Payroll Module Business Logic

1. Module Overview

The Payroll module is a Laravel-based submodule of the ERPFLOW SaaS ERP, located at `backend/Modules/Payroll`. It manages end-to-end payroll operations for a multi-tenant (company-scoped) environment, including:

- Payroll settings and overtime configuration
- Tax slab management
- Salary structure and component definitions
- Employee deductions (loans, advances, fines, other)
- Employee salary payment modes (channel splits)
- Monthly attendance freeze / unfreeze
- Attendance snapshots for payroll immutability
- Payroll run lifecycle (draft → processing → pending approval → approved → paid → locked)
- Payslip generation (chunked via queue)
- Salary advance requests with ceiling enforcement
- Payment allocation per payslip
- Disbursement batching and transmission

All API routes are protected by `auth:api` middleware and permission-based authorization. The module has no web routes (UI is React-based).

Source:

- `backend/Modules/Payroll/routes/api.php`
- `backend/Modules/Payroll/routes/web.php` (empty)
- `backend/Modules/Payroll/module.json`

3. API Route Map

All routes are inside `Route::middleware(['auth:api'])`.

HTTP	URI	Route Name	Controller@method	Middleware / Permission	Request Class
GET	`/health`	`health`	`PayrollController@health`	`auth:api`	—
GET	`/settings`	`settings.show`	`PayrollSettingController@show`	`payroll.settings-manage|advance-manage`	—
PUT	`/settings`	`settings.update`	`PayrollSettingController@update`	`payroll.settings-manage`	`UpdatePayrollSettingsRequest`
GET	`/tax-slabs`	—	`TaxSlabController@index`	`payroll.settings-manage`	—
POST	`/tax-slabs`	—	`TaxSlabController@store`	`payroll.settings-manage`	`StoreTaxSlabRequest`
GET	`/tax-slabs/{id}`	—	`TaxSlabController@show`	`payroll.settings-manage`	—
PUT	`/tax-slabs/{id}`	—	`TaxSlabController@update`	`payroll.settings-manage`	`StoreTaxSlabRequest`
PATCH	`/tax-slabs/{id}/status`	—	`TaxSlabController@updateStatus`	`payroll.settings-manage`	Inline
POST	`/tax-slabs/calculate`	—	`TaxSlabController@calculate`	`payroll.settings-manage`	`CalculateTaxRequest`
GET	`/salary-structures`	`salary-structures.index`	`SalaryStructureController@index`	`payroll.structure-manage`	—
POST	`/salary-structures`	`salary-structures.store`	`SalaryStructureController@store`	`payroll.structure-manage`	`StoreSalaryStructureRequest`
GET	`/salary-structures/{id}`	`salary-structures.show`	`SalaryStructureController@show`	`payroll.structure-manage`	—
PUT	`/salary-structures/{id}`	`salary-structures.update`	`SalaryStructureController@update`	`payroll.structure-manage`	`UpdateSalaryStructureRequest`
PATCH	`/salary-structures/{id}/activate`	`salary-structures.activate`	`SalaryStructureController@activate`	`payroll.structure-manage`	—
PATCH	`/salary-structures/{id}/deactivate`	`salary-structures.deactivate`	`SalaryStructureController@deactivate`	`payroll.structure-manage`	—
GET	`/salary-structures/{id}/components`	`salary-structures.components.index`	`SalaryStructureComponentController@index`	`payroll.structure-manage`	—
POST	`/salary-structures/{id}/components`	`salary-structures.components.store`	`SalaryStructureComponentController@store`	`payroll.structure-manage`	`StoreSalaryStructureComponentRequest`
PUT	`/salary-structure-components/{id}`	`salary-structure-components.update`	`SalaryStructureComponentController@update`	`payroll.structure-manage`	`UpdateSalaryStructureComponentRequest`
DELETE	`/salary-structure-components/{id}`	`salary-structure-components.destroy`	`SalaryStructureComponentController@destroy`	`payroll.structure-manage`	—
POST	`/salary-structures/{id}/components/reorder`	`salary-structures.components.reorder`	`SalaryStructureComponentController@reorder`	`payroll.structure-manage`	`ReorderSalaryStructureComponentsRequest`
POST	`/salary-structures/{id}/components/preview`	`salary-structures.components.preview`	`SalaryStructureComponentController@preview`	`payroll.structure-manage`	`PreviewSalaryStructureComponentsRequest`
GET	`/employee-deductions`	—	`EmployeeDeductionController@index`	`payroll.deduction-manage`	`IndexEmployeeDeductionRequest`
POST	`/employee-deductions`	—	`EmployeeDeductionController@store`	`payroll.deduction-manage`	`StoreEmployeeDeductionRequest`
GET	`/employee-deductions/{id}`	—	`EmployeeDeductionController@show`	`payroll.deduction-manage`	—
PUT	`/employee-deductions/{id}`	—	`EmployeeDeductionController@update`	`payroll.deduction-manage`	`UpdateEmployeeDeductionRequest`

HTTP	URI	Route Name	Controller@method	Middleware / Permission	Request Class
PATCH	`/employee-deductions/{id}/cancel`	—	`EmployeeDeductionController@cancel`	`payroll.deduction-manage`	—
GET	`/employee-deductions/{id}/entries`	—	`EmployeeDeductionController@entries`	`payroll.deduction-manage`	—
GET	`/employee-salaries/{id}/payment-modes`	—	`EmployeeSalaryPaymentModeController@index`	`payroll.payment-mode-manage`	—
PUT	`/employee-salaries/{id}/payment-modes`	—	`EmployeeSalaryPaymentModeController@replace`	`payroll.payment-mode-manage`	`UpdateEmployeeSalaryPaymentModesRequest`
GET	`/monthly-attendance`	—	`MonthlyAttendanceFreezeController@index`	`payroll.month-freeze`	—
POST	`/monthly-attendance/{id}/freeze`	—	`MonthlyAttendanceFreezeController@freeze`	`payroll.month-freeze`	—
POST	`/monthly-attendance/{id}/unfreeze`	—	`MonthlyAttendanceFreezeController@unfreeze`	`payroll.month-freeze`	`UnfreezeMonthlyAttendanceRequest`
GET	`/payroll-runs`	—	`PayrollRunController@index`	`payroll.run-create`	—
GET	`/payroll-runs/readiness`	—	`PayrollRunController@readiness`	`payroll.run-create`	`PayrollRunReadinessRequest`
POST	`/payroll-runs`	—	`PayrollRunController@store`	`payroll.run-create`	`StorePayrollRunRequest`
GET	`/payroll-runs/{id}`	—	`PayrollRunController@show`	`payroll.run-create`	—
GET	`/payroll-runs/{id}/approval-summary`	—	`PayrollRunController@approvalSummary`	`payroll.run-approve`	—
POST	`/payroll-runs/{id}/generate-payslips`	—	`PayslipController@generate`	`payroll.run-create`	`GeneratePayslipsRequest`
GET	`/payroll-runs/{id}/generation-status`	—	`PayslipController@generationStatus`	`payroll.run-create`	—
GET	`/payroll-runs/{id}/payslips`	—	`PayslipController@indexForRun`	`payroll.run-create`	`IndexPayslipsRequest`
POST	`/payroll-runs/{id}/build-snapshots`	—	`AttendanceSnapshotController@build`	`payroll.run-create`	—
GET	`/payroll-runs/{id}/disbursement-readiness`	—	`DisbursementController@readiness`	`payroll.disburse`	—
GET	`/payroll-runs/{id}/channel-summary`	—	`DisbursementController@channelSummary`	`payroll.disburse`	—
POST	`/payroll-runs/{id}/disburse`	—	`DisbursementController@disburse`	`payroll.disburse`	—
GET	`/salary-advances`	—	`SalaryAdvanceController@index`	`payroll.advance-manage`	`IndexSalaryAdvanceRequest`
POST	`/salary-advances`	—	`SalaryAdvanceController@store`	`payroll.advance-manage`	`StoreSalaryAdvanceRequest`
GET	`/salary-advances/{id}`	—	`SalaryAdvanceController@show`	`payroll.advance-manage`	—
PUT	`/salary-advances/{id}`	—	`SalaryAdvanceController@update`	`payroll.advance-manage`	`UpdateSalaryAdvanceRequest`
POST	`/salary-advances/{id}/approve`	—	`SalaryAdvanceController@approve`	`payroll.advance-manage`	—
POST	`/salary-advances/{id}/reject`	—	`SalaryAdvanceController@reject`	`payroll.advance-manage`	—
POST	`/salary-advances/{id}/record-payment`	—	`SalaryAdvanceController@recordPayment`	`payroll.advance-manage`	`RecordAdvancePaymentRequest`
PATCH	`/salary-advances/{id}/cancel`	—	`SalaryAdvanceController@cancel`	`payroll.advance-manage`	—
GET	`/salary-advances/summary`	—	`SalaryAdvanceController@summary`	`payroll.advance-manage`	`SalaryAdvanceSummaryRequest`
GET	`/disbursement-batches`	`disbursement-batches.index`	`DisbursementController@index`	`payroll.disburse`	`IndexDisbursementBatchesRequest`
GET	`/disbursement-batches/{id}`	`disbursement-batches.show`	`DisbursementController@show`	`payroll.disburse`	—
POST	`/disbursement-batches/{id}/send`	`disbursement-batches.send`	`DisbursementController@send`	`payroll.disburse`	—
POST	`/disbursement-batches/{id}/confirm`	`disbursement-batches.confirm`	`DisbursementController@confirm`	`payroll.disburse`	—
POST	`/disbursement-batches/{id}/retry`	`disbursement-batches.retry`	`DisbursementController@retry`	`payroll.disburse`	—
POST	`/disbursement-batches/{id}/acknowledge-all`	`disbursement-batches.acknowledge-all`	`DisbursementController@acknowledgeAll`	`payroll.disburse`	—
GET	`/disbursement-batches/{id}/export`	`disbursement-batches.export`	`DisbursementController@export`	`payroll.disburse`	—
POST	`/disbursement-batch-items/{id}/acknowledge`	`disbursement-batch-items.acknowledge`	`DisbursementController@acknowledgeItem`	`payroll.disburse`	`AcknowledgeDisbursementItemRequest`
GET	`/payslips/me`	`payslips.me`	`PayslipController@me`	`payroll.payslip-view-own`	`ListMyPayslipsRequest`
GET	`/payslips/{id}`	`payslips.show`	`PayslipController@show`	`payroll.payslip-view-own|payroll.payslip-view-all`	—
POST	`/payslips/{id}/regenerate`	`payslips.regenerate`	`PayslipController@regenerate`	`payroll.run-create`	`RegeneratePayslipRequest`
GET	`/payslips/{id}/download`	`payslips.download`	`PayslipController@download`	`payroll.payslip-view-own|payroll.payslip-view-all`	—
GET	`/payslips/{id}/allocations`	`payslips.allocations`	`PayslipController@allocations`	`payroll.run-create|payroll.payslip-view-own|payroll.payslip-view-all`	`ListPayslipAllocationsRequest`
GET	`/attendance-snapshots`	`attendance-snapshots.index`	`AttendanceSnapshotController@index`	`payroll.run-create|payroll.run-override-readiness`	`ListAttendanceSnapshotRequest`
GET	`/attendance-snapshots/{id}/divergence`	`attendance-snapshots.divergence`	`AttendanceSnapshotController@divergence`	`payroll.run-create|payroll.run-override-readiness`	—
GET	`/attendance-snapshots/{id}`	`attendance-snapshots.show`	`AttendanceSnapshotController@show`	`payroll.run-create|payroll.run-override-readiness`	—

