# Golf Calendar

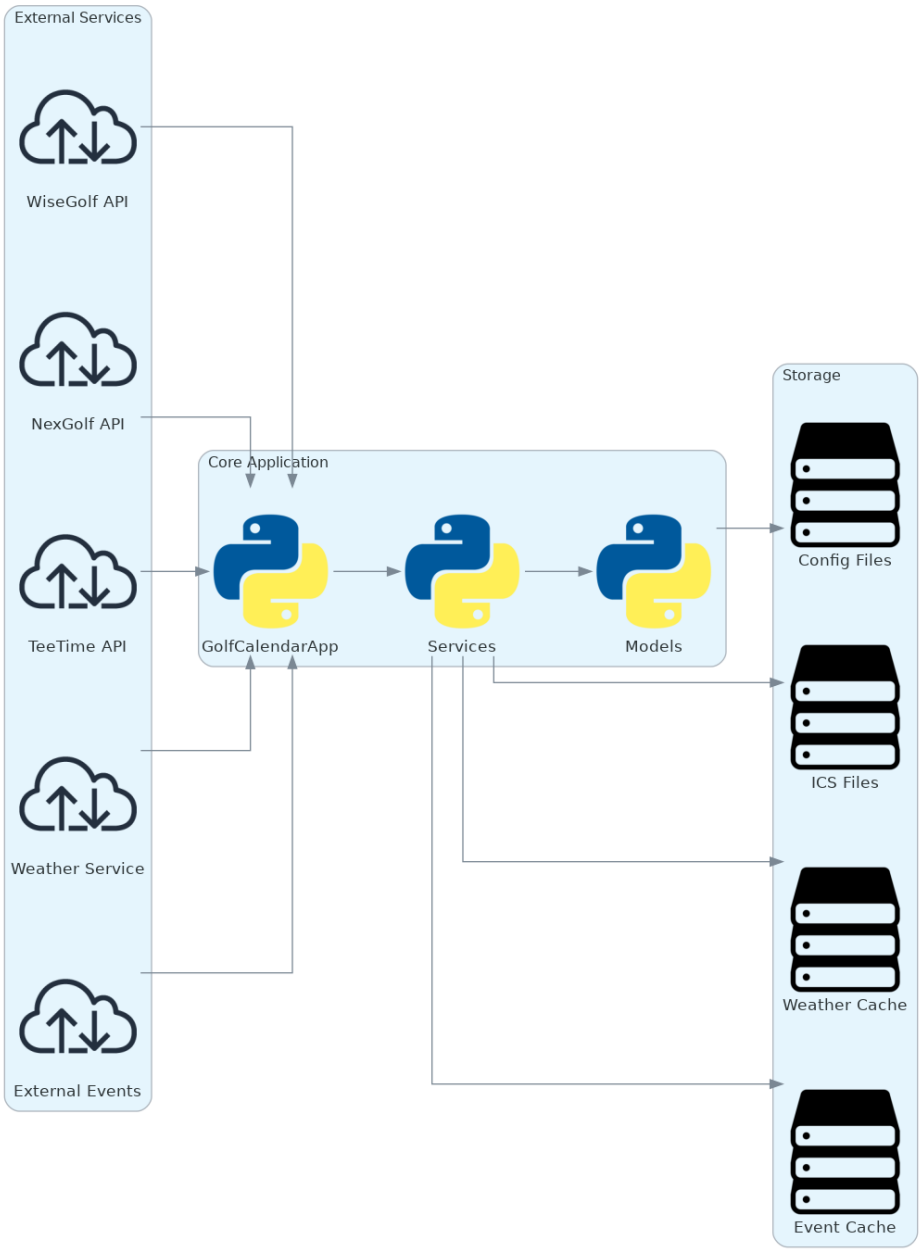
Technical Documentation

Generated: 2024-12-26

## Table of Contents

## 1. System Architecture

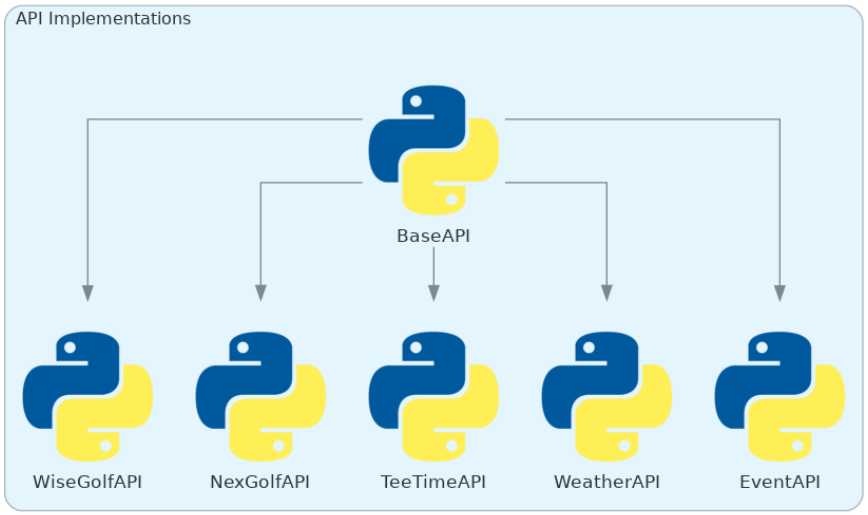
The Golf Calendar application follows a modular architecture with clear separation of concerns. The system is composed of several key components that work together to provide golf reservation management and calendar generation capabilities. The architecture includes integration with weather services and external event systems, with dedicated caching mechanisms for optimal performance.




Golf Calendar System Architecture

## 2. API Layer Analysis

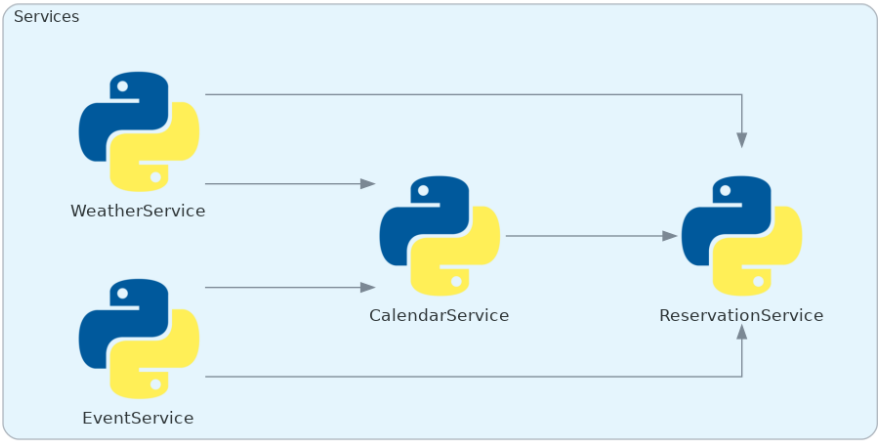
