Stage 1 REST API — Quick Guide

- PHP + MySQL backend with 14 endpoints (7 protected)
- Run locally via PHP built-in or NGINX script
- Includes docs page and two test suites

Agent State Controller

- Manages agent lifecycle: idle, running, paused, error
- CRUD for agents, tasks, tools; logs for execution history
- Bearer token auth for mutating operations
- Front controller via PHP (built-in) or NGINX + PHP-FPM
- Docs page and two test suites included

Prerequisites

- PHP 8+, MySQL (or MariaDB)
- (Optional) NGINX + PHP-FPM
- VS Code (for editing), curl (for tests)

API Endpoints Summary

14 Total Endpoints:

- 7 Public (GET) - no auth required
- 7 Protected - require Bearer token

Resources:

- Agents (5 endpoints)
- Tasks (6 endpoints)
- Tools (1 endpoint)
- Logs (1 endpoint)

1. GET /agents

Purpose: List all agents with their status and details

Auth: None

Response: Array of agent objects

```
[
  {
    "id": 1,
    "name": "DataCollector",
    "description": "Collects and processes data from various sources",
    "status": "idle",
    "created_at": "2025-10-26 22:06:58",
    "updated_at": "2025-10-26 22:06:58"
  }
  // ... more agents
]
```

2. GET /agents/{id}

Purpose: Get specific agent by ID

Auth: None

Parameters: `id` (path, integer)

Response: Agent object or 404


```
{
  "id": 1,
  "name": "DataCollector",
  "description": "Collects and processes data from various sources",
  "status": "idle",
  "created_at": "2025-10-26 22:06:58",
  "updated_at": "2025-10-26 22:06:58"
}
```

3. POST /agents 🔒

Purpose: Create a new agent

Auth: Bearer token required

Body:

```
{
  "name": "NewAgent",
  "description": "Test agent"
}
```

Response (201):

```
{
  "id": "4",
  "message": "Agent created"
}
```

4. PUT /agents/{id} 🔒

Purpose: Update agent details

Auth: Bearer token required

Parameters: `id` (path)

Body: Partial update allowed

```
{
  "status": "paused"
}
```

Response:

```
{
  "message": "Agent updated"
}
```

5. DELETE /agents/{id} 🔒

Purpose: Delete agent (cascades to tasks/logs)

Auth: Bearer token required

Parameters: `id` (path, integer)

Response:

```
{
  "message": "Agent deleted"
}
```

Note: All related tasks and logs are also deleted

6. POST /agents/{id}/execute 🔒

Purpose: Execute an agent and log the action

Auth: Bearer token required

Parameters: `id` (path)

Effect: Sets status to "running", logs execution start

Response:

```
{
  "message": "Agent execution started",
  "agent_id": "2"
}
```

7. GET /tasks

Purpose: List all tasks ordered by priority

Auth: None

Response: Array of task objects

```
[
  {
    "id": 4,
    "agent_id": 3,
    "title": "Check system status",
    "description": "Monitor CPU and memory usage",
    "status": "running",
    "priority": 3,
    "created_at": "2025-10-26 22:06:58",
    "updated_at": "2025-10-26 22:06:58"
  },
  //.... other agent tasks
]
```

8. GET /tasks/{id}

Purpose: Get specific task by ID

Auth: None

Parameters: `id` (path, integer)

Response: Task object or 404


```
{
  "id": 1,
  "agent_id": 1,
  "title": "Fetch weather data",
  "description": "Collect weather information from API",
  "status": "pending",
  "priority": 1,
  "created_at": "2025-10-26 22:06:58",
  "updated_at": "2025-10-26 22:06:58"
}
```

9. GET /tasks/{id}/status

Purpose: Get current status of a task (lightweight)

Auth: None

Parameters: `id` (path, integer)

Response:

```
{
  "task_id": 1,
  "status": "pending"
}
```

Use case: Polling for status updates

10. POST /tasks 🔒

Purpose: Create a new task

Auth: Bearer token required

Body:

```
{
  "title": "New Task",
  "agent_id": 2,
  "priority": 5
}
```

Response (201):

```
{
  "id": "5",
  "message": "Task created"
}
```

11. PUT /tasks/{id} 🔒

Purpose: Update task details

Auth: Bearer token required

Parameters: `id` (path)

Body: Partial update allowed

```
{
  "status": "completed"
}
```

Response:

```
{
  "message": "Task updated"
}
```

12. DELETE /tasks/{id} 🔒

Purpose: Delete task

Auth: Bearer token required

Parameters: `id` (path, integer)

