Example plugins

To get started building, we are making available a set of simple plugins that cover different authentication schemas and use cases. From our simple no authentication todo list plugin to the more powerful retrieval plugin, these examples provide a glimpse into what we hope to make possible with plugins.

During development, you can run the plugin locally on your computer or through a cloud development environment like GitHub Codespaces, Replit, or CodeSandbox.

Plugin quickstart

We created the plugin quickstart as a starting place for developers to get a plugin up and running in less than 5 minutes. If you have not run a plugin yet and want to get acquainted with the minimal steps required to run one, consider beginning with the plugin quickstart repo.

Collapse

Learn how to build a simple todo list plugin with no auth

To start, check out the no authentication page, then define an `ai-plugin.json` file with the following fields:

```
{
    "schema_version": "v1",
    "name_for_human": "TODO List (No Auth)",
    "name_for_model": "todo",
    "description_for_human": "Manage your TODO list. You can add
    "description_for_model": "Plugin for managing a TODO list, y
```

```
    "auth": {
        "type": "none"
    },
    "api": {
        "type": "openapi",
        "url": "PLUGIN_HOSTNAME/openapi.yaml"
    },
    "logo_url": "PLUGIN_HOSTNAME/logo.png",
    "contact_email": "support@example.com",
    "legal_info_url": "https://example.com/legal"
}
```

Note the `PLUGIN_HOSTNAME` should be the actual hostname of your plugin server.

Next, we can define the API endpoints to create, delete, and fetch todo list items for a specific user.

```python
import json

import quart
import quart_cors
from quart import request

# Note: Setting CORS to allow chat.openapi.com is only required
app = quart_cors.cors(quart.Quart(__name__), allow_origin="https

_TODOS = {}


@app.post("/todos/<string:username>")
async def add_todo(username):
    request = await quart.request.get_json(force=True)
    if username not in _TODOS:
        _TODOS[username] = []
```

```
18      _TODOS[username].append(request["todo"])
19      return quart.Response(response='OK', status=200)
20
21
22  @app.get("/todos/<string:username>")
23  async def get_todos(username):
24      return quart.Response(response=json.dumps(_TODOS.get(usernam
25
26
27  @app.delete("/todos/<string:username>")
28  async def delete_todo(username):
29      request = await quart.request.get_json(force=True)
30      todo_idx = request["todo_idx"]
31      if 0 <= todo_idx < len(_TODOS[username]):
32          _TODOS[username].pop(todo_idx)
33      return quart.Response(response='OK', status=200)
34
35
36  @app.get("/logo.png")
37  async def plugin_logo():
38      filename = 'logo.png'
39      return await quart.send_file(filename, mimetype='image/png')
40
41
42  @app.get("/.well-known/ai-plugin.json")
43  async def plugin_manifest():
44      host = request.headers['Host']
45      with open("ai-plugin.json") as f:
46          text = f.read()
47          # This is a trick we do to populate the PLUGIN_HOSTNAME
48          text = text.replace("PLUGIN_HOSTNAME", f"https://{host}"
49          return quart.Response(text, mimetype="text/json")
50
51
52  @app.get("/openapi.yaml")
```

```python
async def openapi_spec():
    host = request.headers['Host']
    with open("openapi.yaml") as f:
        text = f.read()
        # This is a trick we do to populate the PLUGIN_HOSTNAME
        text = text.replace("PLUGIN_HOSTNAME", f"https://{host}"
        return quart.Response(text, mimetype="text/yaml")


def main():
    app.run(debug=True, host="0.0.0.0", port=5002)


if __name__ == "__main__":
    main()
```

Last, we need to set up and define a OpenAPI specification to match the endpoints defined on our local or remote server. You do not need to expose the full functionality of your API via the specification and can instead choose to let ChatGPT have access to only certain functionality.