Source: `backend/Modules/Payroll/routes/api.php`

4. Actors

Authenticated API User

- Any user with a valid API token.
- Must belong to a company (tenant context).
- Specific actions require specific permissions (see Section 5).

Employee (as Data Subject)

- Represented by `employee_id` foreign keys.
- Does not directly call APIs unless the user is also the employee.
- Owns payslips, deductions, advances, and payment modes.

Payroll Administrator / HR User

- A user holding permissions such as `payroll.settings-manage`, `payroll.run-create`, `payroll.advance-manage`, `payroll.disburse`.
- The code does **not** define a hardcoded "HR" role; authorization is purely permission-based.

Approval Gateway / Platform Executor

- The `PayrollRunExecutor` and `SalaryAdvanceExecutor` run as part of the platform approval workflow.
- They transition statuses and finalize data when an approval request completes.

Queue Worker

- Processes `GeneratePayslipChunkJob` for asynchronous payslip generation.

---

5. Permissions & Authorization

Permission	Where Checked	Protects	Without Permission
`payroll.settings-manage`	Route middleware	Settings write, Tax Slab CRUD	403
`payroll.advance-manage`	Route middleware	Settings read (alongside settings-manage), Salary Advance CRUD	403
`payroll.structure-manage`	Route middleware	Salary Structures & Components	403
`payroll.deduction-manage`	Route middleware	Employee Deductions	403
`payroll.payment-mode-manage`	Route middleware	Employee Salary Payment Modes	403
`payroll.month-freeze`	Route middleware + Policy	Monthly attendance freeze/unfreeze	403
`payroll.month-unfreeze-paid`	Policy + Service	Unfreezing a month that has a paid payroll run	403
`payroll.run-create`	Route middleware	Payroll run CRUD, Payslip generation, Attendance snapshots	403
`payroll.run-override-readiness`	Controller + PermissionEngine	Creating a run when employees are not ready	403
`payroll.run-approve`	Route middleware	Run approval summary, approval workflow	403
`payroll.payslip-view-own`	Route middleware	View own payslip / download / allocations	403
`payroll.payslip-view-all`	Route middleware + Controller private method	View any payslip / download / allocations	403
`payroll.disburse`	Route middleware	Disbursement batches & items	403

Source: `backend/Modules/Payroll/routes/api.php`, `MonthlyAttendanceApprovalPolicy.php`, `PayslipController.php`, `PayrollRunController.php`

---

6. Payroll Entities & Database Tables

`payroll_settings`

- **Purpose:** Company-level payroll configuration.
- **Key Fields:** `company_id` (unique), `overtime_multiplier`, `overtime_rate_base` (`gross` / `basic`), `standard_monthly_hours`, `hourly_rate_method`, `prorate_method`, `round_net_pay_to`, `advance_enabled`, `advance_default_method`, `advance_default_value`, `advance_max_percentage`, `advance_max_amount`, `advance_requires_approval`, `updated_by`.
- **Scope:** One row per company.

`tax_slabs`

- **Purpose:** Income tax slab definitions per company per year.
- **Key Fields:** `company_id`, `name`, `effective_year`, `slabs` (JSON array of bands), `status` (`active` / `inactive`), `annual_income`, `created_by`, `updated_by`.
- **Scope:** Company + effective_year. Only one active set per year enforced by code.

`salary_structures`

- **Purpose:** Reusable salary templates.
- **Key Fields:** `company_id`, `name`, `code`, `requires_basic` (bool), `status` (`Active` / `Inactive`), `created_by`, `updated_by`.
- **Scope:** Company. Code must be unique per company.

`salary_structure_components`

- **Purpose:** Line items within a salary structure (earnings/deductions).
- **Key Fields:** `company_id`, `salary_structure_id`, `component_name`, `component_code`, `component_type` (`earning` / `deduction`), `is_basic`, `calculation_type` (`fixed` / `percentage`), `value`, `percentage_base` (`gross` / `basic`), `is_taxable`, `prorated`, `display_order`, `status`.
- **Scope:** Structure. `component_code` unique per structure. Only one `is_basic` allowed per structure.

`employee_deductions`

- **Purpose:** Recurring or one-off deductions for an employee.
- **Key Fields:** `company_id`, `employee_id`, `type` (`loan` / `advance` / `fine` / `other`), `total_amount`, `remaining_balance`, `installment_amount`, `start_month`, `start_year`, `status` (`active` / `completed` / `cancelled`), `remarks`.
- **Scope:** Company + employee.

`employee_deduction_entries`

- **Purpose:** Record of each deduction installment applied to a payroll run.
- **Key Fields:** `employee_deduction_id`, `payroll_run_id`, `payslip_id`, `amount`, `created_at`.

- **Unique:** `employee_deduction_id` + `payroll_run_id`.

`employee_salary_payment_modes`

- **Purpose:** How an employee's net salary is split across payment channels.
- **Key Fields:** `company_id`, `employee_id`, `employee_salary_id`, `channel`, `allocation_type` (`fixed` / `percentage` / `residual`), `value`, `employee_bank_account_id`, `display_order`.
- **Unique:** `employee_salary_id` + `channel`.

`payroll_runs`

- **Purpose:** A payroll period execution.
- **Key Fields:** `company_id`, `month`, `year`, `run_type` (`regular` / `off_cycle`), `status`, `total_employees`, `total_amount`, `processed_at`, `approved_by`, `created_by`.
- **Unique:** `company_id` + `month` + `year` + `run_type`.

