ChatGPT

Tax Intelligence Extension (Binding Compliance) – Brazil Implementation Specification

1. Canonical Tax Map & Determination Engine

The **Tax Intelligence Extension** uses a canonical tax determination model to ensure every transaction is evaluated against Brazilian tax rules consistently. It implements a decision-tree logic that first checks if a transaction is taxable or exempt, then identifies which taxes apply based on the transaction's nature (goods, services, financial, etc.) [1] [2]. For example, a sale of goods in Brazil triggers state ICMS and federal PIS/COFINS (and IPI if applicable), a service triggers municipal ISS and PIS/COFINS, and financial transactions may trigger IOF [1]. The engine also accounts for special cases like exports (which might be zero-rated) or small businesses below thresholds (which might be exempt) [2]. This ensures the correct taxes are determined for **B2B vs B2C** (e.g. interstate B2C sales apply destination-part ICMS) and for regime differences (standard regime vs. Simples Nacional).

Internally, the engine uses a **rule registry** to map transaction attributes to applicable tax rules. It looks up rules by jurisdiction, product/service type, and other flags to gather all candidate taxes, then filters by conditions (e.g. "buyer is consumer" or threshold requirements) [3]. All rules effective on the transaction date are considered, ensuring historical law changes are respected. By using this canonical tax map approach, the extension can systematically decide *which* taxes apply and *why*, providing a clear rationale for each tax on each transaction. This forms the foundation for binding compliance, as the same engine is used for real transactions and for generating official tax reports, guaranteeing consistency. The design also supports Brazil-specific logic such as Simples Nacional: if a company is under Simples, the engine can flag transactions for aggregate Simples calculation instead of per-transaction ICMS/PIS/COFINS (avoiding double-calculation). Likewise, personal income events are tagged for **Carnê-Leão/IRPF** workflows rather than per-transaction tax, integrating individual tax obligations into the canonical model.

2. Data Model & Schemas (TypeScript + SQL)

To implement the above, the extension defines a robust data model in both **TypeScript interfaces** and **SQL tables** for persistence. Key entities include:

- **TaxRule** – the canonical tax rule registry. Each rule record includes fields like `jurisdiction`, `item_type` (transaction category), `tax_type`, `rate`, and date ranges of validity [4]. Additional fields capture the calculation basis (net or gross), thresholds, conditional flags, and legal reference notes [5]. For example, a rule might be: **(jurisdiction = "BR-SP", item_type = "goods", tax_type = "ICMS", rate = 0.18, effective_from = "2025-01-01")** indicating 18% ICMS for goods in São Paulo [4] [6]. Rules are versioned rather than edited in place – if a rate changes, a new record is added with a new effective_from and the old one's effective_to is set to the day before [7]. This preserves history and allows the engine to fetch the correct rule by date.

- **Transaction** – represents a financial transaction (invoice, revenue, expense, etc.) that may have tax implications. Fields include `id`, `date`, `amount`, `currency`, `seller_id`, `buyer_id`, `item_type`, `description`, and flags like `is_export` or `is_exempt`. It also links to any source documents (e.g. NF-e or NFS-e numbers for audit). For Brazil, a transaction also carries the taxpayer's regime (normal, Simples, MEI) and participant tax IDs (CPF/CNPJ) as these affect tax treatment. In TypeScript, for instance:

```typescript
interface Transaction {
  id: string;
  date: Date;
  amount: number;
  currency: string;
  sellerId: string;
  buyerId: string;
  itemType: string;
  taxRegime?: "NORMAL" | "SIMPLES" | "MEI";
  flags?: { export?: boolean; import?: boolean; };
  sourceDoc?: string; // e.g., NF-e key or NFS-e ID
}
```

Corresponding SQL table `transactions` would enforce data types and relationships (e.g. `seller_id` references a taxpayers table, `item_type` perhaps references a catalog or is free text).

- **TaxResult** – stores the outcome of tax calculations for each transaction line or invoice. Each entry includes `transaction_id` (link to Transaction), `tax_type`, `jurisdiction`, `base_amount`, `rate`, `tax_amount`, and flags like `withheld` (if this tax is withheld by the payer) [8]. It also records the `rule_id` (or version) that was applied, providing traceability [9]. This table holds one row per tax applied per transaction. For example, if a sale had ICMS, PIS, COFINS, it would have three TaxResult rows. This normalized design allows easy aggregation by tax type or period.

- **TaxCreditLedger** – logs input tax credits for purchases. When the system identifies a creditable tax on a purchase (e.g. ICMS on inventory purchase for a company in the normal regime), it creates an entry here. Fields include `entity_id` (who can claim the credit), `tax_type`, `amount`, `origin_transaction_id`, and `period` (the period the credit applies to). This ledger can be keyed by period to sum credits for offset in filings [10] [11]. (Simples regime taxpayers would not use this, since they cannot claim such credits [12].)

- **PeriodSummary** – optional table summarizing total taxes by period (month/quarter) for each obligation. This can store the computed total ICMS, ISS, etc., for a given company and month, the status (calculated/paid/filed), and any references to filing documents. These summaries are generated by end-of-period jobs (see *Period Management*).

- **RuleChangeLog / Approval** – a ledger to track any changes to TaxRule or manual tax overrides. This records `change_id`, timestamp, user, the data before/after, and approval status. It works with an **approval workflow**: e.g., a tax analyst proposes a new rule or adjustment, and a supervisor approves it, after which the change is applied to the TaxRule table (ensuring SOX compliance) [13].

- **AuditTrail** – while not a single table, the system ensures all critical computations are logged. For instance, a `calc_events` table could log each calculation event with `transaction_id`, timestamp, user or system trigger, and a hash of inputs/outputs for integrity verification.

All these schema elements can be represented in TypeScript types (for front-end or service logic) and in SQL (PostgreSQL is recommended for its reliability and JSON support for flexible fields) [14]. By using a relational model, we can query and join data for reports or integrity checks (for example, cross-check that sum of TaxResult for ICMS in a period matches the PeriodSummary or matches what was reported in SPED). The unified model also supports both **transactional taxes and periodic taxes**: transactions feed into periodic calculations. For instance, each Transaction has a category that maps to an annual IRPF line or a quarterly corporate tax category, so the system can aggregate income and deductions for periodic filings [15]. This mapping is maintained via configuration tables (see *Starter Artifacts* for an IRPF mapping example). The **data model** is designed to be extensible and Brazil-first: it includes fields for Brazil's unique needs (like NF-e keys, Carnê-Leão flags) but can extend to other countries by adding rules and jurisdictions without changing code (canonical structure).

3. Computation Specification (Order, Rounding, FX, Credits)

Calculation Pipeline: The Extension's calculation engine processes each transaction through a sequence of steps to compute taxes accurately. When a new transaction is recorded (or an existing one updated), the engine performs:

- **Determine Taxable Base:** Start with the transaction's amount as the initial base. If prices are tax-exclusive (common in B2B sales), the base is the full amount. If prices are tax-inclusive (common in retail), the engine **grosses-down** the base for those taxes (solving base = price / (1 + rate) if needed) [16] [17]. For example, if an item price R\$109 includes ICMS at 9%, the base would be R\$100 (109/1.09). The engine also checks any *exemptions or minimums* at this stage: if a rule has a minimum threshold (say sales below R\$50 not taxed), and the transaction amount is below it, the tax base for that rule is effectively 0 [18]. This prevents applying taxes below regulatory thresholds. Multi-item invoices are handled line by line to ensure each item's tax base is correct before summing totals.

- **Apply Tax Rate:** For each applicable rule, the engine retrieves the correct tax rate (already determined via the rule lookup). It then calculates the raw tax amount = base * rate [19]. Most Brazilian transaction taxes are a flat percentage so this is straightforward. The engine supports more complex formulas too: e.g. progressive rates (for income tax) or tiered bases. Progressive taxes (like IRPF brackets or certain federal withholding) can be modeled as multiple rules with conditions on amounts, or handled by a specialized function if needed (e.g. summing portions of income in each bracket). In general, however, for indirect taxes like ICMS/PIS/COFINS, a single rate applies per rule.

- **Rounding:** The calculated tax is then rounded according to currency and legal requirements. The standard is to round to two decimal places (the cent, since Brazilian Real has 2 decimal places) using half-up rounding [20] [21]. For example, a tax amount of R\$15.376 becomes R\$15.38 if rounding half-up. The engine can apply specific rounding directives if a rule specifies (some taxes might always round up, or cumulative rounding might occur at invoice total). By default, each tax line is rounded independently to ensure compliance with how invoices display taxes [22]. (The system can also support alternative rounding modes or aggregate rounding if required by certain regs, but Brazilian filings typically sum rounded line amounts [22].)

- **Order of Operations & Compound Taxes:** The engine respects dependencies between taxes. If a tax must be calculated on an amount that includes other taxes (tax-on-tax), the engine calculates those prerequisite taxes first, then uses their results in the base of the dependent tax [23] [24]. For example, if a hypothetical municipal fee is 2% on the invoice total including ICMS, the engine will compute ICMS first, then add ICMS amount to net price to get the fee's base, then compute the fee. Brazil's ICMS itself sometimes is quoted as included in price (effectively a gross-up scenario), which the engine handles via the base adjustment step (calculating ICMS from a tax-included price requires solving the gross-up formula as above) [24]. All tax rules carry a marker if they should be applied after certain other taxes, and the engine executes in a sorted order to satisfy those constraints (essentially constructing a dependency graph of taxes). In most cases, the natural ordering by tax type suffices (e.g., calculate VAT before gross-receipts taxes). The **calculation spec** explicitly defines these sequences so that, for example, **ISS** (service tax) is computed on the net service amount, and then **PIS/COFINS** (which apply on gross revenue) might include the ISS or not depending on regulation (in Brazil, PIS/COFINS are on gross without ISS, but this can vary by interpretation). Such details are encoded in the rules or engine logic flags.

- **Currency Conversion (FX):** If a transaction is in foreign currency, the engine converts amounts to BRL for tax purposes. Brazil typically mandates using the official exchange rate (e.g. PTAX rate from Brazilian Central Bank) of the transaction date or the previous business day. The system either stores the exchange rate on the Transaction (fetched from an integration) or looks it up in a FX rates table. This ensures that, say, a USD transaction is taxed at the correct BRL equivalent. All subsequent calculations use the BRL amount, and the original currency and rate used are stored for audit. (For example, an export invoice of \$1000 might use an exchange rate of 5.00, so base = R\$5000. The rate used would be logged to justify the tax calculation in BRL.)

- **Input Tax Credits:** If the transaction represents a **purchase or expense** where the user is the buyer, the engine checks each tax for credit eligibility. VAT-type taxes like ICMS or non-cumulative PIS/COFINS are creditable for the buyer if they are in the normal regime. The engine does **not** alter the tax amount (tax is still calculated fully), but it will create a corresponding entry in the TaxCreditLedger for the same amount [25]. For example, if a company buys raw materials for R\$1,000 + R\$180 ICMS, the engine logs a credit of R\$180 ICMS for that company [25]. These credits can later offset output tax due. If the buyer is an end consumer or a Simples taxpayer (who cannot take credits), the engine skips logging credits [11]. The credit rules (creditable or not) are defined in the TaxRule (e.g., a boolean flag `creditable=true` for ICMS/PIS/COFINS rules, false for ISS since ISS has no input credit in Brazil [11]). This step ensures the system accumulates credits as transactions occur, so that compliance reports (like SPED or apuração statements) can compute net payable correctly.

