Front-end integration guide: dynamic form builder

This document describes how the front-end should orchestrate form creation, section/question management, option authoring, and version control using the form builder APIs exposed by the sales service. It covers endpoints, request/response shapes, validation rules, and implementation notes that help keep the UI state aligned with backend expectations. Pair this guide with the platform overview in `docs/form-builder.md` for deeper context on the persistence model and lifecycle semantics shared across channels. F:docs/form-builder.md†L1-L200

Domain model overview

The form builder revolves around a hierarchy of reusable entities:

Entity	Key attributes	Lifecycle
Form	`name`, `description`, `formType`, `companyRegionId`, `status`	Forms represent a bundle of sections and questions. Status can be `active`, `inactive`, or `archived`. The type enumerations are `cds1`, `cds2`, `lead_registration`, or `kyc`. F:models/forms.ts†L4-L47 F:components/form/repository/FormRepository.ts L107
Section	`name`, `order`, `isActive`, `status`, `etag`	Sections are children of a form and have draft/published/archived states with optimistic locking via `etag` and version counters. F:models/form_sections.ts†L5-L157 F:components/formBuilder/service.ts†L1189-L1415

Entity	Key attributes	Lifecycle
Question	`sectionId`, `order`, `questionTemplateId`, `required`, `visibleIfJson`, `optionsApi`, `dependsOn`, `status`, `etag`	Questions reference reusable templates that hold the canonical label, answer type, validation, default value, storage metadata, and static options. They also use optimistic locking and can be archived individually. F:models/form_questions.ts†L4-L199 F:components/formBuilder/service.ts†L1600-L2018
Question template	`tkey`, `label`, `answerType`, `validationJson`, `defaultValue`, `storageMetadata`	Shared definitions that multiple forms reuse. Static options are stored as `question_template_options` ordered by `order` with label/value pairs. F:models/question_templates.ts†L6-L141 F:models/question_template_options.ts†L4-L127
Form version	`versionNumber`, `payloadJson`, `publishedBy`, `notes`	Snapshots of published sections/questions. Used for publishing, rollback, and public delivery flows. F:models/form_versions.ts†L4-L99

All form builder endpoints are mounted beneath `/api/v1` and require a valid JWT except for the public read-only version endpoints. Request interception (encryption) is handled by middleware for form CRUD but not for builder operations. F:routes/index.ts†L41-L71 F:components/form/form.route.ts†L19-L25 F:components/formBuilder/formBuilder.route.ts†L16-L400

Authentication and headers

- **JWT**: Every modifying endpoint uses `requireAuth`. Ensure the browser attaches the bearer token. F:components/formBuilder/formBuilder.route.ts†L18-L180
- **Encrypted payloads**: Legacy form CRUD (`/forms`) additionally pipes through `CryptoService.interceptRequest`, so the UI must keep using the shared request interceptor helper when calling those

endpoints. F:components/form/form.route.ts†L9-L23

- **Optimistic locking**: Section and question update/delete flows expect the latest `etag` in the `If-Match` header. Store the returned `etag` from fetch responses and resend it on subsequent mutations. F:components/formBuilder/formBuilder.route.ts†L55-L183 F:components/formBuilder/service.ts†L1241-L2018

Builder API operations quick reference

The table below lists every builder-facing operation, the HTTP contract, and the key UI responsibilities to keep requests valid. Use it as the one-stop reference when wiring data providers or debugging 4xx responses.

Domain	Method & path	Purpose	Front-end considerations
Forms	`GET /api/v1/forms`	Paginated form directory with filters.	Pass `page`/`limit` query params from the list view store and forward search/filter state.
Forms	`POST /api/v1/forms`	Create a new form shell.	Send the full payload via the encrypted client helper so the interceptor runs; pre-generate slugs if you need stable public URLs.
Forms	`PUT /api/v1/forms/{id}`	Update form metadata.	Submit the entire form object, not a diff, to avoid clearing fields; refresh state when `affectedRows` is `0`.
Forms	`DELETE /api/v1/forms/{id}`	Archive/delete a form.	Treat as archive in the UI and redirect away from builder pages on success.

Domain	Method & path	Purpose	Front-end considerations
Sections	`GET /api/v1/forms/{formId}/sections`	List sections for a form.	Persist returned `etag`/`version` for optimistic updates; apply client sorting for drag-and-drop.
Sections	`POST /api/v1/forms/{formId}/sections`	Create a new section.	Capture the generated slug/order immediately so reorder UIs stay in sync.
Sections	`PATCH /api/v1/sections/{sectionId}`	Update section details/order.	Send `If-Match` header with the stored `etag`; retry with refetch on 409.
Sections	`DELETE /api/v1/sections/{sectionId}`	Archive a section.	Expect cascade archiving of child questions—refresh question lists after completion.
Questions	`GET /api/v1/forms/{formId}/questions`	Paginated questions for form.	Attach section/status/type filters and keep pagination tokens in state.
Questions	`POST /api/v1/forms/{formId}/questions`	Create/reuse a question instance.	Provide either a `questionTemplateId` or inline template definition; include overrides like `visibleIf` and `options`.
Questions	`PATCH /api/v1/questions/{questionId}`	Update an existing question.	Supply `If-Match` and send only changed fields; handle template cloning effects server-side.

Domain	Method & path	Purpose	Front-end considerations
Questions	`DELETE /api/v1/questions/{questionId}`	Archive a question instance.	Remove from active lists and surface auto-publish toasts if the form already has live versions.
Question library	`GET /api/v1/form-questions`	Search reusable questions across forms.	Feed region/form filters to help editors discover templates.
Question library	`POST /api/v1/form-questions`	Seed a template into a new region or create catalog entries.	Restricted to admins; use encrypted client helper to satisfy interceptor.
Bulk ops	`POST /api/v1/forms/{formId}/questions/import`	CSV import for questions/sections.	Upload via multipart key `file`; display per-row validation errors from response.
Bulk ops	`GET /api/v1/forms/{formId}/questions/export`	Export builder payload as CSV/JSON.	Pass `format=json` when needed; otherwise treat response content as CSV text.
Publishing & versions	`POST /api/v1/forms/{formId}/publish`	Publish current draft.	Optional `{ "notes": "..." }` body; refresh version timeline after success.
Publishing & versions	`GET /api/v1/forms/{formId}/versions`	List published versions.	Cache results for version selector widgets.
Publishing & versions	`GET /api/v1/forms/{formId}/versions/{versionNumber}`	Fetch a specific version.	Show 404 when the version no longer exists.