The API layer implements adapters for different golf booking systems, weather services, and external event providers. It follows the adapter pattern with a base API class that defines the common interface. The WeatherAPI provides forecast data integration, while the EventAPI handles external event synchronization.




API Layer Structure

## 3. Service Layer Details

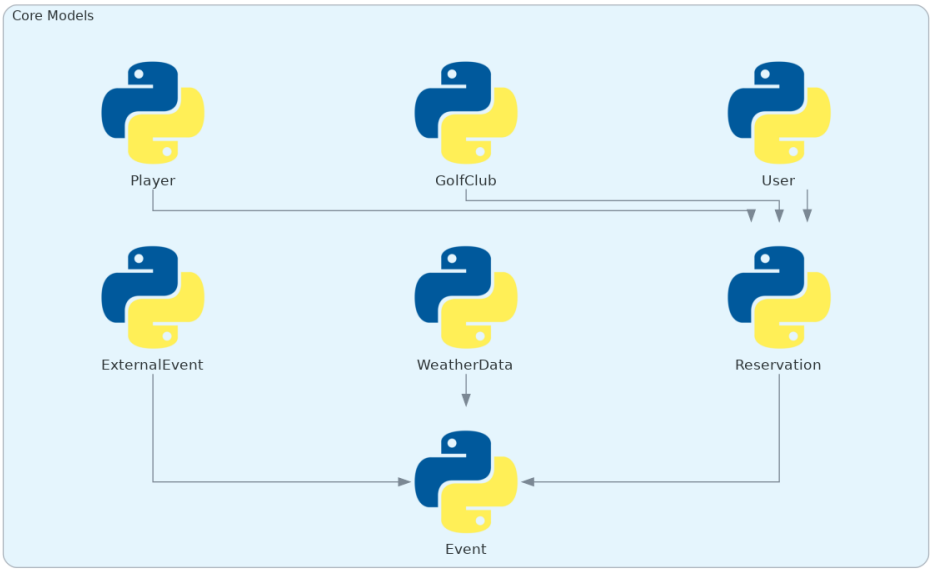
The service layer contains the core business logic of the application. The WeatherService manages weather data retrieval and caching, while the EventService handles external event integration. These services work together with the ReservationService and CalendarService to provide comprehensive event information.




