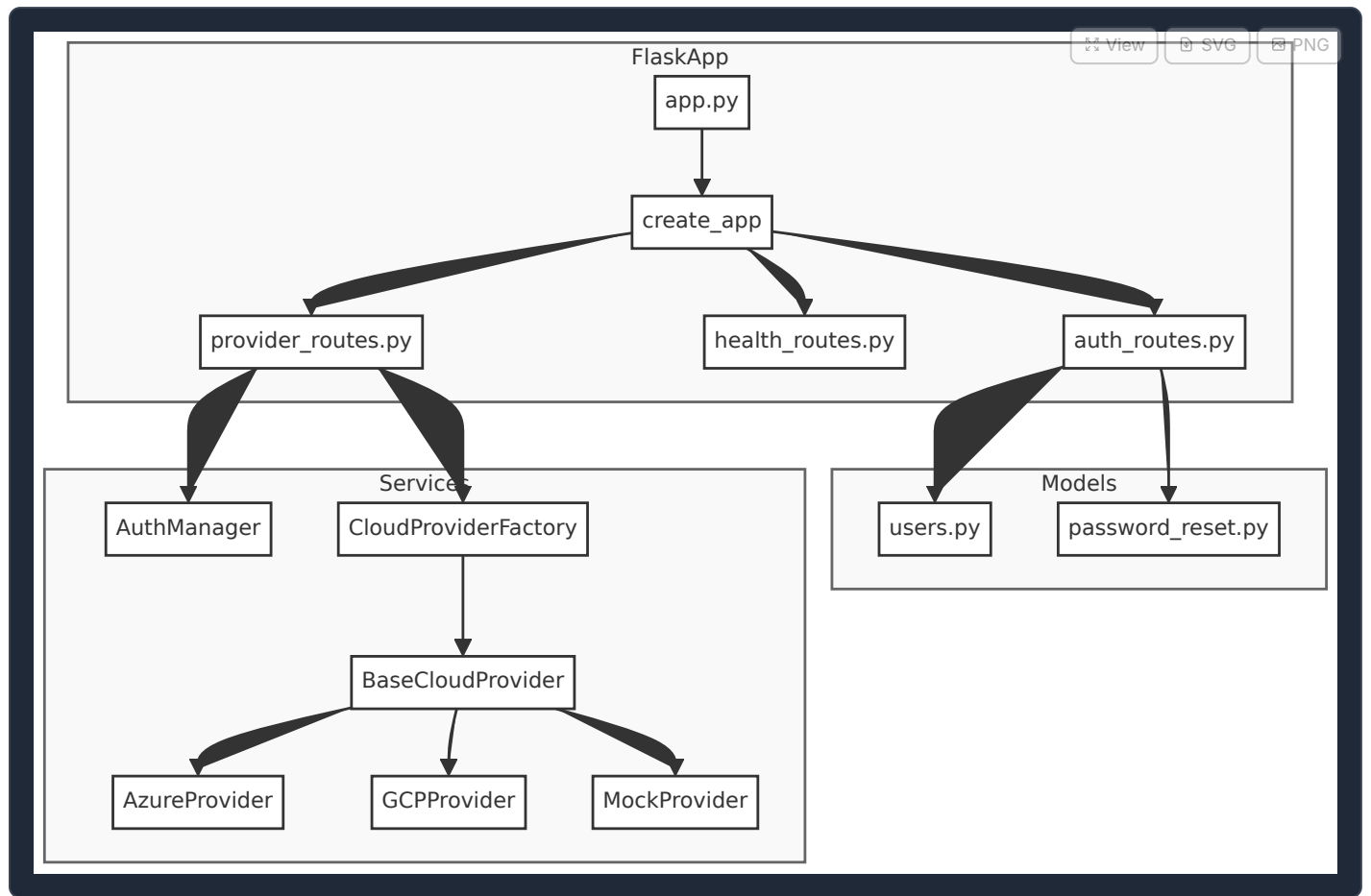


A Flask-based API server that organizes routes, models, and services into modular components. Below is a high-level diagram showing core module relationships:



Main entry point that initializes environment, creates the Flask app, and runs the server.

- Loads `.env` variables before imports using `python-dotenv`
- Calls `create_app()` with `FLASK_ENV` (`development` / `production` / `testing`)
- Instantiates `AuthManager` to log configured providers on startup
- Runs Flask built-in server (for development)

Defines application configurations for different environments.

- Base `Config` class loads from environment with sensible defaults
- `DevelopmentConfig`, `ProductionConfig`, `TestingConfig` inherit from `Config`
- Key settings: `SECRET_KEY`, `JWT_SECRET`, `SQLALCHEMY_DATABASE_URI`, `CORS_ORIGINS`

Specifies Python dependencies:

Package	Purpose
Flask	Web framework
SQLAlchemy	ORM
python-dotenv	Environment variable loader
PyJWT	JWT handling
Werkzeug	Password hashing
azure-identity, azure-mgmt-costmanagement	Azure integration
google-cloud-bigquery, google-cloud-monitoring	GCP integration

Handles user registration, login, token issuance, and password reset flows.

