Case Management

Certified Pega System Architect 23

KNOWLEDGE CHEATSHEET

CASE STRUCTURE

- **Case** — Business outcome → **Stage** — Major phase → **Process** — Related work → **Step** — Task or automation

01 Case Management 33%

Case life cycle and optional work

- A **case type** models a business outcome; a **case** is one instance. **Stages** group major phases, **processes** organize work, and **steps** represent assignments or automations.
- **Primary stages** describe the normal path. **Alternate stages** handle exceptions such as cancellation. Use conditions to skip unnecessary work or choose a different path.
- A **case status** communicates progress independently of the stage name. A status configured on a step is applied when that step starts; resolution uses a Resolved status.
- An **optional action** supports a user-initiated task outside the required sequence. An **optional process** contains multiple steps. Scope either to a stage or the whole case as required.
- Use **automation steps** for work such as sending correspondence, creating cases, updating data or changing stage. An assignment pauses for human input; an automation performs configured system work.

Routing and approval patterns

- A **worklist** holds assignments for one operator. A **work queue** holds shared work that an eligible team member can claim. Use separate queues for distinct skills or departments.
- A **work group** organizes a team and its manager and relates users to work queues. Configure routing on the assignment step, including routing to the requesting employee's manager.
- **Cascading approval** sends work through multiple approvers. A **reporting structure** follows management hierarchy; an **authority matrix** selects approvers from business conditions or responsibility.
- Use **parallel processes** when several approvals can occur independently and in any order. Routing one shared assignment to a queue does not create separate approvals for every member.
- An **Approve/Reject** step can support approval through email and mobile when those options are enabled and configured. A notification email alone does not provide an actionable approval.

Case Management / Data & Integration

Certified Pega System Architect 23

KNOWLEDGE CHEATSHEET

CASE STRUCTURE

Case — Business outcome → **Stage** — Major phase → **Process** — Related work → **Step** — Task or automation

SLAs, waits and child cases

- A **service-level agreement (SLA)** defines time expectations. The **goal** is the desired completion time; the **deadline** is the required completion limit. Escalations can notify users or increase urgency.
- **Urgency** expresses work priority. **Passed deadline** intervals can repeat escalation after a missed deadline; configure advanced interval and repetition behavior in **Dev Studio**.
- A **timer Wait** pauses until an elapsed interval or specified time. An SLA monitors timeliness and triggers escalation; it does not replace a deliberate process pause.
- A **case-dependency Wait** resumes when the configured related-case status condition is met. Choose whether all or any applicable cases must meet it; allow user continuation when the design requires an override.
- A **child case** has its own life cycle while remaining related to its parent. Create a Wait dependency if the parent must await child work; creating a child alone does not express every synchronization requirement.

Decisions and duplicate detection

- A **When rule** returns true or false. Use conditions to select a path, skip a step or process, or control whether an element is visible.
- A **decision table** expresses combinations of conditions as rows with results. It suits classifications based on several attributes; a simple price-times-quantity formula belongs in a calculation.
- A **decision tree** evaluates branching, nested conditions. Choose it when one answer determines which condition is evaluated next.
- In **Search duplicate cases**, **basic conditions** are required matches. **Weighted conditions** add a match score; the configured threshold identifies potential duplicates among candidates.
- Choose duplicate criteria around the business identity of the request, such as location and request type. A name alone may match unrelated requests or miss the same request entered by another person.

02 Data and Integration 23%

Data objects and relationships

- A **data object** models an entity through fields and relationships. A **data record** is an instance, such as a product or request category; records can reside locally or in an external system.
- Use appropriate **field types** for text, dates, Boolean values and numbers. **Single-record** relationships hold one related object; **multiple-record** relationships hold a collection.
- **Embedded data** belongs to the case and is stored with it. A **data reference** points to independently maintained data, often supplied by a data page from a system of record.

Data & Integration

Certified Pega System Architect 23

KNOWLEDGE CHEATSHEET

DATA DESIGN

Model Fields & relations → **Access** Data pages → **Check** Type & business rules → **Persist** System of record

- A **case reference** relates to another case; a **query** relationship retrieves matching records. Choose the relationship according to ownership, reuse and how data is obtained.
- Use **field values** for a mostly static set of allowed or localized values. Use maintained data records and data pages when choices are numerous, change regularly or need additional attributes.

Defaults, calculations and validation

- Set a **default value** during data initialization when a new case needs a starting value. A **data transform** sets, copies or maps properties and can initialize data from another record.
- A **calculated field** derives a value from other fields, such as labor plus materials. Recalculation follows the configured expression; a one-time default and a derived value serve different needs.
- Valid data must have the correct **type** and satisfy **business rules**. Required fields and suitable controls help collection; validate business constraints when the case is submitted or processed.
- A **Validate rule** can check several fields and relationships between values, such as minimum income and maximum debt. Return useful messages that identify the unmet condition.
- An **Edit validate rule** checks the format of a property value. Use business validation for eligibility logic involving several fields; hiding or disabling a control does not enforce that logic.