There are also many tools that will automatically turn your server definition code into an OpenAPI specification so you don't need to do it manually. In the case of the Python code above, the OpenAPI specification will look like:

```yaml
openapi: 3.0.1
info:
    title: TODO Plugin
    description: A plugin that allows the user to create and man
    version: "v1"
servers:
    - url: PLUGIN_HOSTNAME
paths:
    /todos/{username}:
```

```
10          get:
11              operationId: getTodos
12              summary: Get the list of todos
13              parameters:
14                  - in: path
15                    name: username
16                    schema:
17                        type: string
18                    required: true
19                    description: The name of the user.
20              responses:
21                  "200":
22                      description: OK
23                      content:
24                          application/json:
25                              schema:
26                                  $ref: "#/components/schemas/getT
27          post:
28              operationId: addTodo
29              summary: Add a todo to the list
30              parameters:
31                  - in: path
32                    name: username
33                    schema:
34                        type: string
35                    required: true
36                    description: The name of the user.
37              requestBody:
38                  required: true
39                  content:
40                      application/json:
41                          schema:
42                              $ref: "#/components/schemas/addTodoR
43              responses:
44                  "200":
```

```
45                      description: OK
46          delete:
47              operationId: deleteTodo
48              summary: Delete a todo from the list
49              parameters:
50                  - in: path
51                    name: username
52                    schema:
53                        type: string
54                    required: true
55                    description: The name of the user.
56              requestBody:
57                  required: true
58                  content:
59                      application/json:
60                          schema:
61                              $ref: "#/components/schemas/deleteTo
62              responses:
63                  "200":
64                      description: OK
65
66  components:
67      schemas:
68          getTodosResponse:
69              type: object
70              properties:
71                  todos:
72                      type: array
73                      items:
74                          type: string
75                      description: The list of todos.
76          addTodoRequest:
77              type: object
78              required:
79                  - todo
```

```
            properties:
                todo:
                    type: string
                    description: The todo to add to the list.
                    required: true
        deleteTodoRequest:
            type: object
            required:
                - todo_idx
            properties:
                todo_idx:
                    type: integer
                    description: The index of the todo to delete
                    required: true
```

Collapse

Learn how to build a simple todo list plugin with service level auth

To start, check out the service level authentication page and then define an `ai-plugin.json` file with the following fields:

```
{
    "schema_version": "v1",
    "name_for_human": "TODO List (service auth)",
    "name_for_model": "todo",
    "description_for_human": "Manage your TODO list. You can add
    "description_for_model": "Plugin for managing a TODO list, y
    "auth": {
        "type": "service_http",
        "authorization_type": "bearer",
```

```
        "verification_tokens": {
            "openai": "Replace_this_string_with_the_verification
        }
    },
    "api": {
        "type": "openapi",
        "url": "https://example.com/openapi.yaml"
    },
    "logo_url": "https://example.com/logo.png",
    "contact_email": "support@example.com",
    "legal_info_url": "https://example.com/legal"
}
```

Notice that the verification token is required for service level authentication plugins. The token is generated during the plugin installation process in the ChatGPT web UI after you set the service access token.

You will also need to update "Example.com" to the name of your remote server.

Next, we can define the API endpoints to create, delete, and fetch todo list items for a specific user. The endpoints also check that the user is authenticated.

```python
import json

import quart
import quart_cors
from quart import request

app = quart_cors.cors(quart.Quart(__name__))

# This key can be anything, though you will likely want a random
_SERVICE_AUTH_KEY = "REPLACE_ME"
_TODOS = {}

def assert_auth_header(req):
```

```
14      assert req.headers.get(
15          "Authorization", None) == f"Bearer {_SERVICE_AUTH_KEY}"
16
17  @app.post("/todos/<string:username>")
18  async def add_todo(username):
19      assert_auth_header(quart.request)
20      request = await quart.request.get_json(force=True)
21      if username not in _TODOS:
22          _TODOS[username] = []
23      _TODOS[username].append(request["todo"])
24      return quart.Response(response='OK', status=200)
25
26  @app.get("/todos/<string:username>")
27  async def get_todos(username):
28      assert_auth_header(quart.request)
29      return quart.Response(response=json.dumps(_TODOS.get(usernam
30
31  @app.delete("/todos/<string:username>")
32  async def delete_todo(username):
33      assert_auth_header(quart.request)
34      request = await quart.request.get_json(force=True)
35      todo_idx = request["todo_idx"]
36      if 0 <= todo_idx < len(_TODOS[username]):
37          _TODOS[username].pop(todo_idx)
38      return quart.Response(response='OK', status=200)
39
40  @app.get("/logo.png")
41  async def plugin_logo():
42      filename = 'logo.png'
43      return await quart.send_file(filename, mimetype='image/png')
44
45  @app.get("/.well-known/ai-plugin.json")
46  async def plugin_manifest():
47      host = request.headers['Host']
48      with open("ai-plugin.json") as f:
```

```
        text = f.read()
        return quart.Response(text, mimetype="text/json")

@app.get("/openapi.yaml")
async def openapi_spec():
    host = request.headers['Host']
    with open("openapi.yaml") as f:
        text = f.read()
        return quart.Response(text, mimetype="text/yaml")

def main():
    app.run(debug=True, host="0.0.0.0", port=5002)

if __name__ == "__main__":
    main()
```

