You will be developing a GitHub-powered backend tool that supports a ChatGPT app for reviewing pull requests.

This app enables ChatGPT to:

- Retrieve a list of pull requests based on explicit user intent
- Allow the user to select **one pull request** (v1 constraint)
- Retrieve **full context** (metadata, files, diffs) for a selected/specified PR
- Generate review comments (ChatGPT responsibility)
- Post user-approved review comments back to GitHub (backend responsibility)

The backend does not perform code review.
ChatGPT is responsible for all analysis and comment generation.
Your backend is responsible for:

1. Retrieving accurate GitHub data and PR context
2. Executing GitHub write operations when explicitly instructed

The app operates inside the **ChatGPT apps ecosystem**, where ChatGPT may retry requests, repeat calls, or call endpoints out of order. This behavior is expected and must be handled safely.

Critical Operational Requirements (Non-Negotiable)

- ChatGPT may call the same endpoint **multiple times with identical parameters**
- ChatGPT retries must not cause:
    - Duplicate GitHub API calls
    - Duplicate review comments
    - Corrupted state
- The developer **must implement caching for read operations**
- The developer **must implement idempotency for write operations**
- All long-running operations must provide visible feedback to the user

Failure to account for this is a correctness issue.

Scope Constraints (v1)

- Only **one pull request** may be reviewed at a time
- Batch PR review is out of scope
- ChatGPT generates all review comments

- Posting comments requires **explicit user confirmation**
- Permission and access errors must be surfaced clearly
- Formatting and styling are not a priority; correctness and clarity are

---

Reference Links

- ChatGPT Apps SDK: https://openai.com/index/introducing-apps-in-chatgpt/
- GitHub REST API: https://docs.github.com/en/rest
- GitHub GraphQL API: https://docs.github.com/en/graphql

Core Design Principles

1. **ChatGPT is the reviewer**
   The backend never analyzes code or generates review comments.
2. **Separate list vs context retrieval**
   One endpoint lists candidate PRs; a separate endpoint retrieves full context for a specific PR.
3. **Two entry paths to PR context**
   PR context retrieval must work whether:
   - The user selects a PR from the UI list, or
   - The user specifies a PR conversationally (e.g., “Review pr-123”)
4. **Explicit intent for list retrieval**
   Listing PRs is used when the user’s PR target is implicit (“my latest PR”) and they need choices.
5. **User transparency**
   Progress, success, and errors must always be visible.

Endpoint Functionality

1. Get Pull Requests List

(GET /pull-requests)

Used when the user asks for “my PRs to review”, “my most recent PR”, or otherwise does not specify a concrete PR identifier.

ChatGPT may say:

- “What PRs do I need to review?”
- “Show my most recent PR.”
- “List PRs I authored.”

Parameters

- `role` (required)
  Defines the relationship between the queried user and the PRs:
  - `reviewer` – PRs where the user is a requested reviewer
  - `author` – PRs authored by the user
- `target_user` (optional)
  GitHub username to query PRs for.
  - Omitted → authenticated user
  - Provided → specified user (subject to access)

Behavior

- Fetch open pull requests matching:
  - `role`
  - `target_user` or authenticated user
- Exclude merged and closed PRs
- Sort by most recently updated
- Limit to 10 results

Efficiency Requirements

- Results must be cached for a short, reasonable duration
- Repeated calls with identical parameters must reuse cached results
- GitHub rate limits must be respected

Response (conceptual)

Each PR should include enough information for selection and context lookup:

- Pull request ID / number (or other stable PR identifier)
- Repository name / owner
- Pull request title
- Author
- Last updated timestamp
- Review state (optional but helpful)

---

2. Get Pull Request Context

(GET /pull-requests/context)

This endpoint provides **complete review context** for a single PR and is required for the main workflow.