Service Layer

## 4. Data Models

The data models represent the core domain entities of the application. The WeatherData model stores temperature, precipitation, and wind information, while the ExternalEvent model handles additional event data. These models integrate with the core Reservation and Event models to provide comprehensive calendar entries.




Data Models

## 5. Configuration System

The configuration system uses JSON files to store user and club configurations, as well as weather service API keys and external event source settings. It supports flexible configuration of multiple golf clubs, users, authentication methods, and integration points.

## 6. API Data Structures

The Golf Calendar system interacts with various external APIs, each with its own data structures and formats. Below are the key data structures for each API:

### WiseGolf API Data Structures

• Reservation:
  - id: Unique reservation identifier
  - datetime: Tee time date and time
  - duration: Duration in minutes
  - course_id: Reference to course
  - players: Array of player references
  - status: Booking status

• Player:
  - id: Player identifier
  - name: Full name
  - handicap: Current handicap
  - membership: Membership details
  - preferences: Player preferences

• Course:
  - id: Course identifier
  - name: Course name
  - holes: Number of holes
  - par: Course par
  - facilities: Available facilities

### NexGolf API Data Structures

• Booking:
  - booking_id: Unique booking reference
  - time_slot: Reserved time slot
  - member_ids: List of participating members
  - facility_id: Golf facility reference
  - booking_type: Type of reservation

• Member:
  - member_id: Member identifier
  - details: Personal information
  - membership_level: Level of membership
  - active_status: Current status

• Facility:

- facility_id: Facility identifier
- location: Geographic location
- amenities: Available services
- operating_hours: Business hours

## TeeTime API Data Structures

• Tee Time:
- slot_id: Time slot identifier
- start_time: Start time
- end_time: End time
- venue_id: Golf venue reference
- player_count: Number of players
- booking_status: Current status

• User:
- user_id: User identifier
- profile: User profile data
- booking_history: Past bookings
- preferences: User preferences

• Venue:
- venue_id: Venue identifier
- details: Venue information
- availability: Time slot availability
- pricing: Rate information

## Weather API Data Structures

• Current Weather:
- temperature: Current temperature
- precipitation: Precipitation probability
- wind_speed: Wind speed
- wind_direction: Wind direction
- humidity: Humidity percentage

• Forecast:
- hourly: Hourly forecast data
- daily: Daily forecast summary
- alerts: Weather warnings
- location: Forecast location

• Alerts:
- type: Alert type
- severity: Alert severity
- description: Alert details

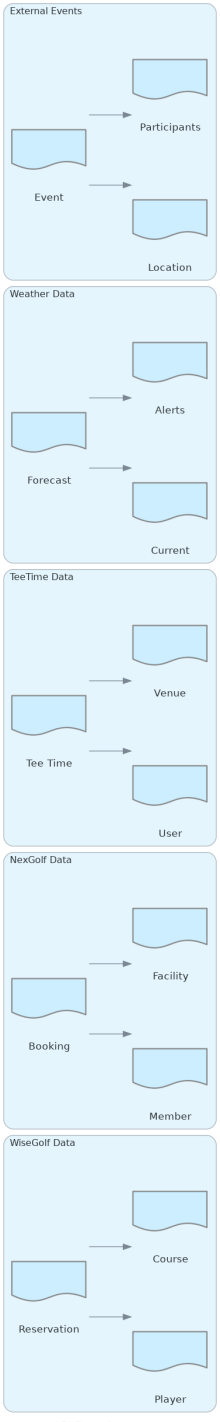
- valid_period: Validity timeframe

## External Events API Data Structures

• Event:
  - event_id: Unique event identifier
  - title: Event title
  - description: Event description
  - start_time: Event start time
  - end_time: Event end time
  - type: Event type

• Location:
  - coordinates: Geographic coordinates
  - address: Physical address
  - venue_name: Location name
  - accessibility: Access information