Domain	Method & path	Purpose	Front-end considerations
Publishing & versions	`POST /api/v1/forms/{formId}/rollback/{versionNumber}`	Create a new draft from a historical snapshot.	Confirm with users that rollback creates a fresh version entry.
Public delivery	`GET /api/v1/forms/{formSlug}/live`	Retrieve latest live snapshot (no auth).	Drive public renderers or preview modes.
Public delivery	`GET /api/v1/forms/{formSlug}/versions/{versionNumber}`	Retrieve a specific published snapshot by slug (no auth).	Use for regulators or QA to view historical payloads.
Auditing	`GET /api/v1/forms/{formId}/audit`	Paginated audit events.	Forward `entity/entityId` filters to scope timelines.

API envelope, error handling, and status codes

All builder routes return the standard envelope `{ status: 'success', data }` or `{ status: false, code, message, data }`. Reuse the shared HTTP client wrapper so toast notifications can inspect `status/message`. Payload validation errors surface detailed Joi messages per field, while domain invariants use explicit error codes such as `FORM_NOT_FOUND`, `SECTION_ARCHIVED`, `OPTIONS_REQUIRED`, and `ETAG_MISMATCH`. Map these codes to user-friendly copy in the UI (for example, translate `OPTIONS_REQUIRED` into "Add at least one option or connect an options API"). F:docs/form-builder.md†L62-L140 F:components/formBuilder/service.ts†L700-L743 F:components/formBuilder/service.ts†L1400-L2018

Managing forms

Base path: `/api/v1/forms` (handled by the legacy controller). Joi schemas enforce valid region/type combinations and guard against missing pagination params before requests hit the repository layer. F:components/form/form.route.ts†L9-L23 F:components/form/schema/categorySchema.ts†L4-L62 Continue using the shared request helper so filters are encoded consistently.

List forms

```
GET /api/v1/forms?page=1&limit=20&status=active
```

- Accepted filters: `status`, `companyRegionId`, and `formType`. Unknown filters are ignored because the repository scrubs `page/limit` from the query

object before forwarding the rest to Sequelize. F:components/form/repository/FormRepository.ts†L37-L59

- Response payload (envelope omitted):

```
{
  "count": 28,
  "rows": [
    {
      "id": 42,
      "name": "Credit Assessment",
      "slug": "credit-assessment-j3k9",
      "description": "Eligibility walkthrough",
      "formType": "cds2",
      "companyRegionId": 1,
      "status": "active",
      "createdAt": "2024-01-05T12:42:18.000Z",
      "updatedAt": "2024-01-12T09:51:13.000Z"
    }
  ],
  "page": 1,
  "limit": 20
}
```

- Builder UI tips: keep pagination controls in sync with `count`, expose the slug (if present) for deep links, and surface archived forms in a separate tab by filtering `status=archived`.

Create form

`POST /api/v1/forms`

```
{
  "name": "Credit Assessment",
  "description": "Eligibility walkthrough",
  "formType": "cds2",
  "companyRegionId": 1,
  "status": "active"
}
```

- `status` defaults to `active`. `slug` is optional—if you want a stable public path, pre-generate it with `generateFormSlug(name)` before sending the payload. F:components/form/repository/FormRepository.ts†L16-L35 F:components/formBuilder/utils/slug.ts†L15-L18
- Success responses include the persisted row so the UI can push it into state without a follow-up fetch.
- Joi validation errors are returned as `422` with `details` entries. Surface them inline next to the relevant input fields.

Update form

`PUT /api/v1/forms/{id}`

- Use the same payload as creation. When `affectedRows` equals 0, the repository responds with `{ status: false, code: 404 }`; prompt the user to refresh since the form may have been archived by someone else. F:components/form/repository/FormRepository.ts†L61-L87
- Because the endpoint replaces the record wholesale, submit the full form state (not a delta) to avoid unintentionally clearing fields.

Soft delete form

`DELETE /api/v1/forms/{id}`

- Updates the row to `status: 'deleted'`. Treat this as an archive toggle in the UI and redirect authors away from builder screens once the request succeeds. F:components/form/repository/FormRepository.ts†L88-L107
- The deleted row is returned so you can display confirmation banners containing the form name and ID.

Managing sections

Sections are accessed through `/api/v1/forms/{formId}/sections`.

List sections

`GET /api/v1/forms/{formId}/sections?status=published`

- Filters on status; defaults to excluding archived sections. Combine with client-side sorting when you allow drag-and-drop reordering in the UI.
- Returns ordered sections with derived `legacyStatus` (mapped to `active/inactive/archived`) and the `etag` needed for edits. F:components/formBuilder/formBuilder.rou L34 F:components/formBuilder/service.ts†L1189-L1240 F:components/formBuilder/service.ts†L520-L608
- Sample response (single item):

```
{
  "id": 101,
  "formId": 42,
  "companyRegionId": 1,
  "formType": "cds2",
  "slug": "42-applicant-details-a1b2",
  "name": "Applicant details",
  "description": null,
  "meta": null,
  "order": 1,
  "isActive": true,
```

```
    "status": "published",
    "legacyStatus": "active",
    "etag": "\"bfb3c1b0\"",
    "version": 3,
    "archivedAt": null,
    "createdAt": "2024-01-05T12:42:18.000Z",
    "updatedAt": "2024-01-12T09:51:13.000Z",
    "createdBy": "auth0|abc",
    "updatedBy": "auth0|xyz"
}
```

- Persist the returned `etag` and `version` in your section store so reorder/save actions can send the correct concurrency token.