`attendance_snapshots`

- **Purpose:** Immutable copy of frozen monthly attendance used for payslip calculation.
- **Key Fields:** `company_id`, `employee_id`, `payroll_run_id`, `month`, `year`, `present_days`, `absent_days`, `leave_days`, `unpaid_leave_days`, `half_days`, `late_count`, `overtime_hours`, `working_hours`, `source_monthly_approval_id`, `snapshot_taken_at`.
- **Unique:** `payroll_run_id` + `employee_id`.
- **Immutability:** Model-level `updating` and `deleting` events throw `AttendanceSnapshotImmutableException`.

`payslips`

- **Purpose:** Employee payslip for a payroll run.
- **Key Fields:** `company_id`, `employee_id`, `payroll_run_id`, `attendance_snapshot_id`, `employee_salary_id`, `gross_earnings`, `total_deductions`, `net_pay`, `advance_paid`, `net_payable`, `advance_carry_forward`, `earnings_breakdown` (JSON), `deductions_breakdown` (JSON), `currency_code`, `status` (`draft` / `finalized` / `paid`), `needs_review`, `generated_at`.
- **Unique:** `payroll_run_id` + `employee_id`.

`payslip_generation_states`

- **Purpose:** Tracks async payslip generation progress.
- **Key Fields:** `company_id`, `payroll_run_id` (unique), `batch_id`, `total`, `processed`, `succeeded`, `failed`, `failures` (JSON), `status`, `started_at`, `finished_at`.

`payslip_payment_allocations`

- **Purpose:** Frozen payment channel split for a generated payslip.
- **Key Fields:** `company_id`, `payslip_id`, `employee_id`, `channel`, `amount`, `employee_bank_account_id`, `source` (`config` / `manual_override`).
- **Unique:** `payslip_id` + `channel`.

`disbursement_batches`

- **Purpose:** Batch of payments per channel for a payroll run.
- **Key Fields:** `company_id`, `payroll_run_id`, `batch_reference`, `payout_channel`, `total_amount`, `status` (`pending` / `sent` / `confirmed` / `failed`), `sent_at`, `failure_reason`.
- **Unique:** `company_id` + `payroll_run_id` + `payout_channel`.

`disbursement_batch_items`

- **Purpose:** Individual payment line within a batch.
- **Key Fields:** `disbursement_batch_id`, `payslip_id`, `payslip_payment_allocation_id`, `employee_id`, `employee_bank_account_id`, `amount`, `payment_reference`, `acknowledged_by`, `acknowledged_at`, `status`, `failure_reason`.
- **Unique:** `payslip_payment_allocation_id`.

`salary_advances`

- **Purpose:** Employee salary advance requests.
- **Key Fields:** `company_id`, `employee_id`, `month`, `year`, `employee_salary_id`, `request_method`, `requested_value`, `basis_gross`, `amount`, `status`, `reason`, `payment_channel`, `payment_reference`, `paid_at`, `paid_by`, `settled_payslip_id`, `settled_at`, `created_by`, `approved_by`.

---

7. Feature-by-Feature Business Logic

7.1 Payroll Settings

What happens: Reads or updates company-level payroll configuration.

Where: `PayrollSettingController`, `PayrollSettingService`, `PayrollSettingRepository`.

Actor: User with `payroll.settings-manage` (write) or `payroll.advance-manage` (read).

Validation (`UpdatePayrollSettingsRequest`):

- `overtime_multiplier`: numeric, min 0, max 5
- `overtime_rate_base`: `in:gross,basic`
- `standard_monthly_hours`: numeric, gt 0, max 744
- `hourly_rate_method`: `in:fixed_monthly_hours,working_days_x_shift_hours`
- `prorate_method`: `in:working_days,calendar_days`
- `round_net_pay_to`: `in:0.01,0.10,1.00,10.00`
- `advance_enabled`: boolean
- `advance_default_method`: `in:fixed,percentage`
- `advance_default_value`: numeric, gt 0
- `advance_max_percentage`: numeric, gt 0, max 100
- `advance_max_amount`: nullable, numeric, gt 0
- Cross-field: if `advance_default_method` = `percentage`, value must not exceed 100 or `advance_max_percentage`. If `fixed`, must not exceed `advance_max_amount`.

Business Rules:

- Updating settings affects future payslip generation only; regenerating a draft picks up new values.
- If `overtime_rate_base` is `basic`, a warning is returned: *"Ensure active salary structures define a basic salary component."*

Database: Upserts `payroll_settings` for the company.

Audit: Not explicitly audited in the inspected code for settings updates.

Response: `PayrollSettingResource` with `meta.warnings` and `meta.recalculation_note`.

7.2 Tax Slabs

What happens: CRUD for progressive tax slabs.

Where: `TaxSlabController`, `TaxSlabService`, `TaxSlabRepository`.

Actor: `payroll.settings-manage`.

Validation (`StoreTaxSlabRequest`):

- `name`: required, string, max 150
- `effective_year`: integer, current year ±5
- `slabs`: required array, min 1
- Each slab: `min_income` required numeric ≥0, `max_income` nullable numeric, `rate` required numeric 0–100
- Custom validator:
  - First slab `min_income` must be 0.
  - Only last slab may have null `max_income`.
  - `min_income` < `max_income` for closed bands.
  - Bands must be contiguous (current `max_income` == next `min_income`).
  - Exactly one open-ended top bracket required.

Business Rules:

- Only one active tax slab set per company per `effective_year`.
- Creating or updating a slab to `active` automatically deactivates all other slabs for that year via `deactivateOthersForYear`.

Database: `tax_slabs` table.

Events: None found.

7.3 Salary Structures

What happens: Manage reusable salary templates.

Where: `SalaryStructureController`, `SalaryStructureService`, `SalaryStructureRepository`.

Actor: `payroll.structure-manage`.

Validation (`StoreSalaryStructureRequest`):

- `name`: required, string, max 150
- `code`: required, string, max 50 (normalized to uppercase)
- `requires_basic`: sometimes, boolean

Business Rules:

- `code` must be unique per company (case-insensitive, stored uppercase).
- If `requires_basic` is true, the structure is created with `status = Inactive`.
- Activation (`activate`) requires the structure to be ready: either `requires_basic = false` or exactly one basic component exists (`isReadyToActivate`).
- Deactivation sets status to `Inactive`.
- Status cannot be changed directly via `update`; only through activate/deactivate endpoints.
- Updating `requires_basic` to true on an already-Active structure that lacks a basic component will auto-deactivate it.

Database: `salary_structures`.

7.4 Salary Structure Components

What happens: Manage line items (earnings/deductions) inside a salary structure.

Where: `SalaryStructureComponentController`, `SalaryStructureComponentService`.

Actor: `payroll.structure-manage`.

Validation (`StoreSalaryStructureComponentRequest` / `UpdateSalaryStructureComponentRequest`):

- `component_name`: required, string, max 100
- `component_code`: required, string, max 50
- `component_type`: `in:earning,deduction`
- `is_basic`: boolean
- `calculation_type`: `in:fixed,percentage`
- `value`: required, numeric
- `percentage_base`: required when `calculation_type = percentage`, `in:gross,basic`
- `is_taxable`: boolean
- `prorated`: boolean
- `display_order`: integer
- `status`: `in:Active,Inactive`

Business Rules (in Service `assertBusinessRules`):

- `component_code` must match `^[A-Z0-9_]+$` and be unique within the structure.
- `value` must be > 0.
- `is_basic` is allowed only when `component_type = earning`.
- Only one `is_basic` component per structure.
- If a component is referenced by a payslip (`structureHasPayslipReference`), its `component_code` is locked (throws `SalaryStructureComponentCodeLockedException` on edit).

Database: `salary_structure_components`.

Additional Endpoints:

- `reorder`: updates `display_order` for a list of component IDs.
- `preview`: calculates earnings/deductions breakdown for a given gross/basic salary using `SalaryComponentCalculator`.

7.5 Employee Deductions

What happens: Create and manage recurring deductions for employees.

Where: `EmployeeDeductionController`, `EmployeeDeductionService`, `EmployeeDeductionRepository`.

Actor: `payroll.deduction-manage`.