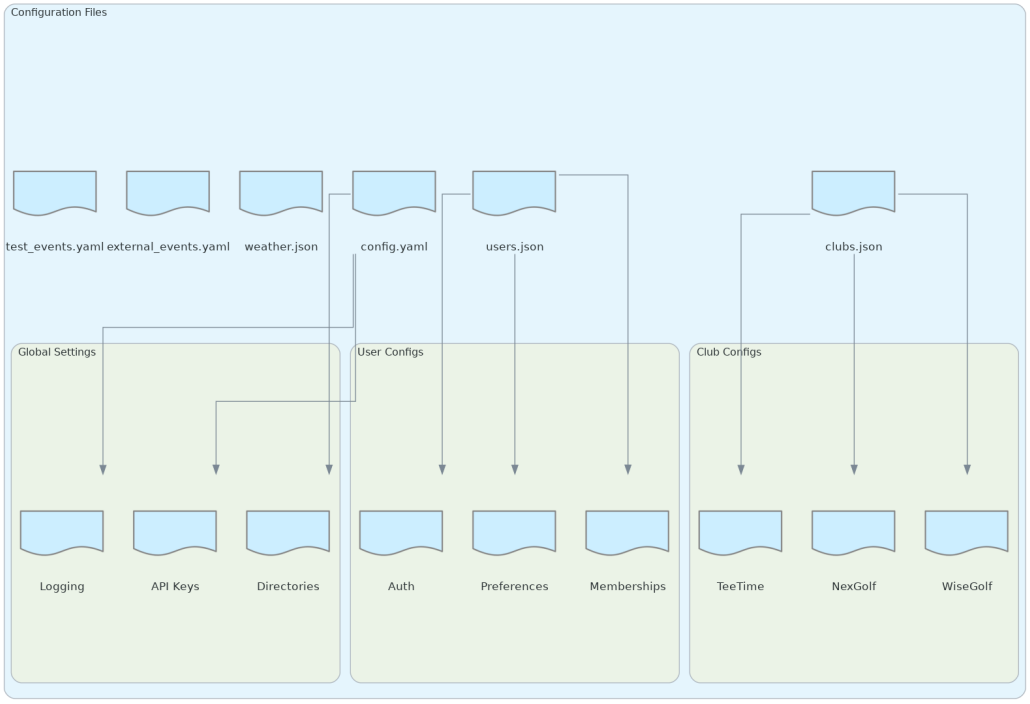
• Participants:
  - capacity: Maximum participants
  - current_count: Current participant count
  - registration_status: Registration state
  - waiting_list: Waiting list status




API Data Structures

## 7. Configuration File Structures

The Golf Calendar system uses several JSON configuration files to manage different aspects of the application. Each configuration file has a specific structure and purpose:




Configuration Files Structure

## users.json

Structure:

```
{
   "user_name": {
      "email": "user@example.com",
      "memberships": [
         {
            "club": "CLUB_CODE",
            "duration": {
               "hours": 2,
               "minutes": 0
            },
            "auth_details": {
```

```
                "token": "auth_token",
                "cookie": "session_cookie",
                "username": "login_username",
                "password": "encrypted_password"
            }
        }
    ],
    "preferences": {
        "default_duration": 120,
        "notification_email": "notifications@example.com",
        "calendar_file": "user_calendar.ics",
        "time_zone": "Europe/Helsinki"
    }
  }
}
```

Description:
- user_name: Unique identifier for each user
- email: Primary contact email
- memberships: Array of club memberships
  - club: Reference to club configuration
  - duration: Default booking duration
  - auth_details: Authentication credentials
- preferences: User-specific settings

## clubs.json

Structure:

```
{
  "CLUB_CODE": {
    "type": "wisegolf|nexgolf|teetime",
    "url": "https://api.club.com/endpoint",
    "public_url": "https://club.com/api/public",
    "cookie_name": "session_cookie",
    "auth_type": "token|cookie|basic",
    "crm": "system_type",
    "address": "Club physical address",
    "clubAbbreviation": "SHG",
    "product_ids": {
      "53": {
        "description": "18 holes",
        "group": "A"
      }
    },
    "coordinates": {
      "latitude": 60.123,
      "longitude": 24.456
    },
```

```
      "facilities": ["restaurant", "pro_shop", "driving_range"],
      "operating_hours": {
         "weekday": "06:00-22:00",
         "weekend": "07:00-21:00"
      }
   }
}
```

Description:
- CLUB_CODE: Unique identifier for each club
- type: Booking system type
- url: API endpoint URL
- auth_type: Authentication method
- product_ids: Available booking types
- coordinates: Location for weather data
- facilities: Available services