- **Withholding Calculation:** For certain taxes, the obligation is on the payer to withhold a portion of the amount and remit it directly to the government. The engine identifies any such taxes after computing the amounts. In Brazil, examples include **IRRF** (income tax withheld at source on certain payments), some ISS cases (where the service buyer withholds ISS), and PIS/COFINS/CSLL withheld in certain service payments (commonly 4.65% combined). The TaxRule for such taxes has a flag `withholding=true`. When the engine sees a withheld tax, it marks the TaxResult as withheld and does **not** add that amount to the supplier's payable total [26] [8]. Instead, it segregates it for reporting as a liability of the payer. For example, if a service invoice of R\$1,000 results in ISS = 50 and IRRF = 150 to be withheld, the seller will only receive R\$800 (having R\$200 withheld), even

though PIS/COFINS etc. might still be added to the invoice [8]. The engine's output will clearly label which amounts are withheld by the customer. It can produce two subtotals: one for taxes that increase the invoice amount vs. those withheld from payment. This is important for invoice presentation and accounting integration. The engine also records metadata, such as which government entity each withheld tax must go to and by when [27] – for instance, IRRF must be paid to federal tax authorities by the 20th of the following month, ISS withheld might be due to the city by the 15th, etc. This metadata is used in generating the payment obligations (like DARF forms, covered later).

- **Output Assembly:** Finally, the engine compiles all computed tax details into a structured result. This result can be output as a JSON payload for the front-end or other systems, and is also saved to the TaxResult table for record. For example, an output JSON for an invoice might look like:

```
{
  "transaction_id": "INV-1001",
  "tax_details": [
    { "tax_type": "ICMS", "jurisdiction": "BR-SP", "base": 1000.00, "rate":
0.18, "amount": 180.00 },
    { "tax_type": "PIS", "jurisdiction": "BR-Federal", "base": 1000.00, "rate":
0.0165, "amount": 16.50 },
    { "tax_type": "COFINS", "jurisdiction": "BR-Federal", "base": 1000.00,
"rate": 0.076, "amount": 76.00 },
    { "tax_type": "ISS", "jurisdiction": "BR-SP (Municipal)", "base": 1000.00,
"rate": 0.05, "amount": 50.00, "withheld": true }
  ]
}
```

Each entry includes the tax type, jurisdiction, base, rate, amount, and a flag if it's withheld (as in the ISS example above). The `rule_id` or version could also be attached for traceability [9]. This breakdown is used to update the UI (the Tax Console will show these line items), to populate invoice printouts (listing taxes), and to feed accounting postings. The system ensures that effective dates were honored – e.g., if this transaction date is in 2024 and ISS was 5% that year, it used 5%; if in 2025 the rate changed to 4%, a transaction dated in 2025 would pick the new rule. The output is also logged along with the rule version and any special notes (e.g., "Export – ICMS at 0% applied" or "Simples regime – tax computed via Simples, see summary") for full auditability [28] [29].

Overall, this computation spec guarantees that all calculations are **reproducible and auditable**. Every formula (tax = base * rate, rounding rules, etc.) is either driven by data in the rules or explicitly coded in one place. The engine avoids hard-coding tax-specific logic beyond what the rules capture – for instance, instead of an if/else for each tax, it iterates through applicable rules and applies their parameters [30]. This data-driven approach means that when tax laws change (e.g., a new rate or a new tax), we update the rule data rather than deep code, reducing maintenance risk. Where complex logic is unavoidable (such as the multi-bracket IRPF calculation or Simples Nacional's percentage vs. deduction formula), the engine modularly implements those as separate calculation modules that plug into the workflow at the appropriate stage (for example, the Simples calculation runs in a monthly job, described later, rather than per transaction). All

results are rounded and stored in a consistent manner, ensuring the totals will match official forms to the centavo.

4. Compliance Workflows (IRPF, Carnê-Leão, MEI/Simples, DARF, NFS-e/NF-e Integration)

In addition to transaction-level calculations, the extension orchestrates compliance workflows specific to Brazilian obligations. These workflows aggregate and transform transaction data into filings or payments required by law, ensuring **end-to-end compliance**. Key workflows include:

- **Annual IRPF (Personal Income Tax) Workflow:** For individual taxpayers using the system, the extension accumulates all relevant income and deductible expenses throughout the year. Each transaction or income event is tagged with an IRPF category (salary, self-employment, rent, capital gains, etc.). At year-end, the system generates an **IRPF summary report** that maps these categories to the official IRPF declaration fields. For example, self-employment income received from individuals or abroad is classified under "Carnê-Leão" income in the annual return, whereas salary from a company with withholding is categorized separately. The extension can produce a **pre-filled IRPF form** or data file that includes total taxable income, taxes already paid (withheld by employers or via Carnê-Leão), and applicable deductions (e.g., contributions, dependents, medical expenses that were logged). This assists the user in filing their IRPF by the deadline (usually end of April). The system will flag if the user's income exceeds the annual filing threshold (e.g. around R\$33,000 of taxable income – it tracks this and alerts if a declaration is mandatory). All the data supporting the IRPF (transaction lists, receipts) are stored as evidence for audit. The goal is that by the time of filing, the user needs only to verify the compiled information and upload it to Receita Federal's IRPF program or possibly use an API if available. (In future, integration with the Gov.br API or e-CPF digital certificate could even allow direct submission, but currently it's prepared for manual filing.) By consolidating the data, the extension ensures no income is omitted and all eligible deductions are included, reducing errors.

- **Monthly Carnê-Leão (Individual Monthly Tax) Workflow:** The Carnê-Leão is a mechanism for individuals to pay income tax on certain types of income monthly, instead of only annually [31] . The extension supports this by maintaining a **monthly ledger of personal incomes** that are subject to Carnê-Leão (e.g. professional service fees received from individuals, rental income, foreign-source income) [31] . Each month, the user (or integrated systems) logs such earnings; the extension can also derive them from transactions marked as personal income. At month-end, the system calculates the tax due for that month using the progressive IRPF table applicable to monthly income. It applies the current exemption and brackets (for example, as of 2025, roughly the first R\$2,640 monthly might be exempt after the standard deduction, then 7.5% on the next portion, etc., up to 27.5%). It also accounts for allowable deductions in Carnê-Leão, such as dependent allowances or INSS contributions, as per Receita Federal's rules [32] . The result is the amount of income tax to be paid for that month. The extension then generates a **DARF payment slip** for the Carnê-Leão (using the appropriate revenue code, currently code 0190 for Carnê-Leão). The DARF includes the taxpayer's CPF, the month reference, and the amount due, and can be paid by the last working day of the following month [31] . The extension can output a PDF with a barcode or a JSON payload with the details to feed into payment systems. If the user is online, it may deep-link to the Receita Federal portal for Carnê-Leão [33] or use stored Gov.br credentials to automate submission. All monthly

payments are recorded. At year-end, the total Carnê-Leão paid is used as a credit in the IRPF annual calculation (the extension will ensure the IRPF summary includes taxes paid via Carnê-Leão so that the individual isn't double-taxed). If the user forgets or is late, the system flags the missed payment and can even calculate the interest and fine for late payment (the rules for late Carnê-Leão are applied). This workflow essentially automates the otherwise manual Carnê-Leão process [34], ensuring individuals stay compliant each month.

- **MEI (Microentrepreneur) Compliance:** For users classified as MEI (Microempreendedor Individual), the compliance is simplified but has strict limits. The extension allows an account to be marked as MEI, in which case it enforces the annual gross revenue limit (currently R\$81,000/year) and tracks it in real time [35] [36]. Each sale recorded adds to the MEI's yearly total; the UI will show a progress bar towards the limit and alert if nearing it. MEIs pay a fixed monthly amount (DAS MEI) which includes INSS and a small amount of ICMS or ISS depending on business type (commerce, services, or industry). The extension will not compute per-transaction taxes for MEI (no ICMS/PIS/COFINS on each sale), because by law MEI's sales are not individually taxed. Instead, it generates a **monthly MEI DAS** amount. For example, if the MEI is in commerce, the monthly tax is around R\$61 (a fixed ICMS + INSS); if in services, around R\$65 (ISS + INSS), values adjustable by government changes. The system can store the exact current MEI rates and auto-populate the monthly DAS. It reminds the user before the 20th of each month (the usual due date) to pay the DAS. Optionally, it can integrate with the PGMEI system to generate the payment slip – for instance, by providing a link or instructions for the user to log in and issue the DAS. Since MEI has no complicated tax calc, compliance workflow here is mostly **limit monitoring and payment reminders**. The extension also ensures that no input credits are tracked for a MEI (since they cannot use credits), and if the MEI exceeds the revenue threshold, the system will flag that they must transition to Simples Nacional (and possibly recalc taxes for the excess amount according to law). It could even initiate that transition in the system (changing the regime and informing the user of next steps).

- **Simples Nacional Compliance:** Small businesses under **Simples Nacional** benefit from a unified tax regime [37] [38]. Instead of calculating various taxes for each transaction, Simples taxpayers pay a single consolidated tax each month (the DAS) which covers federal, state, and municipal taxes (IRPJ, CSLL, PIS, COFINS, ICMS, ISS, etc.) [39]. The extension handles this by **aggregating monthly revenue** and applying the Simples Nacional rate tables. It maintains a mapping of the business's activities to the Simples "Annexes" (there are different tables of rates for commerce, industry, and service sectors) [40]. Each transaction for a Simples company is tagged with an Annex category (the system can derive this from item type or an explicit setting per product/service). At month-end, the total gross revenue for the month is computed for each category. The system looks at the **cumulative revenue of the last 12 months** (since Simples rates are progressive based on annual revenue) [41] [42]. Using the official Simples table, it determines the applicable bracket and effective tax rate. For example, if a service provider in Annex III has R\$500,000 revenue in the last 12 months, the system finds the bracket (e.g., some mid-range tier with an effective rate perhaps around 10%). It then calculates that month's DAS amount = (monthly revenue * rate) – (a fixed deduction if applicable, as per Simples law). The extension produces a breakdown showing how that DAS corresponds to underlying taxes (for internal info: e.g., of the 10% effective rate, X% is ICMS, Y% ISS, Z% federal, as defined by law [37]). It then generates the **DAS payment slip** or data to input into the Simples Nacional portal [43]. Typically, the user would use the Simples National application to generate the official DAS, but our system can streamline this by providing the exact numbers to input (gross revenue for each Annex, etc.). In a future integration, an API or automated browser task could even

submit these numbers and retrieve the DAS PDF. The due date (usually the 20th of the following month) is tracked. The system marks the obligation as fulfilled once the payment is recorded (the user can upload proof or if integrated with bank, it can detect payment). Simples National also requires an annual statement (Declaracao Anual do Simples), which the extension can assist with by summarizing the entire year's revenue and taxes paid. Additionally, the extension monitors if the 12-month revenue exceeds R\$4.8 million; if so, it warns that the company will be ineligible for Simples and may need to switch regimes [35] [44]. This workflow greatly simplifies compliance for small businesses by automating the complex rate lookup and ensuring the correct amount is paid each month in one go [41].

- **DARF Generation & Payment Management:** Many Brazilian tax obligations require a *Documento de Arrecadação de Receitas Federais (DARF)* or similar payment form (e.g., DAS for Simples). The extension centralizes the creation of these payment documents. For each tax that requires periodic payment (be it monthly, quarterly, or ad-hoc), the system knows the **revenue code** (código de receita) and the due date rules. Examples: IRRF on services (code 1708) due by the 20th of next month, monthly Carnê-Leão (0190) due by end of next month, quarterly corporate IRPJ (codes 2089/2090) due on the last day of quarter, etc. The system maintains a reference table of these codes. When the time comes to pay, it populates a DARF template with the taxpayer's CNPJ/CPF, the period (competência), the amount, and the code. It can generate a **bar-code line** for the payment slip according to Receita's specification, which the user can then use to pay via internet banking or at authorized banks. The UI will have a section listing all *Pending Payments*, each with a link to view/download the DARF. When a payment is completed, the user (or bank integration) can mark it as paid, storing the date and a receipt number. This ensures that the system not only computes the taxes but also closes the loop by facilitating payment and recording proof. Overdue payments: if a payment is past due, the system can compute interest and penalties (using Selic rate and 0.33% per day of delay up to 20%, per Brazilian tax law) and update the DARF amount accordingly, advising the user of the new amount. This is particularly useful for Carnê-Leão or occasional DARFs. By managing DARFs in one place, the user has a clear picture of what needs to be paid and can trust that each amount ties back to the calculated taxes. (For withheld taxes, the system prepares DARFs for those as well, which the company needs to pay on behalf of others – e.g., if R\$200 of IRRF was withheld from payments, a DARF for R\$200 code 1708 is generated). The DARF integration is crucial for **binding compliance** because it ensures the taxes calculated are actually remitted; the extension effectively becomes a mini tax payable ledger for the company.

