API Module Documentation

File: `api/game_api.py`

What It Does

Handles all HTTP REST API communication with the cloud-based game server. Manages authentication, game state polling, score submission, and leaderboard retrieval.

Main Class: `GameAPI`

What It Does

1. Authentication - Login with credentials and manage bearer token
2. Game initialization - Poll for game creation (status=initiated)
3. Game start monitoring - Continuously poll until game starts (status=playing)
4. Score submission - Submit final team scores to backend
5. Leaderboard retrieval - Fetch current top teams

Configuration

API Settings (from `config.py`)

```
base_url: str       # API endpoint (production/development)
email: str          # Authentication email
password: str       # Authentication password
game_id: str        # Unique game identifier UUID
game_name: str      # Display name for game

# Timeouts (seconds)
auth_timeout: 30
game_status_timeout: 8
submit_score_timeout: 20
leaderboard_timeout: 12
```

Key Methods

1. `authenticate() -> bool`

Authenticate with API and obtain bearer token

Flow:

1. POST credentials to `/login2`
2. Extract bearer token from response

3. Store token in `self.token`
4. Configure authorization header

Example:

```
api = GameAPI()
if api.authenticate():
    print("Authenticated successfully")
```

---

2. `poll_game_initialization() -> Optional[Dict]`

Poll for game initialization (Step 1 of game flow)

Endpoint: `GET /game-result?status=initiated&load_participant=true&gameID={GAMEID}&limit=1`

Returns: Game data dictionary when status is 'initiated'

Response structure:

```
{
    'id': 'uuid-game-id',
    'name': 'Team Alpha',
    'nodeIDs': [
        {'userID': 'user-uuid-1', 'name': 'Player 1'},
        {'userID': 'user-uuid-2', 'name': 'Player 2'}
    ],
    'status': 'initiated'
}
```

Usage:

```
game_data = api.poll_game_initialization()
if game_data:
    team_name = game_data['name']
    game_result_id = game_data['id']
```

---

3. `poll_game_start_continuous() -> Optional[Dict]`

Continuously poll until game starts (Step 2 of game flow)

Endpoint: `GET /game-result/{game_result_id}`

Parameters:

- `game_result_id` - Game session UUID
- `submit_score_flag_ref` - Reference to external submission flag
- `started_flag_ref` - Reference to game started flag
- `cancel_flag_ref` - Reference to cancellation flag
- `game_stopped_check` - Optional callback to check if game stopped

Polling behavior:

- Polls every 2 seconds until status changes
- Monitors for three states: 'playing', 'cancel', 'submit_triggered'
- Stops polling when game starts or is cancelled

Returns: Game data when status is 'playing' or cancellation info

Response states:

```
# Game Started
{'status': 'playing', 'id': 'uuid', ...}

# Game Cancelled
{'status': 'cancel', 'cancelled': True}

# Score Submission Triggered
{'status': 'submit_triggered'}
```

Usage:

```
game_data = api.poll_game_start_continuous(
    game_result_id=game_id,
    submit_score_flag_ref=lambda: self.submit_score_flag,
    started_flag_ref=lambda: self.started_flag,
    cancel_flag_ref=lambda x: setattr(self, 'cancel_flag', x)
)

if game_data and game_data.get('status') == 'playing':
    # Start the game
    start_game()
```

---

4. `submit_final_scores() -> bool`

Submit final game scores to backend (Step 3 of game flow)

Endpoint: `POST /game-result/scoring`

Request body:

```
{
    "gameResultID": "uuid-game-id",
    "individualScore": [
        {"userID": "user-uuid-1", "nodeID": 1, "score": 45},
        {"userID": "user-uuid-2", "nodeID": 2, "score": 45}
    ]
}
```

Returns: `True` if submission successful

Usage:

```
individual_scores = [
    {"userID": "user1", "nodeID": 1, "score": 30},
    {"userID": "user2", "nodeID": 2, "score": 30}
]

success = api.submit_final_scores(game_result_id, individual_scores)
if success:
    print("Scores submitted successfully")
```

---

5. `get_leaderboard() -> Tuple[List, Dict]`

Retrieve current game leaderboard

Endpoint: `GET /leaderboard/dashboard/based?source=game&nameGame={game_name}`

Returns: Tuple of (top_teams_list, last_played_team)

Response structure:

```
# Top teams list
[
    ("Team Alpha", 90, 450),  # (name, total_score, weighted_points)
    ("Team Beta", 85, 420),
    ...
]

# Last played team
{
    'name': 'Team Gamma',
    'total_score': 60,
    'weighted_points': 300,
    'rank': 15
}
```

Usage:

```
top_teams, last_played = api.get_leaderboard()

for i, (name, score, weighted) in enumerate(top_teams):
    print(f"{i+1}. {name}: {score} points")

if last_played:
    print(f"Your team: {last_played['name']} (Rank:
{last_played['rank']})")
```

HTTP Session Management

Retry Strategy

```
Retry(
    total=3,                    # Max 3 retries
    status_forcelist=[429, 500, 502, 503, 504],
    allowed_methods=["HEAD", "GET", "OPTIONS", "POST"],
    backoff_factor=1            # 1s, 2s, 4s between retries
)
```

