fireservice-adapter

API Documentation

1. Overview

Firestore Adapter is a stateless RESTful microservice that abstracts and simplifies Google Firestore operations for other backend services.

It provides clean endpoints for common Firestore operations (CRUD), so you can integrate Firestore into your apps without directly using Google SDKs or managing low-level permissions.

---

2. Architecture

- **Language/Stack:** Python (Flask), Dockerized
- **Deployment:** Google Cloud Run (stateless, auto-scaled)
- **Authentication:**
  - Service account for backend-to-backend communication (recommended)
  - API key (optional, if enabled)
- **State:** Stateless (all data is stored in Firestore; no persistent state in the service)

---

3. Environment Variables

Variable	Description	Required	Default
`GOOGLE_CLOUD_PROJECT`	GCP project ID	Yes	—
`FIRESTORE_COLLECTION`	Firestore collection name	No	`documents`
`API_KEY`	API key for API-key-based auth (if enabled)	No	—

---

4. Endpoints

Method	Path	Description	Auth Required
GET	`/health`	Health check	No
GET	`/documents`	List all documents	Yes
GET	`/documents/{id}`	Get document by ID	Yes
POST	`/documents`	Create a new document	Yes
PUT	`/documents/{id}`	Update a document	Yes
DELETE	`/documents/{id}`	Delete document by ID	Yes

---

5. Authentication

Supported Methods

- **Google Service Account JWT** (recommended for backend-to-backend)
- **API Key** (if enabled; set `API_KEY` in environment)

How to Authenticate

Send the following HTTP header on all protected endpoints:

```
Authorization: Bearer <service-account-jwt or API key>
```

> *API Key is checked for exact match against the value in the environment variable. There is no management endpoint for API keys; distribution is out-of-band.*

---

6. Data Model

- Any **valid JSON** object is allowed as a document.
- No required fields enforced by the API; all fields are custom.
- Timestamps (`createdAt`, `updatedAt`) are automatically added by the backend.

---

7. API Reference and Examples

GET `/health`

Check service status.

Response:

```
{"status": "ok"}
```

GET `/documents`

List all documents in the collection.

Auth required: Yes

Response:

```
[
  {
    "id": "12345",
    "field1": "value1",
    "createdAt": "2025-05-27T10:20:00Z"
  },
  ...
]
```

> ***Note:*** *Pagination and filters are not supported.*

POST `/documents`

Create a new document.

Auth required: Yes

Request body:

```
{
  "field1": "value1",
  "field2": "value2"
}
```

Response:

```
{
  "id": "12345",
  "field1": "value1",
  "field2": "value2",
  "createdAt": "2025-05-27T10:20:00Z"
}
```

Error responses:

- 400 Bad Request: `{"error": "Invalid JSON"}`
- 401 Unauthorized: `{"error": "Invalid token"}`

GET `/documents/{id}`

Retrieve a document by its ID.

Auth required: Yes

Response:

```
{
  "id": "12345",
  "field1": "value1",
  "createdAt": "2025-05-27T10:20:00Z"
}
```

Error responses:

- 404 Not Found: `{"error": "Document not found"}`
- 401 Unauthorized: `{"error": "Invalid token"}`

PUT `/documents/{id}`

Update an existing document.

Auth required: Yes

Request body:

```
{
  "field1": "newvalue"
}
```

Response:

```
{
  "id": "12345",
  "field1": "newvalue",
  "field2": "value2",
  "updatedAt": "2025-05-27T10:25:00Z"
}
```

Error responses:

- 404 Not Found: `{"error": "Document not found"}`
- 401 Unauthorized: `{"error": "Invalid token"}`

DELETE `/documents/{id}`

Delete a document by ID.

Auth required: Yes

Response:

```
{ "success": true }
```

Error responses:

- 404 Not Found: `{"error": "Document not found"}`
- 401 Unauthorized: `{"error": "Invalid token"}`

8. Error Codes

Status	Example	Meaning
400	`{"error": "Invalid JSON"}`	Bad Request (malformed)
401	`{"error": "Invalid token"}`	Unauthorized
404	`{"error": "Document not found"}`	Not Found
500	`{"error": "Server error"}`	Internal Server Error

9. Deployment & CI/CD

Cloud Build & Cloud Run:

- Docker image is built and pushed to Artifact Registry.
- Cloud Run is used for deployment (stateless, managed).
- CI/CD: Pushing to `main` branch triggers a build and deployment.

Required steps:

1. Set environment variables (`GOOGLE_CLOUD_PROJECT`, `FIRESTORE_COLLECTION`, `API_KEY` if needed).
2. Use the provided `cloudbuild.yaml` for automated builds and deployment.
3. (Optional) Use `deploy.sh` to push code and trigger deployment.

10. Security, CORS & Rate Limits

- **CORS:** Not enabled by default.
- **Rate limiting:** Not enforced in the API.
- **Max payload size:** Not enforced in code.
- **Changelog/versioning:** None currently.

11. Integration Quick Start

Example – Create a Document with Bearer Token:

```
curl -X POST https://<cloud-run-url>/documents ¥
  -H ”Content-Type: application/json” ¥
  -H ”Authorization: Bearer <token>” ¥
  -d '{”field1”: ”value1”, ”field2”: ”value2”}'
```

Replace `<token>` with a Google service account JWT or API key, as configured.