Last, we need to set up and define a OpenAPI specification to match the endpoints defined on our remote server. In general, the OpenAPI specification would look the same regardless of the authentication method. Using an automatic OpenAPI generator will reduce the chance of errors when creating your OpenAPI specification.

```
openapi: 3.0.1
info:
    title: TODO Plugin
    description: A plugin that allows the user to create and man
    version: "v1"
servers:
    - url: https://example.com
paths:
    /todos/{username}:
        get:
            operationId: getTodos
            summary: Get the list of todos
```

```
13              parameters:
14                  - in: path
15                    name: username
16                    schema:
17                        type: string
18                    required: true
19                    description: The name of the user.
20              responses:
21                  "200":
22                      description: OK
23                      content:
24                          application/json:
25                              schema:
26                                  $ref: "#/components/schemas/getT
27          post:
28              operationId: addTodo
29              summary: Add a todo to the list
30              parameters:
31                  - in: path
32                    name: username
33                    schema:
34                        type: string
35                    required: true
36                    description: The name of the user.
37              requestBody:
38                  required: true
39                  content:
40                      application/json:
41                          schema:
42                              $ref: "#/components/schemas/addTodoR
43              responses:
44                  "200":
45                      description: OK
46          delete:
47              operationId: deleteTodo
```

```
48              summary: Delete a todo from the list
49              parameters:
50                  - in: path
51                    name: username
52                    schema:
53                        type: string
54                    required: true
55                    description: The name of the user.
56              requestBody:
57                  required: true
58                  content:
59                      application/json:
60                          schema:
61                              $ref: "#/components/schemas/deleteTo
62              responses:
63                  "200":
64                      description: OK
65
66  components:
67      schemas:
68          getTodosResponse:
69              type: object
70              properties:
71                  todos:
72                      type: array
73                      items:
74                          type: string
75                      description: The list of todos.
76          addTodoRequest:
77              type: object
78              required:
79                  - todo
80              properties:
81                  todo:
82                      type: string
```

```
                    description: The todo to add to the list.
                    required: true
        deleteTodoRequest:
            type: object
            required:
                - todo_idx
            properties:
                todo_idx:
                    type: integer
                    description: The index of the todo to delete
                    required: true
```

Collapse

Learn how to build a simple sports stats plugin

This plugin is an example of a simple sports stats API. Please keep in mind our domain policy and usage policies when considering what to build.

To start, define an `ai-plugin.json` file with the following fields:

```
{
    "schema_version": "v1",
    "name_for_human": "Sport Stats",
    "name_for_model": "sportStats",
    "description_for_human": "Get current and historical stats f
    "description_for_model": "Get current and historical stats f
    "auth": {
        "type": "none"
    },
    "api": {
        "type": "openapi",
```

```
        "url": "PLUGIN_HOSTNAME/openapi.yaml"
    },
    "logo_url": "PLUGIN_HOSTNAME/logo.png",
    "contact_email": "support@example.com",
    "legal_info_url": "https://example.com/legal"
}
```

Note the `PLUGIN_HOSTNAME` should be the actual hostname of your plugin server.

