
        continue-on-error: true

      - name: Preserve history
        run: |
          mkdir -p allure-history
          cp -r gh-pages/* allure-history/ 2>/dev/null || true

      - name: Generate Allure report
        uses: simple-elf/allure-report-action@master
        with:
          allure_results: allure-results-merged
          allure_history: allure-history

      - name: Publish report
        uses: peaceiris/actions-gh-pages@v4
        with:
          publish_branch: gh-pages
          publish_dir: allure-history

      - uses: actions/upload-artifact@v4
        with:
          name: allure-results-merged-${{ github.run_number }}
          path: allure-results-merged
```

Summary

This workflow guarantees that:

- backend and frontend tests run in parallel pipelines,
- Cypress E2E tests execute reliably inside a headless environment,
- Allure results from both test suites are merged,
- global test history is preserved,
- a public Allure report is automatically published to GitHub Pages.

This ensures high-visibility CI/CD validation and traceability for every pull request.

Conclusion

The InFlow production infrastructure combines modern DevOps automation, secure service orchestration and full observability. Using Terraform, Docker Compose and GitHub Actions, the system achieves fully automated provisioning and continuous delivery on Google Cloud.

Installation Guide

Purpose and Scope

This production setup guide serves as the reference for deploying the **InFlow** Payment and Billing System in a production environment. The guide is structured to support system administrators, DevOps engineers, and technical staff responsible for establishing and maintaining the platform infrastructure.

This document assumes deployment on a brand new Ubuntu Server 24.04.3 LTS installation and provides explicit instructions for all required software installations, configurations, and validations. No prior system configuration or preinstalled development tools are required beyond standard Ubuntu Server installation.

High-Level Architecture

The **InFlow** production deployment implements a distributed microservices architecture comprising the following core components:

- **Backend Services:** Java Spring Boot application (`InFlow`) providing RESTful API endpoints and business logic implementation
- **Frontend Application:** React-based single-page application built with Vite (`InFlow_frontend`) delivering the user interface
- **Data Layer:** Dual database architecture utilizing PostgreSQL for relational data and MongoDB for document storage
- **Messaging Infrastructure:** Apache Kafka with ZooKeeper coordination for asynchronous event-driven communication between microservices
- **Identity Management:** Keycloak with dedicated PostgreSQL database providing authentication, authorization, and single sign-on capabilities
- **API Gateway:** Kong Gateway serving as the single entry point for all client requests, managing routing, load balancing, and TLS termination with provided certificate and key files
- **Observability Stack:** Comprehensive monitoring solution including Grafana for visualization, Loki for log aggregation, Promtail for log collection, Tempo for distributed tracing, and OpenTelemetry Collector for telemetry data gathering
- **RAG Search Capability:** Weaviate vector database integrated with Hugging Face vectorizer for semantic search and retrieval-augmented generation
- **Embedding Services:** Ollama providing local large language model capabilities
- **Payment Processing:** Stripe CLI for webhook relay enabling payment event handling in development and production environments without requiring public ingress

This architectural design ensures separation of concerns, independent scalability of components, and fault isolation across the system.

System Preparation

Base System Updates

The initial step in preparing the production environment involves updating the system package repository and installing foundational dependencies. Execute the following commands with administrative privileges:

```
sudo apt update
sudo apt upgrade -y
sudo apt install -y ca-certificates curl gnupg lsb-release git
sudo apt install ubuntu-desktop
```

These commands perform the following operations:

1. Synchronize the package index with Ubuntu repositories
2. Upgrade all installed packages to their latest stable versions
3. Install certificate authorities for HTTPS communications
4. Install `curl` for data transfer operations
5. Install GNU Privacy Guard for cryptographic operations
6. Install `lsb-release` for Linux Standard Base version reporting
7. Install `git` for repository management
8. Install the Ubuntu Desktop environment for GUI access

Container Runtime Installation

Docker Engine

The **InFlow** system utilizes containerization for service deployment and orchestration. Docker Engine installation requires adding the official Docker repository and installing the container runtime:

```
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
    dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/
    keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(
    lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.
    list > /dev/null
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io
sudo usermod -aG docker $USER
newgrp docker
```

```
docker --version
```

The installation process establishes the following configuration:

- Creates secure keyring directory for repository authentication
- Downloads and installs Docker's official GPG key
- Configures Docker's APT repository with architecture-specific packages
- Installs Docker Engine, CLI tools, and containerd runtime
- Adds the current user to the `docker` group for non-privileged container management
- Activates group membership without requiring session logout
- Verifies successful installation through version reporting

Docker Compose

Docker Compose enables declarative multi-container application orchestration. Install the Docker Compose plugin:

```
sudo apt install -y docker-compose-plugin
docker compose version
```

Development Runtime Installation

Java Development Kit

The backend services require Java Runtime Environment and build tools. Install OpenJDK 21 LTS and Apache Maven:

```
sudo apt update
sudo apt install -y openjdk-21-jdk
java -version
sudo apt install -y maven
mvn -version
```

Node.js Runtime

The frontend application requires Node.js runtime for build processes. Install Node.js 22 LTS via NodeSource repository:

```
curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -
sudo apt-get install -y nodejs
node -v
npm -v
```

Source Code Repository

Clone the **InFlow** repository from the institutional GitHub organization:

```
git clone git@github.com:detiuaveiro/group-project-es2526_302.git
cd group-project-es2526_302
```

SSH key configuration with GitHub is required for repository access via SSH protocol. Ensure that SSH keys are properly configured in the GitHub account settings before attempting repository cloning. To create a new SSH key pair, use the following command and follow the prompts:

```
ssh-keygen -t ed25519 -f ~/.ssh/group-project-deploy -C "deploy key for
    detiuaveiro/group-project-es2526_302"
```

This creates:

- /.ssh/group-project-deploy (private key)
- /.ssh/group-project-deploy.pub (public key)

Then on GitHub go to the repository → Settings → Deploy keys → Add deploy key. Write a title (e.g., "Production Deploy Key"), and in the key field paste contents of /.ssh/group-project-deploy.pub. Finaly, click "Add key".

Environment Configuration

The system requires comprehensive environment variable configuration to establish service connectivity and authentication credentials. Environment variables must be configured according to deployment-specific requirements including database credentials, API keys, and service endpoints and shouldent be hardcoded in the source code for security and flexibility.

Access Organization OneDrive to obtain the `.env` easily. Or follow the instructions below to manually create the necessary configuration files.

On the root of the cloned repository, create a `.env` file and write:

```
# ============================================================
# Global
# ============================================================
DB_HOST=localhost
DB_USER=admin
DB_PASS=admin
BACKEND_URL=http://backend:8080

# ============================================================
# PostgreSQL
# ============================================================

POSTGRES_USER=admin
POSTGRES_PASSWORD=admin
POSTGRES_DB=inflow
POSTGRES_PORT=5432

# ============================================================
# MongoDB
# ============================================================

MONGO_PORT=27017
MONGO_DB=inflow

# ============================================================
# Kafka
# ============================================================

ZOOKEEPER_CLIENT_PORT=2181
ZOOKEEPER_TICK_TIME=2000
KAFKA_BROKER_ID=1
KAFKA_PORT=9092
KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1

# ============================================================
# Backend
# ============================================================
```

```
BACKEND_PORT=8080

# ============================================================
# Frontend
# ============================================================

FRONTEND_PORT=5173

# ============================================================
# API Gateway
# ============================================================

POSTGRES_KONG_HOST=kong

STRIPE_SECRET_KEY=sk_test_51SDLtTGJGqnnaqtXmmHM6AaSIAY4\
yVQBO2amNAogen1qmxJ1qSOeSE8nOeg43iTCLAJ7lKzWFVIxCPT3XTT\
pKnQ700BzzOFitD
STRIPE_WEBHOOK_SECRET=whsec_84571b7b771ec068aa30782df84\
da4e67046ab2ade159cf728d94a734fddd211

# ============================================================
# Keycloak (admin) - usados pela stack e pelo backend (programmatic
    admin ops)
# Dentro do Docker, o backend aponta por defeito para http://keycloak
    :8080/
# ============================================================

KEYCLOAK_AUTH_SERVER_URL=https://keycloak:8443/
KEYCLOAK_ADMIN=admin
KEYCLOAK_ADMIN_PASSWORD=admin
KEYCLOAK_IMPORT=/opt/keycloak/data/import/inflow-realm-realm.json
KC_HOSTNAME_STRICT=false
KC_PROXY=edge

# ============================================================
# Vite Frontend Env
# ============================================================
VITE_STRIPE_PUBLISHABLE_KEY=pk_test_51SDLtTGJGqnnaqtXgsh\
VrMvnAQPPMCEnBFZ7ACZ84l5logs1OdhGzQQ1khH7dc3SJge7zuCthfb\
NTzZxk0Ev044M003xBXPDmq
```