Create section

`POST /api/v1/forms/{formId}/sections`

```
{
  "name": "Applicant details",
  "order": 1,
  "isActive": true
}
```

- Payload rules: `name` required, `order` >= 1, `isActive` optional. Validation uses Joi and normalizes alternative property names (`sortOrder`, `status`). F:components/formBuilder/validators.ts†L3-L13 F:components/formBuilder/service.ts†L520-L609 F:components/formBuilder/service.ts†L1241-L1308
- Automatically assigns `order` if omitted, generates a slug (`{formId}-{slugified-name}-{random}`), and sets status to `draft`. After creation, if the form already has published versions, the service auto-publishes to keep live snapshots synchronized, so the UI should be ready to show updated version numbers in the publish history. F:components/formBuilder/service.ts†L1263-L1308 F:components/formBuilder/service.ts†L2342-L2353 F:components/formBuilder/utils/slug.ts†L1-L13
- Response includes the newly generated `etag` and slug—update your drag-and-drop state immediately so reorder handles the new section.

Update section

`PATCH /api/v1/sections/{sectionId}` with `If-Match: {etag}`

```
{
  "name": "Applicant information",
  "order": 2,
  "isActive": false
}
```

- Must include at least one field. Rejects updates to archived sections. Returns the latest `etag` and increments the internal version counter. Auto-publish logic runs after successful updates if a live version exists. F:components/formBuilder/validators.ts†L9-L13 F:components/formBuilder/service.ts†L1324-L1392 F:components/formBuilder/service.ts†L2342-L2353
- Concurrency failures surface `ETAG_MISMATCH` with HTTP 409—listen for that code and trigger a refetch before allowing the user to retry. F:components/formBuilder/service.ts†L1346-L1375

Archive section

`DELETE /api/v1/sections/{sectionId}`

- Marks the section archived, disables it, logs an audit entry, and recursively archives questions within the section (the UI should refresh question lists after this call). F:components/formBuilder/formBuilder.route.ts†L79-L92 F:components/formBuilder/service.ts†L1395-L1445
- Because child questions are archived one by one, show a loading state until the mutation resolves to avoid surfacing partially updated lists.

Managing questions

Questions sit under `/api/v1/forms/{formId}/questions`.

List questions

`GET /api/v1/forms/{formId}/questions?sectionId=12&status=draft&type=dropdown&page=1&pageSize`

- Supports filter parameters for section, status, answer type, and free-text search (`q` filters on label/tkey). Pagination defaults to page 1 / page size 25, capped at 100. Results include resolved template data, static options, visibility rules, and `etag`. F:components/formBuilder/formBuilder.route.ts†L94-L126 F:components/formBuilder/service.ts†L1447-L1509 F:components/formBuilder/service.ts†L520-L744
- Response payload structure:

```
{
  "items": [
    {
      "id": 555,
      "formId": 42,
      "sectionId": 101,
      "order": 1,
      "questionTemplateId": 45,
      "tkey": "applicant_full_name",
      "label": "Applicant full name",
```

```
      "helperText": "As on national ID",
      "answerType": "text",
      "required": true,
      "validation": { "min": null, "max": null },
      "visibleIf": [],
      "defaultValue": null,
      "optionsApi": null,
      "dependsOn": [],
      "options": [],
      "storage": { "tableName": "applicants", "columnName": "full_name" },
      "status": "published",
      "legacyStatus": "active",
      "etag": "\"a81d3e2c\"",
      "version": 4,
      "createdAt": "2024-01-05T12:42:18.000Z",
      "updatedAt": "2024-01-12T09:51:13.000Z"
    }
  ],
  "page": 1,
  "pageSize": 25,
  "total": 12
}
```

- Each item merges template metadata with per-form overrides. Cache `questionTemplateId` and `tkey` locally for reuse pickers when authors add follow-up questions.

Create question

`POST /api/v1/forms/{formId}/questions`

Common scenarios:

1. **Reuse an existing template**

```
{
  "sectionId": 12,
  "questionTemplateId": 45,
  "order": 3,
  "required": true,
  "visibleIf": [
    { "questionId": "household_income", "operator": "gte", "value": 20000 }
  ]
}
```

   - The backend resolves the template, enforces option requirements, and stores visibility rules after validating referenced questions. F:components/formBuilder/service.ts†L1649-L1776

2. **Inline template definition** (for ad-hoc questions)

```
{
  "sectionId": 12,
  "tkey": "kyc_birth_date",
  "label": "Date of birth",
  "answerType": "date",
  "helperText": "DD/MM/YYYY",
  "required": true,
  "validation": { "past": true }
}
```

   - At least one of `questionTemplateId`, `template`, or the trio `tkey`/`label`/`answerType` is required. The Joi validator enforces this and ensures enumerated types are lowercase versions of the supported answer types. F:components/formBuilder/validators.ts†L61-L94

3. **Enumerated options (static list)**

```
{
  "sectionId": 12,
  "tkey": "employment_status",
  "label": "Employment status",
  "answerType": "dropdown",
  "options": [
    { "tkey": "employed", "label": "Employed", "value": "employed", "order": 1 },
    { "tkey": "self_employed", "label": "Self-employed", "value": "self", "order": 2 }
  ]
}
```

   - For dropdown/radio questions, at least one static option or a dynamic `optionsApi` must be supplied. Non-enumerated types must not include options. The service normalizes and validates these invariants and rejects inconsistent payloads. F:components/formBuilder/service.ts†L1665-L1674 F:components/formBuilder/service.ts†L1681-L1763 F:components/formBuilder/service.ts†L2338

4. **Dynamic options**

```
{
  "sectionId": 12,
  "questionTemplateId": 60,
  "optionsApi": "/api/v1/catalogue/products",
  "dependsOn": ["business_type"],
  "defaultValue": null
}
```

   - `optionsApi` can be relative; the backend stores it as-is after URI validation. Use `dependsOn` to list template keys that should

trigger refreshes when they change (front end should wire watchers accordingly). F:components/formBuilder/validators.ts†L61-L114 F:components/formBuilder/service.ts†L1649-L1763

Additional creation patterns and payload tricks

The validator accepts the broader legacy payload surface documented in `formQuestionSwagger.ts`, so the UI can support power users without hand-writing adapters. F:docs/formQuestionSwagger.ts†L1-L160 F:docs/formQuestionSwagger.ts†L160-L320 When surfacing advanced configuration panels, account for the following patterns:

5. **Create-and-share a catalog template in one call**

```
{
  "sectionId": 44,
  "template": {
    "tkey": "kyc.business_registration",
    "label": "Business registration number",
    "answerType": "text",
    "question": "What is the official registration number?",
    "helperText": "Exact value from the certificate",
    "validationJson": { "maxLength": 30 },
    "defaultValue": null,
    "storage": {
      "tableName": "business_profiles",
      "columnName": "registration_number",
      "persistToTable": true
    },
    "options": []
  },
  "required": true
}
```

   - Sending a `template` object creates or updates catalog rows before the instance is inserted, so subsequent forms can discover the new `tkey` through the library search without extra API calls. F:components/formBuilder/service.ts†L390-L460 F:components/formBuilder/service.ts†L1649-L1763 Store the `questionTemplateId` from the response to ensure edit dialogs point back to the shared template.

6. **Enumerations with expected answers**

```
{
  "sectionId": 52,
  "tkey": "loan.repayment_frequency",
  "label": "Preferred repayment frequency",
```

```
  "answerType": "dropdown",
  "expectedAnswers": [
    { "tkey": "weekly", "label": "Weekly", "value": "weekly", "order": 1 },
    { "tkey": "monthly", "label": "Monthly", "value": "monthly", "order": 2 }
  ],
  "defaultValue": "monthly"
}
```

- `expectedAnswers` and `answers` are accepted aliases for `options`; the service normalizes all three to the canonical option array so scoring jobs and expected-answer APIs stay in sync. F:components/formBuilder/service.ts†L380-L408 F:docs/formQuestionSwagger.ts†L1-L160 Document both field names in tooltips so legacy CSV imports continue to round-trip without losing data.

7. **Dynamic options with dependent reload and custom defaults**

```
{
  "sectionId": 61,
  "questionTemplateId": 99,
  "optionsApi": "/api/v1/credit-products?country={country_code}",
  "dependsOn": ["country_code", "loan_amount"],
  "defaultValue": {
    "tkey": "popular_product",
    "label": "Popular product"
  }
}
```

- `dependsOn` is de-duplicated and trimmed server side, so the UI can allow comma-separated entry while still sending clean arrays. Encourage editors to use template keys in the dependency selector—the resolver will map those keys to actual question IDs once the dependent questions exist. F:components/formBuilder/service.ts†L300-L360 F:components/formBuilder/service.ts†L783-L920 The dynamic default shown above is preserved as-is and can be interpreted by custom renderers.

Reusing questions across forms and regions

The form builder now allows the same logical question to live on multiple forms so multinational teams can share catalogue content while tailoring the experience per country. Authoring flows should treat the template as the shared asset and create form-specific instances that point to it.

- **Surface a question library picker** – Fetch reusable candidates via `GET /api/v1/form-questions?formId={sourceFormId}&tkey=...&page=1&limit=50`. Results include template metadata plus region coverage, making it easy to confirm whether the template is already enabled

for another market. F:components/questions/question.route.ts†L17-L21 F:components/questions/repository/QuestionRepository.ts†L214-L329 Allow switching the source form to browse different catalogues.

- **Attach to another form** – When the author chooses a question, call `POST /api/v1/forms/{targetFormId}/questions` with the `questionTemplateId` (and any per-form overrides such as visibility rules or default values). The service reuses the shared template, validates enumerations, and persists a new `form_questions` row linked to the target form. F:components/formBuilder/formBuilder.route.ts†L94-L144 F:components/formBuilder/service.ts†L1649-L1776 Example payload:

```
{
  "sectionId": 31,
  "questionTemplateId": 45,
  "order": 5,
  "required": true,
  "visibleIf": [
    { "questionId": "country_code", "operator": "eq", "value": "KE" }
  ],
  "defaultValue": null
}
```

  The response contains the cloned instance with its own `id`, `etag`, and region-safe template metadata. Store the returned `id` in the target form state just like newly authored questions.

- **Country coverage** – Each form carries a `companyRegionId`, and templates have explicit region mappings enforced through `question_template_regions`. Attempting to reuse a template that is not yet enabled for the target region returns `QUESTION_TEMPLATE_REGION_UNAVAILABLE`; prompt the user to either switch to a form that shares the same region or extend the template's coverage. F:models/forms.ts†L17-L123 F:models/question_template_regions.ts†L1-L94 F:components/formBuilder/service.ts†L999-L1186

- **Extending a template to a new country** – Provide an admin-only action that calls `POST /api/v1/form-questions` with the template payload and the new form/region. The legacy question controller creates the missing `question_template_regions` join when the template is first introduced to that region, after which subsequent builder calls can reuse it freely. F:components/questions/question.route.ts†L17-L24 F:components/questions/repository/QuestionRepository.ts†L214-L329 Display a confirmation banner so editors know the template is now globally available.

- **Audit implications** – Every form still maintains its own section/question history, so updates to a shared template's label/options should be coordi-

nated. Surface warnings when an edit will impact multiple forms and link to the activity log for quick verification. F:components/formBuilder/service.ts†L2185-L2338 F:components/formBuilder/formBuilder.route.ts†L242-L305

On success, the response merges instance state (required flag, order, `visibleIf`, etc.) with template metadata (`label`, `answerType`, default value, options) so the front end can render without extra lookups. F:components/formBuilder/service.ts†L520-L744 F:components/formBuilder/service.ts†L1681-L1769 Auto-publish is triggered if the form already has a live snapshot. F:components/formBuilder/service.ts†L1958-L1965

Question persistence and storage contract

The `createQuestion` flow ultimately writes to the `form_question_instances` table, so UI payloads must map cleanly to its schema and the related template tables. F:components/formBuilder/service.ts†L1649-L1758 F:models/form_questions.ts†L28-L214 Key storage rules:

- **Instance fields** – Every record stores `formId`, `sectionId`, `questionTemplateId`, `order`, `required`, `helperText`, `visibleIfJson`, `optionsApi`, `dependsOn`, `status`, `version`, and the optimistic `etag`. The service also sets `createdBy/updatedBy` from the actor ID, so ensure the UI forwards authenticated requests even during background saves. F:components/formBuilder/service.ts†L1695-L1757 F:models/form_questions.ts†L28-L214
- **Template linkage** – Labels, answer types, validation, default value, storage metadata, and static options live in the reusable template tables. When you send inline template data, the backend creates/updates `question_templates` + `question_template_options` before inserting the instance. Reusing an existing template simply references the `questionTemplateId` and pulls the shared option set. F:components/formBuilder/service.ts†L1649-L1763 F:models/question_templates.ts†L6-L92 F:models/question_template_options.ts†L4-L88
- **Region gating** – The `question_template_regions` join ensures each template is only available in allowed markets; attempting to save a question in a disallowed region fails early. Use the region picker UI to call the admin `/form-questions` endpoint when you need to extend coverage. F:components/formBuilder/service.ts†L1013-L1128 F:models/question_template_regions.ts†L5-L64 F:components/questions/question.route.ts†L19-L23
- **Unique per form** – `form_question_instances` enforces a unique index on `(formId, questionTemplateId)`. Prevent duplicate adds in the UI by disabling already-attached templates or prompting authors to clone the template when they truly need divergent copies. F:models/form_questions.ts†L200-L214 F:components/formBuilder/service.ts†L1649-L1763

The response includes the persisted row and resolved template metadata so you

can synchronise state stores without extra fetches:

```
{
  "id": 912,
  "formId": 42,
  "sectionId": 101,
  "questionTemplateId": 45,
  "label": "Applicant full name",
  "answerType": "text",
  "required": true,
  "visibleIf": [],
  "options": [],
  "optionsApi": null,
  "dependsOn": [],
  "status": "draft",
  "version": 1,
  "etag": "\"a1b2c3d4\"",
  "createdAt": "2024-02-12T09:01:12.000Z",
  "updatedAt": "2024-02-12T09:01:12.000Z"
}
```

After a successful save:

1. Store the `id`, `version`, and `etag` in your question cache so subsequent edits can pass optimistic-lock headers. F:components/formBuilder/service.ts†L1695-L1758
2. Merge the template metadata (`label`, `answerType`, options) directly into presentation state; the API already resolved and normalised it to match renderer expectations. F:components/formBuilder/service.ts†L1681-L1769
3. Update dependency graphs—`visibleIfJson` and `dependsOn` persist as arrays, so keep them in sync locally to avoid sending stale references on the next PATCH. F:models/form_questions.ts†L36-L153 F:components/formBuilder/service.ts†L1690-L1737
4. Reflect status/auto-publish side-effects by refreshing version widgets whenever the service bumps a live snapshot. The create response may include the same version number but the auto-publish hook triggers background updates you should surface to editors. F:components/formBuilder/service.ts†L1950-L1969

Question response payload reference

Every `GET/POST/PATCH` response for questions conforms to the `FormBuilderQuestionResponse` type exported to the front end. Use the following field summary when typing stores or testing selectors. F:types/formBuilder/FormBuilderQuestion.ts†L1-L108 F:components/formBuilder/service.ts†L600-L704

Field	Origin	Notes
`companyRegionId`, `formType`	Form metadata	Populated so region-aware UIs can warn when questions cross markets.
`categoryId`, `position`	Section instance	Mirrors `sectionId`/`order` for backwards-compatible consumers; treat them as read-only aliases.
`questionKey`, `label`, `question`, `validationJson`, `storage`	Template	Remain identical across forms unless the template is cloned; always show the template helper text when the instance override is null.
`expectedAnswers`	Template options	Mirrors `options` so scoring engines and exports can keep using legacy property names.
`isRequired`, `legacyStatus`	Derived	Computed by the mapper to satisfy legacy widgets that expect string enums.
`etag`, `version`, `createdBy`, `updatedBy`	Instance	Required for optimistic updates and audit stamps—store them even if your UI does not display the values.

The mapper sorts options by `order` and ensures each option contains `optionKey`, `name`, and `answer` aliases, so dropdown widgets can keep using whichever property they previously relied on. F:components/formBuilder/service.ts†L600-L704

Visibility, dependency, and uniqueness rules

- **Condition references** – `visibleIf.questionId` may be a numeric question ID or a template `tkey`. The resolver cross-checks both forms and throws `VISIBLE_IF_INVALID_REFERENCE` when the referenced question is missing, so the UI should validate selectors and surface inline errors before submitting. F:components/formBuilder/service.ts†L783-L920
- **Dependency dedupe** – `dependsOn` arrays are trimmed, lower-noise strings. When editors type comma-separated keys, send

them as-is; the normalizer removes blanks and duplicates for you. F:components/formBuilder/service.ts†L300-L344

- **Unique index guard** – Attempting to attach the same template twice to a form triggers the unique constraint. Offer a "Create form-specific copy" affordance that calls `POST /form-questions` with a new `tkey` so the backend persists a sibling template before adding the instance. F:models/form_questions.ts†L200-L214 F:components/formBuilder/service.ts†L1649-L1763

Expected answers and scoring integrations

Some business flows store answer keys for downstream scoring or eligibility checks using the expected-answer endpoints. The builder should expose a management panel only when the answer type supports enumerations.

- **List expected answers** – `GET /api/v1/expected-answers?questionId={id}` returns catalogued answers for scoring workflows. Use it to hydrate side drawers when reviewers inspect conditional logic. F:docs/formExpectedAnswerSwagger.ts†L1-L120
- **Create/update expected answers** – `POST /api/v1/expected-answers` and `PUT /api/v1/expected-answers/{id}` accept the same fields as option payloads (`tkey`, `answer`, `field`, `name`, `thetype`) and reuse the encrypted transport header. Serialize builder edits through these endpoints when authors define auto-validation rules. F:docs/formExpectedAnswerSwagger.ts†L1-L200
- **Builder sync** – Whenever questions with enumerations are saved, mirror the option array to expected answers if the question participates in automated grading flows. The normalization helper already feeds `expectedAnswers` in the response, so the UI can render current values without extra lookups. F:components/formBuilder/service.ts†L380-L408 F:components/formBuilder/service.ts†L600-L704

Question library API deep dive

Use the library endpoints to seed picker dialogs or admin consoles that curate reusable templates.

