**JIL File Generator - Project Documentation**

**1. Overview**

**Info Panel: What is the JIL File Generator?**

The JIL File Generator is a web-based tool designed to help users create JIL (Job Information Language) files for job scheduling. It replaces the complex, error-prone manual process of writing JIL scripts with a user-friendly, dynamic form. Users fill in job details through a guided interface, and the application automatically generates a valid and complete JIL file.

**1.1. Purpose & Key Benefits**

This application serves to streamline and standardize the creation of job scheduling configurations.

* **Simplifies Complexity:** Users do not need to know the specific syntax of JIL. The form guides them through all required parameters.
* **Reduces Errors:** Built-in validation, conditional logic, and read-only fields ensure that the generated files are accurate and adhere to standards.
* **Increases Efficiency:** Drastically reduces the time required to create and configure new jobs.
* **Highly Maintainable:** The form's questions, structure, and logic are defined in external JSON files, allowing for easy updates without changing the application's source code.

**2. System Architecture & Core Concepts**

The JIL File Generator is built on a modular and dynamic architecture. The user interface is not hard-coded but is rendered at runtime based on a series of JSON configuration files.

* **Dynamic Form Rendering:** The core of the application is its ability to build Angular forms on-the-fly from JSON templates.
* **Question Bank:** A centralized question-bank.json file acts as a single source of truth for every possible field (question) that can appear in the form. This includes the field's ID, label, type, validators, and default value.
* **Subform Architecture:** The main form is composed of smaller, reusable components called "subforms" (e.g., top, box, cmd, fw). Each subform has its own JSON template that defines which questions from the bank it contains and in what order.
* **Function-Job Mapping:** A dedicated function-job-mapping.json file decouples the business logic of job functions from the code, defining which subforms are available for specific job types.

[💡 **Visualization Suggestion: High-Level Architecture Diagram**]

Insert a diagram here illustrating the main components and their interaction.

**Example Diagram Flow:**  
User Interface (Angular) → interacts with → Dynamic Form Service → which reads from → [JSON Config Files: question-bank.json, subform-templates.json, function-job-mapping.json] → to generate → Angular FormGroups/FormArrays → which produces the final → JIL File

**3. How It Works: A User's Guide**

This section describes the step-by-step process a user follows to create a JIL file.

**Step 1: Configure the Main Job**

The user starts with the "Top" subform, which is always present. Here, they define the primary characteristics of the job, such as the job function, type, and the target environments (DEV, PROD, UAT, etc.). All fields in this initial section are editable.

[🖼️ **Visualization Suggestion: Screenshot of the Top Subform**]

Insert a screenshot here showing the initial form view with the "Top" subform and its editable fields.

**Step 2: Add Additional Job Sections (Subforms)**

Based on the job's requirements, the user can add more sections (subforms) like box, cmd, fw, or cfw using checkboxes or "Add" buttons.

**Step 3: Fill in Section-Specific Details**

As subforms are added, new sections appear in the form. The user fills out the fields in these new sections.

* **Smart Form Behavior:** The form intelligently shows or hides fields based on previous selections.
  + **Environment Logic:** If the user selected only "DEV" and "UAT" in Step 1, only the fields relevant to DEV and UAT will appear in the subsequent subforms.
  + **Conditional Logic:** For example, selecting "Weekly" as the Load Frequency will display a Days of Week checkbox group, while selecting "Custom" will display a Run Calendar text field.
* **Read-Only Logic:** Key fields like Job Function and Job Type, which were set in the Top subform, are displayed in the additional subforms but are read-only to ensure consistency.

[🖼️ **Visualization Suggestion: Screenshot of Conditional Fields**]

Insert a screenshot here showing how the form changes when a user selects a different frequency, highlighting the fields that appear or disappear.

**Step 4: Preview and Download**

Once all the necessary information is filled in, the user can click a "Preview" button to see the generated JIL code. After verifying, they can download the final .jil file.

**4. Technical Deep Dive**

This section provides technical details for developers working on the project.