- **Electronic Invoicing (NF-e and NFS-e) Reconciliation:** In Brazil, every sale of goods or service is documented by an electronic invoice (NF-e for goods, NFS-e for services in many municipalities). The extension integrates with these systems to ensure tax computations align 100% with issued invoices. For **NF-e (Nota Fiscal Eletrônica)**: the system can import NF-e XML data either via integration with the company's ERP or by using the SEFAZ web services (with the company's digital certificate) to query issued invoices. Each NF-e contains fields for all taxes (ICMS, IPI, PIS, COFINS, etc.) that were applied. The extension cross-checks that against its own calculations: any discrepancy (e.g., if an NF-e shows R\$180 ICMS but our engine calculated R\$178) is flagged as an anomaly for investigation. This reconciliation ensures that the tax engine's rules mirror what is on legally issued invoices, which is critical since the NF-e is the legal record [45]. If our system is driving the calculation pre-invoice, ideally no discrepancies occur. Additionally, the extension monitors **NF-e cancellations and returns**. Brazilian law allows an NF-e to be canceled within a limited window (often 24 hours, varies by state) if a mistake or transaction void [46]. If an NF-e is canceled, the system will mark the corresponding

Transaction as **canceled** and exclude it from tax totals (or rather, include a negative entry if the period had already been summed) [47]. The audit trail will keep the canceled record with a flag, and link to the cancellation event (including the NF-e cancellation protocol number as evidence). For partial returns of goods (devolução), which are issued as their own NF-e (with reference to the original), the extension will treat those as negative transactions that often generate tax credits for the buyer. It will compute the tax on the return NF-e (usually symmetrical to the sale) and log appropriate credits (e.g., buyer receives ICMS credit back) [48]. All such document references (NF-e keys) are stored, enabling easy retrieval of the actual XML/PDF if an auditor asks. For **NFS-e (service invoices)**: since each city has its own system, the extension supports at least linking out to those systems. Users can input their city and either upload NFS-e files or use an API if the city provides one. For example, São Paulo has a web service for NFS-e issuance; the extension could integrate so that when a service invoice is issued, it automatically records the details in the tax engine. The extension also generates any **municipal report** needed – some cities require a monthly summary of services and ISS paid [49]. The system can produce this summary from its data (a list of all NFS-e issued, their numbers and ISS amounts, which the user can then submit to the municipality). By reconciling with NF-e/NFS-e, the extension achieves compliance on documentation: every transaction in the tax system is backed by a legal invoice, and vice versa, ensuring nothing falls through the cracks.

In summary, these compliance workflows extend the tax calculation engine into full **compliance automation**. They cover the periodic filing and payment tasks that businesses and individuals must perform, using the engine's data as the source of truth. The design ensures that totals in the system will match the values reported in official filings, such as SPED contributions or DCTF, since those can be directly drawn from the database aggregations [45]. The system can generate SPED records or at least provide the data to fill them (e.g., total PIS/COFINS for each quarter, total ICMS for each state, etc.) [50], and prepare annual withholding statements like DIRF from its records [50]. By linking calculation with compliance, any discrepancy is eliminated, and the company's compliance status becomes a continuously monitored aspect of the system (if something wasn't calculated, it won't be on the obligation list, highlighting potential missing data). This tight integration of calculation, reporting, and payment is what makes the compliance "binding" – the outputs of the tax engine are directly used to meet legal obligations, with evidence at every step.

5. Evidence & Auditability (Hashes, Provenance, Approval Rules)

A core principle of this extension is that **every number can be traced back to its source**. To achieve strong auditability and trust, the system implements multiple evidence-preserving features:

- **Provenance of Calculations:** Each tax calculation is stored with references to *how* it was derived. As noted, every TaxResult entry links to the TaxRule (by ID/version) that produced it. The system also stores the relevant context, such as any condition flags (e.g., "buyer is final consumer = true") that were applied. This means an auditor or user can inspect a tax entry and see the exact rule and even the legal citation behind it. For example, if an auditor asks "Why was 7% ICMS applied on this interstate sale?", the system can show that rule (with id and version) which corresponds to the law (Complementary Law 87/96 interstate rate) and note that the buyer was a final consumer in another state [51]. The `TaxRule.notes` field often contains a short reference to the legal basis (like "ICMS interstate 4% per Res.13/2012" or "ISS per LC 116/2003") which is output alongside, so the

justification is clear. This provenance data is accessible via the UI by drilling into a TaxResult or by generating an audit report for a period.

- **Detailed Audit Trail & Change Logging:** All modifications to critical data (transaction amounts, tax rules, etc.) are logged. The system maintains an **immutable ledger** of changes where each record change is recorded as an event (with old value, new value, timestamp, user). Importantly, changes to tax configuration go through an *approval workflow*: a tax manager must approve rule changes before they become active [52]. For instance, if a tax analyst updates the PIS rate or adds a new municipal tax rule, the system can require a second user's approval (with a proper role) to move it from draft to active. This enforces segregation of duties in line with SOX/internal control best practices [13]. The change log allows auditors to see exactly *when* and *who* made any update, and to verify that two-person integrity was observed for compliance-critical changes. The UI provides a "Changes" history view that can filter by date or user, showing, for example, that "On 2025-04-01, user J.Doe updated rule #45 – ICMS rate from 17% to 18% effective 2025-04-01, approved by M.Smith on 2025-04-02".

- **Digital Signatures & Hashing:** To guard against undetected data tampering, the system computes hashes for critical records. Each Transaction and TaxResult might have a hash field that is included in the audit log events. Additionally, at period close, the system can generate a **hash of the period's summary** (like a checksum of all tax values in the period) and store it. This is similar to creating a sealed record of what was filed. If someone were to alter a transaction after period closure (which the app normally forbids without reopening the period), the hashes would not match, alerting to potential tampering [53]. We leverage the database's cryptographic functions (e.g., SHA-256) to generate these hashes. For even stronger integrity, the system could export these hashes or summaries to an external immutable store (or even blockchain, if desired) but even within the database, the snapshot history via techniques like Append-Only logging or using an **Apache Iceberg** table for transactions provides time-travel auditability [54]. The use of Iceberg or a similar versioned data store means that every change creates a new snapshot; an auditor can query the state of the data as of any date to see if records were altered after the fact [55].

- **Evidence Retention:** All source documents and calculations are kept for the statutory retention period and beyond. The system will store at least 5 years of data online (and older in archives) since Brazilian tax authorities can audit at least 5 years back [56]. In practice, we keep 6+ years to be safe. Instead of deleting or overwriting data when filings are done, the system marks periods as *closed* (read-only) but retains everything [57]. This includes storing copies of invoices (NF-e XML, NFS-e PDFs) and generated forms (the system can save PDFs of filed SPED reports, DARFs, etc., or at least the data). A **period close routine** might produce an *immutable audit file* – for example, a PDF summary of the period's taxes with a digital signature, listing every transaction and tax [58]. This file can be archived offline as well to present to auditors. By having these concrete evidence files, management and auditors get confidence that what was reported is backed by detailed logs.

- **Reproducibility:** Given that rules are versioned and data is never lost, the system can **recompute any historical period** and get the same result. To facilitate that, when rules change over time, the old versions remain accessible and the engine allows running in "as of date" mode. If an auditor in 2026 wants to know how 2024 taxes were computed, the engine can be pointed to 2024 data and it will automatically use rule versions effective in 2024 to recalc (which should match the stored results). This is an important check. We also implement a **simulation mode** (e.g., via a Jupyter notebook or an internal module) that can run a set of transactions through the engine outside of the

normal workflow [59]. This is used for testing and validation – e.g., before deploying a rule update, run last month's data through the new rule to see differences [60]. It's also evidence to regulators that we have strong internal controls to verify accuracy.

- **Automated Anomaly Detection:** The extension includes internal controls that automatically flag suspicious or unexpected results for review. For example, if in one month a particular tax amount drops or rises dramatically compared to trend, an alert is generated [61]. If a rule yields a negative tax (which shouldn't happen except in credit notes), it's flagged. If a user manually overrides a tax calculation (the system allows adjustments in special cases), it records the override and triggers an alert for management approval. These anomaly checks are logged and shown in the UI's "Anomalies" section. While not evidence per se, they are a governance tool ensuring that any irregularities are documented and explained, which itself becomes part of the audit trail (e.g., "Why was April's PIS so low?" might be answered by "Anomaly flagged: one large sale was reclassified as export, removing PIS – reviewed by tax team on May 5, 2025"). By catching anomalies early, the system helps prevent compliance issues (like underpayment) or at least ensures they are intentionally addressed (with evidence of the decision).

- **Access and Security Compliance:** Only authorized users can view or alter the data, enforced by role-based access control. For instance, a normal user might only view their own tax summary, whereas a tax admin can approve rule changes or make corrections. All access is logged. This means if an auditor asks "Who had access to change tax rates?", we can show an access log and permissions list. Moreover, personal data (like individuals' financial info for IRPF) is protected in line with LGPD (Brazil's data protection law) – e.g., sensitive fields can be encrypted at rest, and only accessible to users with a need to know. This ensures that in an audit, the company can demonstrate not only tax correctness but also data handling compliance.

In essence, the extension is built to produce an **audit trail** that satisfies both internal auditors and external tax inspectors. Every tax figure can be drilled down into: from annual return to monthly summary to individual transaction to the rule that caused the tax, and finally to the law or regulation that underpins the rule [51]. The combination of change approvals, hashing, and detailed logs means the data is *forensically sound* – one can prove it wasn't silently altered. Such evidence rigor is crucial given the heavy penalties for non-compliance in Brazil (fines, etc.), and it gives management confidence to rely on the system's outputs. The design aligns with standards like **Sarbanes-Oxley (SOX)** controls, ensuring accuracy and preventing fraudulent manipulation [62]. All these measures collectively turn the tax engine into a **system of record** for tax compliance, where outputs are legally defensible and trusted.

6. Period Management (Locking, Restatements)

Handling data by fiscal periods is critical in tax, since filings are usually done monthly, quarterly, or annually. The extension incorporates robust period management to control the lifecycle of tax data:

- **Period Definition:** The system organizes tax data into periods (e.g., monthly periods for VAT/ICMS/PIS/COFINS and ISS, annual for IRPF, etc.). Each relevant tax has an associated period type. A calendar of periods is defined (for example, Jan 2025, Feb 2025, … for monthly obligations). This allows the system to group transactions and results by the period in which they must be reported.

- **Closing/Locking Periods:** Once all transactions for a period are entered and the taxes computed and validated, the period can be "closed". Closing a period in the system means it becomes **read-only** for those transactions and tax results [63]. No further edits or additions are allowed that would alter that period's figures, unless the period is re-opened explicitly. This mimics the accounting practice of closing the books. For example, after January 2025's ICMS is calculated and the return filed, January 2025 is locked. Any late invoices or adjustments affecting January would either have to be accounted for in February (as an adjustment) or require reopening January with careful control. Locking ensures that the data used in the compliance filings remains intact and consistent with what was actually filed. The UI will visually indicate closed periods (and not allow editing transactions dated in those periods). If a user attempts to change something, a warning is shown that the period is closed and they must create an adjusting entry in a current period if needed.

