Jira `WORKFLOW.md` Generation Prompt

Create a production-ready `WORKFLOW.md` in the repository root for running OpenAI Symphony against a Jira Cloud project.

Use the public Jira workflow as the authoritative structural starting point:

https://github.com/openai/symphony/blob/main/elixir/WORKFLOW.jira.md

Do not copy its example-specific Jira site, project, statuses, paths, or Git strategy. Discover and validate the target configuration before generating the file.

Phase 1: Required environment variables

Before inspecting the repository, querying Jira, or generating `WORKFLOW.md`, verify that all four of these environment variables exist and contain non-empty values:

- `JIRA_BASE_URL`
  - The Jira Cloud site origin.
  - Example: `https://example.atlassian.net`
  - It must be an HTTPS URL without a path, query, or fragment.
- `JIRA_PROJECT_KEY`
  - The Jira project/space key.
  - Example: `ENG`
- `JIRA_EMAIL`
  - The email address associated with the Jira API token.
- `JIRA_API_TOKEN`
  - A Jira Cloud API token with permission to browse the project and inspect its issue types, statuses, and workflow configuration.

Check for these variables without printing or otherwise exposing their secret values.

If any variable is missing or blank:

1. Stop immediately.
2. Do not query Jira.
3. Do not create or modify `WORKFLOW.md`.
4. Tell the user exactly which variables are missing.
5. Ask the user to configure them in the environment and confirm when ready.
6. Never ask the user to paste `JIRA_API_TOKEN` into the generated file, source code, chat output, or another repository-owned file.

Example shell configuration:

```bash
export JIRA_BASE_URL="https://example.atlassian.net"
export JIRA_PROJECT_KEY="ENG"
export JIRA_EMAIL="user@example.com"
export JIRA_API_TOKEN="<secret>"
```

The final `WORKFLOW.md` must refer to these settings through environment-variable references:

```yaml
tracker:
  kind: jira
  provider:
    base_url: "$JIRA_BASE_URL"
    project_key: "$JIRA_PROJECT_KEY"
    email: "$JIRA_EMAIL"
```

```
    api_token: "$JIRA_API_TOKEN"
```

Do not substitute literal credentials into `WORKFLOW.md`.

Phase 2: Ask for repository configuration

After confirming that the required Jira environment variables exist, ask the user for the repository-specific configuration.

At minimum, ask for:

1. Source repository
   - A local source repository path supplied as `PROJECT_REPO`; or
   - A Git clone URL.
2. Workspace root
   - Example: `~/symphony-workspaces`.
3. Default branch
   - Default: `main`.
4. Completion strategy
   - Directly merge the issue branch into the default branch; or
   - Push an issue branch and create/use a pull request.
5. Git remote
   - Default: `origin`.
6. Git author name and email for automated commits.
7. Required setup commands
   - Dependency installation, environment setup, code generation, or similar.
8. Required validation commands
   - Formatting, linting, static analysis, targeted tests, and full tests.
9. Required Jira labels
   - Default: `[]`.
10. Any additional instructions or guardrails that every unattended agent must follow.
11. Whether network access is required during agent turns.
    - Default: `true` when dependency managers, Jira, or remote Git operations are needed.

Do not require the user to provide commands that can be reliably discovered from the repository. Inspect files such as:

- `README.md`
- `AGENTS.md`
- `Makefile`
- package-manager manifests
- build-tool configuration
- CI workflow files
- repository-local agent skills
- existing workflow files

Present the discovered commands to the user and ask them to confirm or correct them.

If the user does not know the repository path, workspace root, or clone source, stop and ask for the missing information rather than inventing it.

Phase 3: Offer sensible Symphony defaults

Ask whether the user wants to customize the Symphony and Codex runtime settings. If not, use these defaults from the existing Jira workflow pattern:

```yaml
polling:
  interval_ms: 5000
```

````
agent:
  max_concurrent_agents: 10
  max_turns: 20

codex:
  command: codex --config shell_environment_policy.inherit=all --config
'model="gpt-5.5"' --config model_reasoning_effort=xhigh app-server
  approval_policy: on-request
  thread_sandbox: danger-full-access
  turn_sandbox_policy:
    type: workspaceWrite
    networkAccess: true
```

Explain the defaults briefly:

- `max_concurrent_agents: 10`
  - Runs at most ten issue agents concurrently.
- `max_turns: 20`
  - Allows up to twenty consecutive turns while an issue remains active.
- `polling.interval_ms: 5000`
  - Polls Jira every five seconds.
- `codex.command`
  - Starts Codex in app-server mode using the command pattern from the existing
workflow.
- `networkAccess: true`
  - Allows package managers, Jira operations, and remote Git operations when
needed.