Data pages, persistence and Clipboard

- A **data page** provides reusable access to data and separates consumers from retrieval details. **Page** structure holds one record; **List** structure holds a collection.
- Data pages can load from **local data**, report definitions or **connectors** to external services. Parameters select the needed data; refresh settings control when cached data is reloaded.
- A **savable data page** defines how to persist changes. **Database save** writes a mapped data instance to its database table; a **connector** save plan invokes an external persistence operation.
- The **Clipboard** displays server-side pages and properties in memory. **pyWorkPage** normally holds the current case; **data pages** hold sourced data, and requestor pages carry session context.
- Keep the **selected record**, a **list of candidate records**, and **session information** distinct. Copy data into case fields only when the case needs its own values or a snapshot.

Data & Integration / DevOps

Certified Pega System Architect 23

KNOWLEDGE CHEATSHEET

ACCESS DECISIONS

Identity Authenticate → **Context** Access group → **Roles** Authorize → **Data & actions** Enforce

Configuration settings and sets

- **Configuration settings** expose parameters that control application behavior, such as feature switches or thresholds. A **Configuration set** groups related settings by business feature or case type.
- Create and maintain sets in **App Studio**, under Settings > Configurations. Generated rules are visible in Dev Studio; settings are data instances of **Data-Configuration-Setting**.
- Reference settings through the **Condition Builder**. **Role-based access control (RBAC)** governs who can read or change them; authorized users can adjust behavior without rewriting business rules.

03 Security 5%

Personas, roles and permissions

- **Authentication** establishes identity; **authorization** determines permitted actions. Password and authentication policies protect entry, while application permissions control access after sign-in.
- A **persona** represents a user category and its responsibilities. Apply **least privilege**: define a suitable persona or role combination when existing categories grant too much access.
- An **access group** connects operators to an application context and access roles. **Access of Role to Object (ARO)** rules govern class-level permissions such as opening, modifying or deleting instances and running reports.
- ARO permissions can use **production-level** thresholds or access conditions. Interpret the configured operation and environment together; instance permissions and rule-editing permissions are separate.
- Enforce access to **cases, actions, reports and data** through security rules. A hidden menu or widget does not provide sufficient protection. Use auditing when changes to sensitive data must be tracked.

04 DevOps 5%

Unit tests, scenarios and coverage

- A **PegaUnit test** verifies a rule's behavior with controlled inputs and assertions. Examples include checking data-transform output or whether a data page returns expected data without errors.
- A **scenario test** records UI interactions through a portal or case flow. Add explicit **assertions** for field values and expected UI results, then replay the scenario to detect regressions.
- Use **test suites** to group related tests. Keep test data repeatable and isolate dependencies; passing tests should reflect expected behavior rather than accidental environment state.

DevOps / User Experience

Certified Pega System Architect 23

KNOWLEDGE CHEATSHEET

CONSTELLATION VIEWS

- **Full page** — Main interface
- **Partial** — Read-only detail
- **List** — Collection
- **Form** — Collect & edit

- The **Test coverage** tool identifies rules executed during tests and rules left uncovered. Execution coverage helps find gaps but does not prove that every important outcome was asserted.
- **Performance testing** evaluates response time and concurrency. **Deployment validation** checks a migrated application. Neither substitutes for a focused unit test or a recorded UI scenario.

05 User Experience 12%

Constellation and view types

- **Constellation** combines a design system with a view-based UI architecture. Configure prescribed components and views in App Studio; traditional UI uses sections, harnesses and skins.
- A **Full Page View** presents a case or data object's main interface. A **Partial View** presents reusable read-only information. A **List View** displays a collection from a data source.
- A **Form View** collects or edits data and can also contain read-only fields. Configure a form from a case step or create a reusable Form View on the UX tab of a case type or data object.
- Build the **data model** before assembling views so fields and relationships are available. Reuse meaningful groups of related fields rather than duplicating the same presentation throughout the application.

Controls and conditional presentation

- Use a **checkbox** for a Boolean choice, **radio buttons** to expose a short single-choice list, and a **dropdown** for a compact single-choice selector.
- Use **autocomplete** with a suitable data page for large searchable lists. Filter candidates as users type instead of loading thousands of choices into a dropdown.
- A **visibility condition** shows or hides an element. A **disable condition** leaves it visible but unavailable for editing. Apply the condition to the field or group whose behavior must change.
- In **traditional UI**, a **layout group** organizes related layouts as tabs, accordions or similar groupings. Responsive layouts change arrangement; repeating layouts display collections of records.