This must be callable in two ways:

1. **UI path:** user selects a PR from the list view (single-select checkbox) → app calls this endpoint
2. **Conversational path:** user says, "Can you review pr-123 for me?" → ChatGPT calls this endpoint directly

Parameters

- `pr_name` (required, lowercase)
  A concrete PR identifier used to resolve the PR (e.g., `pr-123`). This must be mandatory in v1.

Optional supporting identifiers may be used at the developer's discretion (e.g., repository owner/name) if needed to uniquely resolve the PR, but the core requirement is that **a specific PR must be addressable**.

Behavior

- Resolve the specified PR unambiguously (or return a clear error if ambiguous/not found)
- Fetch:
  - PR metadata (title, description, repository, author)
  - List of changed files
  - Unified diffs / patches
- Return structured context suitable for ChatGPT to generate comments:
  - File paths
  - Patch hunks / line references when available
  - Any lightweight supporting metadata needed for anchoring inline comments

The backend must **not interpret or analyze** the code.

Efficiency Requirements

- Context responses (including diffs) must be cached for a short, reasonable duration
- Repeated calls for the same PR must reuse cached results
- Avoid refetching unchanged data unnecessarily
- Respect GitHub rate limits

---

3. Post Review Comments

(POST /pull-requests/review/post)

Triggered only after **explicit user approval** of the comments that ChatGPT generated.

Behavior

- Accept approved review comments from ChatGPT
- Post comments to GitHub:
    - Inline comments when line context is provided
    - General comments otherwise
- Default review type:
    - `COMMENT`
- `APPROVE` or `REQUEST_CHANGES` only if explicitly requested by the user

Response Requirements

- Return a success/failure summary of posted comments
- Return the **GitHub pull request URL** so the user can confirm everything is correct

Idempotency Requirements

- Duplicate comments must not be created if the request is retried
- The endpoint must be safe under repeated ChatGPT calls
- Posting logic must detect and prevent duplication

Error & Permission Handling (Mandatory)

At any stage (list, context retrieval, posting), if an error occurs:

- Detect the failure explicitly
- Identify whether it is due to:
    - Insufficient repository access
    - Insufficient write permissions
    - Rate limiting
    - Transient GitHub API issues
    - Ambiguous or unresolved PR identifiers (context endpoint)

Surface a clear, user-visible message explaining:

- What failed
- Why it failed
- Whether user action is required

Silent failures are unacceptable.

UI Requirements (Consolidated)

Global Progress Feedback

Whenever the system is:

- Fetching the PR list
- Fetching PR context
- Posting review comments

A visible **progress loader** must be shown so the user knows the app is functioning.

Pull Request List UI

- Display up to 10 pull requests returned by `GET /pull-requests`
- Show key metadata:
  - Repository
  - Title
  - Author
  - Last updated timestamp
- Each PR must include a **checkbox**
- **Only one PR may be selected at a time**
  - Selecting a new PR deselects the previous one
- Upon selection:
  - Call `GET /pull-requests/context` with the selected PR's identifier
  - Show a progress loader until context retrieval completes
  - If context retrieval fails, surface the error clearly

Review Posting Confirmation UI

- Require explicit confirmation before posting comments
- Show a progress loader while posting
- After successful posting:
  - Display the **GitHub pull request URL** returned by the backend
  - Allow the user to click through to verify the review was applied
- If posting fails due to permissions or other errors, surface the error clearly

---

Final Notes for the Developer

- Endpoint shapes are recommendations; observable behavior is not
- Mandatory for v1:
    - `GET /pull-requests` lists PRs (no single-PR context here)
    - `GET /pull-requests/context` retrieves full context for a **required** `pr_name`
    - `POST /pull-requests/review/post` posts approved comments and returns the PR URL
    - Caching for reads and idempotency for writes to handle repeated ChatGPT calls
    - Clear, user-visible error and permission handling
    - Progress loaders at all async stages
    - Single-select checkbox UI for PR list
    - GitHub PR link displayed after posting