## weather.json

Structure:

```
{
   "api_key": "weather_service_api_key",
   "providers": {
      "primary": {
         "name": "openweathermap",
         "url": "https://api.openweathermap.org/data/2.5",
         "update_interval": 3600,
         "units": "metric"
      },
      "fallback": {
         "name": "weatherapi",
         "url": "https://api.weatherapi.com/v1",
         "update_interval": 7200
      }
   },
   "cache": {
      "enabled": true,
      "duration": 3600,
      "file": "weather_cache.json"
   },
   "alerts": {
      "enabled": true,
      "threshold": {
         "wind_speed": 10,
         "precipitation": 70
      }
   }
}
```

Description:
- api_key: Weather service authentication
- providers: Available weather services
- cache: Caching configuration
- alerts: Weather alert settings

## external_events.yaml

Structure:
```
events:
  - name: "Event Name"
    location: "Golf Club Name"
    coordinates:
      lat: 60.2859
      lon: 24.8427
    users:
      - "User1"
      - "User2"
    start: "2024-11-13T18:00:00"
    end: "2024-11-13T21:00:00"
    timezone: "Europe/Helsinki"
    address: "Club Address"
    repeat:
      frequency: "weekly"    # weekly or monthly
      until: "2025-04-02"    # end date for recurring events
```

Description:
- name: Event display name
- location: Golf club or venue name
- coordinates: Geographic coordinates for weather data
- users: List of users to include in the event
- start/end: ISO format datetime strings
- timezone: IANA timezone identifier
- address: Full venue address
- repeat: Optional recurring event settings
  - frequency: weekly/monthly
  - until: End date for recurrence

## test_events.yaml

Structure:
```
- name: "Test Event Name"
  location: "Test Golf Club"
  coordinates:
    lat: 59.8940
```

```
    lon: 10.8282
  users:
    - "User1"
  start_time: "tomorrow 10:00"
  end_time: "tomorrow 14:00"
  timezone: "Europe/Oslo"
  address: "Test Club Address"
```

Description:
- name: Test event identifier
- location: Test venue name
- coordinates: Location for weather testing
- users: Test users to include
- start_time/end_time: Supports dynamic time formats:
  - "tomorrow HH:MM"
  - "N days HH:MM"
  - "today HH:MM"
- timezone: IANA timezone for testing
- address: Full test venue address

Used for testing different scenarios:
1. Time ranges: tomorrow, 3 days, 7 days
2. Times of day: morning, afternoon, evening
3. Regions: Nordic, Spain, Portugal, Mediterranean

## Configuration File Attributes

Detailed documentation of all available configuration attributes and their purposes.

### users.json Attributes

| Attribute | Type | Description |
|---|---|---|
| email | string | Primary email address for user notifications and identification |
| memberships | array | List of club memberships with authentication and booking preferences |
| preferences | object | User-specific settings including default duration, timezone, and notification preferences |
| default_duration | integer | Default booking duration in minutes (typical values: 120 for 18 holes, 60 for 9 holes) |
| time_zone | string | User's timezone in IANA format (e.g., "Europe/Helsinki") |

### clubs.json Attributes

| Attribute | Type | Description |
|---|---|---|
| type | string | Booking system type (wisegolf, nexgolf, teetime). Determines API integration method |
| url | string | Primary API endpoint URL for the booking system |
| auth_type | string | Authentication method (token, cookie, basic). Each type requires specific auth_details |
| product_ids | object | Mapping of booking types to internal IDs with descriptions and grouping |
| coordinates | object | Geographic coordinates for weather data retrieval and distance calculations |
| operating_hours | object | Business hours for different days, affects booking time validation |

### weather.json Attributes

| Attribute | Type | Description |
|---|---|---|
| api_key | string | Authentication key for weather service API access |
| providers | object | Configuration for primary and fallback weather data providers |
| update_interval | integer | Time in seconds between weather data updates (typical: 3600-7200) |
| cache | object | Weather data caching settings to minimize API calls and improve performance |
| alerts | object | Weather alert thresholds and notification settings |

### events.json Attributes

| Attribute | Type | Description |
|---|---|---|
| sources | object | External event data sources with authentication and update settings |
| filters | object | Event filtering rules based on type, priority, and other criteria |
| sync | object | Synchronization settings including frequency and future event horizon |
| categories | array | List of supported event categories for filtering and display |

### logging.json Attributes

| Attribute | Type | Description |
|---|---|---|
| formatters | object | Log message format configurations for different output types |
| handlers | object | Log output handlers for file and console with rotation settings |
| loggers | object | Logger configurations for different components with log levels |
| maxBytes | integer | Maximum log file size before rotation (default: 10MB) |
| backupCount | integer | Number of rotated log files to keep (default: 5) |