- **Adjustments and Restatements:** In the event that an error is discovered in a closed period or a new document arrives that legitimately belongs to that period, the system supports **restatement procedures**. There are two main approaches: (1) **In-period adjustment** – reopen the period, incorporate the change, and mark that period's data as having been amended (and potentially require refiling an amended return); or (2) **Subsequent period adjustment** – keep the period closed and instead record a correction in the next period. The choice depends on regulatory allowance. For Brazilian federal taxes, usually one can file an amended return for a past month (with penalties/interest if it increases tax). The system will allow an admin to reopen a period (with an audit log entry noting who reopened and why). Once changes are made, the period can be reclosed and the system can mark it as "Amended" and even track multiple versions of a period's summary (original vs amended). Each run (original filing vs amendment) is logged with differences. This ties into the Jobs/Runs structure (see Backlog section) – an amended run would be a separate run for that period's job. If instead the user chooses to adjust in the next period, the system provides a way to input an "Adjustment Transaction" in, say, February that explicitly tags itself as correcting January. For instance, a forgotten invoice from January could be entered in February with a note, and the February calculations would include it (and perhaps an explanation line in February's return). This approach is often used for minor corrections to avoid refiling. The extension supports both, but encourages proper refiling if material.

- **Period Aggregation & Rollforward:** During an open period, users might want to see interim figures (e.g., mid-month how much ICMS so far). The system can generate live period-to-date summaries. At period close, final aggregation is done and stored in PeriodSummary. Those summaries (totals per tax) are what get reported or used to generate returns. The system ensures that the sum of individual transactions (from TaxResult) equals the PeriodSummary totals (this can be a check performed automatically at close). PeriodSummary also stores metadata like filing status (unfiled, filed, amended) and references to any government receipt numbers (for instance, the protocol number of a SPED file or the receipt of a DCTF submission).

- **Data Retention & Archiving:** Once a period is closed and perhaps a certain time passes, data might be archived. The system's design, however, **never deletes** transactional tax data after filing – it retains it for the audit window and beyond [56]. Old closed periods can be migrated to an archive database or data lake (like moving year 2018 transactions to an archival table) to keep the live database performant, but the data remains accessible when needed. The design with Apache Iceberg or similar means that even archived data can be queried seamlessly if needed for an audit, but doesn't burden daily operations [56]. Additionally, **backups** are taken at each period close (e.g., a

full backup of the database or an export of that period's data) and stored securely off-site [64]. This guards against catastrophic data loss (which itself could cause compliance issues if records are lost).

- **Period Locking Mechanism:** Technically, period locking might be implemented by a field `period_status` in the PeriodSummary or a separate Period table. When status is "closed", the application will reject any CRUD operations on Transactions in that period (the ORM or database triggers can enforce this). For example, an update trigger on `transactions` can check the transaction's date against the Period table and abort if the period is closed. This ensures even at the database level no accidental changes occur.

- **Overlap with Accounting Periods:** Often, accounting periods and tax periods align (monthly). The extension can integrate with the general ledger close process. For example, it can lock a period only after it has confirmed that all accounting entries for that period are finalized and the tax amounts have been reconciled to the accounting books. If a discrepancy is found (say the accounting says R\$10,100 of PIS due but our system says R\$10,000), that's resolved before locking. This cross-check can be part of the closing workflow.

- **Multi-year Considerations:** Annual obligations like IRPF or corporate income tax have their own period (fiscal year). The system will "lock" an annual period after the annual return is filed. If any adjustments are needed (e.g., a corrected IRPF), those are handled via an amendment record. The system could allow unlocking the annual period, but more typically one would file an amended return and keep track of the changes in a reconciliation statement. The extension could track something like "201X IRPF original vs amended difference".

- **Audit of Period Closure:** Closing a period is a significant event. The system logs who closed it and when. It could also require a second person sign-off (especially if adjustments were made) – e.g., a manager approves closing to ensure all tasks (payments, filings) are done. When closed, the system can generate a **period closing report** with all pertinent info: total taxes, list of anomalies resolved, etc., which is then archived (possibly signed) as mentioned [58]. This report can serve as a management sign-off document for compliance.

By implementing strict period management, the extension prevents the common pitfall of "drifting" data – where what's in the system no longer matches what was reported because someone edited history. Any restatement is deliberate, documented, and reflected in the system's outputs (like marking returns as amended). This gives confidence to stakeholders that once a period is done, those numbers are final and trustworthy. Moreover, in an audit scenario, one can retrieve the exact state of the system as of filing time, because of our no-delete policy and versioning. Period management functions like the **guardrails** of the compliance process, ensuring timeliness (periodic checks and close processes), accuracy (reconciliation before close), and completeness (no forgetting to file something because open periods will be visible in the dashboard until closed).

7. UI Specification (Tax Console: Obligations Table, Drilldowns, Exports, Anomalies)

The user interface, dubbed the **Tax Console**, is designed to give users (tax analysts, accountants, or individual taxpayers) a clear window into their tax obligations and the data behind them. Built with **Next.js**

and Tailwind CSS, it provides a responsive and accessible experience, even with offline support. Key UI components and features include:

- **Obligations Dashboard:** The main screen is an **Obligations table or calendar view** that lists all upcoming and recent tax obligations. Each row in the table represents a tax return or payment for a specific period (e.g., "ICMS – March 2025", "Simples DAS – Apr 2025", "Annual IRPF 2025"). Columns show the due date, status (e.g., *Pending*, *Filed on [date]*, *Paid*, *Overdue*), and amount due or filed. This provides at-a-glance awareness of what needs attention. Users can sort or filter by status or tax type. For instance, an accountant might filter to see all *pending* obligations in the next 30 days. Each obligation entry has an action button (like *"View/Resolve"*). Overdue items are highlighted (e.g., in red text), upcoming due soon maybe in amber. This table is essentially the compliance task list generated by the system's calculations.

- **Drilldown Details:** Clicking an obligation opens a **drilldown view** with detailed data. For a monthly tax, it shows how the system arrived at the amount. For example, clicking "ICMS – March 2025" would show a breakdown: total ICMS = R\$X, which is the sum of all taxable sales in March * relevant rates. The UI might list the top-level summary (state-wise if needed, e.g., ICMS SP: R\$1000, ICMS RJ (due to interstate): R\$100) and then allow the user to expand to see **transaction-level detail**. One could see each invoice contributing to ICMS, with columns like Invoice No., Date, Amount, ICMS Base, ICMS Tax. This is essentially the data from the TaxResult table filtered for that period and tax. Similarly, for Simples DAS, drilling in would show the revenue breakdown by categories and the applied rates (e.g., "Commerce: R\$50k * 4.5% = 2250; Services: R\$20k * 6% = 1200; total DAS = 3450"). The drilldown for IRPF would show each category of income and deduction aggregated. The interface enables exporting these details (see next point) and also supports **annotations** – a user can add a note to an obligation (e.g., "Reviewed by John, all good" or "Adjusted for missing invoice on Apr 5"). These notes are stored for audit trail and visible in the UI. Drilldowns also provide links to source documents: e.g., next to an invoice entry, an icon to view the NF-e PDF or the original record.

- **Data Export & Reporting:** For any detailed view, the UI provides export options (CSV, Excel, PDF). A tax manager can, for instance, export the full list of transactions for a period with all their taxes to share with an external auditor or to keep as a workpaper. The console also can generate standardized reports: e.g., a **SPED output file** (or at least a CSV ready to import into the SPED program) for PIS/COFINS or ICMS. Similarly, an **IRPF report** for individuals (which could be a pre-filled IRS declaration in PDF format summarizing incomes and taxes paid). Another export is the **DARF/DAS payment forms**. From the obligation view, the user can click "Generate DARF" and get a PDF to print or the code to pay online. All exports and generated files use templates with the company or user's info. The UI ensures that images like barcodes and the official layout are clear. Additionally, the UI might allow *scheduled emails* of certain reports (for example, email the monthly tax summary to the CFO each month once filed). For data backup or integration, the user can also export entire datasets (like all tax rules, or all transactions in a year). These features ensure the user can easily get data out of the system in human-readable or system-readable form as needed.

- **Anomalies & Alerts:** The Tax Console includes an **Anomalies panel** that highlights any issues the system detected. This might be shown on the dashboard (e.g., a section "⚠ Anomalies detected: 2" which links to details). In the anomalies view, each item is listed with a description: e.g., "Transaction #INV1005: Negative tax amount calculated (possible credit note) – check if correct," or "July 2025: PIS/COFINS amount is 50% lower than previous month – verify if revenue drop is expected." The UI

offers suggestions or next steps if possible (for a negative tax, it might suggest this is okay if it's a refund; for a drop, it might suggest checking if some sales were reclassified). The user can mark an anomaly as reviewed/cleared or leave it open. If an anomaly is still open when trying to close a period, the system will warn the user. This ensures that potential issues are acknowledged. The anomalies panel might also display any integration issues (e.g., "5 invoices from ERP were not found in Tax Engine" if any discrepancy in data ingestion, or "NFS-e report for São Paulo not uploaded for last month" if something expected is missing). Essentially, it's the UI's way to bring attention to things that could jeopardize compliance if not fixed. The design of this uses clear icons and coloring (e.g., yellow triangle for warning, red for critical anomalies). The user can click an anomaly to go to the relevant screen (for example, clicking the missing invoice anomaly could take them to an "Import Invoices" screen or the specific period's detail where data is missing).

- **Tax Rule Management UI:** Although much of the UI is focused on obligations, the extension also provides an interface to view and edit tax rules (for authorized users). This UI shows the list of rules (filterable by jurisdiction, tax type, etc.), and allows adding new rules or updating rates. It would include input validation (e.g., cannot overlap effective dates for the same jurisdiction/tax). Because changes require approval, when a user proposes a change, the UI will show the rule in a "pending approval" state to others. A supervisor can then approve it in the UI. All rule changes and their status are visible in a change log interface, as mentioned. Having a friendly UI for rules ensures that when laws change, the team can quickly update the system without needing a developer, which is essential for staying current. This UI can also show comparisons (like old vs new rates) and perhaps even link to external tax law references for convenience.

- **Offline Mode & Autosave:** The Tax Console is built as a modern web app with offline capabilities. Using service workers and client-side storage, it can allow certain functions even without internet. For example, if connectivity is lost, the user could still open recent data (which is cached on the client) and even input new transactions or adjustments. These would be queued locally and synced to the server once connectivity returns. This is especially useful for users recording expenses or incomes on the go (like a self-employed user might be at a client site and record a payment offline). The **autosave** feature ensures that as the user inputs data into any form (e.g., adding a manual transaction or editing a category), it's saved in real-time. There's no heavy "Save" button usage; instead, forms continuously save field by field or on short intervals. A small indicator might show "All changes saved" to assure the user. If offline, it might say "Offline – changes will sync when online" and those changes are stored locally until syncing. This prevents data loss and gives a seamless experience. Under the hood, every change the user makes (like adjusting a transaction's classification or adding a note) is recorded as a *change-set* which is appended to the ledger (with a temporary local ID if offline). When sync happens, the server consolidates these changes preserving order and integrity (the change-set ledger concept). The UI will then update with any recalculated results due to those changes. For example, if a user reclassifies a sale from taxable to exempt and that change saves, the system will recalc that period's taxes and the UI will refresh the obligation amount automatically. This live update feedback is achieved via WebSockets or polling so that users always see the latest picture.

- **Visual Design and Accessibility:** Using Tailwind CSS with modern design principles, the UI is clean and not overwhelming despite the data-heavy content. Colors are chosen in OKLCH format to achieve precise control and to ensure high contrast using the **APCA** contrast method. For instance, important text (like amounts or warnings) uses colors that meet APCA guidelines for readability on

both light and dark backgrounds. We adhere to accessibility standards: all interactive elements are keyboard-navigable and have appropriate ARIA labels (especially important in a complex dashboard). The design uses a responsive layout – on large screens you might see a dashboard with charts and tables side by side, whereas on mobile it might stack or simplify. The UI might also incorporate visualization for trends (simple sparkline charts for tax amounts over months, etc., to help identify anomalies visually). However, we avoid overly decorative elements in favor of clarity – tables, filters, and buttons are styled simply with Tailwind utility classes. A consistent design system (spacing, typography, color usage) ensures the console looks professional and is easy to scan.