- **Search catalog** – `GET /api/v1/form-questions?formId=42&tkey=address&page=1&limit=20` returns template rows enriched with region metadata, helper text, and storage bindings. The swagger file documents optional filters like `categoryId`, `companyRegionId`, and free-text search; wire them to your filter chips for a consistent experience. F:docs/formQuestionSwagger.ts†L1-L160 F:components/questions/repository/QuestionRepository.ts†L214-L329 Display badges for regions returned in `question_template_regions` so editors know whether reuse is allowed immediately.
- **Extend catalog coverage** – Admins can call `POST /api/v1/form-questions` with either `questionTemplateId` + `companyRegionId` to enable an exist-

ing template in a new market, or with a full `template` payload to create a brand-new entry. Surface success banners stating that the template is now available in the selected country so editors understand why it appears in subsequent searches. F:docs/formQuestionSwagger.ts†L1-L200 F:components/questions/repository/QuestionRepository.ts†L214-L329

- **Instance hydration** – After the admin flow completes, reuse the standard `POST /api/v1/forms/{formId}/questions` call documented above. Because the repository ensures template-region joins exist, the builder can attach the template without special casing per market logic. F:components/formBuilder/service.ts†L999-L1186 F:components/formBuilder/service.ts†L1649-L1763

Update question

`PATCH /api/v1/questions/{questionId}` with `If-Match: {etag}`

- Accepts the same fields as creation (all optional). Changes that affect template-driven properties (label, answer type, options) cause the backend to resolve or clone the appropriate template before applying updates. Visibility rules are revalidated and stored when modified. The response includes the refreshed template and options. F:components/formBuilder/service.ts†L1752-L1970
- Auto-publish runs after updates when previous versions existed, so the UI should refresh version history widgets if they are visible. F:components/formBuilder/service.ts†L1958-L1969
- Handle `ETAG_MISMATCH` errors the same way as sections: refetch and replay unsaved changes if the backend detects concurrent edits. F:components/formBuilder/service.ts†L1888-L1944

Archive question

`DELETE /api/v1/questions/{questionId}` simply marks the question archived (no payload). UI should remove the question from lists unless showing archived items. F:components/formBuilder/formBuilder.route.ts†L171-L184 F:components/formBuilder/service.ts†L1972-L2018 * Archiving a question does **not** delete its template. If the form had been published previously, the auto-publish hook will snapshot the change automatically—surface a toast indicating the live version bumped. F:components/formBuilder/service.ts†L1958-L1965 F:components/formBuilder/service.ts†L1972-L2018

Template resolution, storage metadata, and region gating

When a question payload arrives, the backend determines whether to reuse a catalog template or create/update one on the fly. Understanding this flow helps the front end present the right affordances. F:components/formBuilder/service.ts†L985-L1186

- **Template lookup priority** – If `questionTemplateId` is supplied, the API fetches that template and verifies it's available for the form's `companyRegionId`. Without a numeric ID, the service reuses/creates a template based on the provided `tkey` + metadata. Warn authors if they try to reuse a template that isn't enabled for their region (`QUESTION_TEMPLATE_REGION_UNAVAILABLE`). F:components/formBuilder/service.ts†L999-L1128
- **Inline definitions** – Sending `{ template: { tkey, label, answerType, ... } }` or the inline `tkey`/`label`/`answerType` trio instructs the service to create or reuse a template. Helper text, validation JSON, default values, storage bindings, and static options are all normalized before persistence. F:components/formBuilder/validators.ts†L45-L94 F:components/formBuilder/service.ts†L1076-L1186
- **Storage metadata** – Passing `storage` (or `template.storage`) lets authors map answers to downstream tables/columns. The helper sanitizes column names, fills defaults (`columnName` from `tkey`), and ignores empty configs so the UI can let users toggle persistence on/off without manual cleanup. F:components/formBuilder/service.ts†L934-L982 F:components/formBuilder/service.ts†L1025-L1054
- **Region mapping** – Whenever a template is reused/created, the service ensures `question_template_regions` contains the form's region. If not, it throws `QUESTION_TEMPLATE_REGION_UNAVAILABLE`, so offer a region picker or duplicate-template workflow in the UI to resolve the conflict. F:components/formBuilder/service.ts†L1013-L1128
- **Option cloning** – Static options are bulk-created when a brand-new template is saved. When reusing an existing template, the canonical option list is pulled straight from the catalog so every form stays in sync. F:components/formBuilder/service.ts†L1169-L1186

Option authoring patterns

Dropdown and radio questions carry extra requirements that the UI should enforce ahead of time. F:components/formBuilder/validators.ts†L38-L83 F:components/formBuilder/service.ts†L722-L743 F:components/formBuilder/service.ts†L2303-L2338

- **Option schema** – Each option needs a non-empty `tkey`. `label`/`value` may be blank, but the UI should encourage descriptive values for analytics. The service auto-fills `order` when omitted, so drag handles can simply send the list order back to the server. F:components/formBuilder/validators.ts†L38-L44 F:components/formBuilder/service.ts†L758-L760
- **Dynamic vs static options** – If `answerType` is `dropdown` or `radio`, ensure either `options` contains at least one entry or `optionsApi` is a valid URI; mixing both is allowed. For other answer types, omit both to avoid `OPTIONS_NOT_ALLOWED` errors. F:components/formBuilder/service.ts†L722-L743 F:components/formBuilder/validators.ts†L61-L114

- **Duplicate prevention** – During publish, the backend rejects duplicate option `tkey` values. The UI should enforce uniqueness client-side and visually flag duplicates during editing. F:components/formBuilder/service.ts†L2322-L2336
- **Dynamic option dependencies** – `dependsOn` accepts template keys so the renderer can refresh the options API after prerequisite answers change. Provide a multi-select UI limited to existing question `tkey` values to avoid validation failures. F:components/formBuilder/validators.ts†L61-L114 F:components/formBuilder/service.ts†L650-L652