Validation (`StoreEmployeeDeductionRequest`):

- `employee_id`: required, exists in `employee_personal_infos`
- `type`: `in:loan,advance,fine,other`
- `total_amount`: required, numeric, min 0.01
- `installment_amount`: required, numeric, min 0.01, `lte:total_amount`
- `start_month`: 1–12
- `start_year`: 2020–2100

Business Rules:

- On creation, `remaining_balance` is set to `total_amount` and `status` to `Active`.
- Only `Active` deductions can be edited or cancelled.
- `installment_amount` cannot exceed `total_amount` on update.
- `applyInstallment` (called during payslip generation):
  - Skips if not active.
  - Idempotent per payroll run (unique constraint on `employee_deduction_id` + `payroll_run_id`).
  - If an entry already exists for this run, it reverses the previous balance before re-applying.
  - Applies `min(installment_amount, remaining_balance)`.
  - If `projectedNetPay - amountToApply < 0`, flags for review (appends to `remarks`) and applies 0.
  - Deducts from `remaining_balance`; if balance reaches 0, status becomes `Completed`.
- `reverseInstallmentsForPayslip`: restores balance and deletes entries when a payslip is regenerated.

Database: `employee_deductions`, `employee_deduction_entries`.

7.6 Employee Salary Payment Modes

What happens: Define how an employee's net salary is split across payment channels.

Where: `EmployeeSalaryPaymentModeController`, `EmployeeSalaryPaymentModeService`.

Actor: `payroll.payment-mode-manage`.

Validation (`UpdateEmployeeSalaryPaymentModesRequest`):

- Uses `EmployeeSalaryPaymentModesRule` which delegates to `EmployeeSalaryPaymentModesValidator`.

Business Rules (`EmployeeSalaryPaymentModesValidator`):

- At most one row per channel.
- Exactly one `residual` allocation is required.
- `residual` rows must have null/empty `value`.
- `fixed` / `percentage` rows require `value` > 0.
- Percentage value cannot exceed 100.
- `bank` / `mobile_banking` channels require a valid `employee_bank_account_id` belonging to the same employee.
- If fixed + percentage allocations exceed gross salary, a warning is returned (but not blocked).
- Payment modes can only be edited on the active salary revision.
- If the salary already has finalized/paid payslips, a warning is returned that existing payslips are unaffected.

Database: `employee_salary_payment_modes` (replaced entirely on update).

Audit: `activityLogService->log` event `payroll.payment_modes_replaced` with full payload.

7.7 Monthly Attendance Freeze

What happens: Freeze or unfreeze a monthly attendance approval record so it can be used for payroll.

Where: `MonthlyAttendanceFreezeController`, `MonthlyAttendanceFreezeService`.

Actor: User with `payroll.month-freeze` (checked via policy and service).

Business Rules:

- `freeze`:
  - Only an approved attendance month (`status = approved`) can be frozen.
  - Sets `frozen_at = now()`, `frozen_by = actorId`.
  - Dispatches `MonthFrozen` event.
  - Audits via `ActivityLogServiceContract` with action `month.freeze`.
- `unfreeze`:
  - Requires `unfreeze_reason` (required, string, max 255).
  - If a payroll run for the same month/year has status `paid`, requires additional permission `payroll.month-unfreeze-paid`.
  - Clears `frozen_at`, `frozen_by`, sets `unfreeze_reason`.
  - Audits via `ActivityLogServiceContract` with action `month.unfreeze`, noting `paid_month_unfreeze` and `elevated_permission_used`.

Database: `monthly_attendance_approvals` (Attendance module table).

7.8 Attendance Snapshots

What happens: Build immutable attendance snapshots for a payroll run.

Where: `AttendanceSnapshotController`, `AttendanceSnapshotService`, `AttendanceSnapshotRepository`.

Actor: `payroll.run-create` or `payroll.run-override-readiness`.

Business Rules:

- `buildForRun`:
  - Run must be in `draft` status (`PayrollRunNotDraftException` otherwise).
  - Resolves the employee set using the same readiness logic as run creation.
  - If any employee is not ready because attendance is not frozen, throws `AttendanceNotFrozenException` with employee list.
  - If any employee is missing a monthly attendance approval, throws `MonthlyApprovalMissingException` with employee list.
  - Creates one `attendance_snapshots` row per employee, copying fields from `MonthlyAttendanceApproval` (source columns mapped in `AttendanceSnapshot::SOURCE_COLUMNS`).
  - Unique constraint prevents duplicate snapshots for the same run/employee.
- `divergence`: compares a snapshot against the current live attendance approval. Returns field-by-field differences or a note if the approval no longer exists.

Database: `attendance_snapshots` (immutable; model throws on update/delete).

7.9 Payroll Runs

What happens: Create and manage a payroll execution for a period.

Where: `PayrollRunController`, `PayrollRunService`, `PayrollRunRepository`.

Actor: `payroll.run-create` (create/read), `payroll.run-approve` (approval summary).

Validation (`StorePayrollRunRequest`):

- `month`: 1–12
- `year`: current year ±5
- `run_type`: `regular` or `off_cycle`
- `override`: boolean (optional)

Business Rules:

- `readiness`: checks which employees are ready for payroll for the given month/year/run_type. Returns `ready` and `not_ready` arrays with reasons.
- `createRun`:
  - Duplicate run check: throws `DuplicatePayrollRunException` (409) if a run already exists for the same company/month/year/run_type.
  - If employees are not ready and `override = false`, throws `PayrollRunReadinessException` (422) with the list of not-ready employees.
  - If `override = true`, requires `payroll.run-override-readiness` permission; otherwise 403.
  - On creation, status is `draft`, `total_employees` is set to the count of ready employees, `total_amount` is 0, `created_by` is the user ID.
  - If the company has approval configured for `payroll.run-approve`, the run is created and an approval request may be raised later during the approval flow.
- `approvalSummary`: returns summary data for the approval workflow.
- `approveRun` (called by `PayrollRunExecutor`):
  - Only works if run status is not already `approved`, `paid`, or `locked`.
  - Requires payslip generation to be complete (`PayrollRunGenerationIncompleteException` otherwise).
  - Finalizes all payslips (`status = finalized`), settles paid advances, creates carry-forward deductions if needed, updates run status to `approved`, `processed_at = now()`, `approved_by = actorId`.
  - Dispatches `PayrollRunApproved` and `PayslipPublished` (per payslip).
  - If any advance carry-forward deductions are created, logs activity.
- `payRun` (called by `DisbursementService`):
  - Updates run status to `paid`.
  - Dispatches `PayrollRunPaid`.

Status Transitions: See Section 8.

Database: `payroll_runs`.

7.10 Payslip Generation

What happens: Generate payslips for all employees in a payroll run.

Where: `PayslipController`, `PayslipService`, `PayslipCalculatorService`, `PayslipFinaliser`, `PaymentAllocator`.

Actor: `payroll.run-create`.

Business Rules:

- `queueGeneration`:
  - Run status must be `draft` or `processing` (`PayslipRunNotGeneratableException` otherwise).
  - If generation is already `running`, throws `PayslipGenerationInProgressException`.
  - Resolves employee IDs from attendance snapshots + ready employees.
  - Asserts tax slabs exist if any employee is tax-applicable and not exempt (`TaxSlabRequiredException`).
  - Creates/updates `payslip_generation_states` with status `Queued`, then dispatches `GeneratePayslipChunkJob` chunks (size from config, default 100).
- `processChunk` (job handler):
  - Updates state to `Running`.
  - For each employee, calls `PayslipCalculatorService::generateForEmployee`.
  - Tracks success/failure counts and updates state to `Completed` or `Failed`.
- `generateForEmployee`:
  - Requires an attendance snapshot for the employee.
  - Requires an active `EmployeeSalary` as of the period end.
  - Requires the salary structure to be `Active`.
  - Calculates earnings/deductions using `SalaryComponentCalculator`.
  - Prorates based on `working_days` (or `calendar_days` depending on settings) and unpaid days.
  - Calculates overtime based on `overtime_multiplier`, `overtime_rate_base`, and `hourly_rate_method`.
  - Calculates tax from annual taxable income using active tax slabs (if applicable and not exempt).
  - Applies active employee deductions via `EmployeeDeductionService::applyInstallment`.
  - Rounds net pay per `round_net_pay_to` setting.
  - Saves draft payslip with `status = Draft`.
- `PayslipFinaliser::finalise`:
  - Sums paid advances for the period (`SalaryAdvanceStatus::Paid`).
  - Computes `net_payable = max(0, net_pay - advance_paid)`.
  - Computes `advance_carry_forward = max(0, advance_paid - net_pay)`.

- If carry-forward > 0, sets `needs_review = true`.
- Calls `PaymentAllocator::allocate`.