- **User Assistance:** The interface provides contextual help – e.g., tooltips or info icons next to complex fields (like an explanation of "Difal" or "IRRF"). There might also be a help menu with documentation and the ability to summon an **AI assistant**. For instance, a user could ask the AI in natural language: "Why did my ICMS for March decrease?" and if integrated, the AI (using the system's data) might respond: "It looks like you had fewer taxable sales in March and one large sale was marked as export (ICMS zero-rated) [65], hence the drop." This kind of integration (if implemented) would appear as a chat or Q&A panel.

Overall, the Tax Console UI is the command center for compliance: it tells the user what needs to be done, provides the data to support each task, and ensures that interacting with the tax system is as efficient as possible. By implementing features like obligations tracking, one-click drilldowns, and automatic saves, we reduce the chance of user error (e.g., forgetting to include something) and make it easy to verify and trust the system's outputs. The offline and autosave capabilities cater to real-world scenarios where network or time to manually save shouldn't hinder compliance work. The UI is not just a read-only dashboard; it's an interactive tool where **compliance actions** (like filing, paying, adjusting) can be initiated and monitored, bringing the user into an active partnership with the automation.

8. Integration Points (Existing Engines, OCR, AI Agent, Observability)

The Tax Intelligence Extension is designed to **integrate seamlessly** with other systems and services in the fintech and compliance ecosystem. Key integration points include:

- **Integration with Existing Tax/Accounting Engines:** Many organizations already have ERP systems or accounting software with some tax capabilities. The extension provides APIs to integrate with these. For example, an ERP can call the extension's REST API endpoint `GET /taxes/calculate?transaction_id=XYZ` to retrieve real-time tax breakdown for an invoice [66]. Conversely, when a new invoice is created in the ERP, it can `POST` that transaction to our system (`POST /transactions`) so that the extension records it and computes taxes. This two-way integration ensures that the extension can slot into existing workflows: the ERP remains the source of transaction data and invoicing, while the extension is the source of tax logic and compliance. The API uses JSON payloads (aligned with our data model; e.g., including items, amounts, buyer info, etc.). For real-time needs (like e-commerce checkout tax calculation), the REST API provides answers within milliseconds given the rules are in memory [67]. For batch needs (like end-of-day posting of thousands of transactions), the extension can also consume messages via a queue (e.g., Kafka) – the ERP could publish events (new transaction, update transaction) and the extension's worker service subscribes and processes them asynchronously [68]. This decoupling is important for performance

and reliability at scale. If the organization has an existing global tax engine or a legacy system, the extension can be configured to either override it for Brazil or work in tandem (perhaps the legacy handles basic VAT, and our extension provides the additional Brazil-specific compliance workflows like SPED or Carnê-Leão, feeding results back to the main system). We ensure compatibility by offering data transformation mappings if needed (for example, mapping our canonical tax types to the codes used in SAP or Oracle systems). Additionally, the extension can output data in formats needed by other systems, such as CSV exports or database views that the accounting system can read for journal entries.

- **OCR and Document Digitization:** Not all inputs will come structured. The extension integrates with **OCR (Optical Character Recognition)** services to ingest unstructured tax documents. For example, if a user has paper receipts or PDF invoices that were issued outside the system, they can upload them. The extension uses an OCR engine to extract key data: dates, amounts, tax IDs, and tax breakdowns from these documents. A common use case is an accountant receiving a PDF of an NFS-e (service invoice) from a provider – instead of manually typing in the details, they upload it, and the OCR extracts the service value, ISS amount, provider CNPJ, etc., creating a transaction in the system. Another case is reading payment receipts: e.g., a PDF of a DARF payment confirmation can be parsed to mark an obligation as paid (the OCR can detect the barcode or the validation code on the receipt). We might use a combination of open-source tools (like Tesseract) and specialized cloud APIs (Google Vision or AWS Textract) for higher accuracy, especially for structured documents like invoices. For NF-e XML files, we don't even need OCR since they are already structured; we directly parse the XML. The OCR integration is mostly for scanned or PDF documents that aren't in our database. The UI will facilitate this by allowing drag-and-drop of files in relevant places (like an "Import Invoice" button). The system then presents the extracted data for review (since OCR can have errors) and once confirmed, it's inserted into the tax calc pipeline. This integration reduces manual data entry and helps ensure *all* relevant data enters the system. It's also useful for audit evidence: the system can attach the image of the document to the transaction record, so anyone reviewing can see the original source side-by-side with the extracted numbers.

- **AI Agent Integration:** The extension incorporates an **AI assistant** to enhance user experience and system intelligence. This AI agent can serve multiple roles:

- *User Query Assistant:* Users can ask natural language questions about their tax data or obligations, and the AI (with access to the system's data via a secure interface) can answer. For instance, a user might ask, "What is the total PIS I paid last year?" and the AI can compute and respond using the database. Or "Show me transactions with no NFS-e attached," and it can list them. This leverages the system's data in a conversational way, making it easier to navigate for non-expert users.
- *Explaining Tax Logic:* The agent can also explain complex tax concepts or the reason behind calculations (essentially translating what the system did into plain language). This is useful if a user is unsure why something was taxed a certain way – the AI could say, for example: "Invoice #123 was exempt from ICMS because the customer is outside SP state and the product was for export [65]. Thus, 0% ICMS was applied, which is why ICMS shows R\$0. PIS and COFINS were also not applied because exports are zero-rated for those [1]." This is pulling from the rules and references we maintain.
- *Legislative Watch:* We can integrate the AI with external data (like news or official diaries) to monitor tax law changes. For example, using NLP to scan for any legislative changes (e.g., "New law increases COFINS rate effective July 2025"). The AI could flag such news and even suggest the rule changes

needed. While final rule updates would be done by humans (or at least reviewed), this helps in staying ahead of changes.

- *Automation and Classification:* The AI can assist in categorizing transactions. If a user imports a bank statement or a generic expense list, an AI model could classify each line into tax categories (e.g., identify that a line with description "Uber" is a transport expense maybe with no invoice, or "John Doe Consulting" likely requires withholding). This speeds up data preparation for tax calculation by auto-suggesting how to treat items. The user can review the AI's suggestions.

The AI agent is accessible through the UI (perhaps a chat widget or a "Help me decide" button on forms). It is built with careful consideration of data privacy – it uses only the user's data and relevant tax knowledge, and does not expose sensitive info across users. Technically, this could be implemented with a service calling OpenAI's models or similar, with fine-tuned context (including our own documentation and rules as part of the prompt). The integration of an AI assistant is forward-looking, aiming to reduce the manual burden and provide educational value to users (many of whom are not tax experts).

- **Observability and Monitoring:** From a devops and reliability standpoint, the extension integrates with observability tools to ensure the system is running smoothly and to detect issues early. This includes:
- *Logging:* The application produces structured logs of key events (e.g., "Calculated taxes for transaction 123 in 0.05s", "Filed ICMS for 2025-03 for CNPJ X", "Error: Failed to fetch NF-e data for key Y"). These logs are integrated with monitoring systems (like Datadog, Splunk, or ELK stack) so developers and support teams can track the system's behavior. No personal tax data is in logs (to avoid sensitive info leakage), but enough context to troubleshoot (IDs, error codes).
- *Metrics:* The system emits metrics such as number of transactions processed per minute, time taken for batch jobs, success/failure counts of jobs, etc. For example, we track how long the monthly close job runs and whether it completes. We also track external integration health (like if the SEFAZ NF-e service is down, we might have a metric for failed NF-e fetch attempts). These metrics are fed to a dashboard and can trigger alerts. For instance, if a daily job fails or if an API endpoint latency spikes or if the anomaly count jumps unexpectedly, an alert can notify the support team via email/Slack.
- *Traceability:* Using something like OpenTelemetry, we can trace requests through the system. If a user complains that a calculation is slow, the trace might show it waited on the database or an external API. This helps performance tuning. Also, for complex operations like "Close period", we have sub-traces for each step (calculating, exporting, sending email) to pinpoint any slow or failing component.
- *High Availability & Sync:* If deployed in a cloud environment, the system integrates with load balancers and perhaps runs multiple instances. Observability includes heartbeat checks – the system has a health-check endpoint that monitoring pings to ensure the service is up. If we have a separate background worker for jobs, it registers its status as well.
- *Error Monitoring:* Integration with Sentry or a similar service captures exceptions on both backend and front-end. If, say, an unexpected condition causes a calculation to throw an error, it gets logged with context so developers can fix it. On the front-end, any UI errors (maybe a user sees a blank screen due to a JavaScript bug) also send an error report. This ensures the team can react swiftly to any bugs or edge cases that slip through.

Observability is crucial because any downtime or undetected error can lead to missed filings or incorrect taxes, which is unacceptable. By integrating these tools, we ensure the system's reliability is actively monitored. For example, if a scheduled job fails, not only does it raise an alert, but the system can also

surface that in the UI (perhaps an admin dashboard showing "Job failed, retry or contact support"). This transparency and quick detection mitigate the risk of non-compliance due to system issues.

- **Integration with Government Systems:** Beyond invoices and filings, there are other government touchpoints. For example, integration with the **Receita Federal e-CAC** portal via APIs (if available) to fetch things like the *Darf* status or to submit declarations (like using an automated script to upload DCTF files). The extension will explore such integration wherever possible. For instance, some cities provide API for NFS-e issuance which we would integrate to automatically issue service invoices from our system (if the user chooses) – so the tax calculation and invoice generation become one step. Another example: checking CNPJ or CPF statuses via the government API (to validate if a given ID is active or if a SIMPLES option is active for a company). These integrations ensure our data stays accurate (like not applying Simples calc if a company lost Simples status mid-year – we could query the Simples option API to verify status periodically).

All these integration points illustrate that the extension is **not an island**; it's designed to enhance and work with the existing software landscape. By providing open APIs and adopting standard formats, it's easy to plug into an enterprise's architecture. The OCR and AI bring external intelligence into the system, reducing human workload on data entry and analysis. And robust observability and monitoring integrations ensure that from a maintenance perspective, the system can be trusted to remain operational and correct – any deviation will be noticed and addressed. In a compliance context, this is vital: it's not just the code's correctness, but also the infrastructure's reliability that ensures deadlines are met and correct filings are made. Thus, integration with ops tools (logging/alerts) is just as important as integration with data sources.

9. Acceptance Criteria (Testable Scenarios & Thresholds)

To validate the system, a comprehensive set of acceptance tests and scenarios are defined. Each scenario has expected outcomes to ensure the implementation meets the requirements, especially for numeric accuracy and compliance rules. Below are several key test cases:

1. **Domestic Sale – Tax Calculation:** Given a sale transaction on 2025-01-15 of R\$1,000 for goods sold within São Paulo (seller and buyer in SP, normal regime), the system should determine **ICMS 18%** and **PIS/COFINS** at 1.65%/7.6%. The expected tax results: ICMS = R\$180.00, PIS = R\$16.50, COFINS = R\$76.00 (rounded to 2 decimals) [69]. There should be no ISS (not a service) and no withholding. The transaction's tax detail in the UI and JSON output should list these three taxes with correct amounts and the jurisdiction tags (ICMS as state SP, PIS/COFINS as Federal) [69]. The sum of taxes (R\$272.50) would be marked as part of the invoice total.

2. **Service Sale with Withholding:** Given a service invoice on 2025-02-10 of R\$1,000 issued by a service provider in Rio de Janeiro to a business customer also in Rio (thus ISS 5% applicable) – and assume PIS 1.65%, COFINS 7.6%, and also **IRRF 15%** must be withheld on service fees. The system should calculate ISS = R\$50.00, PIS = R\$16.50, COFINS = R\$76.00, and IRRF = R\$150.00. It should mark ISS and IRRF as **withheld** (based on configuration that ISS in RJ is withheld by the customer, and IRRF always withheld) [8]. The invoice's net payable to the provider should then be R\$1,000 + PIS + COFINS (which are not withheld) minus ISS – IRRF = 1000 + 92.5 – 200 = R\$892.50 net [8]. Acceptance criteria: The TaxResult entries for ISS and IRRF are flagged `withheld=true`, and the output clearly separates the withheld amount of R\$200 [8]. Also, the system should generate two

payment obligations for the customer: one DARF for IRRF (R\$150) and one municipal payment for ISS (R\$50) due to the city.