Register User

Export to Postman

Create a new user account.

POST

http://localhost:5000/api/auth/register

Request body

JSON payload required for this request.

```
{  "username": "john",  "password": "secret",  "email": "john@example.com",  "role": "viewer"}
```

Code examples

```
curl -X POST "http://localhost:5000/api/auth/register" \
  -H "Content-Type: application/json" \
  -d '{
    \"username\": \"john\",
    \"password\": \"secret\",
    \"email\": \"john@example.com\",
    \"role\": \"viewer\"
  }'
```

Responses

201 400 409 500

User registered successfully

```
{
  "ok": true,
  "message": "User registered successfully.",
  "user": {"username": "john", "role": "viewer"}
}
```

Missing credentials or invalid input

Username or email already exists

Server error

Login User

[Export to Postman](#)

Authenticate user and return JWT.

POST

Request body

JSON payload required for this request.

```
{
  "username": "john",
  "password": "secret"
}
```

Code examples

```
curl -X POST "http://localhost:5000/api/auth/login" \
  -H "Content-Type: application/json" \
  -d '{
    \"username\": \"john\",
    \"password\": \"secret\"
  }'
```

Responses

200 400 401 500

Login successful

```
{
  "token": "<jwt>",
  "user": {"username": "john", "role": "viewer"},
  "expires_at": "...Z",
  "expires_at_local": "...+00:00",
  "client_tz": "UTC"
}
```

Missing credentials

Invalid credentials

Server error

Get Current User

[Export to Postman](#)

Return user info for authenticated requests.

GET

Headers

Authorization string • header required

Bearer <token>

Code examples

```
curl -X GET "http://localhost:5000/api/auth/me" \
  -H "Authorization: Bearer <token>"
```

Responses

200 401

User data

```
{"user": {"username": "john", "role": "viewer"}}
```

Unauthorized

Logout User

 Export to Postman

Invalidate the current session (client-side).

POST

Headers

Authorization string • header required

Bearer <token>

Code examples

```
curl -X POST "http://localhost:5000/api/auth/logout" \  
-H "Authorization: Bearer <token>"
```

Responses

Logout successful

```
{"ok": true}
```

Request Password Reset

 Export to Postman

Send or return a reset token for password recovery.

POST

Request body

JSON payload required for this request.

```
{
  "username": "john",
  "email": "john@example.com"
}
```

Code examples

```
curl -X POST "http://localhost:5000/api/auth/forgot" \
  -H "Content-Type: application/json" \
  -d '{
    "username": "john",
    "email": "john@example.com"
  }'
```

Responses

200 500

Instructions sent

```
{"ok": true, "message": "If the account exists...", "reset_token": "abc123"}
```

Server error

Reset Password

[Export to Postman](#)

Complete password reset using a valid token.

POST

Request body

JSON payload required for this request.

```
{
  "token": "abc123",
  "new_password": "newsecret"
}
```

Code examples

```
curl -X POST "http://localhost:5000/api/auth/reset" \
  -H "Content-Type: application/json" \
  -d '{
    \"token\": \"abc123\",
    \"new_password\": \"newsecret\"
  }'
```

Responses

200 400 404 500

Password updated

```
{\"ok\": true, \"message\": \"Password updated successfully.\"}
```

Missing fields or invalid/expired token

User not found

Server error

Monitors service liveness and provides a quick human-readable page.

Index Page

[Export to Postman](#)

Basic HTML overview with links.

GET

Code examples

```
curl -X GET "http://localhost:5000/"
```

Responses

HTML page with API hints

Health Check

[Export to Postman](#)

Return JSON status for monitoring.

GET

Code examples

```
curl -X GET "http://localhost:5000/api/health"
```

Responses

Service operational

```
{"ok": true, "service": "multi-cloud-dashboard"}
```

Unified multi-cloud endpoints guarded by JWT and roles.
Key route groups:

- **/api/providers** — list active providers
- **/api/costs** — MTD (`/<provider>/costs/mtd`), daily (`/<provider>/costs/daily`), summary
- **/api/metrics** — live metrics (`/metrics/live/<provider>`), timeseries
- **/api/anomalies** — provider & all
- **/api/forecast** — provider & all
- **/api/recommendations** — provider & all
- **/api/alerts** — provider & all
- **/api/budgets** — provider & all
- **/api/services/breakdown** — provider & all

Relationships: uses `AuthManager` for config, `CloudProviderFactory` to instantiate providers, each provider subclass for data fetching.

Defines `User` SQLAlchemy model:

- Fields: `username`, `email`, `password_hash`, `role`, `created_at`
- Unique/index constraints for fast lookups

- `__repr__` hides sensitive data

Used by auth routes and token decorators.