Next, we define a mock API for a simple sports service plugin.

```python
import json
import requests
import urllib.parse

import quart
import quart_cors
from quart import request

# Note: Setting CORS to allow chat.openapi.com is only required
app = quart_cors.cors(quart.Quart(__name__), allow_origin="http
HOST_URL = "https://example.com"

@app.get("/players")
async def get_players():
    query = request.args.get("query")
    res = requests.get(
        f"{HOST_URL}/api/v1/players?search={query}&page=0&per_p
    body = res.json()
    return quart.Response(response=json.dumps(body), status=200


@app.get("/teams")
async def get_teams():
```

```python
    res = requests.get(
        "{HOST_URL}/api/v1/teams?page=0&per_page=100")
    body = res.json()
    return quart.Response(response=json.dumps(body), status=200


@app.get("/games")
async def get_games():
    query_params = [("page", "0")]
    limit = request.args.get("limit")
    query_params.append(("per_page", limit or "100"))
    start_date = request.args.get("start_date")
    if start_date:
        query_params.append(("start_date", start_date))
    end_date = request.args.get("end_date")

    if end_date:
        query_params.append(("end_date", end_date))
    seasons = request.args.getlist("seasons")

    for season in seasons:
        query_params.append(("seasons[]", str(season)))
    team_ids = request.args.getlist("team_ids")

    for team_id in team_ids:
        query_params.append(("team_ids[]", str(team_id)))

    res = requests.get(
        f"{HOST_URL}/api/v1/games?{urllib.parse.urlencode(query
    body = res.json()
    return quart.Response(response=json.dumps(body), status=200


@app.get("/stats")
async def get_stats():
```

```
59      query_params = [("page", "0")]
60      limit = request.args.get("limit")
61      query_params.append(("per_page", limit or "100"))
62      start_date = request.args.get("start_date")
63      if start_date:
64          query_params.append(("start_date", start_date))
65      end_date = request.args.get("end_date")
66
67      if end_date:
68          query_params.append(("end_date", end_date))
69      player_ids = request.args.getlist("player_ids")
70
71      for player_id in player_ids:
72          query_params.append(("player_ids[]", str(player_id)))
73      game_ids = request.args.getlist("game_ids")
74
75      for game_id in game_ids:
76          query_params.append(("game_ids[]", str(game_id)))
77      res = requests.get(
78          f"{HOST_URL}/api/v1/stats?{urllib.parse.urlencode(query
79      body = res.json()
80      return quart.Response(response=json.dumps(body), status=200
81
82
83  @app.get("/season_averages")
84  async def get_season_averages():
85      query_params = []
86      season = request.args.get("season")
87      if season:
88          query_params.append(("season", str(season)))
89      player_ids = request.args.getlist("player_ids")
90
91      for player_id in player_ids:
92          query_params.append(("player_ids[]", str(player_id)))
93      res = requests.get(
```

```
94              f"{HOST_URL}/api/v1/season_averages?{urllib.parse.urlen
95          body = res.json()
96          return quart.Response(response=json.dumps(body), status=200
97
98
99      @app.get("/logo.png")
100     async def plugin_logo():
101         filename = 'logo.png'
102         return await quart.send_file(filename, mimetype='image/png'
103
104
105     @app.get("/.well-known/ai-plugin.json")
106     async def plugin_manifest():
107         host = request.headers['Host']
108         with open("ai-plugin.json") as f:
109             text = f.read()
110             # This is a trick we do to populate the PLUGIN_HOSTNAME
111             text = text.replace("PLUGIN_HOSTNAME", f"https://{host}
112             return quart.Response(text, mimetype="text/json")
113
114
115     @app.get("/openapi.yaml")
116     async def openapi_spec():
117         host = request.headers['Host']
118         with open("openapi.yaml") as f:
119             text = f.read()
120             # This is a trick we do to populate the PLUGIN_HOSTNAME
121             text = text.replace("PLUGIN_HOSTNAME", f"https://{host}
122             return quart.Response(text, mimetype="text/yaml")
123
124
125     def main():
126         app.run(debug=True, host="0.0.0.0", port=5001)
127
128
```

```python
if __name__ == "__main__":
    main()
```