- `PaymentAllocator::allocate`:
  - Reads employee salary payment modes.
  - If no modes exist, creates a single fallback residual mode (cash, no bank account).
  - Splits `net_payable` across fixed, percentage, and residual channels.
  - Merges by channel, rounds to 2 decimals.
  - Asserts bank accounts exist for bank/mobile_banking channels with amount > 0.
  - Asserts sum of allocations equals `net_payable` (`PaymentAllocationInvariantException`).
  - Persists to `payslip_payment_allocations` (replacing existing) and dispatches `PaymentAllocationsFrozen`.

Database: `payslips`, `payslip_generation_states`, `payslip_payment_allocations`, `employee_deduction_entries`.

Events: `PaymentAllocationsFrozen`, `PayslipPublished` (on approval).

7.11 Salary Advances

What happens: Employees can request an advance against their upcoming salary.

Where: `SalaryAdvanceController`, `SalaryAdvanceService`, `SalaryAdvanceRepository`.

Actor: `payroll.advance-manage`.

Validation (`StoreSalaryAdvanceRequest`):

- `employee_id`: required, exists
- `month`: 1–12
- `year`: current year ±1
- `request_method`: `fixed` or `percentage`
- `requested_value`: numeric, gt 0. If percentage, max 100.
- Server-owned fields (`amount`, `basis_gross`, `employee_salary_id`, `status`, etc.) are `prohibited`.

Business Rules:

- Advances are only allowed if `payroll_settings.advance_enabled = true` for the company (`AdvanceConflictException` otherwise).
- The period must be open (no regular payroll run in `approved` / `paid` / `locked` status) (`AdvanceConflictException::periodClosed`).
- The employee must have an active salary as of the period end.
- The advance amount is resolved server-side:
  - `basis_gross` is frozen from the employee's active salary at request time.
  - `amount` = `requested_value` (fixed) or `basis_gross * requested_value / 100` (percentage).
- Ceiling enforcement:
  - `percentageCeiling = basis_gross * advance_max_percentage / 100`
  - `amountCeiling = advance_max_amount` (if set)
  - `takenSoFar` = sum of existing non-rejected/non-cancelled advances for the same employee/month/year.
  - Throws `AdvanceRuleViolationException` if the new advance would breach either ceiling.
- If `advance_requires_approval = true`, the advance is created with `status = pending_approval` and an approval request is raised via the platform approval gateway.
- If `advance_requires_approval = false`, it is immediately `approved` via `SalaryAdvanceService::approve()`.
- `approve`: sets `status = approved`, `approved_by = null` (executor sets it later? Actually executor doesn't set approved_by; the service `approve()` just updates status to Approved and dispatches `SalaryAdvanceApproved`).
- `reject`: sets `status = rejected`.
- `recordPayment`: sets `status = paid`, `payment_channel`, `payment_reference`, `paid_at`, `paid_by`. Requires reference for `cheque` / `bank`. Dispatches `SalaryAdvancePaid`.
- `cancel`: only allowed for `Draft`, `PendingApproval`, `Approved`. Sets `status = cancelled`. Cancels open approval request if the canceller is the original requester.
- `summary`: returns `AdvanceCeilingSnapshot` for an employee/month/year.

Database: `salary_advances`.

Events: `SalaryAdvanceApproved`, `SalaryAdvancePaid`, `SalaryAdvanceSettled`.

7.12 Disbursement Batches

What happens: After a payroll run is approved, create payment batches per channel and transmit/confirm them.

Where: `DisbursementController`, `DisbursementService`.

Actor: `payroll.disburse`.

Business Rules:

- `readiness`: returns payable count, total, excluded list, and settled-by-advance list for a run.
- `channelSummary`: returns total amount per `PaymentChannel` for the run.
- `disburse`:
  - Run must be `approved` (`DisbursementInvariantException` otherwise).
  - Classifies payslip allocations into payable vs excluded (e.g., cash/cheque channel or net payable ≤ 0).
  - Creates one `disbursement_batches` per channel with items linking to `payslip_payment_allocations`.
  - Unique constraint prevents duplicate batch per company/run/channel.
  - Auto-settles advances for payslips where `advance_paid > net_pay` (carry-forward) by creating an `employee_deductions` row with type `advance` and linking the advance `settled_payslip_id`.
  - Updates run status to `paid` if all items are confirmed.
- `send`:
  - Batch must be `pending`.
  - Register channels (cash/cheque) cannot be transmitted (`DisbursementConflictException::registerNotTransmitted`).
  - Calls `DisbursementTransmitterInterface::send` (default: `StubDisbursementTransmitter` which marks all successful).
  - Updates items to `sent` or `failed` with references. Batch status becomes `sent` or `failed`.
- `confirm`:
  - Batch must be `sent`.
  - Marks batch and all items as `confirmed`.
  - Marks linked payslips as `paid`.
  - Checks if all batches for the run are confirmed; if so, updates run to `paid` and dispatches `PayrollRunPaid`.

- `retry`:
  - Batch must be `failed` or `sent` (with failures).
  - Retries failed items via transmitter.
- `acknowledgeAll` / `acknowledgeItem`:
  - Only for register channels (cash/cheque).
  - Marks items as `confirmed` with optional `payment_reference`.
  - Closes payslip and potentially the run if all items confirmed.

Database: `disbursement_batches`, `disbursement_batch_items`.

Events: `PayrollRunPaid`, `PayslipPaid`.

---

8. Payroll Status Lifecycle

PayrollRunStatus

```
 DRAFT
  ↓ (generate payslips, build snapshots)
PROCESSING
  ↓ (generation complete)
PENDING_APPROVAL  ← approval gateway or direct
  ↓ (approval executor applies)
APPROVED
  ↓ (disbursement confirms all items)
PAID
  ↓ (final lock)
LOCKED
```

- `FAILED` can be reached from `PROCESSING`.
- No backward transitions are allowed (`canTransitionTo` enforces this).

PayslipStatus

```
 Draft
  ↓ (run approved)
Finalized
  ↓ (disbursement item confirmed)
Paid
```

SalaryAdvanceStatus

```
 Draft
  ↓ (submit / bypass approval)
PendingApproval
  ↓ (approve)          ↓ (reject)
Approved               Rejected
  ↓ (record payment)
Paid
  ↓ (payslip finalized nets it off)
Settled
```

- `Cancelled` can be reached from `Draft`, `PendingApproval`, or `Approved`.

DisbursementBatchStatus

```
 Pending
  ↓ (send)
Sent
  ↓ (confirm / retry)
Confirmed   ← or →   Failed (retryable)
```

PayslipGenerationStatus

```
 Queued
  ↓ (job starts)
Running
  ↓ (all chunks done)
Completed   ← or →   Failed
```

DeductionStatus

```
 Active
  ↓ (balance reaches 0)
Completed
  ↓ (manual cancel)
Cancelled
```

---

9. Payroll API Request Flow

Typical Flow (e.g., Create Payroll Run):

```
 Authenticated User
  ↓
POST /api/v1/payroll/payroll-runs
  ↓
auth:api middleware
  ↓
permission:payroll.run-create middleware
  ↓
StorePayrollRunRequest validation (month, year, run_type, override)
  ↓
PayrollRunController@store
  ↓
TenantContext::id() → company_id
```

```
  ↓
PermissionEngineContract::userCan('payroll.run-override-readiness') [if override]
  ↓
PayrollRunService::createRun
  ↓
PayrollRunRepository::exists (duplicate check)
  ↓
Readiness check (MonthlyAttendanceApprovalRepository)
  ↓
PayrollRunRepository::create
  ↓
ActivityLogServiceContract::log
  ↓
PayrollRunResource JSON response (201)
```

Typical Flow (e.g., Generate Payslips):

```
 Authenticated User
  ↓
POST /api/v1/payroll/payroll-runs/{id}/generate-payslips
  ↓
PayslipController@generate
  ↓
PayslipService::queueGeneration
  ↓
PayrollRunService::getRun
  ↓
PayslipGenerationStateRepository::findForRun
  ↓
Employee IDs resolved from snapshots + readiness
  ↓
Tax slab assertion
  ↓
PayslipGenerationStateRepository::upsert (Queued)
  ↓
Bus::batch of GeneratePayslipChunkJob
  ↓
202 Accepted response
  ↓
Queue Worker → PayslipService::processChunk
  ↓
PayslipCalculatorService::generateForEmployee (per employee)
  ↓
PayslipFinaliser::finalise
  ↓
PaymentAllocator::allocate
  ↓
State updated to Completed / Failed
```

10. Cross-Module Dependencies

Source Module	Trigger / Relationship	Payroll	Result
Attendance	`MonthlyAttendanceApproval` model	Freeze/unfreeze, snapshot source	Frozen attendance is prerequisite for payroll run creation.
Attendance	`AttendancePolicy` , `Shift` models	`WorkingDaysInMonthService`	Resolves working days and holidays for proration.
Employee	`EmployeePersonalInfo`	Deductions, advances, payslips	Employee is the subject of all payroll records.
Employee	`EmployeeSalary`	Payslip generation, advances, payment modes	Active salary drives gross/basic calculations.
Employee	`EmployeeBankAccount`	Payment modes, disbursement items	Required for bank/mobile banking channels.
Employee	`EmployeeTaxProfile`	`PayslipCalculatorService`	Determines tax exemption.
Platform	`ApprovalGatewayContract` , `ApprovalEngineContract`	Payroll runs, salary advances	Approval workflow integration.
Platform	`ActivityLogServiceContract`	Freeze, payment modes, run creation	Audit logging.
Platform	`PermissionEngineContract`	All controllers	Authorization checks.
Company/Tenant	`TenantContext` , `Company` model	All models	Company scoping.

