Dashboard & Job Enquiry Documentation

Overview

- **Purpose**: capture how the Peter the Possum & Bird Man dashboard and job inquiry (aka new enquiry) experiences are wired together so implementers can safely extend the flows.
- **Audience**: front-end devs and low-code builders using the VitalStats SDK bundle included under `ptpm/js`.
- **Scope**: web assets living in `ptpm/index.html` (dashboard) and `ptpm/pages/new-enquiry.html` (job enquiry) plus their related MVC-style controllers, models, and views.

Shared bootstrap & dependencies

- `ptpm/js/app.js` bootstraps both screens once the DOM loads. It instantiates `VitalStatsSDK`, switches to the Peterpm plugin models (deal/job/property/contact/etc), and conditionally spins up controllers based on the `data-page` attribute on `<body>`.
- Shared front-end stack: Tailwind via CDN, vanilla JS modules, Day.js (with timezone + UTC plugins for the dashboard), Flatpickr for date pickers, and Google Places (lazy-loaded) for property search on the enquiry form.
- All controllers follow a thin MVC pattern: the model encapsulates SDK queries/mutations, the view performs DOM writes/listeners, and controllers glue them together.

Dashboard module

Entry points & layout

- Page: `ptpm/index.html` (body has `data-page="dashboard"`).
- Initialization happens in `app.js` → `App.maybeInitDashboard()`, which only runs once Day.js and calendar/table containers exist.

Model (`js/models/dashboard.js`)

- Wraps `dayjs` to build a rolling 14-day calendar (`#buildCalendarDays`) aligned to `Australia/Brisbane`.
- Persists filter state, selected date, status colour classes (`DASHBOARD_STATUS_CLASSES`), and cached query handles for each tab (deals, quotes, jobs, payments, active jobs, urgent calls).
- Each `fetch*` method builds SDK queries with `.andWhere` filters derived from `collectAllFiltersFromUI()` (dates become epoch ranges, text fields converted to SQL `like`).
- `initScheduledTotals()` precomputes the per-day counts shown on the calendar heatmap; `getRowsForDate()` scopes the day's table rows.
- External data sources: `ptpmDealModel`, `ptpmJobModel`, payment-related models exposed via `plugin.switchTo`.

Controller (`js/controller/dashboard.js`)

- Owns dropdown wiring (`initServiceProviderDropdown`, `initAccountTypeDropdown`, `initSourceDropdown`, `initStatusDropdown`), calendar click handling, global search, and filter chips.
- Maintains `currentTab` state (default `inquiry`) plus `filters` object with every UI filter (global text, account, resident, source, service provider, price min/max, date ranges, etc.).
- `init()` queues Day.js totals, renders calendar, wires notification icon, search bar, filter apply/reset buttons, and tab navigation via `view.initTopTabs`.
- `handleTabChange()` delegates to per-tab fetchers:
  - `fetchDealsAndRenderTable()` for Inquiry list.
  - `fetchQuotesAndRenderTable()` for Quote tab.
  - `fetchPaymentsAndRenderTable()` for Payment tab.
  - `fetchJobsAndRenderTable()` for Jobs tab.
  - `fetchActiveJobsAndRenderTable()` for Active Jobs tab.
  - `fetchUrgentCallsAndRenderTable()` for Urgent Calls tab.
- `renderTable()` selects the correct view renderer per tab (`renderQuoteTable`, `renderPaymentTable`, etc.) and passes status colour mappings + date formatter.
- Error handling: every fetch is wrapped in `try/catch`, writes to console, and clears table state on failures.

View (`js/views/dashboard.js`)

- Provides `renderDynamicTable()` helper plus tab-specific renderers (Inquiry/Quote/Payment/Jobs/Active Jobs/Urgent Calls) that describe headers, row formatters, and inline action buttons.
- `renderCalendar()` draws the 14-day grid with totals + selection styling.
- `initTopTabs()` attaches click listeners to `data-tab` buttons and keeps ARIA attributes in sync.
- Extra UX helpers: toast notifications, notification modals, inline loading skeletons, filter chip renderer.