Visibility rules and inter-question dependencies

Conditional display logic relies on the `visibleIf` array. The service resolves each condition to an actual question instance ID so the UI can safely edit rules even when authors refer to template keys. F:components/formBuilder/validators.ts†L23-L27 F:components/formBuilder/service.ts†L1684-L1694 F:components/formBuilder/service.ts†L1684-L1737 F:components/formBuilder/service.ts†L1689-L1699 F:components/formBuilder/service.ts†L1690-L1747

- Acceptable operators are enforced via Joi (`eq`, `neq`, `gt`, `gte`, `lt`, `lte`, `in`, `not_in`, `exists`, `not_exists`). Use dropdowns to restrict the selection. F:components/formBuilder/validators.ts†L23-L27
- Authors can reference either numeric question IDs or template `tkey` values. The service cross-checks the references within the same form and throws `VISIBLE_IF_INVALID_REFERENCE` if anything is missing. Present searchable dropdowns rather than free-text inputs to avoid typos. F:components/formBuilder/service.ts†L1690-L1732 F:components/formBuilder/service.ts†L1733-L1788
- When the UI reorders or deletes questions, recalc the visibility graph. If a referenced question is archived, prompt the author to update or remove dependent rules before publishing.

Retrieval helpers for detail panels

Besides the list endpoints, the service exposes `GET /api/v1/sections/{sectionId}` and `GET /api/v1/questions/{questionId}` to hydrate detail drawers. Both endpoints reuse the list response mappers (`toSectionResponse`, `toQuestionResponse`), so caching them in your state store guarantees consistent shapes between list and detail views. F:components/formBuilder/formBuilder.route.ts†L35-L183 F:components/formBuilder/service.ts†L1189-L1238 F:components/formBuilder/service.ts†L1220-L1239 Use these when deep-linking to a specific question from notifications or audit logs.

Bulk import & export

- **Import CSV** – `POST /api/v1/forms/{formId}/questions/import` with multipart form-data key `file`. Expected headers: `section_name,`

`section_order, tkey, label, helper_text, answer_type, required, question_order, validation_json, visible_if_json, options_json, default_value, storage_json`. Missing columns are ignored, but malformed values surface per-row errors in the response. F:components/formBuilder/formBuilder.route.ts†L18 L215 F:components/formBuilder/validators.ts†L116-L130 Example snippet:

```
section_name,section_order,tkey,label,answer_type,required,options_json,validation_json
Applicant details,1,applicant_full_name,Applicant full name,text,true,,
Applicant details,1,marital_status,Marital status,dropdown,true,"[{""tkey"":""single"",
```

The importer will create sections when missing, reuse templates by `tkey`, and call `updateQuestion` if the form already contains that template. The response reports `{ inserted, updated, errors: [{ row, message }] }`; render these results in an import review modal so admins can fix rows directly. F:components/formBuilder/service.ts†L2020-L2080

- **Export** – `GET /api/v1/forms/{formId}/questions/export?format=csv|json`. Defaults to CSV; JSON returns an array identical to the builder list payload. CSV rows serialise nested fields (`validation_json`, `visible_if_json`, `options_json`, `storage_json`) for round-trip editing. Offer buttons for both formats and link back to this import guide so users understand the expected column names. F:components/formBuilder/formBuilder.route.ts†L217-L240 F:components/formBuilder/service.ts†L2124-L2182

Publishing and version control

- **Publish form** – `POST /api/v1/forms/{formId}/publish` (optional body `{ "notes": "message" }`). Publishes all non-archived sections/questions after validating that every enumerated question has options and that all questions belong to active sections. Publishing snapshots the form into `form_versions`, updates section/question statuses to `published`, logs an audit entry, and returns the new version number and timestamp. F:components/formBuilder/formBuilder.route.ts†L242-L260 F:components/formBuilder/service.ts†L2185-L2338
- **List versions** – `GET /api/v1/forms/{formId}/versions` returns versions newest first; fetch a specific version with `GET /api/v1/forms/{formId}/versions/{versionNumber}` (404 if missing). F:components/formBuilder/formBuilder.route.ts†L262-L305 F:components/formBuilder/service.ts†L2404-L2418
- **Rollback** – `POST /api/v1/forms/{formId}/rollback/{versionNumber}` clones the requested snapshot into a new version and logs the rollback source. The front end should warn users that rollback creates a new version rather than mutating the historical one. F:components/formBuilder/formBuilder.route.ts†L307-L325 F:components/formBuilder/service.ts†L2420-L2467
- **Auto-publish behavior** – When a form already has at least one published version, creating or updating sections/questions automatically calls `publishForm` so the live snapshot stays fresh. Pro-

vide UI feedback (e.g., toast, activity indicator) and refresh live previews accordingly. F:components/formBuilder/service.ts†L1300-L1308 F:components/formBuilder/service.ts†L1958-L1965 F:components/formBuilder/service.ts†L2342-L2353

Public delivery endpoints

No authentication required:

- `GET /api/v1/forms/{formSlug}/live` – Returns the most recent published snapshot for a slug, or 404 if no version exists. Use this to render customer-facing questionnaires without builder privileges.
- `GET /api/v1/forms/{formSlug}/versions/{versionNumber}` – Returns a specific published snapshot by slug + version (404 if missing). F:components/formBuilder/formBuilder.route.ts†L327-L369 F:components/formBuilder/service.ts†L2470-L2495

Both endpoints emit the `payloadJson` structure produced by `composeSnapshot`: ordered sections containing ordered questions with template-resolved options and visibility metadata. This structure is suitable for directly rendering read-only experiences. F:components/formBuilder/service.ts†L2360-L2401

Auditing

`GET /api/v1/forms/{formId}/audit?entity=form_section&entityId=37&page=1&pageSize=20`

- Returns paginated audit log rows filtered by entity type, entity ID, and/or form. Behind the scenes the service inspects `beforeJson`/`afterJson` to include changes related to the requested form even when the entity is a section/question. Use this to power activity timelines in the UI. F:components/formBuilder/formBuilder.route.ts†L371-L400 F:components/formBuilder/service.ts†L2498-L2547