11. Company/Tenant Data Isolation

- **Mechanism:** Every model uses `company_id` (via `BelongsToCompany` trait or explicit column).
- **Resolution:** `TenantContext::id()` provides the current company ID in controllers.
- **Query Scoping:** All repositories filter by `company_id` :
  - `PayrollRunRepository::paginateRuns` → `where('company_id', $companyId)`
  - `PayslipRepository::findById` → `where('company_id', $companyId)`
  - `SalaryAdvanceRepository::listByCompany` → `forCompany($companyId)`
- **Policy Scoping:** `MonthlyAttendanceApprovalPolicy::freeze/unfreeze` checks permission against `$approval->company_id` .
- **No Global Scope:** The code does not appear to use a global `BelongsToCompany` query scope automatically; each repository/controller explicitly passes/scopes `company_id` .

12. Events / Listeners / Jobs

Events (all found in `app/Events/`)

Event	Triggered By	Listener Found?	Purpose (from code comments)
`MonthFrozen`	`MonthlyAttendanceFreezeService::freeze()`	**No**	Marks attendance as frozen.
`PaymentAllocationsFrozen`	`PaymentAllocator::persistAndFreeze()`	**No**	Marks payslip allocations as frozen.
`PayrollRunApproved`	`PayrollRunService::approveRun()`	**No**	Run approved and payslips finalized.
`PayrollRunPaid`	`DisbursementService::maybeCloseRun()`	**No**	All disbursements confirmed.
`PayslipPublished`	`PayrollRunService::finalizePayslips()`	**No**	Payslip moved draft → finalized.
`SalaryAdvanceApproved`	`SalaryAdvanceService::approve()`	**No**	Advance cleared for handover.
`SalaryAdvancePaid`	`SalaryAdvanceService::recordPayment()`	**No**	Money handed over recorded.
`SalaryAdvanceSettled`	`PayrollRunService::settlePaidAdvances()`	**No**	Advance closed against payslip.

Jobs

Job	Trigger	Purpose
`GeneratePayslipChunkJob`	`PayslipService::queueGeneration()` dispatches via Laravel Bus batch.	Processes a chunk of employee IDs to generate payslips asynchronously.

Note: No event listeners were found in the inspected Payroll module codebase. Event classes contain comments such as *"No listeners yet — see spec §10"*.

---

13. Audit Logging

The module reuses the project's existing `ActivityLogServiceContract` (platform-level).

Actions Logged

Action Key	Trigger	Actor	Data Logged
`month.freeze`	`MonthlyAttendanceFreezeService::freeze()`	`actorId`	`frozen_at`, `frozen_by`, month, year, employee_id
`month.unfreeze`	`MonthlyAttendanceFreezeService::unfreeze()`	`actorId`	`reason`, `paid_month_unfreeze`, `elevated_permission_used`
`payroll.payment_modes_replaced`	`EmployeeSalaryPaymentModeService::replace()`	`actorUserId`	`employee_id`, `employee_salary_id`, `mode_count`, `modes` array
`payroll.health_viewed`	`PayrollService::getHealthStatus()`	Current user	`moduleId`, `actionKey`, `subjectType`
`payroll.run_created`	`PayrollRunService::createRun()`	`userId`	`payroll_run_id`, month, year, run_type, total_employees
`payroll.advance_carry_forward_created`	`PayrollRunService::approveRun()`	System	`employee_id`, `amount`, `payroll_run_id`

Source: `MonthlyAttendanceFreezeService.php`, `EmployeeSalaryPaymentModeService.php`, `PayrollService.php`, `PayrollRunService.php`

---

14. Error Handling

Condition	Exception / Behavior	HTTP Status	Response Structure
Auth failure	`auth:api` middleware	401	Laravel default
Missing permission	`PermissionDeniedException` / `AccessDeniedHttpException`	403	`{"success":false,"message":"..."}`
Route/model not found	`NotFoundHttpException` / Eloquent `ModelNotFoundException`	404	Laravel default or controller catch
Duplicate payroll run	`DuplicatePayrollRunException`	409	`{"success":false,"message":"A payroll run already exists for this period."}`
Period closed for advance	`AdvanceConflictException::periodClosed()`	409	`{"success":false,"message":"..."}`
Freeze non-approved attendance	`HttpException`	409	`{"success":false,"message":"Only an approved attendance month can be frozen."}`
Disbursement invariant	`DisbursementInvariantException`	409	`{"success":false,"message":"..."}`
Advance ceiling breach	`AdvanceRuleViolationException`	422	`{"success":false,"message":"...", "meta": {...}}`
Payslip generation in progress	`PayslipGenerationInProgressException`	422	`{"success":false,"message":"..."}`
Run not draft	`PayrollRunNotDraftException`	422	`{"success":false,"message":"..."}`
Generation incomplete	`PayrollRunGenerationIncompleteException`	422	`{"success":false,"message":"..."}`
Validation failure	`ValidationException`	422	Laravel validation error bag
Missing tax slab	`TaxSlabRequiredException`	422	`{"success":false,"message":"..."}`
Generic failure	`Throwable` catch-all	500	`{"success":false,"message":"Failed to ..."}`

Source: Various controllers and exception classes in `app/Exceptions/`.

---

15. Actor Responsibility Matrix

Actor	Feature	Action	Permission	Actual Restriction	Source
Authenticated User	Health check	View	None (just auth)	Must be authenticated	`PayrollController@health`
Settings Manager	Payroll settings	Read/Write	`payroll.settings-manage`	Read also allows `payroll.advance-manage`	`api.php`
Settings Manager	Tax Slabs	CRUD	`payroll.settings-manage`	Company-scoped	`api.php`
Structure Manager	Salary Structures	CRUD	`payroll.structure-manage`	Company-scoped; code uniqueness	`api.php`
Structure Manager	Salary Components	CRUD/Reorder/Preview	`payroll.structure-manage`	Structure-scoped	`api.php`
Deduction Manager	Employee Deductions	CRUD/Cancel	`payroll.deduction-manage`	Company-scoped	`api.php`
Payment Mode Manager	Payment Modes	List/Replace	`payroll.payment-mode-manage`	Active salary revision only	`api.php`
Freeze Manager	Monthly Freeze	Freeze/Unfreeze	`payroll.month-freeze`	Unfreeze paid month needs `payroll.month-unfreeze-paid`	`api.php`, `MonthlyAttendanceFreezeService`
Run Creator	Payroll Runs	Create/List/Show	`payroll.run-create`	Readiness override needs `payroll.run-override-readiness`	`api.php`, `PayrollRunController`
Run Approver	Payroll Runs	Approval summary	`payroll.run-approve`	Via approval gateway	`api.php`
Disburser	Disbursement	All batch ops	`payroll.disburse`	Run must be approved	`api.php`, `DisbursementService`
Advance Manager	Salary Advances	CRUD/Payment	`payroll.advance-manage`	Must have `advance_enabled` setting	`api.php`, `SalaryAdvanceService`
Employee (Self)	Payslips	View own	`payroll.payslip-view-own`	Own employee_id only	`api.php`, `PayslipController`
Payroll Viewer	Payslips	View all	`payroll.payslip-view-all`	Any employee	`api.php`, `PayslipController`

---

16. API Business Logic Matrix

API	Controller	Validation	Authorization	Business Rule	DB Effect	Event/Job	Respon
GET /settings	`PayrollSettingController@show`	—	`payroll.settings-manage|advance-manage`	—	Read `payroll_settings`	—	`Payrol`