Timeout Configuration

- Authentication: 30 seconds
- Game status polling: 8 seconds
- Score submission: 20 seconds
- Leaderboard: 12 seconds

Error Handling

Exception Types

- `ConnectionError` - Network connectivity issues
- `Timeout` - Request exceeded timeout duration
- `RequestException` - General request failure
- `ValueError` - JSON parsing errors

Logging

All API operations are logged with detailed information:

- Request URL and parameters
- Response status code and timing
- Error details and stack traces
- Authentication state changes

Example log output:

```
2025-01-15 14:30:00 - api.game_api - INFO -
========================================
2025-01-15 14:30:00 - api.game_api - INFO - 🔐 AUTHENTICATION ATTEMPT 1/3
2025-01-15 14:30:00 - api.game_api - INFO -
========================================
2025-01-15 14:30:00 - api.game_api - INFO - 📡 POST
https://api.example.com/login2
2025-01-15 14:30:01 - api.game_api - INFO - Response received in 0.85
seconds
2025-01-15 14:30:01 - api.game_api - INFO - ✅ Status Code: 200
2025-01-15 14:30:01 - api.game_api - INFO -
========================================
2025-01-15 14:30:01 - api.game_api - INFO - ✅ AUTHENTICATION SUCCESSFUL!
2025-01-15 14:30:01 - api.game_api - INFO -
========================================
```

Game Flow Sequence

Complete API Communication Flow

```
1. AUTHENTICATION
   api.authenticate()
   ↓ (bearer token obtained)

2. POLL FOR INITIALIZATION
   while True:
       game_data = api.poll_game_initialization()
       if game_data:
           break
       sleep(3 seconds)
   ↓ (game created by admin)

3. POLL FOR START
   while True:
       game_data = api.poll_game_start_continuous(game_result_id)
       if game_data['status'] == 'playing':
           break
       sleep(2 seconds)
   ↓ (admin clicked "Start Game")

4. GAMEPLAY
   (React to triggers - button presses or serial events)
   Track correct/wrong/miss counts
   ↓ (timer reaches 0:00)

5. SUBMIT SCORES
   success = api.submit_final_scores(game_result_id, scores)
```

```
    ↓ (scores saved to database)

6. GET LEADERBOARD
    top_teams, last_played = api.get_leaderboard()
    ↓ (display results)
```

Advanced Features

1. Continuous Polling Pattern

The API supports continuous polling that automatically handles:

- Game start detection
- Game cancellation detection
- External submission triggers
- Game stopped checks

Example:

```
def game_stopped_check():
    # Custom logic to detect if game has stopped
    return not gameStarted

game_data = api.poll_game_start_continuous(
    game_result_id=game_id,
    submit_score_flag_ref=lambda: self.submit_flag,
    started_flag_ref=lambda: self.started_flag,
    cancel_flag_ref=lambda x: setattr(self, 'cancel_flag', x),
    game_stopped_check=game_stopped_check
)
```

2. Authorization Header Management

The API automatically manages the bearer token authorization header:

```
headers = {"Authorization": f"Bearer {token}"}
```

Automatic re-authentication if token is missing:

```
if not self._ensure_authenticated():
    return None
```

3. Connection Information

Get current API connection status:

```
info = api.get_connection_info()
# Returns:
# {
#     'base_url': 'https://api.example.com',
#     'game_id': 'uuid-123',
#     'game_name': 'Fast Reaction',
#     'authenticated': True,
#     'token_length': 254
# }
```

Dependencies

Required Packages

```
import requests                    # HTTP client
from requests.adapters import HTTPAdapter
from urllib3.util.retry import Retry
from requests.exceptions import ConnectionError, Timeout, RequestException
```

Internal Dependencies

```
from config import config          # Configuration settings
from utils.logger import get_logger  # Logging utilities
```

Testing & Debugging

Check Initialization

```
if api.is_initialized():
    print("API ready to use")
```

Verify Authorization

```
if api.verify_authorization_header():
    print("Bearer token configured correctly")
```

Get Game Flow Status

```
status = api.get_game_flow_status()
print(f"Current flow step: {status['flow_step']}")
# Possible values:
# - 'authentication_required'
# - 'waiting_for_initialization'
# - 'initialized_waiting_for_start'
# - 'game_active'
# - 'submitting_scores'
```

FastReaction-Specific Notes

Scoring Format

FastReaction uses simple integer scores representing reaction performance:

- Typical range: 0-100 points
- Score calculation: +1 correct, -1 wrong/miss
- Final score can be positive or negative

Game Mechanics

The API handles discrete reaction events:

- Each correct/wrong/miss is a separate event
- Score accumulates over game duration
- Timer-based gameplay (default 60 seconds)

API Usage Pattern

```
# FastReaction typical flow
1. Authenticate
2. Poll for game initialization
3. Poll for game start
4. Play game (track reactions locally)
5. Submit final score (sum of all reactions)
6. Display leaderboard
```