## config.yaml

Structure:

```
# Global configuration parameters
timezone: "Europe/Helsinki"

# Directory paths
directories:
  ics: "ics"
  config: "config"
  logs: "logs"

# ICS file paths (override default naming)
ics_files:
  User1: "ics/User1_golf_reservations.ics"
  User2: "ics/User2_golf_reservations.ics"

# API Keys
api_keys:
  weather:
    # Spanish Meteorological Agency (AEMET)
    aemet: "your-aemet-api-key"
    # OpenWeather API (Mediterranean region)
    openweather: "your-openweather-api-key"

# Logging configuration
logging:
  dev_level: "DEBUG"
  verbose_level: "INFO"
  default_level: "WARNING"
```

Description:

- timezone: Default timezone for the application
- directories: Path configurations
  - ics: Calendar file storage location

  - config: Configuration files location
  - logs: Log files location
- ics_files: Custom calendar file paths per user
- api_keys: External service API keys
  - weather: Weather service authentication
    - aemet: Spanish weather service
    - openweather: OpenWeather API
- logging: Log level configurations
  - dev_level: Development mode logging
  - verbose_level: Verbose mode logging
  - default_level: Default logging level

### config.yaml Attributes

| Attribute | Type | Description |
|---|---|---|
| timezone | string | Default application timezone in IANA format (e.g., "Europe/Helsinki"). Used when no user-specific timezone is set. |
| directories | object | Path configurations for different file types. Supports relative and absolute paths. |
| ics_files | object | Custom calendar file path mappings per user. Overrides default naming convention. |
| api_keys.weather | object | Weather service API keys. Supports multiple providers with region-specific configurations. |
| logging | object | Logging configuration with different levels for development, verbose, and default modes. |

## YAML Configuration Files

The application uses YAML format for configuration files that require more structured and readable formats. Below are the detailed attributes for each YAML file:

### config.yaml (Primary Configuration)

| Attribute | Type | Description |
|---|---|---|
| timezone | string | Default application timezone (e.g., "Europe/Helsinki"). Required. Used as fallback when user timezone is not set. |
| directories.ics | string | Calendar files directory path. Default: "ics". Can be relative or absolute path. |
| directories.config | string | Configuration files directory path. Default: "config". Can be relative or absolute path. |
| directories.logs | string | Log files directory path. Default: "logs". Can be relative or absolute path. |
| ics_files | map<string,string> | Map of username to custom calendar file path. Optional. Overrides default "{username}_golf_reservations.ics" naming. |
| api_keys.weather.aemet | string | Spanish Meteorological Agency API key. Required for Spanish weather data. Get from: https://opendata.aemet.es/ |
| api_keys.weather.openweather | string | OpenWeather API key. Required for Mediterranean region. Default key provided but can be overridden. |
| logging.dev_level | string | Development mode log level. Default: "DEBUG". Options: DEBUG, INFO, WARNING, ERROR, CRITICAL |
| logging.verbose_level | string | Verbose mode log level. Default: "INFO". Options: DEBUG, INFO, WARNING, ERROR, CRITICAL |
| logging.default_level | string | Default log level. Default: "WARNING". Options: DEBUG, INFO, WARNING, ERROR, CRITICAL |

### external_events.yaml (External Events)

| Attribute | Type | Description |
|---|---|---|
| events[].name | string | Event display name. Required. Used in calendar entries and logs. |
| events[].location | string | Golf club or venue name. Required. Used for location display and weather lookup. |
| events[].coordinates.lat | float | Venue latitude. Required. Used for weather data retrieval. Range: -90 to 90. |
| events[].coordinates.lon | float | Venue longitude. Required. Used for weather data retrieval. Range: -180 to 180. |
| events[].users | string[] | List of usernames to include in event. Required. Must match user configuration names. |
| events[].start | datetime | Event start time in ISO format (YYYY-MM-DDTHH:MM:SS). Required. Must be in specified timezone. |
| events[].end | datetime | Event end time in ISO format (YYYY-MM-DDTHH:MM:SS). Required. Must be after start time. |
| events[].timezone | string | IANA timezone identifier. Required. Used for time conversions and weather data. |
| events[].address | string | Full venue address. Optional. Used in calendar location field. |
| events[].repeat.frequency | string | Recurrence frequency. Optional. Values: "weekly" or "monthly". Creates recurring events. |
| events[].repeat.until | date | Recurrence end date in ISO format (YYYY-MM-DD). Required if repeat is set. Last instance date. |