API	Controller	Validation	Authorization	Business Rule	DB Effect	Event/Job	Respon
PUT /settings	`PayrollSettingController@update`	`UpdatePayrollSettingsRequest`	`payroll.settings-manage`	Cross-field advance validation	Update `payroll_settings`	—	JSON w
GET /tax-slabs	`TaxSlabController@index`	—	`payroll.settings-manage`	—	Read `tax_slabs`	—	`TaxSla`
POST /tax-slabs	`TaxSlabController@store`	`StoreTaxSlabRequest`	`payroll.settings-manage`	Single active per year	Create `tax_slabs`	—	`TaxSla`
POST /salary-structures	`SalaryStructureController@store`	`StoreSalaryStructureRequest`	`payroll.structure-manage`	Code unique per company	Create `salary_structures`	—	`Salary`
POST /salary-structures/{id}/components	`SalaryStructureComponentController@store`	`StoreSalaryStructureComponentRequest`	`payroll.structure-manage`	One basic max, code regex	Create `salary_structure_components`	—	JSON w
POST /employee-deductions	`EmployeeDeductionController@store`	`StoreEmployeeDeductionRequest`	`payroll.deduction-manage`	Installment ≤ total	Create `employee_deductions`	—	`Employ`
PUT /employee-salaries/{id}/payment-modes	`EmployeeSalaryPaymentModeController@replace`	`UpdateEmployeeSalaryPaymentModesRequest`	`payroll.payment-mode-manage`	One residual, bank accounts valid	Replace `employee_salary_payment_modes`	Audit log	JSON w
POST /monthly-attendance/{id}/freeze	`MonthlyAttendanceFreezeController@freeze`	—	`payroll.month-freeze`	Status must be approved	Update `monthly_attendance_approvals`	`MonthFrozen`	`Monthly`
POST /monthly-attendance/{id}/unfreeze	`MonthlyAttendanceFreezeController@unfreeze`	`UnfreezeMonthlyAttendanceRequest`	`payroll.month-freeze`	—	Update `monthly_attendance_approvals`	—	`Monthly`
POST /payroll-runs	`PayrollRunController@store`	`StorePayrollRunRequest`	`payroll.run-create`	No duplicate; readiness or override	Create `payroll_runs`	Audit log	`Payrol`
POST /payroll-runs/{id}/generate-payslips	`PayslipController@generate`	`GeneratePayslipsRequest`	`payroll.run-create`	Run draft/processing; tax slab check	Upsert `payslip_generation_states`	`GeneratePayslipChunkJob`	JSON (2
POST /payroll-runs/{id}/disburse	`DisbursementController@disburse`	—	`payroll.disburse`	Run approved	Create `disbursement_batches` + items	`PayrollRunPaid` (conditional)	JSON (2
POST /disbursement-batches/{id}/send	`DisbursementController@send`	—	`payroll.disburse`	Batch pending; not register	Update items/batch status	—	`Disburs`
POST /disbursement-batches/{id}/confirm	`DisbursementController@confirm`	—	`payroll.disburse`	Batch sent	Update batch/items/payslip status	`PayrollRunPaid` (conditional)	`Disburs`
GET /payslips/me	`PayslipController@me`	`ListMyPayslipsRequest`	`payroll.payslip-view-own`	Own employee only	Read `payslips`	—	Paginate
POST /salary-advances	`SalaryAdvanceController@store`	`StoreSalaryAdvanceRequest`	`payroll.advance-manage`	Enabled, period open, ceilings	Create `salary_advances`	Approval request (conditional)	`Salary`
POST /salary-advances/{id}/record-payment	`SalaryAdvanceController@recordPayment`	`RecordAdvancePaymentRequest`	`payroll.advance-manage`	Status approved; channel ref rules	Update `salary_advances`	`SalaryAdvancePaid`	`Salary`

17. File-to-Business-Logic Mapping

File	Role
`routes/api.php`	Defines all Payroll API entry points and permission middleware.
`PayrollController.php`	Health check endpoint.
`PayrollSettingController.php`	Reads/writes company payroll settings.
`TaxSlabController.php`	CRUD and calculation for tax slabs.
`SalaryStructureController.php`	CRUD and status toggles for salary structures.
`SalaryStructureComponentController.php`	Manages components, reordering, and preview calculations.
`EmployeeDeductionController.php`	CRUD for employee deductions and viewing entries.
`EmployeeSalaryPaymentModeController.php`	Lists and replaces employee payment mode splits.
`MonthlyAttendanceFreezeController.php`	Freeze/unfreeze monthly attendance with policy checks.
`AttendanceSnapshotController.php`	Builds, lists, and divergences immutable attendance snapshots.
`PayrollRunController.php`	Creates runs, checks readiness, returns approval summaries.
`PayslipController.php`	Queues generation, lists, downloads, regenerates payslips.
`SalaryAdvanceController.php`	Full advance lifecycle (create, approve, pay, cancel).
`DisbursementController.php`	Manages disbursement batches and item acknowledgements.
`PayrollRunService.php`	Core run logic: creation, approval, finalization, advance settlement.
`PayslipService.php`	Queues and processes payslip generation chunks.
`PayslipCalculatorService.php`	Computes earnings, deductions, tax, overtime per employee.
`PayslipFinaliser.php`	Nets off advances and triggers payment allocation.
`PaymentAllocator.php`	Splits net payable across channels and persists allocations.
`SalaryAdvanceService.php`	Ceiling checks, approval workflow, payment recording.
`DisbursementService.php`	Batch creation, transmission, confirmation, retry.
`AttendanceSnapshotService.php`	Builds immutable snapshots from frozen attendance.
`MonthlyAttendanceFreezeService.php`	Business rules for freeze/unfreeze + audit logging.
`EmployeeDeductionService.php`	Installment application with idempotency and negative-net guard.
`PayrollSettingService.php`	Settings retrieval and update with warnings.
`TaxSlabService.php`	Slab CRUD and tax calculation with single-active enforcement.
`SalaryStructureService.php`	Structure CRUD with code uniqueness and readiness checks.
`SalaryStructureComponentService.php`	Component CRUD with business rule assertions.
`EmployeeSalaryPaymentModeService.php`	Payment mode replacement with validation and audit.
`WorkingDaysInMonthService.php`	Resolves working days from shift/holiday calendar.
`PayrollRunExecutor.php`	Approval executor: finalizes payslips on run approval.
`SalaryAdvanceExecutor.php`	Approval executor: transitions advance to approved/rejected.

File	Role
`GeneratePayslipChunkJob.php`	Queue job that processes a chunk of employee IDs.
`MonthlyAttendanceApprovalPolicy.php`	Gates freeze/unfreeze on `payroll.month-freeze`.

18. End-to-End Business Flow Diagrams

Payroll Run Creation & Payslip Generation Flow

```
flowchart TD
    A[User: payroll.run-create] -->|POST /api/v1/payroll/payroll-runs| B[StorePayrollRunRequest]
    B --> C{Ready?}
    C -->|No + No override| D[422 Not Ready]
    C -->|Yes / Override| E[PayrollRunService::createRun]
    E --> F[Create payroll_runs row<br/>status: draft]
    F --> G[User: POST /generate-payslips]
    G --> H[PayslipService::queueGeneration]
    H --> I{Run status draft/processing?}
    I -->|No| J[422 Not Generatable]
    I -->|Yes| K[Dispatch GeneratePayslipChunkJob]
    K --> L[Queue Worker]
    L --> M[PayslipCalculatorService::generateForEmployee]
    M --> N[Create/Update payslips<br/>status: draft]
    N --> O[PayslipFinaliser::finalise]
    O --> P[PaymentAllocator::allocate]
    P --> Q[payslip_payment_allocations]
    Q --> R[Generation State: Completed]
```

Payroll Approval & Disbursement Flow

```
flowchart TD
    A[Run: approved status] -->|POST /disburse| B[DisbursementService::disburse]
    B --> C[Create disbursement_batches<br/>per channel]
    C --> D[Create disbursement_batch_items]
    D --> E[Settle advances<br/>create deductions if carry-forward]
    E --> F[Run status: paid?]
    F -->|All confirmed| G[PayrollRunPaid event]
    G --> H[Run status: paid]
    C --> I[POST /send]
    I --> J[StubDisbursementTransmitter]
    J --> K[Items: sent]
    K --> L[POST /confirm]
    L --> M[Items: confirmed<br/>Payslips: paid]
```