Last, we define our OpenAPI specification:

```yaml
openapi: 3.0.1
info:
    title: Sport Stats
    description: Get current and historical stats for sport pla
    version: "v1"
servers:
    - url: PLUGIN_HOSTNAME
paths:
    /players:
        get:
            operationId: getPlayers
            summary: Retrieves all players from all seasons who
            parameters:
                - in: query
                  name: query
                  schema:
                      type: string
                  description: Used to filter players based on
            responses:
                "200":
                    description: OK
    /teams:
        get:
            operationId: getTeams
            summary: Retrieves all teams for the current season
            responses:
                "200":
                    description: OK
    /games:
```

```yaml
    get:
      operationId: getGames
      summary: Retrieves all games that match the filters
      parameters:
        - in: query
          name: limit
          schema:
            type: string
          description: The max number of results to ret
        - in: query
          name: seasons
          schema:
            type: array
            items:
              type: string
          description: Filter by seasons. Seasons are r
        - in: query
          name: team_ids
          schema:
            type: array
            items:
              type: string
          description: Filter by team ids. Team ids can
        - in: query
          name: start_date
          schema:
            type: string
          description: A single date in 'YYYY-MM-DD' fo
        - in: query
          name: end_date
          schema:
            type: string
          description: A single date in 'YYYY-MM-DD' fo
      responses:
        "200":
```

```
                    description: OK
    /stats:
        get:
            operationId: getStats
            summary: Retrieves stats that match the filters spe
            parameters:
                - in: query
                  name: limit
                  schema:
                      type: string
                  description: The max number of results to ret
                - in: query
                  name: player_ids
                  schema:
                      type: array
                      items:
                          type: string
                  description: Filter by player ids. Player ids
                - in: query
                  name: game_ids
                  schema:
                      type: array
                      items:
                          type: string
                  description: Filter by game ids. Game ids can
                - in: query
                  name: start_date
                  schema:
                      type: string
                  description: A single date in 'YYYY-MM-DD' fo
                - in: query
                  name: end_date
                  schema:
                      type: string
                  description: A single date in 'YYYY-MM-DD' fo
```

```
            responses:
                "200":
                    description: OK
    /season_averages:
        get:
            operationId: getSeasonAverages
            summary: Retrieves regular season averages for the
            parameters:
                - in: query
                  name: season
                  schema:
                      type: string
                  description: Defaults to the current season.
                - in: query
                  name: player_ids
                  schema:
                      type: array
                      items:
                          type: string
                  description: Filter by player ids. Player ids
            responses:
                "200":
                    description: OK
```

Collapse

Learn how to build a simple OAuth todo list plugin

To create an OAuth plugin, we start by defining a `ai-plugin.json` file with the auth type set to `oauth`:

```
{
```

```
    "schema_version": "v1",
    "name_for_human": "TODO List (OAuth)",
    "name_for_model": "todo_oauth",
    "description_for_human": "Manage your TODO list. You can add
    "description_for_model": "Plugin for managing a TODO list, y
    "auth": {
        "type": "oauth",
        "client_url": "PLUGIN_HOSTNAME/oauth",
        "scope": "",
        "authorization_url": "PLUGIN_HOSTNAME/auth/oauth_exchang
        "authorization_content_type": "application/json",
        "verification_tokens": {
            "openai": "Replace_this_string_with_the_verification
        }
    },
    "api": {
        "type": "openapi",
        "url": "PLUGIN_HOSTNAME/openapi.yaml"
    },
    "logo_url": "PLUGIN_HOSTNAME/logo.png",
    "contact_email": "contact@example.com",
    "legal_info_url": "http://www.example.com/legal"
}
```

Next, we need to define our OAuth service. This OAuth example is not intended for production use cases but rather to highlight what a simple OAuth flow will look like so developers can get experience building towards a production solution.

```
import json

import quart
import quart_cors
from quart import request