### test_events.yaml (Test Configuration)

| Attribute | Type | Description |
|---|---|---|
| name | string | Test event identifier. Required. Used to identify test case purpose. |
| location | string | Test venue name. Required. Used to test location handling. |
| coordinates | object | Geographic coordinates for weather testing. Required. Tests different weather service regions. |
| users | string[] | Test user list. Required. Must exist in user configuration. |
| start_time | string | Dynamic start time. Required. Formats: "tomorrow HH:MM", "N days HH:MM", "today HH:MM". |
| end_time | string | Dynamic end time. Required. Same format as start_time. Must be after start_time. |
| timezone | string | IANA timezone. Required. Tests timezone handling and conversions. |
| address | string | Test venue address. Optional. Tests address formatting and display. |

## Core Configuration Files

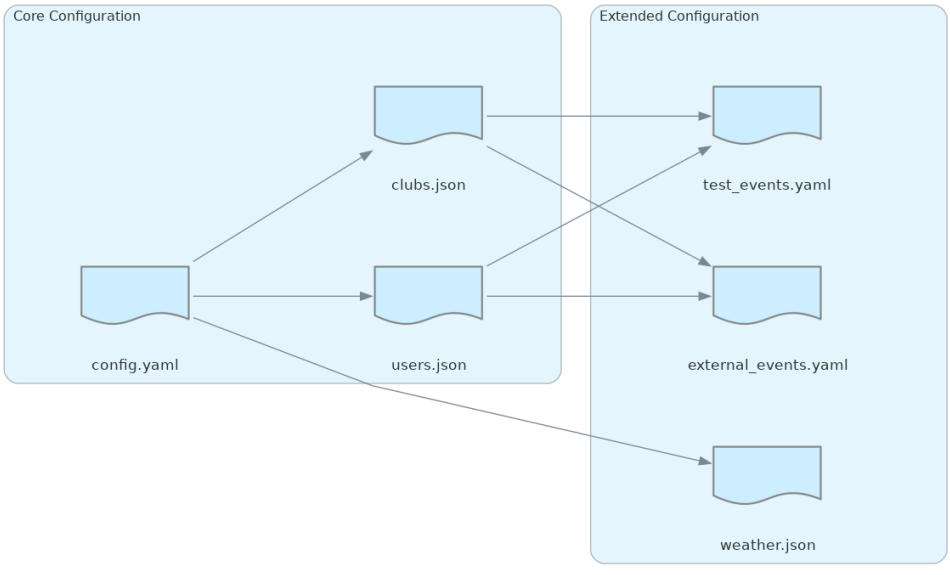
The application relies primarily on three key configuration files that must be properly configured for basic functionality:

1. config.yaml - Global Application Settings
   - Contains essential application-wide settings
   - Defines directory structures and paths
   - Manages API keys and authentication
   - Controls logging behavior
   - Required for application startup

2. users.json - User Management
   - Stores user profiles and credentials
   - Manages club memberships
   - Defines user preferences
   - Handles authentication details
   - Required for booking functionality

3. clubs.json - Golf Club Configuration
   - Defines supported golf clubs
   - Contains API endpoints and authentication
   - Manages booking types and products
   - Stores facility information
   - Required for reservation system

These files form the core configuration backbone and must be present and properly formatted for the application to function correctly.




Configuration Dependencies

# Weather Service APIs

The application uses multiple weather service APIs based on geographic location. Each service has its own request format, response structure, and specific features.

### Weather Service Selection

The appropriate weather service is selected based on coordinates:

• Nordic Region (55°N-72°N, 3°E-32°E):
  - Service: MET Norway (met.no)
  - Coverage: Norway, Sweden, Finland
  - Free service, no API key required
  - Hourly forecasts up to 48 hours

• Portugal (-9.5°W to -6.2°W):
  - Service: IPMA (Portuguese Institute for Sea and Atmosphere)
  - Coverage: Portugal mainland and islands
  - Free service, no API key required
  - Daily forecasts with 3-hour resolution

• Spain (-7°W to 5°E):
  - Service: AEMET (Spanish State Meteorological Agency)
  - Coverage: Spain mainland and islands
  - Requires API key from opendata.aemet.es
  - Hourly and daily forecasts

• Mediterranean Region (Other locations):
  - Service: OpenWeather API
  - Global coverage as fallback
  - Requires API key
  - Various forecast products