Domain-specific error codes & UX recommendations

Surface backend error codes with tailored remediation guidance so authors understand how to resolve issues without reading server logs. F:components/formBuilder/service.ts†L700-L742 F:components/formBuilder/service.ts†L1013-L1186 F:components/formBuilder/service.ts†L1689-L1988 F:components/formBuilder/service.ts†L2320-L2336

Code	When it appears	Suggested UI response
`FORM_NOT_FOUND` / `SECTION_NOT_FOUND`	Editing a form/section/question that was archived or deleted in another tab.	Show a warning banner and redirect back to the form list after refreshing state. F:components/formBuilder/service.ts†L700-L741 F:components/formBuilder/service.ts†L1329-L1405
`SECTION_ARCHIVED` / `QUESTION_ARCHIVED`	Attempting to update an archived entity (often due to race conditions).	Refresh the entity, disable inputs, and explain that archived items must be restored or recreated. F:components/formBuilder/service.ts†L1395-L1444 F:components/formBuilder/service.ts†L1972-L2018
`QUESTION_TEMPLATE_REGION_UNAVAILABLE`	Requested template that is not enabled for the form's region.	Prompt the author to clone the template for their region or switch regions before proceeding. F:components/formBuilder/service.ts†L101 L1128
`OPTIONS_REQUIRED` / `OPTIONS_NOT_ALLOWED`	Dropdown/radio questions missing options, or non-enumerated questions providing options.	Highlight the option editor and display helper text clarifying when options are mandatory. F:components/formBuilder/service.ts†L722 L743
`VISIBLE_IF_INVALID_REFERENCE`	Visibility condition references a question that doesn't exist in the current form.	Auto-remove invalid rows from the builder UI and ask the author to pick a different trigger question. F:components/formBuilder/service.ts†L1690-L1774
`QUESTION_OPTIONS_MISSING` / `QUESTION_OPTION_DUPLICATE`	Publishing detected dropdown/radio questions without options or with duplicate `tkey` values.	Focus the problematic question in the review panel and require authors to fix options before publishing. F:components/formBuilder/service.ts†L2325 L2336

Code	When it appears	Suggested UI response
`ETAG_MISMATCH` (HTTP 409)	Optimistic concurrency failure during section/question update.	Show a modal asking the author to refresh and replay unsaved changes (keep the diff client-side). F:components/formBuilder/service.ts†L1346-L1375 F:components/formBuilder/service.ts†L1888-L1944

End-to-end authoring workflow example

The sequence below illustrates how a builder session should orchestrate API calls, UI state, and optimistic updates when launching a new questionnaire. Each step links back to the relevant endpoints documented above.

1. **Create the form shell** – Call `POST /api/v1/forms` with name, region, and type. On success, store the returned `id`/`slug` and route the user to the form designer. F:components/form/repository/FormRepository.ts†L16-L35
2. **Seed sections** – For each planned chapter, call `POST /api/v1/forms/{formId}/sections`. Immediately insert the returned payload into local state and render editable cards; note the generated slug/order for drag handles. F:components/formBuilder/service.ts†L1241-L1308
3. **Add questions** – Use the question catalog picker to choose templates (`questionTemplateId`) or allow inline definitions. After `POST /api/v1/forms/{formId}/questions`, merge the response into the section tree so the UI displays resolved labels/options without extra requests. F:components/formBuilder/service.ts†L1680-L1758
4. **Configure visibility and dependencies** – When authors add rules, validate the payload client-side against the allowed operators, then submit via `PATCH /api/v1/questions/{questionId}` with `If-Match`. On `ETAG_MISMATCH`, refresh and replay unsaved changes. F:components/formBuilder/service.ts†L1888-L1944
5. **Publish** – Once every section/question is ready, call `POST /api/v1/forms/{formId}/publish`. Inspect the response version number and append it to the publish history timeline in the UI. F:components/formBuilder/service.ts†L2185-L2338
6. **Verify public payload** – Optionally fetch `GET /api/v1/forms/{slug}/live` to preview the customer-facing structure using the same renderer deployed in production. F:components/formBuilder/service.ts†L2360-L2401
7. **Monitor activity** – Load the audit feed via `GET /api/v1/forms/{formId}/audit` to show who edited what, and surface import/publish events in a chronological sidebar. F:components/formBuilder/service.ts†L2498-L2547

Implementation checklist for the front end

1. **State normalization** – Mirror the backend response structure (sections keyed by id, questions grouped per section). Persist `etag`, `version`, and `status` metadata for concurrency-aware editing. F:components/formBuilder/service.ts†L520-L744
2. **Error handling** – Surface validation errors returned by the service (e.g., `OPTIONS_REQUIRED`, `SECTION_INVALID`, CSV row errors). Many originate from Joi validation or explicit guard clauses in `createQuestion/publishForm`.
3. **Option editors** – Enforce unique `tkey` values within a question, maintain ordering, and restrict options to enumerated answer types to avoid server-side rejections. F:components/formBuilder/validators.ts†L38-L83 F:components/formBuilder/service.ts†L1665-L1674 F:components/formBuilder/service.ts†L2303-L2338
4. **Visibility rules** – Provide pickers for existing questions by `tkey` so the `visibleIf` payload matches `{ questionId, operator, value }`. Operators must be from the allowed set (`eq`, `neq`, `gt`, etc.). F:components/formBuilder/validators.ts†L23-L27
5. **Dynamic options** – When `optionsApi` is set, configure the UI to fetch options at runtime and refresh when dependencies listed in `dependsOn` change. F:components/formBuilder/service.ts†L1689-L1712 F:components/formBuilder/service.ts†L2386-L2397
6. **Publishing UX** – Show publish status, version numbers, and audit trail integration so builders understand when auto-publish events fire after edits. F:components/formBuilder/service.ts†L2185-L2353 F:components/formBuilder/service.ts†L2498-L2547

By aligning the front-end workflow with these contract details, the form builder UI can provide reliable authoring, preview, and delivery experiences that stay consistent with the backend's validation, versioning, and auditing guarantees.