Salary Advance Lifecycle Flow

```
flowchart TD
    A[User: payroll.advance-manage] -->|POST /salary-advances| B[StoreSalaryAdvanceRequest]
    B --> C[SalaryAdvanceService::create]
    C --> D{Advance enabled?}
    D -->|No| E[409 Conflict]
    D -->|Yes| F{Period open?}
    F -->|No| G[409 Period Closed]
    F -->|Yes| H{Within ceilings?}
    H -->|No| I[422 Ceiling Breach]
    H -->|Yes| J[Create salary_advances<br/>status: pending_approval or approved]
    J --> K[Approval Gateway<br/>if required]
    K --> L[SalaryAdvanceExecutor::apply]
    L --> M[Status: approved]
    M --> N[POST /record-payment]
    N --> O[Status: paid]
    O --> P[Payslip generation nets off]
    P --> Q[Status: settled]
```

19. Undocumented / Unclear Behavior

- **Event Listeners:** All 8 events in the Payroll module explicitly contain comments stating *"No listeners yet"*. No listeners were found in the inspected codebase. It is unclear what downstream behavior (notifications, accounting entries) is intended.
- **Audit Log Storage Format:** The module calls `ActivityLogServiceContract::log`, but the exact schema and retention of the activity log table were not inspected in this module.
- **Disbursement Transmitter:** The default bound implementation is `StubDisbursementTransmitter`, which always returns success. The real banking integration behavior is not defined in this module.
- **Approval Gateway Configuration:** The approval gateway (`ApprovalGatewayContract`) determines whether approval is required for payroll runs and advances. The exact configuration UI or seeder for these rules lives outside the Payroll module and was not inspected.
- **Currency Handling:** `currency_code` is stored on `payslips`, but no currency conversion logic was found. It appears to be a static field.
- **Overtime Calculation Details:** `hourlyRate` uses either `standard_monthly_hours` or `workingDaysInMonth * shiftHours`. The `shiftHours` value (default 8.0) is hardcoded in the service signature; no per-employee shift hour override was found.
- **Tax Profile Source:** `EmployeeTaxProfileRepositoryInterface` is used to check `tax_exemption`, but its model and fields were not inspected as part of this module.
- **Payroll Run "Failed" Status:** The `FAILED` status exists in the enum and is a valid transition from `PROCESSING`, but no code path in the inspected services actually sets a run to `failed`. It may be reserved for future use.
- **Employee Salary Payment Mode "Manual Override" Source:** The `PayslipAllocationSource` enum includes `manual_override`, but no API or service logic was found that creates allocations with this source.

20. Verified Technical Observations

- **Immutability Enforcement:** `AttendanceSnapshot` model uses `static::updating` and `static::deleting` booted events to throw `AttendanceSnapshotImmutableException`. This is a code-level guarantee, not just a convention.
- **Idempotent Deductions:** `EmployeeDeductionService::applyInstallment` uses a unique constraint on `employee_deduction_id` + `payroll_run_id` to prevent duplicate entries. It reverses prior entries before re-applying.
- **Money Precision:** The module consistently uses `Money::round2()` (PHP `round($amount, 2)`) and `Money::roundTo($amount, $increment)` for net pay rounding to avoid floating-point drift.
- **Tenant Leak Prevention:** `SalaryAdvanceController::requireAdvance` checks `assertAdvancesEnabled($companyId)` *before* looking up the record, ensuring a disabled company always receives 409 rather than 404.
- **Company Context Strictness:** `TaxSlabController` throws `PermissionDeniedException('company.context')` if `TenantContext::id()` is missing.

- **Approval Executor Rejection Safety:** `PayrollRunExecutor::reject` does **not** modify the payroll run or payslips; it only validates the rejection reason length. This ensures rejection is a safe no-op.
- **Chunked Generation:** Payslip generation is explicitly designed for queues using `config('payroll.payslip_generation_chunk_size')` (default 100) via `GeneratePayslipChunkJob`.

---

21. Source References

All business rules documented above are derived from the following source files within `backend/Modules/Payroll`:

- `routes/api.php` — API route definitions and permission middleware.
- `app/Http/Controllers/*` — HTTP layer and request delegation.
- `app/Http/Requests/*` — Validation rules.
- `app/Services/PayrollRunService.php` — Run lifecycle, approval, finalization.
- `app/Services/PayslipService.php` — Generation queue and chunk processing.
- `app/Services/PayslipCalculatorService.php` — Earnings/deductions/tax/ot calculation.
- `app/Services/PayslipFinaliser.php` — Advance netting and allocation trigger.
- `app/Services/PaymentAllocator.php` — Channel split logic.
- `app/Services/SalaryAdvanceService.php` — Advance ceiling and state machine.
- `app/Services/DisbursementService.php` — Batch creation and confirmation.
- `app/Services/AttendanceSnapshotService.php` — Snapshot build and divergence.
- `app/Services/MonthlyAttendanceFreezeService.php` — Freeze/unfreeze with audit.
- `app/Services/EmployeeDeductionService.php` — Installment application.
- `app/Models/*` — Eloquent models, casts, relationships, and immutability guards.
- `app/Enums/*` — Status enums and transition rules.
- `app/Approval/PayrollRunExecutor.php` — Approval-side run finalization.
- `app/Approval/SalaryAdvanceExecutor.php` — Approval-side advance transition.
- `app/Jobs/GeneratePayslipChunkJob.php` — Async payslip generation.
- `app/Policies/MonthlyAttendanceApprovalPolicy.php` — Freeze authorization.
- `app/Support/Money.php`, `AdvanceCeilingCalculator.php`, `SalaryComponentCalculator.php`, `EmployeeSalaryPaymentModesValidator.php` — Business rule arithmetic.
- `database/migrations/*` — Schema definitions and constraints.
- `tests/Feature/PayrollRunControllerTest.php` — Confirmed behavior for readiness override and duplicate runs.
- `tests/Unit/PayrollRunStatusTest.php` — Confirmed status transition matrix.

Summary

Confirmed Payroll Business Flows

1. **Settings & Configuration:** Company-level overtime, proration, rounding, and advance policy.
2. **Tax Slabs:** Progressive tax band management with single-active enforcement per year.
3. **Salary Structures:** Template management with component-level earnings/deductions and activation readiness.
4. **Employee Deductions:** Installment-based deduction tracking with idempotent payroll application.
5. **Payment Modes:** Channel splitting (cash, cheque, bank, mobile banking) with residual allocation.
6. **Monthly Freeze:** Locking attendance approvals before payroll processing.
7. **Attendance Snapshots:** Immutable copies of frozen attendance for historical payslip reproducibility.
8. **Payroll Runs:** Period-based execution with readiness checks, duplicate prevention, and optional override.
9. **Payslip Generation:** Chunked async generation from snapshots, salary structures, tax slabs, and deductions.
10. **Salary Advances:** Ceiling-guarded part-payments of salary with approval workflow integration.
11. **Disbursement:** Channel-based batching, stub transmission, confirmation, and payroll run closure.

Confirmed Actors

- Authenticated API user (permission-based)
- Employee (as data subject)
- Queue worker (payslip generation)
- Approval executor (platform-integrated)

Confirmed Permissions

`payroll.settings-manage`, `payroll.advance-manage`, `payroll.structure-manage`, `payroll.deduction-manage`, `payroll.payment-mode-manage`, `payroll.month-freeze`, `payroll.month-unfreeze-paid`, `payroll.run-create`, `payroll.run-override-readiness`, `payroll.run-approve`, `payroll.payslip-view-own`, `payroll.payslip-view-all`, `payroll.disburse`.

Confirmed API Endpoints

~45 endpoints covering settings, tax, structures, deductions, payment modes, freeze, snapshots, runs, payslips, advances, disbursements.

Confirmed Module Dependencies

Attendance (freeze/approvals/shifts), Employee (salary/personal info/bank accounts/tax profiles), Platform (approval/audit/permissions).

Confirmed Database/State Changes

Immutable snapshots, draft→finalized→paid payslips, active→completed deductions, pending→sent→confirmed batches, draft→approved→paid→locked runs.

Confirmed Events/Jobs/Auditing

8 events dispatched (no listeners found in module), 1 queue job (`GeneratePayslipChunkJob`), audit logging via platform `ActivityLogServiceContract` for freeze, payment modes, and run creation.

Unclear or Undocumented Areas

- No downstream listeners for any Payroll event.
- Real disbursement transmitter implementation is a stub.
- Approval gateway configuration is external to this module.
- No explicit code path sets a payroll run to `failed`.

- `manual_override` allocation source is unused in inspected code.