Response:

```
{
  "message": "Task deleted"
}
```

13. GET /tools

Purpose: List all available tools for agents

Auth: None

Response: Array of tool objects (sorted by category, name)

```
[
  {
    "id": 5,
    "name": "Email Sender",
    "description": "Sends email notifications",
    "category": "communication",
    "created_at": "2025-10-26 22:06:58"
  },
  // .. other agent tools
]
```

14. GET /logs

Purpose: Get execution logs

Auth: None

Query Parameters:

- `limit` (integer, default 100) - Max results
- `agent_id` (integer, optional) - Filter by agent

Response:

```
{
  "id": 1,
  "agent_id": 1,
  "task_id": 1,
  "level": "info",
  "message": "Task started successfully",
  "created_at": "2025-10-26 22:06:58"
}
// ... more logs
```

Authentication Example

Protected endpoints require Bearer token:

```
curl -X POST http://localhost:8000/agents \
  -H "Authorization: Bearer test_token_12345abcdef67890" \
  -H "Content-Type: application/json" \
  -d '{"name":"NewAgent","description":"Test agent"}'
```

Generate token:

```
cd code && ./generate_token.sh
```

Error Responses

All endpoints return consistent error format:

```
{
  "error": "Agent not found"
}
```

HTTP Status Codes:

- 200 OK, 201 Created
- 400 Bad Request, 401 Unauthorized, 404 Not Found
- 405 Method Not Allowed, 500 Internal Server Error

NGINX Config Anatomy

- listen 8000; root code/
- `try_files $uri /index.php?$args`
- fastcgi_pass 127.0.0.1:9000
- minimal, self-contained config

PHP-FPM Quick Start

- macOS: `brew services start php`
- Linux/WSL: `sudo systemctl start php-fpm`
- Script auto-starts if not running

Windows / WSL Note

- Recommended: WSL (Ubuntu) then follow Linux steps
- Native: nginx.exe + php-cgi.exe (not primary path)

Schema Overview

- agents, tasks, tools, logs, api_tokens
- Foreign keys: tasks→agents, logs→(agents,tasks)
- Indexes on status and timestamps

Seed Data (Optional)

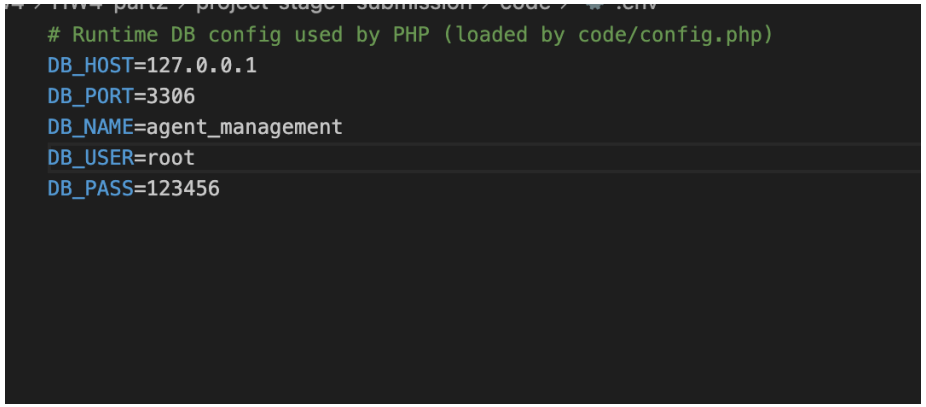
- `./setup.sh` prompts to load seed.sql
- Provides sample agents/tasks/tools/logs

1) Configure Database (.env)

Create `code/.env`:

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_NAME=agent_management
DB_USER=root
DB_PASS=123456
```

Tip: Keep this file local. Values above are for local dev only.

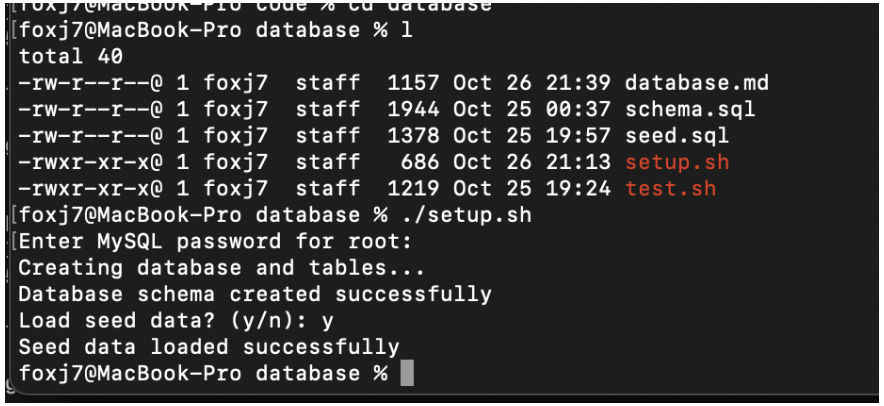


2) Initialize Database

From `code/database/`:

```
DB_HOST=127.0.0.1 DB_PORT=3306 DB_USER=root DB_PASS=123456 ./setup.sh
```

- Creates DB and tables, optional seed data



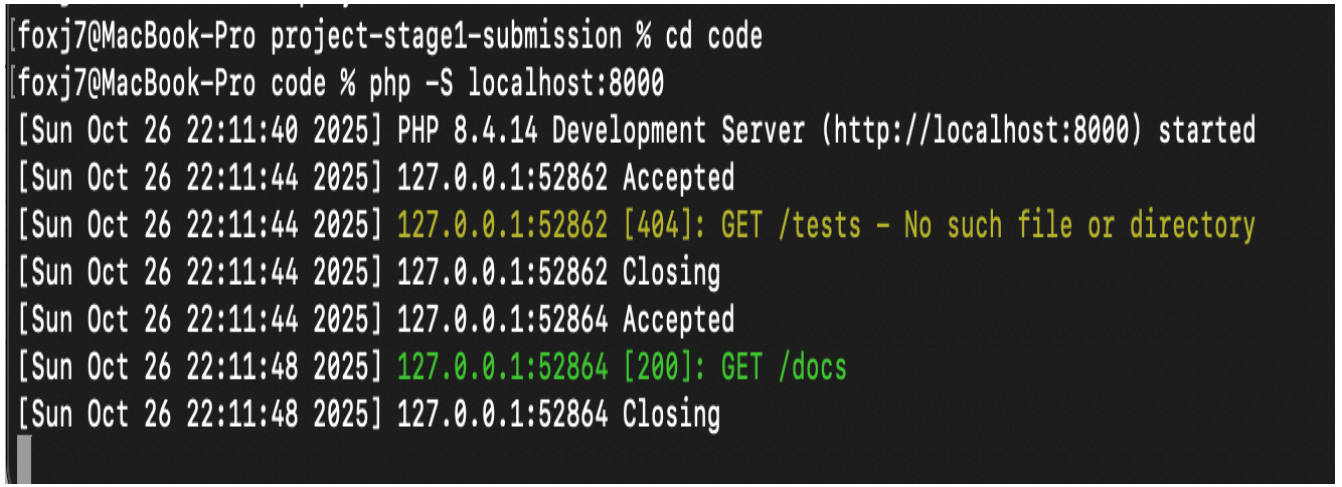
3) Start Server (PHP built-in)

From `code/`:

```
php -S localhost:8000
```

Visit:

- http://localhost:8000/docs



4) Start Server (NGINX script)

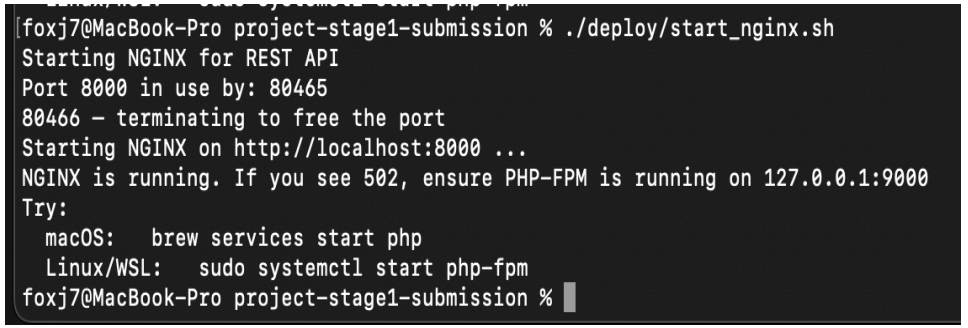
From project root:

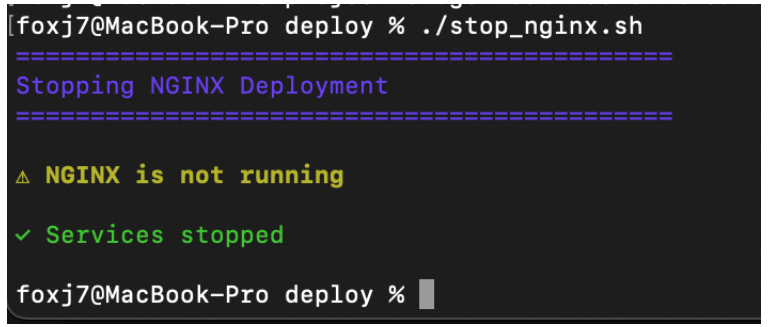
```
./deploy/start_nginx.sh
```

- Auto-generates config, starts services if needed
- Visit http://localhost:8000/docs

Stop:

```
./deploy/stop_nginx.sh
```



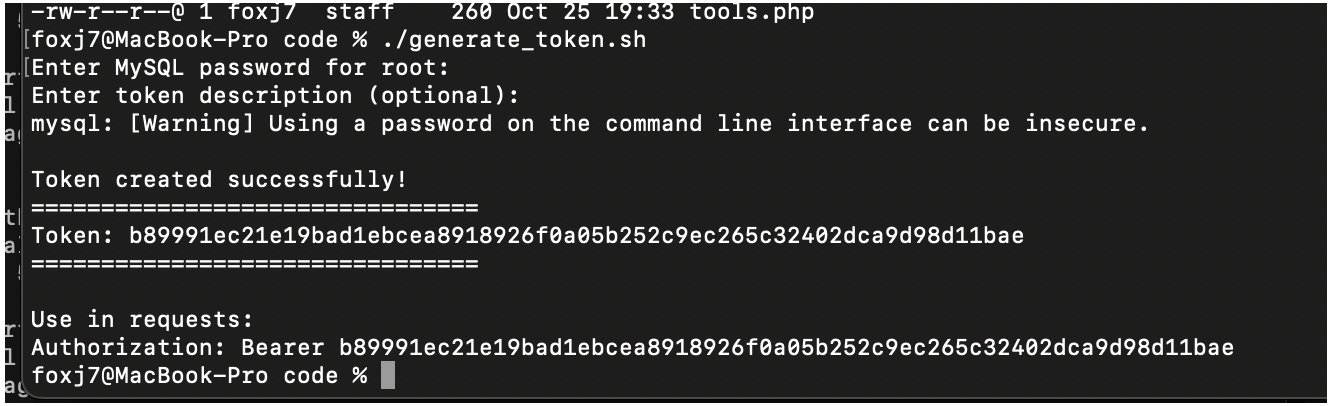


5) Generate Auth Token

From `code/`:

```
./generate_token.sh
```

- Inserts token into DB
- Use “Authorization: Bearer <token>” in protected calls

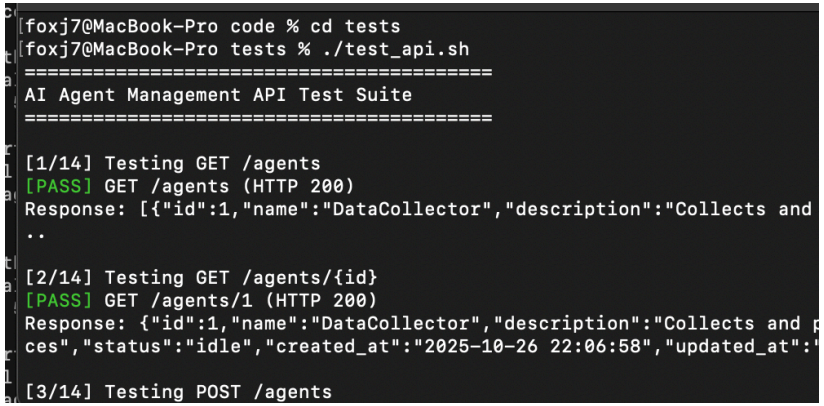


6) Test — Shell Suite

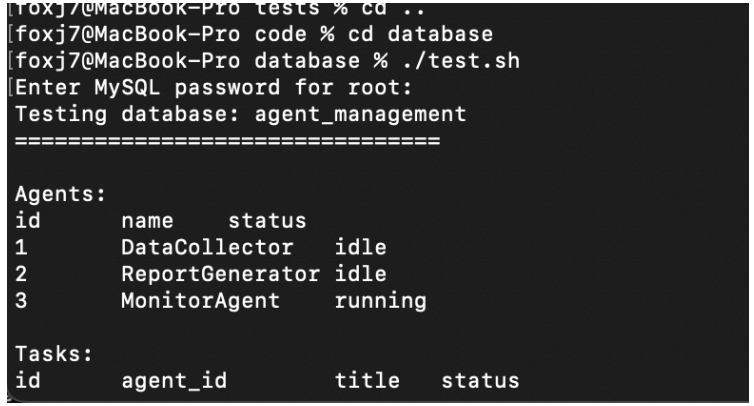
From `code/`:

```
./tests/test_api.sh
```

- Runs cURL tests against all endpoints
- Shows HTTP codes and pass/fail



cURL run



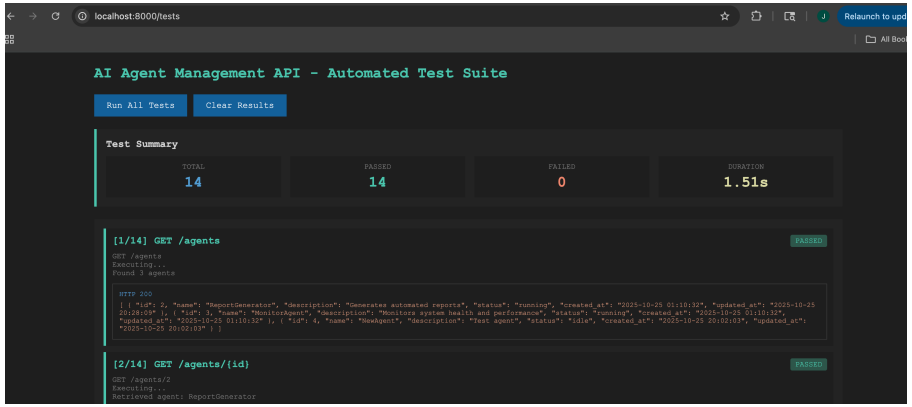
DB verification

7) Test — Browser Suite

Open:

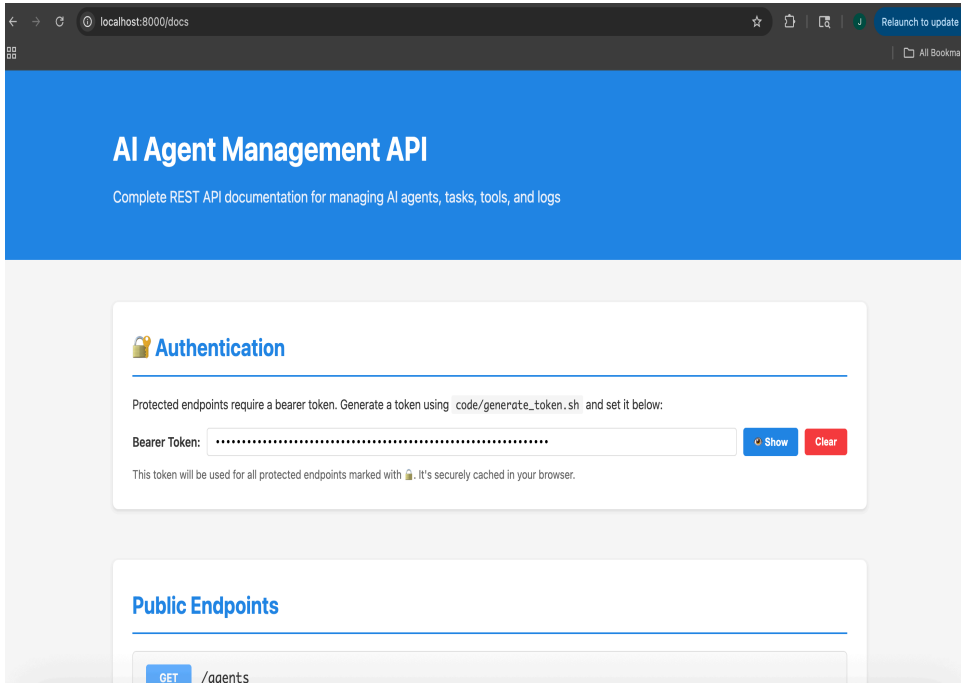
```
open code/tests/test.html        # macOS
# or
xdg-open code/tests/test.html    # Linux
```

- Click “Run All Tests”
- See per-endpoint results



8) Docs Page

- http://localhost:8000/docs
- Try endpoints interactively
- Paste the Bearer token for protected routes



9) Troubleshooting (Quick)

- DB errors: check `code/.env` values
- 502 in NGINX: ensure PHP-FPM is running
  - macOS: `brew services start php`
  - Linux/WSL: `sudo systemctl start php-fpm`
- Port busy: script frees 8000 automatically

10) What’s Included

- REST API (PHP) with MySQL (PDO)
- Auth via `api_tokens` table
- `./deploy/start_nginx.sh` + `./deploy/stop_nginx.sh`
- Docs page + two test suites

DB Troubleshooting

- Access denied: check `DB_USER/DB_PASS`
- Unknown DB: run setup.sh
- Not running: start MySQL (brew/systemctl)

Ports & Tips

- Default port: 8000 (script frees if busy)
- Use `php –S` for quick dev, NGINX for parity