Tab reference

Tab	Controller fetcher	Primary data source	Key
Inquiry	`fetchDealsAndRenderTable`	`ptpmDealModel` deal query filtered by `inquiry_status`	Creat accou resid prope sourc assig provi status actio
Quote	`fetchQuotesAndRenderTable`	`dashboardHelper.fetchQuotes` (or fallback) mapped via `DashboardHelper.mapQuoteRows`	Quot refere amou status conta

Tab	Controller fetcher	Primary data source	Key
			menu
Jobs	`fetchJobsAndRenderTable`	`DashboardHelper.mapJobsRows` from job query	Job # name appo wind crew an, st
Active Jobs	`fetchActiveJobsAndRenderTable`	`dashboardHelper.mapActiveJobRows`	Appo servi job/ir refere progr
Payment	`fetchPaymentsAndRenderTable`	`DashboardHelper.mapPaymentsRows` from invoice data	Invoi numb accou outst total, dates sync
Urgent Calls	`fetchUrgentCallsAndRenderTable`	`dashboardHelper.mapUrgentCallRows`	Task date, flags instru colun

Filter reference (UI ids → filter keys)

UI element	Filter key	Notes
`#global-search`	`global`	Text search hits first/last name, email, sms number
`#filter-account-name`	`accountName`	Company name wildcard
`#filter-resident`	`resident`	Matches contact first/last name
`#filter-address`	`address`	Property `address_1` wildcard
Source dropdown (`data-source`)	`source`	Multi-select list with

UI element	Filter key	Notes
		"None" toggle synced to controller
Status dropdown (`data-status`)	`statuses`	Tab aware (Inquiry statuses, Quote statuses, etc.)
`#filter-serviceman`	`serviceman`	Technician/staff filter for job-based tabs
Account type dropdown (`data-account-type`)	`accountTypes`	Multi-select (Res/Com) used for company filters
Service provider dropdown (`data-service-provider`)	`serviceProviders`	Multi-select toggles + "All" button
`#price-min` / `#price-max`	`priceMin`, `priceMax`	Numeric range applied where available
`#date-from` / `#date-to`	`dateFrom`, `dateTo`	Applied to creation date across tabs
Payment-only fields	`invoiceDateFrom`, `invoiceDateTo`, `dueDateFrom`, `dueDateTo`, `billPaidDateFrom`, etc.	Only read when those inputs exist in the DOM
Task checkboxes `#task-due-today`, `#task-assigned-to-me`	`taskDueToday`, `taskAssignedToMe`	Used for Urgent Calls / tasks tab

Extending the dashboard

1. **Add a new tab**: place a button with `data-tab` inside the nav + a matching `data-panel` section in HTML. Extend `handleTabChange` with a new case that fetches + maps data, then add a renderer inside `DashboardView` (or reuse `renderDynamicTable`).
2. **Add filters**: give the input a unique id or `data-*` attribute, extend `collectAllFiltersFromUI()` to read it, then use the value while building queries in `DashboardModel`.
3. **New status pills**: extend `inquiryStatues`/`quoteStatuses`/etc arrays in the controller and update `DASHBOARD_STATUS_CLASSES` so new statuses have colour tokens.
4. **SDK queries**: the model always reuses `plugin.switchTo` handles. Keep query building pure (no DOM) and prefer `.deSelectAll().select([...])` to minimize payload size.

Operational considerations

- Day.js timezone plugins must be present globally; otherwise the constructor throws an error early

(helps catch missing CDN scripts).
- `App.maybeInitDashboard()` short-circuits when required DOM nodes are missing so the JS bundle can be shared across multiple pages.
- When filters change, `bindApplyFilters()` writes chips via `renderAppliedFilters()`; keep chip labels short to avoid overflow on small screens.