```
VITE_KEYCLOAK_URL=https://deti-engsoft-12.ua.pt:8081/
VITE_REDIRECT_URI=https://deti-engsoft-12.ua.pt/inflow/
VITE_BACKEND_URL=http://backend:8080
VITE_KEYCLOAK_TOKEN_URL_VM=https://deti-engsoft-12.ua.pt:8081/realms/
    inflow-realm/protocol/openid-connect/token
VITE_HMR_HOST=deti-engsoft-12.ua.pt
VITE_HMR_CLIENT_PORT=443
VITE_HMR_PROTOCOL=wss

# ============================================================
# SMTP
# ============================================================
SMTP_HOST=smtp.gmail.com
SMTP_USER=inflow.contacts@gmail.com
SMTP_PASSWORD=bjmu lsdf hqsf ehip

# ============================================================
# Flagsmith
# ============================================================
VITE_FLAGSMITH_ENV_KEY=Xi2doqGUtqKL9MBEtJYJh5

# ============================================================
# Observability (Elastic APM + OpenTelemetry)
# ============================================================
ELASTIC_APM_SECRET_TOKEN=ApiKey NGJSOTdwbOJsV01qem9KX2do\
WHM6YzlCYXg5TkRwTGRmV2Izc0c5SEhIZw==
OTEL_ENDPOINT=https://fe001816c490435a8b5996613bdf4640.ingest.europe-
    west1.gcp.elastic.cloud/

# ============================================================
# Conversational Assistant (Hugging Face)
# https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
# ============================================================
HUGGINGFACE_API_KEY=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
HF_MODEL=Qwen/Qwen2.5-7B-Instruct:together
HF_API_URL=https://router.huggingface.co/v1
```

Then on the InFlow_frontend folder, create a `.env` file and write:

```
# ==========================
# Keycloak (local)
```

```
# ==========================
VITE_KEYCLOAK_URL=https://localhost:8081/
VITE_KEYCLOAK_TOKEN_URL=https://localhost:8081/realms/inflow-realm/
    protocol/openid-connect/token
VITE_REDIRECT_URI=https://localhost:5173/inflow/

# ==========================
# Backend proxy (o /api ja e redirecionado pelo Kong)
# ==========================
# durante desenvolvimento local
VITE_BACKEND_URL=http://backend:8080

# ==========================
# Stripe
# ==========================
VITE_STRIPE_PUBLISHABLE_KEY=pk_test_51SDLtTGJGqnnaqtXgsh\
VrMvnAQPPMCEnBFZ7ACZ84l5logs1OdhGzQQ1khH7dc3SJge7zuCthfb\
NTzZxk0Ev044M003xBXPDmq

# ==========================
# Keycloak (producao na VM)
# ==========================
VITE_KEYCLOAK_TOKEN_URL_VM=https://deti-engsoft-12.ua.pt:8081/realms/
    inflow-realm/protocol/openid-connect/token

# ==========================
# Flagsmith
# ==========================
VITE_FLAGSMITH_ENV_KEY=Xi2doqGUtqKL9MBEtJYJh5
```

You can create and edit these files using `nano` or any text editor of your choice.

```
nano .env
nano InFlow_frontend/.env
```

System Deployment

Service Initialization

Execute the deployment script to build and initialize all system services:

Development Mode (with RAG Ingestion):

```
./run-project.sh dev --ingest
```

Production Mode:

```
./run-project.sh prod
```

Verification

Verify that all containers are running correctly:

```
./check-containers.sh
```

Important Consideration: Container initialization includes dependency resolution, database schema creation, and service health verification. Even after the deployment script completes, individual containers may require additional time to achieve full operational readiness. Monitor service logs to verify successful initialization.

Service Access Points

Upon successful deployment, the following services are accessible at their designated network endpoints:

- **Kong API Gateway:** https://<host-or-domain>/ (ports 80/443 mapped to internal ports 8000/8443)
- **Backend API:** http://<host-or-domain>:8080
- **Frontend Application:** http://<host-or-domain>:5173 (development mode; production deployments should serve built artifacts via Nginx or Kong routing configuration)
- **Keycloak Administration:** https://<host-or-domain>:8081 (Needs initial TLS handshake since Keycloak is configured to use HTTPS without a certificate. Please accept the browser warning on the endpoint before login.)
- **Grafana Dashboard:** http://<host-or-domain>:3000 (default credentials: admin/admin)
- **Loki Log Aggregation:** Port 3100
- **Tempo Distributed Tracing:** Port 3200

- **OpenTelemetry Protocol Endpoint:** Port 4318
- **Kafka Message Broker:** Port 9092
- **PostgreSQL Database:** Port 5432
- **Weaviate Vector Database:** Port 8085
- **Ollama Model Service:** Port 11434

Backend Build Configuration

The backend service (`InFlow`) utilizes containerized build processes defined in the `Dockerfile` within the `InFlow` directory.

Frontend Build Configuration

Production frontend builds utilize `Dockerfile.frontend`. Build arguments are injected from environment variables defined in the frontend `.env` file, enabling build-time configuration of API endpoints, authentication parameters, and feature flags.

Kong Gateway Configuration

Kong utilizes declarative configuration defined in `kong.yaml`. TLS configuration requires `cert.pem` and `key.pem` files present in the repository root directory. These certificates must be valid for the target domain and properly configured for the deployment environment.

Observability Infrastructure

The observability stack is configured through provisioning files:

- `grafana/provisioning`: Datasource definitions and dashboard configurations
- `loki`, `promtail`, `tempo`, `otel-collector`: Service-specific configuration files located in respective subdirectories

Weaviate Configuration

Weaviate vector database integrates with Hugging Face router endpoints utilizing the configured API key. Ensure adequate API quotas and service availability before deployment.

Ollama Configuration

Model data persistence is managed through the `ollama_data` Docker volume, ensuring model availability across container restarts.

Conclusion

The InFlow Payment and Billing System represents a comprehensive implementation of modern software engineering principles applied to a business-critical domain. Through the development of this platform, the project team has successfully demonstrated the practical application of microservices architecture, event-driven design, and DevOps practices in building a scalable, maintainable, and secure financial operations system.

Key Achievements

The project has delivered a production-ready system with the following notable accomplishments:

Architectural Excellence: The microservices-based architecture successfully separates concerns across independent services (Invoicing, Payments, Accounts, Audit, Reporting, Notifications, Conversational Assistant), each with dedicated data stores and clear API boundaries. This design enables independent development, deployment, and scaling of system components.

Comprehensive User Experience: The platform serves four distinct user personas—Billing Administrators, Customers, Sales Representatives, and Auditors—each with tailored interfaces and functionality. The system implements 27 user stories spanning authentication, invoice management, payment processing, notifications, audit capabilities, and sales representative functions.

Operational Maturity: The deployment infrastructure implements industry-standard practices including Infrastructure-as-Code (Terraform), containerization (Docker), automated CI/CD pipelines (GitHub Actions), and comprehensive observability (Grafana, Loki, Tempo, Promtail, OpenTelemetry). These practices ensure reproducible deployments, rapid iteration, and operational visibility.

Security and Compliance: The system implements role-based access control through Keycloak, TLS 1.3 encryption for data in transit, immutable audit trails with timestamped logging, and GDPR-compliant data handling practices. These measures establish a foundation for operating in regulated environments.

Event-Driven Integration: The Kafka-based messaging infrastructure enables asynchronous communication between microservices, supporting eventual consistency patterns and decoupled service interactions. This design improves system resilience and enables complex workflows without tight coupling.

Developer-Friendly Tooling: The project includes comprehensive documentation, automated testing workflows with Allure reporting, local development environment setup, and clear onboarding instructions. These resources reduce friction for new team members and support collaborative development.

Project Impact

The InFlow platform demonstrates practical capability in several dimensions:

Business Value: The system addresses real business needs in billing and payment management, providing automation, accuracy, and transparency that reduce operational costs and improve customer satisfaction.