3. **Purchase with Credit Logging:** Given a purchase transaction on 2025-03-05 where our user (in SP, normal regime) buys goods of R\$1,000 + ICMS 18% from a supplier. The system will compute input ICMS = R\$180.00 on that purchase. Expected behavior: it logs an **ICMS credit** of R\$180 in the TaxCreditLedger for March 2025, attributed to the buyer's account [25]. This credit should appear in the UI (perhaps in a credits summary) and be available to offset output ICMS in the March return. Acceptance is that the credit ledger entry exists with correct amount and link to the purchase transaction, and that the March ICMS payable in PeriodSummary is effectively (output ICMS – input ICMS). (If no output ICMS in March, then it might carry to next month depending on regime, which is another scenario.)

4. **Threshold Exemption:** Given a small sale of R\$30 on 2025-03-20 by a small seller where rules say "if monthly gross sales < R\$100, no ISS is due" (some municipalities have de minimis exemptions). Suppose this sale would normally have 5% ISS = R\$1.50. The system should check the threshold and since perhaps the monthly total for that seller is still < R\$100, **ISS is not applied** [18]. Expected result: The TaxResult for ISS is either absent or present with amount R\$0 and a note "Below minimum base, exempt" [18]. PIS/COFINS might still apply if no such exemption, but ISS is skipped. The acceptance criteria is that the UI or output explicitly shows that no ISS was charged due to the threshold rule, matching the configured threshold in the rule. If later in the month the threshold is exceeded, the system should retroactively apply ISS (depending on law: some thresholds are per invoice, some per month – test both configurations).

5. **Interstate Sale – DIFAL:** Given an interstate sale on 2025-07-10 of R\$1,000 from a company in São Paulo (state SP) to an end consumer in Minas Gerais (state MG). By 2025, the interstate **DIFAL** (split ICMS) applies: origin state ICMS 7% and destination state ICMS 10% (example rates). The system should apply two ICMS rules: one for SP (origin) at 7% and one for MG (destination) at 10%, yielding R\$70 and R\$100 respectively, total ICMS R\$170. The acceptance is that the TaxResults include both lines with appropriate jurisdiction labels and descriptions (one might be labeled "ICMS (interestadual origem)", the other "ICMS DIFAL dest.) [70]. The sum (R\$170) matches the expected combined rate of 17% (since interstate consumer sales share 60%/40% in this hypothetical). Also, the system should allocate these amounts correctly in obligations: the R\$70 will be paid to SP (via the normal ICMS route) and R\$100 to MG (likely via a GNRE or similar, but at least flagged for MG). This scenario tests multi-jurisdiction tax on one transaction and passes if the split is correctly implemented per the rule definitions (which we set up with conditions "if interstate consumer then rule for dest ICMS").

6. **Simples Nacional Calculation:** Given a Simples Nacional company (Annex I: commerce) with revenue of R\$200,000 from April 2025 to March 2026 (12-month sum). In March 2026, they have revenue of R\$20,000. According to Simples rates (hypothetical: Annex I for that range might be ~8% effective rate), the system should calculate the March DAS at ~8% of 20,000 = R\$1,600. We expect the system to pick the correct bracket for R\$200k/year and output the unified tax. Acceptance: The PeriodSummary for March 2026 Simples shows R\$1,600 due. The breakdown might show that corresponds, for example, to R\$200 IRPJ, R\$180 CSLL, R\$300 ICMS, etc., internally [71]. We also expect the due date (April 20, 2026) to be assigned and a DAS form generated. If the company's revenue was, say, R\$4.9 million (exceeding the limit), the system should flag that and perhaps split the calculation (above limit revenue taxed outside Simples). But within range, it should

straightforwardly apply the correct rate from the table. This test passes if the computed DAS matches manually applying the Simples table for that revenue range, and the system generates the proper obligations (one combined payment).

7. **Annual IRPF Summary:** Given an individual user who during 2025 had: salary income R\$50,000 (with R\$5k withheld via payroll), Carnê-Leão income R\$12,000 (on which they paid monthly R\$1,500 total), and some deductible expenses like R\$2,000 of medical receipts. The system should aggregate these by category: Salary as "Income with tax withheld", Carnê as "Income from PF/Abroad (Carnê-Leão)" and medical as "Deduction – medical". It should compute the annual tax: e.g., total taxable income ~R\$62,000, compute progressive tax (say ~R\$7k), subtract the R\$5k + R\$1.5k already paid, resulting in ~R\$500 tax due (numbers illustrative). Acceptance: The IRPF workflow produces a report showing those totals and that the remaining tax due is R\$500. It should also list the amounts paid in Carnê-Leão that were credited [32]. The user would then file this and pay the R\$500 via DARF which our system can generate. The test is successful if the output aligns with an independent manual IRPF calculation for that scenario and if all components (incomes, withholdings, deductions) are correctly classified.

8. **NF-e Cancellation Handling:** Given an invoice #1001 on 2025-08-01 for R\$5,000 with taxes (ICMS, etc.) that was included in August's calculations, and that invoice gets canceled on 2025-08-05 via NF-e cancellation. The system should, upon receiving that cancellation (through integration or user input), mark invoice #1001 as canceled and **remove its tax impact** from August. Expected results: The TaxResults for that invoice are either voided or have corresponding negative entries, so that the August ICMS, etc. are reduced accordingly [46]. The system might create adjustment transactions on Aug 5 that negate the original. Acceptance criteria: The August PeriodSummary after cancellation should exclude that R\$5,000 – i.e., taxes lower by the amount that invoice had. Additionally, the audit trail for August will note "Invoice 1001 canceled, removed R\$X taxes" [47]. If the period was already closed, this should trigger either a reopening or a next-period adjustment. This scenario passes if after cancellation, the obligations match the new reality and the UI shows the invoice as canceled (maybe with a strike-through or a status) with no double counting. Also the system should enforce that cancellation happened within the allowed time window (if configured, e.g., within same month) – if not, it might require a different handling (like a credit note next month).

9. **Audit Trail and Reproducibility:** A non-numeric but crucial test: pick a past period, say September 2025 for a company, and reproduce its tax calculation from scratch. We input all transactions in a test environment and run the engine with rules effective as of 2025. The expected outcome is exactly the same totals that were originally produced and maybe filed. The acceptance is that using the same data and rule versions, the engine output matches to the cent (proving no data drift). Additionally, verify that for a random transaction, the UI's audit detail shows the correct rule citation [51] (for instance, an interstate sale's audit detail should mention the interstate ICMS rule and reference law 87/96 as we put in notes). This test ensures the auditability and versioning work as intended.

10. **Performance and Load Threshold:** While not a functional result, we define that the system should handle, say, **100 transactions per second** in calculation throughput, or process **10,000 transactions batch** within a minute without error. A load test is run with synthetic data. Acceptance: The system remains stable (no crashes) and results remain correct. Memory and CPU usage within acceptable bounds. This ensures that even at end of month with large volumes, the calculations complete in

time. Another threshold might be UI response: the obligations dashboard should load in <3 seconds even if there are hundreds of obligations listed, and drilldowns with thousands of lines should paginate or virtualize so that user experience remains smooth.

Each of the above scenarios will be tested with unit tests or integration tests. For numeric outcomes, a tolerance of 0.01 (one cent) is generally allowed for rounding differences (which are expected to be handled as specified). Many scenarios (like tax calc) are exact. Where randomness or AI suggestion is involved (like OCR or AI classification), acceptance is more about proper handling (e.g., AI suggestion can be wrong but the system should allow override easily; OCR extraction should capture at least 95% of fields correctly on standard documents in testing).

By meeting these acceptance criteria, we ensure that the extension works correctly for typical and edge cases, and that it upholds the compliance rules. Stakeholders (tax managers, auditors, end-users) can have confidence that if the system passes these tests, it will produce accurate and legally compliant results in production.

10. Starter Artifacts (JSON/CSV Examples, DARF Template, IRPF Mapping Table)

To accelerate implementation and ensure clarity, the project provides several **starter artifacts**. These artifacts serve as examples, templates, and initial data to be used in the development and testing of the extension:

- **Canonical Tax Rules CSV:** A sample **tax_rule.csv** file is provided to bootstrap the TaxRule registry. It contains a broad set of Brazilian tax rules as of the current date. Each row has fields like `jurisdiction, item_type, tax_type, rate, basis, threshold, effective_from, effective_to, notes`. For example, entries include:

```
jurisdiction,item_type,tax_type,rate,effective_from,effective_to,notes
BR-SP,Goods (domestic),ICMS,0.18,2025-01-01,9999-12-31,ICMS São Paulo internal [72]
BR-SP,Service (general),ISS,0.05,2024-01-01,2024-12-31,ISS São Paulo 2024 (5%) [73]
BR-Federal,All (non-cumulativ),PIS,0.0165,2017-01-01,9999-12-31,PIS non-cumulativo 1.65% [74]
BR-Federal,All (non-cumulativ),COFINS,0.076,2017-01-01,9999-12-31,COFINS non-cumulativo 7.6% [75]
BR-SP→BR-MG,Goods (B2C),ICMS Dest.,0.10,2022-01-01,9999-12-31,ICMS DIFAL dest. MG (10%) [70]
BR-SP→BR-MG,Goods (B2C),ICMS Orig.,0.07,2022-01-01,9999-12-31,ICMS DIFAL origin SP (7%)
... (more rules covering all states, services, Simples, etc.)
```

This CSV covers common scenarios so developers can load it into the database and have a working base. It includes Simples Nacional placeholders (though Simples is calculated differently, we list a generic rule like

“BR-Federal,All,Simples, [n/a rate] …” mainly to flag regime). It also contains some US and EU examples as placeholders to illustrate global extensibility (from previous project scope), but those can be ignored for Brazil-first development. The sample ensures that from day one, the engine can do a basic calc (e.g., it has an ICMS 18% rule for SP goods) [6]. During implementation, these rules will be verified against official tables, and any updates (like 2025 new rules) applied. The notes column provides context and can be used in the UI hover tooltips or audit logs.

- **Example Transactions & Expected Output (JSON):** A set of **JSON examples** is provided demonstrating input and output of the engine. For instance, `example_invoice_input.json` might describe an invoice with multiple items, and `example_invoice_output.json` shows the expected tax breakdown. We include cases like:

- A domestic goods sale JSON with expected ICMS/PIS/COFINS output.

- A service sale JSON with withholding, showing the `"withheld": true` flags.
- A Simples taxpayer sale JSON, where expected output perhaps just flags “Simples regime – taxes consolidated” rather than listing ICMS/PIS, etc.
- A purchase JSON with a flag `"purchase": true` leading to a credit entry in output.

For each, the output JSON follows the format in our spec (transaction_id, tax_details array) and these examples double as tests. For example, one JSON output might be:

```
{
  "transaction_id": "TXN-1001",
  "tax_details": [
    {"tax_type": "ICMS", "jurisdiction": "BR-SP", "amount": 180.00, "rule_id":
1},
    {"tax_type": "PIS", "jurisdiction": "BR-Federal", "amount": 16.50,
"rule_id": 4},
    {"tax_type": "COFINS", "jurisdiction": "BR-Federal", "amount": 76.00,
"rule_id": 5}
  ]
}
```

This corresponds to the earlier example of R\$1000 sale in SP [69]. By comparing the engine’s actual output to these examples during development, we validate correctness. We also provide a **batch example** where multiple transactions are input and an aggregate report is expected, to demonstrate the period summary creation.