Job / New Enquiry module

Entry points & layout

- Page: `ptpm/pages/new-enquiry.html` with `data-page="new-enquiry"`. Contains three major sections: **Contact selection**, **Property information**, and **Inquiry/Feedback forms** plus supporting modals for contact/entity details.
- `App.initNewEnquiry()` (in `app.js`) injects `NewEnquiryModel`, `NewEnquiryView`, and `NewEnquiryController`, and binds the controller's `initAutocomplete` as the Google callback.

Model (`js/models/new-enquiry.js`)

- Provides handles to VitalStats models: contacts, affiliations, properties, deals, companies, jobs, etc. All retrieved via `plugin.switchTo("Peterpm*")`.
- Maintains a local cache of contacts (`contacts`) plus a map of `relatedCache` keyed by email to avoid duplicate property/job lookups.
- Core responsibilities:
  - `loadContacts()` primes the contact list using `.query().fetchAllRecords()` with `maxRecords` limit and stores normalized entries for the view.
  - CRUD helpers for contacts, affiliations, companies, properties, and deals (new inquiry). Mutations all go through `model.mutation().createOne()` or `.update()` followed by `.execute(true).toPromise()`.
  - `fetchRelated(email)` hydrates existing properties/jobs/inquiries for the selected contact/entity (used to render the side panel).
  - Google Places helpers store API key/session tokens when needed (current UI loads Maps via `<script>` and the controller handles parsing).

Controller (`js/controller/new-enquiry.js`)

- Seeds dropdown definitions for noises, pest locations, time-of-day observations, state options, inquiry metadata (service, source, type, referral source).
- Binds UI events on construction: contact field change watchers, add affiliation/property contact buttons, modal save buttons, submit button, view detail link, entity add button, property add button, Google address auto-complete, etc.
- Flow summary:
  1. **Contact lookup**: `this.view.onContactSelected` surfaces matches from the view search widget. When a contact is selected, the controller clears property inputs and loads related deals/properties.
  2. **Manual add**: `onManualAdd` resets selection so the user can enter contact details from scratch.
  3. **Contact save**: `#handleSave()` normalizes payload, validates required fields, detects duplicates by email, and either selects the existing record or creates a new contact via the model.

4. **Related panel**: `#loadRelated(email)` fetches properties/jobs/inquiries tied to the contact; request IDs guard against stale responses.
5. **Dropdown cards**: `#renderDropdownOptionsForTab()` + `#initDropdown()` generate “select all/none” behaviours for noises, locations, and times.
6. **Property modals**: hooking `showHideAddAddressModal`, `onAddAffiliationButtonClicked`, `onAddPropertyContactButtonClicked`, and `onAddPropertyButtonClick` toggles modals, syncs selected contact/entity, and eventually calls `model.createNewProperties()`.
7. **Inquiry submission**: `onSubmitButtonClicked()` merges `[data-inquiry-id]` and `[data-feedback-id]` inputs, injects account type + primary contact/company IDs, requires a property selection, calls `model.createNewInquiry()`, and surfaces success/failure state via the status modal.

- Other helpers: `createInquiryDetailOption()` populates service/type/source selects, `renderDropdownForStates()` feeds state options, `onSameAsContactCheckboxClicked()` copies contact address into property fields, `initAutocomplete()` wires Google Places.

View (`js/views/new-enquiry.js`)

- Wraps DOM nodes for both **Individual** and **Entity** tabs and exposes methods controllers call:
  - Contact search panel with built-in dropdown (includes empty states, add button, manual entry toggle).
  - Field getters such as `getValuesFromFields(selector, attr)` and `getFormValues()` powering the submit payload.
  - Modal helpers (`toggleModal`, `resetAffiliationModal`, `setAffiliationContacts`, etc.), loader overlay, status modal, and toast-like feedback element (`data-contact-feedback`).
  - Related panel management: `renderRelated()`, `showRelatedLoading()`, `setRelatedTab()` for properties/jobs/inquiries subsections.
  - Visual affordances: highlight selected property card, duplicate contact address for property when “same as contact” is checked, `createOptionsForSelectbox()` to populate selects from controller config.
- Maintains UI state such as the currently active tab, selected contact/property IDs, cached contacts filtered by search, and loader counters so nested async sections can show/hide a shared spinner.