Manages one-time, time-limited password reset tokens:

- `token` generated via `secrets.token_urlsafe()`
- `created_at` timestamp, `expires_at` helper
- `issue_for()` static method and `is_expired()` instance check

Integrated into `/forgot` and `/reset` flows.

Loads cloud provider configurations from:

1. JSON file (`CLOUD_CONFIG_PATH`)
2. Environment variables (`GCP_PROJECT_ID` , `AWS_ACCOUNT_ID` , `AZURE_SUBSCRIPTION_ID`)
3. `USE MOCK DATA` override

Provides:

- `is_provider_configured(name)`
 - `get_config(name)`
 - `get_active_providers()` (for `/api/providers`)
-

Abstract class defining interface for cloud data providers:

- `get_mtd_costs()` , `get_daily_costs()` , `get_live_metrics()` , `get_timeseries(...)`
 - Default stubs for new features: anomalies, forecast, recommendations, alerts, budgets, services breakdown
-

Implements Azure integration via Azure SDK:

- Validates `subscription_id` and SDK installation
 - Queries Cost Management API for MTD and daily costs
 - Placeholder methods for metrics/time series
-

Implements GCP integration via BigQuery & Cloud Monitoring:

- Validates `project_id` and SDK libs
- Queries BigQuery for MTD costs
- Fetches CPU utilization from Monitoring API
- Time series stub

Provides comprehensive fake data for demos:

- 140+ services per provider
- Simulated anomalies, forecasts, recommendations, alerts, budgets
- Used when `USE MOCK DATA=true`

Factory pattern to instantiate real or mock providers:

- Maps provider keys (`aws` , `azure` , `gcp` , `mock`) to classes
- Honors `USE MOCK DATA` env var for dev mode
- Extensible via `register_provider()`

A React + Vite SPA with:

- **Authentication** components under `src/components/auth`
- **Common UI** (`CostsTable` , `MetricsCard`) under `src/components/common`
- **Dashboard** pages and plots under `src/components/dashboard`
- **Custom hooks** in `src/hooks/useCloudData.js`
- **API client** in `src/services/api.js`

Mounts React app into `#root` , wraps in `React.StrictMode` , and renders `<App/>` . `filecite` `turn?file?`

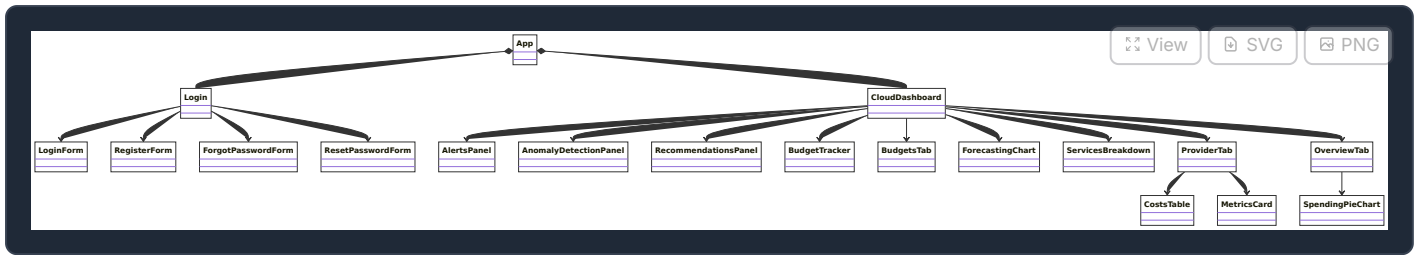
Defines routes with `react-router-dom` :

- `/login` → `<Login/>` (public)
- `/dashboard` → `<CloudDashboard/>` (protected by `RequireAuth`)
- Catch-all redirects to `/login`

Wraps all in an `ErrorBoundary` for graceful error handling. `filecite` `turn?file?`

Central HTTP client featuring:

- JWT token injection
 - `X-Timezone` header for locale handling
 - Automatic 401 handling (logout redirect)
 - `AbortController` support
 - Typed methods for every backend endpoint (auth, health, summary, provider, anomalies, forecast, recommendations, alerts, budgets, services)
-



Each component uses corresponding hooks from `useCloudData.js` to fetch data via `api.js`, ensuring separation of concerns and reusability.

- **frontend/index.html**: HTML template mounting `#root`
- **frontend/package.json**: npm scripts (`dev`, `build`, `lint`, `preview`), dependencies (`react`, `react-router-dom`, `recharts`)
- **vite.config.js**: Vite setup with React plugin and dev proxy to backend
- **eslint.config.js**: Linting rules for React, hooks, and modern JS

□ This documentation covers the core files you requested, outlines their purpose, and illustrates how they interconnect. For further details on individual React component props or service methods, consult inline JSDoc comments.