Let the user override any of these values.

Do not silently change the model, reasoning effort, approval policy, sandbox policy, concurrency, or turn limit. Use the defaults only when the user accepts them or says they have no preference.

Warn the user that `danger-full-access` and `on-request` are permissive settings intended for trusted environments. Offer a safer sandbox configuration if requested, but ensure the resulting policy still supports the repository's required setup, validation, Jira, and Git operations.

## Phase 4: Validate Jira connectivity

Use the required environment variables to authenticate to the public Jira Cloud REST API.

Authentication must use Basic authentication constructed from:

- Username: `JIRA_EMAIL`
- Password: `JIRA_API_TOKEN`

Do not log the generated Authorization header or token.

Start with a lightweight authenticated request, such as:

```http
GET /rest/api/3/myself
```

Then validate the target project:

```http
GET /rest/api/3/project/{JIRA_PROJECT_KEY}
```
````
````

If authentication fails:

1. Stop.
2. Report the HTTP status and a sanitized error.
3. Ask the user to verify `JIRA_EMAIL` and `JIRA_API_TOKEN`.
4. Do not create `WORKFLOW.md`.

If the project cannot be accessed:

1. Stop.
2. Report the project key and sanitized API error.
3. Ask the user to verify `JIRA_BASE_URL`, `JIRA_PROJECT_KEY`, and project permissions.
4. Do not create `WORKFLOW.md`.

A `401` normally indicates invalid or missing authentication. A `403` normally indicates insufficient permission. A `404` may indicate an incorrect project key, incorrect site, inaccessible project, or unavailable endpoint.

Phase 5: Discover issue types and workflows

Retrieve the Jira workflow and status information for every issue type in the configured project.

At minimum, query the Jira project-status endpoint:

```http
GET /rest/api/3/project/{JIRA_PROJECT_KEY}/statuses
```

This response associates the project's issue types with their available statuses.

When the authenticated Jira APIs and permissions support it, also inspect the project's workflow scheme and workflow details using the applicable public Jira Cloud REST API endpoints. Use those results to understand which workflows apply to which issue types.

The discovery must account for all issue/work types returned for the project, including, when present:

- Epic
- Story
- Task
- Bug
- Sub-task
- Service-management request types
- Custom issue types

Do not assume every issue type uses the same workflow.

Build a normalized table containing:

Issue type	Workflow, if available	Status	Status category	Proposed Symphony role

Use Jira's status categories as evidence, but do not rely on them alone:

- `new` usually indicates queued or not started.
- `indeterminate` usually indicates active or review work.
- `done` usually indicates a terminal state.

Status categories do not reliably distinguish implementation from review. Use

status names, workflow transitions, and issue-type mappings to propose the final classification.

Identify:

1. Queued statuses
   - Eligible to be picked up and transitioned into active work.
2. Active statuses
   - Work is currently being implemented or corrected.
3. Review statuses
   - Implementation is complete and awaiting review, verification, merge, or acceptance.
4. Terminal statuses
   - No further agent work should run.
5. Out-of-scope statuses
   - Statuses that Symphony must not automatically process.
6. Blocked or paused statuses
   - Decide with the user whether these remain active, are excluded, or require special handling.

Check whether all issue types share a compatible status model.

If workflows differ significantly by issue type, do not collapse them into one unsafe status list. Instead, explain the differences and propose one of these options:

- Limit Symphony to issue types sharing a compatible workflow.
- Use required labels to scope eligible work.
- Select a safe union of statuses only when the same status meaning is consistent across issue types.
- Update the Jira workflow so relevant issue types share compatible status names.
- Create separate Symphony workflow configurations for incompatible workflow groups.