Channels, dashboards and accessibility

- A **channel interface** lets users interact with the application through a delivery channel. A **portal** supplies navigation, pages and work areas tailored to user responsibilities.
- A **dashboard** summarizes useful work and information through widgets. Configure shared portal content for the intended audience and enforce the underlying report or Insight access.

User Experience / Application Development

Certified Pega System Architect 23

KNOWLEDGE CHEATSHEET

APPLICATION DELIVERY

Scope MLP release → **Build** Rules & rulesets → **Inspect** Tracer & Clipboard → **Collaborate** Stories & Pulse

- Design clear **labels**, instructions, validation messages, keyboard navigation and focus behavior. Do not rely on color alone to communicate status or required actions.
- **Responsive breakpoints** trigger presentation changes at selected widths. Preserve logical reading order, useful content and usable controls as layouts adapt to smaller screens.

06 Application Development 12%

Studios, rules and rulesets

- **App Studio** supports business-focused low-code authoring. **Dev Studio** exposes detailed rule configuration and debugging. **Admin Studio** supports runtime administration; **Prediction Studio** supports decisioning models.
- A **field** is backed by a property; a **data object** is modeled by a data class; a **process** is implemented through flow rules. UI implementation depends on whether the application uses Constellation or traditional UI.
- A **rule** defines reusable application behavior. **Classes** organize rules and data; inheritance allows specialized behavior to reuse more general definitions.
- A **ruleset** packages related rules. Use a concise, meaningful name without spaces or a reserved Pega prefix. A ruleset version follows **major-minor-patch**, such as 01-02-05.
- **Major** versions mark substantial application changes; **minor** versions group enhancements; **patch** versions group smaller corrections. Lock released ruleset versions and develop changes in an appropriate new version.

Debugging execution and values

- The **Tracer** records rule execution events, including data transforms and declarative processing. Inspect event details to see rule inputs, outputs and failures in execution order.
- Use **event and ruleset filters** to focus a trace. **Breakpoints** pause at a relevant event; **Watch** tracks selected values. Save useful trace output for later analysis.
- Use the **Clipboard** to inspect current property values and page structure. Combine the stored state with Tracer execution history when a calculated value differs from expectations.
- The **Developer assistant** provides contextual suggestions during development. Review its guidance alongside the application requirements and guardrails.

Application Development / Reporting

Certified Pega System Architect 23

KNOWLEDGE CHEATSHEET

APPLICATION DELIVERY

Scope MLP release → **Build** Rules & rulesets → **Inspect** Tracer & Clipboard → **Collaborate** Stories & Pulse

Delivery scope and collaboration

- A **Minimum Lovable Product (MLP)** delivers a focused outcome that users value. Tag planned **channels** and **data objects or interfaces** with releases in Case Designer to support scope and sizing.
- The **Estimator** uses scope and complexity information to estimate effort. Out-of-the-box components still require configuration and testing; release tags alone do not mean work is complete.
- Track **user stories**, acceptance criteria, feedback and bugs through development collaboration tools such as Agile Workbench. Acceptance criteria describe observable business behavior.
- **Pulse** supports contextual case conversations. **Tags** label cases for discovery and organization. Dashboards summarize information but do not replace case discussion.

07 Reporting 5%

Reports, aggregations and Insights

- **Business reports** analyze business facts such as revenue, regions or products. **Process reports** analyze workflow performance, such as case volume, assignment duration and timeliness.
- A **report definition** selects columns, filters, sorting and aggregation. To show requests per cost center, group by cost center and **count case IDs**.

Reporting / Mobility

Certified Pega System Architect 23

KNOWLEDGE CHEATSHEET

MOBILE EXPERIENCE

- **Channel** — Navigation
- **Brand** — Theme & assets
- **Touch** — Taps & swipes
- **Verify** — Preview & devices

- Use **filters** for the intended population and time window, then choose **count, sum, average, minimum or maximum** to match the business measure. Sorting alone does not aggregate records.
- In Constellation, use **Explore data** to create an **Insight** with a suitable dataset, visualization and filters. Case status plus a count can summarize the current workload.
- Place an **Insight widget** on the appropriate shared portal dashboard when a whole audience needs the visualization. Align sharing and permissions with that audience; a private dashboard serves its owner.

08 Mobility 5%

Mobile channel and touch design

- Configure a **mobile channel** in App Studio for the mobile experience, including navigation, branding, supported swipe actions and app security behavior such as inactivity locking.
- Plan **theme colors and image assets** for the mobile app. Follow the asset requirements in the channel configuration; platform-specific icon sizes are separate from business workflow design.
- Design for **finger taps**, readable text and limited screen space. Keep frequent actions easy to reach and avoid interactions that depend on hovering or precision pointer movement.
- Use **mobile preview** to check responsive behavior, then test on target devices for touch interactions and orientation. Persona and role permissions continue to control access to mobile work.