```

```
7   app = quart_cors.cors(quart.Quart(__name__), allow_origin="*")
8
9   _TODOS = {}
10
11  @app.post("/todos/<string:username>")
12  async def add_todo(username):
13      request = await quart.request.get_json(force=True)
14      if username not in _TODOS:
15          _TODOS[username] = []
16      _TODOS[username].append(request["todo"])
17      return quart.Response(response='OK', status=200)
18
19  @app.get("/todos/<string:username>")
20  async def get_todos(username):
21      print(request.headers)
22      return quart.Response(response=json.dumps(_TODOS.get(usernam
23
24  @app.delete("/todos/<string:username>")
25  async def delete_todo(username):
26      request = await quart.request.get_json(force=True)
27      todo_idx = request["todo_idx"]
28      # fail silently, it's a simple plugin
29      if 0 <= todo_idx < len(_TODOS[username]):
30          _TODOS[username].pop(todo_idx)
31      return quart.Response(response='OK', status=200)
32
33  @app.get("/logo.png")
34  async def plugin_logo():
35      filename = 'logo.png'
36      return await quart.send_file(filename, mimetype='image/png')
37
38  @app.get("/.well-known/ai-plugin.json")
39  async def plugin_manifest():
40      host = request.headers['Host']
41      with open("manifest.json") as f:
```

```
        text = f.read()
        text = text.replace("PLUGIN_HOSTNAME", f"https://{host}"
        return quart.Response(text, mimetype="text/json")

@app.get("/openapi.yaml")
async def openapi_spec():
    host = request.headers['Host']
    with open("openapi.yaml") as f:
        text = f.read()
        text = text.replace("PLUGIN_HOSTNAME", f"https://{host}"
        return quart.Response(text, mimetype="text/yaml")

@app.get("/oauth")
async def oauth():
    query_string = request.query_string.decode('utf-8')
    parts = query_string.split('&')
    kvps = {}
    for part in parts:
        k, v = part.split('=')
        v = v.replace("%2F", "/").replace("%3A", ":")
        kvps[k] = v
    print("OAuth key value pairs from the ChatGPT Request: ", kv
    url = kvps["redirect_uri"] + f"?code={OPENAI_CODE}"
    print("URL: ", url)
    return quart.Response(
        f'<a href="{url}">Click to authorize</a>'
    )

# Sample names
OPENAI_CLIENT_ID = "id"
OPENAI_CLIENT_SECRET = "secret"
OPENAI_CODE = "abc123"
OPENAI_TOKEN = "def456"

@app.post("/auth/oauth_exchange")
```

```python
async def oauth_exchange():
    request = await quart.request.get_json(force=True)
    print(f"oauth_exchange {request=}")

    if request["client_id"] != OPENAI_CLIENT_ID:
        raise RuntimeError("bad client ID")
    if request["client_secret"] != OPENAI_CLIENT_SECRET:
        raise RuntimeError("bad client secret")
    if request["code"] != OPENAI_CODE:
        raise RuntimeError("bad code")

    return {
        "access_token": OPENAI_TOKEN,
        "token_type": "bearer"
    }

def main():
    app.run(debug=True, host="0.0.0.0", port=5002)

if __name__ == "__main__":
    main()
```

Last, like with our other examples, we define a simple OpenAPI file based on the endpoints:

```yaml
openapi: 3.0.1
info:
    title: TODO Plugin
    description: A plugin that allows the user to create and man
    version: "v1"
servers:
    - url: PLUGIN_HOSTNAME
paths:
    /todos/{username}:
```

```
10          get:
11              operationId: getTodos
12              summary: Get the list of todos
13              parameters:
14                  - in: path
15                    name: username
16                    schema:
17                        type: string
18                    required: true
19                    description: The name of the user.
20              responses:
21                  "200":
22                      description: OK
23                      content:
24                          application/json:
25                              schema:
26                                  $ref: "#/components/schemas/getT
27          post:
28              operationId: addTodo
29              summary: Add a todo to the list
30              parameters:
31                  - in: path
32                    name: username
33                    schema:
34                        type: string
35                    required: true
36                    description: The name of the user.
37              requestBody:
38                  required: true
39                  content:
40                      application/json:
41                          schema:
42                              $ref: "#/components/schemas/addTodoR
43              responses:
44                  "200":
```

```
45                      description: OK
46          delete:
47              operationId: deleteTodo
48              summary: Delete a todo from the list
49              parameters:
50                  - in: path
51                    name: username
52                    schema:
53                        type: string
54                    required: true
55                    description: The name of the user.
56              requestBody:
57                  required: true
58                  content:
59                      application/json:
60                          schema:
61                              $ref: "#/components/schemas/deleteTo
62              responses:
63                  "200":
64                      description: OK
65
66  components:
67      schemas:
68          getTodosResponse:
69              type: object
70              properties:
71                  todos:
72                      type: array
73                      items:
74                          type: string
75                      description: The list of todos.
76          addTodoRequest:
77              type: object
78              required:
79                  - todo
```

```
            properties:
                todo:
                    type: string
                    description: The todo to add to the list.
                    required: true
        deleteTodoRequest:
            type: object
            required:
                - todo_idx
            properties:
                todo_idx:
                    type: integer
                    description: The index of the todo to delete
                    required: true
```

Collapse

Learn how to build a semantic search and retrieval plugin

The ChatGPT retrieval plugin is a more fully featured code example. The scope of the plugin is large, so we encourage you to read through the code to see what a more advanced plugin looks like.

The retrieval plugin includes:

- Support for multiple vector databases providers
- All 4 different authentication methods
- Multiple different API features

Collapse