Phase 6: Obtain end-user approval of Jira statuses

Before generating or modifying `WORKFLOW.md`, present the Jira discovery results to the user.

Include:

1. Every discovered issue type.
2. Its associated statuses.
3. Its workflow name when available.
4. The proposed classification of each status.
5. The exact proposed Symphony configuration:

```yaml
tracker:
  active_states:
    - <queued-or-active-status>
    - <active-status>
    - <review-status>
  terminal_states:
    - <terminal-status>
```

Also identify which configured active states represent:

- queued work,
- implementation,
- rework,
- review.

The public Jira template places queued, implementation, and review statuses in `active_states` because Symphony must continue polling and reconciling issues in all those states. Only genuinely completed or canceled statuses should normally be terminal.

Ask the user to explicitly approve or correct:

- queued statuses,
- active implementation statuses,
- review statuses,
- terminal statuses,
- blocked/paused behavior,
- issue types eligible for automation,
- required labels, if any.

Do not create `WORKFLOW.md` until the user validates this mapping.

Preserve status names exactly as returned by Jira, including capitalization, spacing, and punctuation. Jira polling uses exact status matching.

Phase 7: Inspect the repository

After the Jira mapping has been approved, inspect the repository and determine:

- default branch,
- Git remote configuration,
- setup/bootstrap commands,
- formatting command,
- lint/static-analysis command,
- targeted test commands,
- full test command,
- build command,
- package-manager requirements,
- repository-local instructions,
- available Jira and SCM tools,
- branch and pull-request conventions.

Do not invent commands or conventions.

Present any uncertain or conflicting findings to the user for validation before generating the workflow.

Phase 8: Generate `WORKFLOW.md`

After the user has validated the Jira statuses and repository configuration, generate `WORKFLOW.md` at the repository root.

Preserve the Symphony file structure:

1. YAML front matter containing runtime configuration.
2. A Markdown body containing the Codex session prompt.

The front matter must include:

```yaml
---
tracker:
  kind: jira
  provider:
    base_url: "$JIRA_BASE_URL"
    project_key: "$JIRA_PROJECT_KEY"
    email: "$JIRA_EMAIL"
    api_token: "$JIRA_API_TOKEN"
```

````
  required_labels: []
  active_states:
    - <USER_APPROVED_EXACT_STATUS>
  terminal_states:
    - <USER_APPROVED_EXACT_STATUS>

polling:
  interval_ms: 5000

workspace:
  root: <USER_APPROVED_WORKSPACE_ROOT>

hooks:
  after_create: |
    <REPOSITORY_BOOTSTRAP_AND_BRANCH_SETUP>
  after_run: |
    <USER_APPROVED_COMPLETION_STRATEGY>
  before_remove: |
    set -euo pipefail

agent:
  max_concurrent_agents: 10
  max_turns: 20

codex:
  command: codex --config shell_environment_policy.inherit=all --config
'model="gpt-5.5"' --config model_reasoning_effort=xhigh app-server
  approval_policy: on-request
  thread_sandbox: danger-full-access
  turn_sandbox_policy:
    type: workspaceWrite
    networkAccess: true
---
```

Replace defaults with user-approved overrides where applicable.

If the source repository is a local path, prefer an environment reference such
as:

```bash
: "${PROJECT_REPO:?PROJECT_REPO must be set by the launcher}"
git clone "$PROJECT_REPO" .
```

If the source repository is a remote URL, use the validated clone URL and
authentication approach. Do not embed access tokens or credentials.

## Phase 9: Preserve required runtime template variables

Preserve these Symphony/Liquid variables exactly:

- `{{ issue.identifier }}`
- `{{ issue.title }}`
- `{{ issue.state }}`
- `{{ issue.labels }}`
- `{{ issue.url }}`
- `{{ issue.description }}`
- `{{ attempt }}`

Preserve valid Liquid control flow:

```liquid
{% if attempt %}
````

````
Follow-up context:

- This is follow-up attempt #{{ attempt }}.
- Resume from the current workspace state.
{% endif %}
```

And:

```liquid
{% if issue.description %}
{{ issue.description }}
{% else %}
No description provided.
{% endif %}
```

Do not resolve these variables while generating the file. Symphony resolves them for each issue at runtime.