**4.1. Key File Structure**

|  |  |
| --- | --- |
| File Path | Description |
| src/assets/json/question-bank.json | Defines all possible questions/fields, their types (text, dropdown, environments-group), and validators. |
| src/assets/json/box-subform.json | Template for the 'box' subform. Contains an array of questionIds from the question bank. |
| src/assets/json/[...]-subform.json | Other subform templates (e.g., cmd-subform.json, fw-subform.json). |
| src/assets/json/function-job-mapping.json | Maps job functions to specific job types and the subforms that can be used with them. |
| src/app/services/dynamic-form-builder.service.ts | The core service that reads the JSON files and programmatically builds the Angular FormGroup. |
| src/app/components/dynamic-form-viewer/... | The Angular component responsible for rendering the form controls based on the generated FormGroup. |
| src/app/pages/dynamic-form-page/... | The main page component that manages the collection of subforms, navigation, and state. |

**4.2. Form Generation Logic**

1. The dynamic-form-page component determines which subforms to display.
2. For each subform, it calls the dynamic-form-builder.service.
3. The service reads the subform's JSON template (e.g., box-subform.json) to get the list of questionIds.
4. For each questionId, it looks up the full question definition in question-bank.json.
5. Based on the question's type and validators, it creates an appropriate FormControl. Repeatable sections like conditions are created as a FormArray.
6. These FormControls are grouped into a FormGroup representing the subform.
7. The final FormGroup is passed to the dynamic-form-viewer component, which uses an \*ngSwitch or similar directive to render the correct HTML input for each control.

[💻 **Code Snippet Suggestion: Form Builder Service**]

Insert a simplified code block from dynamic-form-builder.service.ts showing the core loop that iterates through question IDs and creates FormControls.

**4.3. Conditional Field Logic (\*ngIf)**

The visibility of fields like days\_of\_week and run\_calendar is managed within the component's TypeScript and template.

1. **Component Logic:** The component subscribes to the valueChanges observable of the frequency form control.
2. **Template Logic:** The \*ngIf directive is used in the HTML template to check the current value of the frequency control and display the relevant field.
3. **Value Clearing:** When the frequency changes, the subscription callback clears the value of the field that is now hidden to prevent orphaned data.

[💻 **Code Snippet Suggestion: Conditional Logic**]

Show a paired code snippet:

**5. Configuration & Customization**

The application is designed to be easily configured without code changes.

**5.1. How to Add a New Question**

1. **Open question-bank.json**.
2. **Add a new JSON object** to the main array. Define its id, label, type, placeholder, and any validators.

[💻 **Code Snippet Suggestion: question-bank.json**]

Display an example of a new question object.

{  
 "id": "new\_custom\_field",  
 "label": "New Custom Field",  
 "type": "text",  
 "placeholder": "Enter value here...",  
 "validators": [  
 { "name": "required", "message": "This field is mandatory." },  
 { "name": "maxLength", "value": 50, "message": "Value cannot exceed 50 characters." }  
 ]  
}

1. **Open the relevant subform template** (e.g., box-subform.json).
2. **Add the new id** ("new\_custom\_field") to the questionIds array in the desired position.

**5.2. How to Edit a Subform**

1. **Open the JSON file** for the subform you want to change (e.g., cmd-subform.json).
2. **Modify the questionIds array**:
   * To **add** a field, insert its ID.
   * To **remove** a field, delete its ID.
   * To **reorder** fields, change the order of the IDs in the array.

**5.3. How to Update Job Functions**

1. **Open function-job-mapping.json**.
2. Modify the JSON to add or edit the mappings between job functions, job types, and their associated subforms.

[💻 **Code Snippet Suggestion: function-job-mapping.json**]

Display an example snippet from this file to illustrate the structure.

{  
 "MyNewFunction": {  
 "job\_type": "BOX",  
 "subform\_types": ["box", "cmd"]  
 }  
}

**6. Project Setup for Local Development**

**6.1. Prerequisites**

* Node.js (v18.x or later recommended)
* NPM (v9.x or later recommended)
* Angular CLI (npm install -g @angular/cli)

**6.2. Installation & Running**

1. **Clone the repository:**

git clone <your-repository-url>

1. **Navigate to the project directory:**

cd JIL\_File\_Generator

1. **Install dependencies:**

npm install

1. **Start the frontend development server:**

ng serve

1. **Access the application** in your browser at http://localhost:4200.

**7. Support**

For questions, bug reports, or feature requests, please open an issue in the project's repository. Please provide detailed information, including steps to reproduce, screenshots, and expected vs. actual behavior.