Form sections & key fields

Section	Selectors	Notes
Contact search	`[data-search-root="contact-individual"]`	Debounced search, manual entry toggle, add-new button opens contact modal
Individual contact fields	`[data-contact-field]` inputs	Includes first/last name, email, sms, office phone, work requested by
Entity tab	`[data-contact-section="entity"]`	Contains company search/add UI, entity-specific `data-contact-field="entity-id"` hidden input
Property	`[data-property-id]` fields +	Captures address lines, suburb, state, postal code,

Section	Selectors	Notes
information	Google Places input	property meta
Inquiry details	`[data-inquiry-id]` inputs	Service inquiry, status, job type, priority, scheduling windows, etc. (see `inquiryConfigs` for select options)
Feedback & observation	`[data-feedback-id]` inputs	Multi-select dropdowns for noises, pest location, times active
Submission controls	`#submit-btn`, `#cancel-btn`, `#reset-btn`	Controller resets/unsets fields but actual cancel navigation is left to host app

External integrations

- **VitalStats SDK**: all CRUD flows run through the plugin instance injected when the page loads. Keep long-lived references in `NewEnquiryModel` so multiple controller calls reuse the same handles.
- **Google Places Autocomplete**: script tag appended 9 seconds after load to reduce blocking; once Google calls `initAutocomplete`, the controller stores parsed address parts via `view.setGoogleSearchAddress()`.
- **Flatpickr**: `flatpickr(".date-picker", { dateFormat: "d/m/Y", allowInput: true })` attaches to inputs with `.date-picker` class for scheduling fields.

Submission guardrails & UX notes

- Inquiry submission requires both a contact/company **and** a property. Missing prerequisites trigger the shared status modal with failure copy.
- Duplicate contacts (same email) are automatically surfaced; the existing record is selected and its related data refreshed instead of creating a new contact.
- Long-running mutations use `view.showLoader(message)` / `hideLoader()` so the CTA footer shows progress.
- `relatedRequestId` patterns prevent race conditions when multiple async related-data requests overlap (e.g., fast contact switching).

Extending the enquiry workflow

1. **New dropdown chips**: add entries to `this.inquiryConfigs` in the controller, then update the HTML template with matching `data-inquiry-id` selects.
2. **Persist more fields**: ensure the HTML input has a `data-inquiry-id`/`data-feedback-id`, update `NewEnquiryModel.createNewInquiry()` to map the field into the mutation payload, and adjust any validation logic in `#validate()`.
3. **Custom validations**: extend `#validate()` with new rules (e.g., require phone number for certain services) and surface messages via `view.showFeedback()`.
4. **Property search**: hook additional autocomplete providers by swapping `initAutocomplete` or by setting `this.propertySearchData` in the view and reusing the existing prediction UI skeleton.

Combined operational runbook

1. **Dashboard**: open `index.html`, ensure Day.js + Tailwind CDNs resolve, and confirm the controller logs `App.controllers.dashboard` in DevTools. Use the top-right filters to scope the dataset and verify rows change accordingly.
2. **Job enquiry**: open `pages/new-enquiry.html`, wait for the contact list to load, select or create a contact, attach/create a property, fill in inquiry + feedback details, then hit **Submit**. A success modal indicates the mutation executed.
3. **PDF generation**: the HTML companion file `docs/dashboard-and-job-enquiry.html` plus the exported PDF (see below) are regenerated via `npx --yes marked docs/dashboard-and-job-enquiry.md -o docs/dashboard-and-job-enquiry.html` and `libreoffice --headless --convert-to pdf docs/dashboard-and-job-enquiry.html --outdir docs`.

Artifact list

- Markdown source: `docs/dashboard-and-job-enquiry.md` (this file).
- HTML rendition: `docs/dashboard-and-job-enquiry.html`.
- Combined PDF: `docs/dashboard-and-job-enquiry.pdf`.