Technical Proficiency: The project showcases competence in modern technology stacks including Spring Boot, React, PostgreSQL, Kafka, Keycloak, Kong Gateway, and comprehensive observability tools. The team successfully integrated these technologies into a cohesive system.

Software Engineering Discipline: The development process followed software engineering best practices including requirements analysis, architectural design, iterative development, continuous integration, automated testing, and comprehensive documentation.

Teamwork and Collaboration: The successful delivery of a complex distributed system required effective collaboration, clear communication, task coordination, and shared responsibility among team members.

Final Remarks

The InFlow Payment and Billing System successfully demonstrates that well-designed distributed systems can deliver complex business functionality while maintaining operational excellence. The project team has built a platform that is not only functional but also maintainable, observable, and positioned for future evolution.

The knowledge, skills, and practices developed through this project—spanning system architecture, distributed computing, DevOps automation, and secure software development—provide a strong foundation for future professional endeavors in software engineering.

The InFlow platform stands as evidence of what can be achieved through disciplined

application of modern software engineering principles, collaborative teamwork, and persistent attention to both functional requirements and quality attributes.

References and Resources

During the development of the **InFlow** platform, we will use several resources and tools that will be fundamental to the success of the project. Below are the main libraries, services, and references that we used in the elaboration and production of these report:

Atlassian. (n.d.). Continuous integration vs. delivery vs. deployment. Atlassian. Retrieved September 30, 2025, from https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Cloud IAM. (n.d.). Password policies. Cloud IAM Documentation. Retrieved September 30, 2025, from https://documentation.cloud-iam.com/resources/password-policies.html

Cloudflare. (n.d.). Role-based access control (RBAC). Cloudflare. Retrieved September 30, 2025, from https://www.cloudflare.com/pt-br/learning/access-management/role-based-access-control-rbac/

Cloudflare. (n.d.). Why use TLS 1.3?. Cloudflare. Retrieved September 30, 2025, from https://www.cloudflare.com/pt-br/learning/ssl/why-use-tls-1.3/

Commission of the European Union. (n.d.). Data protection. European Commission. Retrieved September 30, 2025, from https://commission.europa.eu/law/law-topic/data-protection_en

CookieYes. (2022, March 29). GDPR logging and monitoring: A complete guide. CookieYes. Retrieved September 30, 2025, from https://www.cookieyes.com/blog/gdpr-logging-and-monitoring/

CrowdStrike. (n.d.). The three pillars of observability. CrowdStrike. Retrieved September 30, 2025, from https://www.crowdstrike.com/en-us/cybersecurity-101/observability/three-pillars-of-observability/

Docker. (n.d.). What is a container?. Docker. Retrieved September 30, 2025, from https://www.docker.com/resources/what-container/

Integrate.io. (2023, July 21). Top REST API best practices for data integration. Integrate.io. Retrieved September 30, 2025, from https://www.integrate.io/blog/top-rest-api-best-practices-for-data-integration

Kubernetes. (n.d.). Kubernetes concepts overview. Kubernetes Documentation. Retrieved September 30, 2025, from https://kubernetes.io/docs/concepts/overview/

Microsoft. (n.d.). Scaling out vs. scaling up. Microsoft Azure. Retrieved September 30, 2025, from https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/scaling-out-vs-scaling-up

Microsoft. (n.d.). Microservices architecture. Microsoft Learn. Retrieved September 30, 2025, from https://learn.microsoft.com/en-us/dotnet/architecture/microservices/

Mozilla. (n.d.). Responsive design. MDN Web Docs. Retrieved September 30, 2025, from https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design

QAWolf. (2021, July 19). Guide to continuous deployment. QA Wolf. Retrieved September 30, 2025, from https://www.qawolf.com/content/guide-to-continuous-deployment

SRE Google. (n.d.). Service level objectives. SRE Google. Retrieved September 30, 2025, from https://sre.google/sre-book/service-level-objectives/

Swagger. (n.d.). OpenAPI Specification. Swagger. Retrieved September 30, 2025, from https://swagger.io/specification/

This Person Does Not Exist. (n.d.). AI-generated human faces. This Person Does Not Exist. Retrieved September 30, 2025, from https://this-person-does-not-exist.com/en

Veeam. (2021, February 8). Recovery time objective (RTO) and recovery point objective (RPO). Veeam Blog. Retrieved September 30, 2025, from https://www.veeam.com/blog/recovery-time-recovery-point-objectives.html

W3C. (2018, June 5). Web content accessibility guidelines (WCAG) 2.1. W3C. Retrieved September 30, 2025, from https://www.w3.org/TR/WCAG21/

Xavor. (2023, May 2). Principles of fault tolerance in microservices. Xavor Blog. Retrieved September 30, 2025, from https://www.xavor.com/blog/principles-of-fault-tolerance-in-microservices/

Medium. (2020, December 11). Why JSON is the preferred format for APIs. TechTrends Digest. Retrieved September 30, 2025, from https://medium.com/techtrends-digest/why-json-is-the-preferred-format-for-apis-af706fa8ff10