- **DARF Generation Template:** An artifact is provided to detail how to construct a DARF form or the “linha digitável” (barcode text) from its components. This includes a **DARF template document** (perhaps a PDF or image) highlighting each field and a JSON schema for DARF data. For example, a JSON template:

```
{
  "taxpayer_id": "12345678900",
```

```
  "taxpayer_name": "Fulano de Tal",
  "code": "0190",
  "reference_period": "03/2025",
  "due_date": "30/04/2025",
  "principal_amount": 1500.00,
  "interest": 0.00,
  "penalty": 0.00,
  "total": 1500.00
}
```

And guidelines: the line barcode is Numeric and constructed as per Receita's spec using these fields (with modulo calculations for checksum). We include an example: for the above, the barcode might be something like `8366 0000 0123 4567 8900 0190 2504 1500 0000 0001500 00` (illustrative format). The developers can use this to implement the DARF generator module. Additionally, we list common **revenue codes**: 0190 (Carnê-Leão), 1708 (IRRF services), 0561 (salary withholding), etc., and where applicable the extension will use them. The template ensures the format of printed DARF (like a PDF) meets the official requirements (correct spacing, etc.). We also include a sample filled DARF PDF for reference. This artifact speeds up the implementation of the payment generation feature.

- **IRPF Mapping Table:** A crucial artifact is a **mapping of internal categories to IRPF declaration fields**. We supply this as a spreadsheet or CSV with columns: `category_code, description, IRPF_field, field_description`. For instance:

```
category_code,category_description,IRPF_field_code,IRPF_field_description
SALARY,Salary Income (with tax withheld),3A,"Rendimentos Tributáveis - Trabalho
assalariado"
CARNELAO,Self-Employment (Carnê-Leão),6A,"Rendimentos Tributáveis de PF/Exterior
(Carnê-Leão)"
DIVIDEND,Dividends,10,"Rendimentos Isentos - Dividendos"
MEDICAL,Medical Expenses,12,"Deduções - Despesas Médicas"
...
```

This table corresponds to sections of the Brazilian IRPF form. The system uses it to know where to place each sum. For example, category SALARY goes to line 3A of the IRPF form (which is typically salary from which tax was withheld). The code might also indicate how to treat withheld tax for that category (like for SALARY, the withheld tax is reported in DIRF and also credited in the IRPF). We include all common categories such as: salary, freelance income, rental income, foreign income, interest, dividends (exempt portion), capital gains (which in IRPF are on a separate schedule but at least indicated), etc., and deductions like education, health, dependents. During implementation, developers can load this mapping so that when generating IRPF output, the system can sum all transactions in each category and produce a structured report or even a dummy IRPF input file. This mapping artifact spares the team from manually figuring out form lines and ensures consistency with tax regulations (we compiled it from official IRPF guidance). It will

be updated if IRPF codes change year to year (some codes do update; the mapping might need an update for each tax year, which the system can accommodate by versioning this table per year).

- **Simples Nacional Rate Table:** As an aid for Simples Nacional calculations, we provide the official **Simples tables (Annex I to V)** as of the current year. This is given in a structured format (CSV or JSON) where for each Annex and revenue range, the applicable nominal rate and deductible portion are listed [40] [39]. For example (Annex I sample):

```
annex,from,to,rate,deduction
I,0.00,180000.00,0.04,0.00
I,180000.01,360000.00,0.073,5940.00
I,360000.01,720000.00,0.095,13860.00
... (up to 4.8M)
```

We do this for each Annex I through V. The extension will use this to calculate effective rates. We also note any special rules (like Annex V progressive deduction formula or situations where Annex III can shift to Annex V rates depending on payroll ratio – these complexities are documented in comments). Having this table saves time in coding those rates and ensures accuracy. It should be validated with official sources, which we have cited in the sources.

- **Sample Compliance Outputs:** As a final artifact, we include examples of compliance output files:
- A **SPED EFD-Contribuições** sample (for PIS/COFINS) in the text layout, showing how our data would populate the records (e.g., a block of C100, C170 records with our amounts).
- A **DCTF** summary example (maybe just a mock, since DCTF is submitted via software – we show what totals would go where).
- A **Municipal ISS report** example for a city like São Paulo (e.g., the total services value and ISS for that month).
- An **audit report** example: a PDF listing transactions and taxes for a period with a hash at the end.

These artifacts serve both as templates and verification tools. During development, the team will load and use these to test the system. For instance, after implementing rule processing, we'll load the sample rules and ensure that the example transactions produce the example outputs. When implementing IRPF logic, we'll use the mapping table on a test set of categorized incomes and see if the summary matches expected results. The DARF template will be used to generate a test DARF and visually confirm it matches the official layout (including scanning the barcode to ensure it encodes the right data).

Providing these artifacts greatly reduces ambiguity. They effectively capture the business rules and expected behavior in concrete form. For future maintainers, these also act as documentation. We will include them in the project repository (likely under a docs or resources folder). Some (like the rule CSV and mapping table) will actually become part of the product's initial data load. Others (like PDFs) are for reference.

In conclusion, the starter artifacts ensure that from day one the implementation team works with real-life scenarios and official formats, aligning the development with compliance needs. They also function as a validation suite – if the system can reproduce all the provided examples correctly, it is a strong sign of correctness and readiness to handle production data.

11. Backlog (Jobs → Runs → Steps)

The extension's computation and compliance processes are organized into a **Jobs/Runs/Steps** framework, which also reflects in the implementation backlog (the plan to build and deploy these features). We outline the key jobs the system performs, how they will be executed (runs and steps), and the backlog items to implement them:

- **Job Orchestration Framework:** First, a generic **Job Scheduler/Runner** will be implemented. This likely involves a database table `jobs` (defining the type of job, schedule or trigger, and relevant parameters like which entity or period) and a table `job_runs` for each execution attempt (with status, start/end timestamps, etc.). Each job can be broken into **steps** which are smaller tasks that can be checkpointed. In code, we might create a Job service that knows how to execute a given job type through its steps. We will implement the ability to retry failed runs, and to log step-level progress (so if a job fails at step 3, we know where and why). This is a backlog item because it's foundational: before creating specific jobs, we need the infrastructure. Acceptance: We can create a dummy job with 3 steps and see it go through them, mark success or fail properly.

- **Monthly Tax Calculation Job:** A core recurring job is *Monthly Close/Calculation* for each tax or set of taxes. For example, "Close taxes for March 2025 for Company X". This job would be scheduled to run after the period ends (e.g., on the 1st of next month or triggered by user when ready). **Steps** for this job:

- **Aggregate Transactions:** Query all transactions in the target period (that are not already closed) and run the tax determination engine on them to ensure all TaxResults are up-to-date. (If we calculate taxes at entry time, this step may just verify or recompute all to catch any late changes.) This step produces a raw list of tax entries.
- **Compute Period Summaries:** Sum up the tax entries by tax type to get total ICMS, total PIS, etc. Save these in PeriodSummary for that period (marking as draft).
- **Generate Filing Outputs:** Create any filing files needed (e.g., SPED or other returns). Populate forms or data for review. Not actually submitting, just preparing.
- **Generate Payment Slips:** For each payable tax (ICMS, PIS, etc.), generate the DARF/DAS forms with the amounts from summary.
- **Notify/Review:** Perhaps send a notification that "Taxes for March 2025 are ready for review" or mark the job as waiting for user approval.
- **Finalize/Lock Period:** Once user confirms, mark the period as closed in the database (so no more transaction changes). Then update filing status as filed. Log a final snapshot (e.g., store a JSON of results or hash).

Each of these steps is implemented and can fail independently. E.g., if generating a SPED file fails (maybe a formatting issue), that's step 3 failing; the system can retry just that after fixing the format bug. The backlog items include implementing each step's logic and testing them. We would maintain idempotence where possible (running step 1 again should yield same results, etc.). This breakdown ensures that even if something like an external system is down, we can retry that part without redoing everything.

- **Carnê-Leão Monthly Job:** A specific job for *Carnê-Leão monthly processing* for individual users. It triggers at month-end or when user requests. Steps:
- Sum up all personal income transactions of relevant categories for that month.

- Apply deductions (the system might prompt user to input any deductible expenses like dependents monthly if they haven't).
- Calculate tax due using progressive table for that month's income.
- Generate DARF for Carnê-Leão (via the DARF module).
- (Optional) If auto-pay integration, schedule the payment or at least notify the user via email with the DARF attached.

These steps then mark the Carnê-Leão for that month as completed. Backlog tasks: implementing the progressive calculation (taking into account if any new law like as of May 2023, there's a R\$528 monthly exemption as "simplified deduction" – the logic for that would be coded in step 3). Also tasks to fetch any user-specified info (like number of dependents which affect deduction).

- **Quarterly/Annual Jobs:** For corporate taxes (IRPJ/CSLL) if applicable, a quarterly job might aggregate profit/loss data and compute those, though if the extension doesn't manage full accounting, this might be limited to capturing numbers from outside. Similarly, an annual IRPF job:
- Gather all incomes from year (maybe it runs just after year-end).
- Fetch all amounts of tax paid (withholdings, Carnê).
- Prepare an IRPF draft summary or file.
- Perhaps even pre-populate the government's receipt system (though that might not have an API).
- Notify user that "Your IRPF draft is ready for review."

Steps here revolve around data aggregation and formatting for output. Backlog includes implementing this aggregation (using mapping table) and generating a human-readable report or even the official software import format if possible (maybe a .dec file).

- **Data Import/Sync Jobs:** There are jobs to regularly import data from external sources:
- **NF-e Sync Job:** Maybe runs daily to fetch any new NF-es issued to the company (via SEFAZ distribution if user's certificate is set up) and import them as transactions. Steps: connect to API, download new docs, parse each NF-e, upsert transactions, run tax calc on them to ensure consistency. Mark any differences or new credits from purchase NF-es.
- **Exchange Rate Update Job:** Daily job to fetch official FX rates from Bacen and store them, so any foreign currency conversion uses up-to-date rates.
- **Legislation Update Check (AI-assisted):** Could be a weekly job where the AI agent scrapes official gazettes or a feed (like CONFAZ updates) and reports if any tax changes are upcoming. Not fully automated rule change, but it could log a message or create a task for admin.

Backlog tasks: build connectors or scripts for these external data. For NF-e, for example, implement reading from SEFAZ web service (which might be complex, possibly skip if out of scope, but at least design for it).

- **Observability & Maintenance Jobs:** Some jobs ensure system health:
- **Anomaly Scan Job:** While anomalies are flagged in real-time during calc, a periodic job could scan recent data for patterns (like end of month to catch anything before filing). It might run a series of queries (e.g., check if sum of credits <= sum of debits for certain taxes, etc.) and log anomalies.
- **Data Backup Job:** Nightly, dump critical tables to a secure location or initiate a cloud backup.
- **Archive Job:** Yearly or as configured, move older closed periods to archive tables/files to keep DB lean (if needed).

These are more technical tasks. The backlog for these includes writing scripts or using cloud features.

- **UI and Manual Trigger Integration:** Many jobs can be triggered by user actions in the UI (e.g., "Run calculation for this month now" or "Regenerate Carnê-Leão DARF"). We ensure that the UI can kick off a job and the job runner handles it asynchronously (perhaps showing status in the UI, like "In Progress… done"). The backlog includes implementing a lightweight UI for job monitoring: showing recent job runs, status, and logs if failed (so a user or admin can see error messages like "Failed to connect to city API"). The job logs might also be accessible for audit (especially for filings).

- **Run Logs & Notifications:** After implementing jobs, we must ensure that when a job (run) completes, appropriate notifications or updates occur. E.g., when the Monthly Close job finishes, it might automatically send an email "Taxes for March 2025 have been calculated. Please review in the Tax Console." Or if a job fails, alert an admin. These are tasks in backlog to integrate with a notification service.