## Phase 10: Agent workflow requirements

The Markdown prompt must tell the unattended agent to:

1. Fetch the Jira issue by its explicit issue key.
2. Inspect its current status before performing work.
3. Route behavior according to the user-approved status map.
4. Assign the issue to the executing Jira user when possible.
5. Transition queued work into the approved implementation status before coding.
6. Maintain exactly one persistent `## Codex Workpad` Jira comment.
7. Inspect repository instructions and current Git state.
8. Create and maintain a concrete implementation plan.
9. Record acceptance criteria derived from the Jira issue.
10. Implement the requested change.
11. Run the repository's validated targeted and full checks.
12. Record validation commands and outcomes in the workpad.
13. Follow the approved direct-merge or pull-request strategy.
14. Move the issue into the approved review status only after the completion bar is satisfied.
15. Move the issue into a terminal status only when the approved workflow says the work is complete.
16. Stop without modifying anything if the issue is already terminal.
17. Avoid asking a human to execute routine steps.
18. Report true external blockers in the workpad with:
    - blocker,
    - impact,
    - attempted alternatives,
    - exact unblock action.

## Phase 11: Jira-access fallback behavior

Keep the public Jira template's access fallback behavior:

1. Prefer Symphony's injected `jira_rest` tool.
2. Otherwise use an available Jira MCP client.
3. Otherwise use direct Jira REST calls based on the configured host-side credentials.
4. Treat Jira access as blocked only after all available options have been attempted.
5. Never expose Jira credentials to logs, comments, source files, commits, or final output.

Do not require a human to perform routine Jira operations when an authenticated client is available.
````
````

Phase 12: Workpad template

Include this exact workpad structure in the generated Markdown prompt:

````markdown
## Codex Workpad

```text
<hostname>:<abs-path>@<short-sha>
```

### Plan

- [ ] 1\. Parent task
  - [ ] 1.1 Child task
  - [ ] 1.2 Child task

### Acceptance Criteria

- [ ] Criterion 1
- [ ] Criterion 2

### Validation

- [ ] targeted tests: `<command>`
- [ ] required full checks: `<command>`

### Notes

- <timestamped progress note>

### Confusions

- <only include when something was confusing during execution>
````

Phase 13: Consistency and safety validation

Before writing the file, verify:

- All four required Jira environment variables were detected.
- Jira authentication succeeded.
- The target project was retrieved successfully.
- All project issue types were inspected.
- The user explicitly approved the status mapping.
- Status names exactly match Jira.
- The selected Git strategy is internally consistent.
- Direct merge and pull-request instructions are not mixed.
- Repository commands were discovered rather than invented.
- No credentials or authorization headers appear in the file.
- Hooks operate only inside the issue workspace or explicitly approved repository path.
- The selected sandbox and network policies support the required commands.
- No references remain to:
  - `plai2.atlassian.net`,
  - project `PD`,
  - `PD-123`,
  - PD-specific statuses,
  - example workspace paths,
  - example repository locations,
  - unapproved Git identities.

The only unresolved runtime placeholders allowed in the final file are:

- Symphony/Liquid issue variables;
- `$JIRA_BASE_URL`;
- `$JIRA_PROJECT_KEY`;
- `$JIRA_EMAIL`;
- `$JIRA_API_TOKEN`;
- `$PROJECT_REPO`, if using a local source repository;
- other explicitly approved environment-backed configuration.

Phase 14: Final response

After creating `WORKFLOW.md`, return:

1. The path of the created file.
2. The approved issue types and status mapping.
3. The effective Symphony settings:
   - polling interval,
   - concurrent agents,
   - maximum turns,
   - Codex command,
   - sandbox and approval policies.
4. The repository setup and validation commands included.
5. The selected Git completion strategy.
6. The required environment variables, without their values.
7. Any assumptions or limitations.
8. Confirmation that the YAML front matter was parsed or otherwise validated.
9. Confirmation that no Jira credentials were written into the file.

Required environment variables:

- `JIRA_BASE_URL`
- `JIRA_PROJECT_KEY`
- `JIRA_EMAIL`
- `JIRA_API_TOKEN`
- `PROJECT_REPO` when the workspace clones from a local source repository