From a **development backlog perspective**, the Jobs→Runs→Steps model implies a need to methodically implement each piece. We might break the development process as:

1. Implement base job scheduler and database schema for jobs and runs.
2. Implement a simple example job (like a dummy or a simple daily data fetch) to test the framework.
3. Implement the Monthly Close job (which is complex) step by step: first get step 1 (calc all transactions) working and test on sample data, then step 2 (aggregation) etc. Use toggles to simulate success/fail scenarios and ensure run state updates correctly.
4. Implement Carnê-Leão job similarly.
5. Implement Simples job (maybe integrated with monthly close or separate).
6. Implement IRPF annual job.
7. Implement import jobs (NF-e, etc.) as needed.
8. Add UI triggers and status display.

Throughout, each job will have test cases (as per acceptance criteria above) to verify correctness. For instance, after Monthly Close job run, verify the PeriodSummary matches manual calcs and that the job's run record is marked success with timestamps.

Traceability in backlog: Every backlog item (user story or task) should map to a requirement in this spec. For example, a story "As a user, I want the system to lock a period after filing" corresponds to implementing the period locking step in the Monthly Close job and UI changes to enforce read-only, which we have covered. We ensure nothing is missed by cross-referencing the sections above with tasks.

Finally, **documentation of jobs** will be part of the system docs so that operators know, for example, that "Monthly Close Job" runs automatically on the 1st but can be re-run manually if needed, etc. This will be in backlog as a deliverable as well.

In summary, the backlog structured around jobs -> runs -> steps ensures that the development focuses on delivering the automation in manageable chunks, and the system itself, when live, has a robust way to manage complex processes in a controlled, auditable fashion [76]. By implementing this, we reduce the chance of human error (jobs run consistently) and we can recover gracefully from issues (re-run only what's needed). Each step and job ties back to the compliance requirements, ensuring completeness.

12. Risks & Mitigations

Implementing a comprehensive tax compliance system entails various risks. Below we outline major risk areas along with mitigation strategies integrated into our design:

- **Risk: Regulatory Changes and Law Complexity.** Tax laws in Brazil change frequently (new rules, rates, or even whole new taxes like the proposed CBS replacing PIS/COFINS) [77]. If the system isn't updated promptly, it could calculate wrong taxes, leading to non-compliance. *Mitigation:* We establish a **continuous monitoring process** for tax legislation [77]. The system is designed for easy updates: the tax rule registry can be updated without code changes, and we've included an approval workflow for rule changes to ensure accuracy. We also plan to subscribe to official updates and use the AI assistant to flag known upcoming changes (e.g., if Congress passes a law changing a rate, a task is created to update the rules effective that date). Additionally, extensive regression tests (including those starter examples and prior period recalculations) will be run when rules change, to ensure nothing else is broken. Our architecture's flexibility (data-driven rules) mitigates the risk of law changes making the system obsolete. In essence, we can deploy a rule update (or even a code update for something like a new tax formula) in short order to adapt. We will also maintain a buffer: for example, if a major reform is expected in 2026, we begin development ahead of time as soon as details are known, so the system is ready on day one.

- **Risk: Calculation Errors or Edge-Case Bugs.** Given the complexity (multiple taxes, rounding, etc.), there is a risk of the system calculating an incorrect amount (even a small rounding issue can accumulate or lead to fines). *Mitigation:* **Extensive testing and validation** will be conducted. We use real-world examples and cross-verify results with independent calculations or known-good systems (like verifying a sample invoice in a trusted tax software or manually). We've enumerated acceptance scenarios (above) which cover typical and edge cases – these will be automated tests. Additionally, the **simulation module** allows us (and users) to simulate scenarios easily to verify outcomes before real data is processed [60]. If any discrepancy is found in production (perhaps via user feedback or anomaly detection), we'll have logging to pinpoint the rule or step that went wrong. The system's design of isolating rules and formulas simplifies debugging (for example, if a specific combination yields a wrong result, we can adjust that rule without side effects). The anomaly detection also serves here: if an output tax is negative or far off expectation, the system flags it [61], effectively catching many potential errors automatically for review. This reduces the chance that an error goes unnoticed into filings.

- **Risk: Data Integrity and Audit Trail Gaps.** If data were lost or altered without record (due to bug or malicious action), compliance could be compromised (auditors might suspect tampering, or required evidence might be missing). *Mitigation:* We enforce **immutability and backups** strongly. The change-set ledger records all changes; nothing gets deleted – even voided transactions remain with a canceled status. We utilize database constraints to prevent accidental deletion of key records (perhaps only allow logical delete flags). Regular backups (and the period snapshots) ensure we can restore any lost data. The use of checksums and Iceberg-like snapshots means we can detect and prove if data was altered post hoc [55]. Access control mitigates malicious changes: only authorized persons can, say, reopen a period or edit a rule, and those actions are logged with user ID. In case of an audit, we have prepared an extensive audit trail, so the risk of *"no evidence for how number X was derived"* is mitigated – we can always produce the chain of calculations and the exact rule (and law

reference) [51] . This is critical for trust: the design essentially assumes we will be audited and needs to withstand that scrutiny.

- **Risk: Missing a Filing or Payment Deadline.** This could happen if the system fails to generate an obligation or if a job doesn't run on time, etc. *Mitigation:* The **Obligations dashboard** and notifications are designed to make it very clear what is due when. We also integrate with calendar/alert systems to remind users ahead of deadlines. On the system side, every scheduled job has a monitoring – if a job doesn't run (due to server downtime or bug), an alert is sent to admins. We keep failsafe: e.g., if for some reason the system is down near a deadline, users can still extract raw data (transactions list) to manually file as a backup. Having offline capabilities means users can at least view needed figures even if connectivity issues to server exist. We also plan redundancy in deployment (so a single server failure doesn't stop a job). Testing of the scheduler under various conditions (like heavy load on last day of month) will be done to ensure reliability.

- **Risk: Performance and Scalability.** If the volume of data grows (e.g., a large enterprise with millions of transactions) or near deadlines usage spikes, the system might slow down or crash, causing delays. *Mitigation:* The architecture separates heavy batch jobs from interactive use. Calculations can be done incrementally – we don't need to recalc the entire year every time, just the new transactions. We will implement caching where appropriate (like caching rule queries, or reusing results if nothing changed). The system can be scaled horizontally: for example, multiple worker processes can handle different companies or different jobs in parallel. Database indexing and partitioning (by company, by period) will be used to optimize queries (e.g., looking up transactions for one period is fast). We also used proven technologies (PostgreSQL, possibly Apache Iceberg for large logs) which handle large data well. We include performance tests in our pipeline. If certain operations are slow (say generating a huge SPED file), we might refactor or use streaming to handle it without consuming too much memory. The UI also has virtual scrolling for large tables, etc. Essentially, by design, any NFS-e or NF-e import that could bring thousands of records is done asynchronously (not blocking a user interface). These mitigations ensure that even high volume clients or peak times won't break the system. In worst-case, we have the ability to temporarily allocate more resources (cloud scaling) around deadlines. Monitoring will catch performance issues early (like if a daily job suddenly takes much longer, we investigate before it becomes a deadline problem).

- **Risk: Security and Privacy (LGPD).** The system handles sensitive personal and financial data. A breach or misuse could lead to legal penalties and loss of trust. *Mitigation:* We follow best security practices: encryption of data at rest and in transit, especially personal data like CPF, addresses, etc. We implement RBAC so users only see data they should. Audit logs for access mean we can tell if someone accessed data improperly. For LGPD compliance, we provide data export/delete capabilities if a user requests their data (though as a tax system, certain data cannot be deleted without violating tax retention laws; we'd anonymize if needed). All integrations (e.g., with AI or external APIs) are vetted not to leak data – for instance, if using OpenAI, we'd use a model that doesn't learn from our prompts, or we scrub personal identifiers from the query. Penetration testing will be done before go-live. The **Security & Compliance Framework** from the organization (if any provided guidelines like encryption, OAuth, etc.) will be adhered to (like using OAuth2 for user auth, ensuring strong passwords, 2FA for admin accounts). These reduce the risk of unauthorized data access. Backups are encrypted and stored securely to avoid a breach via backup files. In short, we treat

security on equal footing with functionality because a lapse could mean not only fines (under LGPD, etc.) but also attackers manipulating tax data (worst case, to commit fraud or cause mischief).

- **Risk: User Error or Resistance.** Users might input wrong data (e.g., classify something incorrectly) or be overwhelmed by the system's complexity, leading to errors or lack of adoption. *Mitigation:* We invest in a **user-friendly UI and guidance**. The system has validations on inputs (e.g., if a user tries to put a future date in a past closed period, it stops them). We also allow **review and approval** for critical changes by design (so one user's mistake can be caught by another) [13] . For adoption, we include training materials and possibly an onboarding wizard (for instance, to help initial setup of their regimes, etc.). The AI assistant can answer "How do I ..." questions to reduce confusion. We also ensure the system's output aligns with familiar forms (so users see the relation and trust it). Gradual rollout strategy can be employed: parallel run with their old method for a month to build confidence. By making the system supportive (autosave, suggestions, clear error messages), we mitigate the risk of user-caused compliance issues or them giving up on the software.

- **Risk: Integration Failures.** Dependence on external systems (like SEFAZ for invoice data or bank for payments) means if those integrations break, some automation fails. *Mitigation:* We handle integration errors gracefully: if NF-e import fails one day, the system logs it and tries later, and informs the user to possibly enter something manually if urgent. We keep the core calculation independent – e.g., even if OCR fails, user can enter data manually. Redundancy: if one data source fails, maybe use another (e.g., if an exchange rate API is down, have a secondary API or allow manual rate input). Also, when designing integrations, use abstraction so that if one service changes API, we can swap out the module without affecting the rest. Observability will catch integration failures immediately (e.g., an API key expired – we'd get an alert and fix it). For critical payments integration, we prefer generating output for user to pay rather than auto-paying (to avoid depending on a bank API being up at that moment – which might introduce risk of failed payment). Essentially, the system doesn't become non-functional just because an integration fails; it falls back to manual steps with clear user alerts.

- **Risk: Timeline and Scope Creep.** (From a project management view) There's a risk that developing all these features (especially advanced ones like AI integration) could take longer than expected, possibly delaying delivery and leaving insufficient time for testing. *Mitigation:* We prioritize core compliance features first (calculation engine, workflows for filings) – those are must-haves. More advanced nice-to-haves (AI assistant, full automation of NF-e fetching) can be phased in. Our spec clearly delineates what needs to be done, and our backlog is structured to implement in logical increments (as described in Backlog section). We'll deliver an MVP that covers primary obligations (e.g., ICMS, ISS, Carnê-Leão, Simples) and ensure those are stable, then iterate. By having the starter artifacts and a clear spec, we reduce development guesswork. Regular reviews with stakeholders will keep the project on track. This mitigates the risk of not meeting compliance by a certain tax deadline – the system will at least be able to support manual compliance with its data if some automation isn't finished. For instance, if AI agent isn't done by launch, that's fine (it's not critical), but if calculation wasn't done – that's unacceptable. So we schedule accordingly.

In conclusion, while the project is ambitious in scope, the design choices explicitly address these risks. By monitoring legislative changes [77] , thoroughly testing for correctness [59] , maintaining strong audit and security practices, and designing for resiliency and user-friendliness, we significantly mitigate the chances of compliance failure or system failure. Each risk has been thought through and countered with features or

processes in the system, aligning with the goal of a reliable, **binding** compliance solution that users and auditors can trust even as conditions change around it.

---

1 2 3 4 5 6 7 8 9 10 11 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 72 73 74 75 76 77 2-Tax Intelligence Guide for a Financial Entity Tax Map.pdf
file://file-5qUBhH5yYsUTZHXh7g2AkR

12 35 36 37 38 39 40 41 42 43 44 71 Simples Nacional Guide: understand and make the most of it
https://clmcontroller.com.br/en/taxes/national-simple/

31 32 33 34 Carnê-Leão — Receita Federal
https://www.gov.br/receitafederal/pt-br/assuntos/meu-imposto-de-renda/pagamento/carne-leao