### MET Norway (met.no)

Request Format:
GET https://api.met.no/weatherapi/locationforecast/2.0/complete
Parameters:
- lat: Latitude (-90 to 90)
- lon: Longitude (-180 to 180)
- altitude: Optional elevation in meters

Headers Required:
- User-Agent: Application identifier (required)
- Accept: application/json

Example Response:
{

```
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [24.8427, 60.2859, 15]
  },
  "properties": {
    "timeseries": [
      {
        "time": "2024-01-26T12:00:00Z",
        "data": {
          "instant": {
            "details": {
              "air_temperature": -2.3,
              "wind_speed": 4.2,
              "wind_from_direction": 180,
              "relative_humidity": 85.0
            }
          },
          "next_1_hours": {
            "summary": {
              "symbol_code": "cloudy"
            },
            "details": {
              "precipitation_amount": 0.2
            }
          }
        }
      }
    ]
  }
}
```


Key Attributes:
- time: ISO 8601 timestamp
- air_temperature: Celsius
- wind_speed: meters/second
- wind_from_direction: degrees (0-360)
- precipitation_amount: millimeters
- symbol_code: Weather condition code
- relative_humidity: percentage

### IPMA (Portuguese Weather Service)

Request Format:
GET https://api.ipma.pt/open-data/forecast/meteorology/cities/daily/{city_id}.json

Parameters:
- city_id: IPMA location identifier

Example Response:

```
{
   "data": [
      {
         "precipitaProb": 85.0,
         "tMin": 12.4,
         "tMax": 18.7,
         "predWindDir": "S",
         "idWeatherType": 6,
         "classWindSpeed": 2,
         "forecastDate": "2024-01-26"
      }
   ],
   "globalIdLocal": 1110600,
   "dataUpdate": "2024-01-26T09:32:51"
}
```


Key Attributes:
- precipitaProb: Precipitation probability (0-100)
- tMin/tMax: Temperature range in Celsius
- predWindDir: Wind direction (N,S,E,W,NE,SE,SW,NW)
- idWeatherType: Weather condition code
- classWindSpeed: Wind speed class (1-9)
- forecastDate: YYYY-MM-DD format

### AEMET (Spanish Weather Service)

Request Format:
1. Get Data URL:
GET https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/horaria/{municipality_id}
Headers:
- api_key: Your AEMET API key

2. Fetch Forecast:
GET {data_url} (from step 1 response)

Example Response:

```
{
   "elaborado": "2024-01-26T09:00:00",
   "prediccion": {
      "dia": [
         {
            "fecha": "2024-01-26",
            "temperatura": {
               "dato": [
                  {
                     "hora": 6,
                     "valor": 15.4
                  }
```

```
            ]
         },
         "precipitacion": {
            "dato": [
               {
                  "hora": 6,
                  "valor": 0.0
               }
            ]
         },
         "viento": [
            {
               "direccion": "S",
               "velocidad": 15
            }
         ]
      }
   ]
}
}
```


Key Attributes:
- elaborado: Forecast generation time
- fecha: Date YYYY-MM-DD
- hora: Hour (0-23)
- temperatura.valor: Celsius
- precipitacion.valor: mm/hour
- viento.velocidad: km/h
- viento.direccion: Wind direction

### OpenWeather API (Mediterranean/Fallback)

Request Format:
GET https://api.openweathermap.org/data/2.5/onecall
Parameters:
- lat: Latitude
- lon: Longitude
- appid: API key
- units: metric
- exclude: Optional parts to exclude

Example Response:


```
{
   "lat": 36.7584,
   "lon": 31.5876,
   "timezone": "Europe/Istanbul",
   "current": {
      "dt": 1706277600,
      "temp": 18.2,
```

```
      "feels_like": 17.8,
      "humidity": 65,
      "wind_speed": 3.6,
      "wind_deg": 180,
      "weather": [
        {
          "id": 800,
          "main": "Clear",
          "description": "clear sky"
        }
      ]
    },
    "hourly": [
      {
        "dt": 1706281200,
        "temp": 18.5,
        "pop": 0.2,
        "weather": [{"id": 800, "main": "Clear"}]
      }
    ]
}
```


Key Attributes:
- dt: Unix timestamp
- temp: Temperature in Celsius
- feels_like: Apparent temperature
- humidity: Relative humidity %
- wind_speed: meters/second
- wind_deg: degrees (meteorological)
- pop: Probability of precipitation
- weather.id: Condition code
- weather.description: Human readable description

### Weather Data Processing

The application processes weather data uniformly regardless of the source:

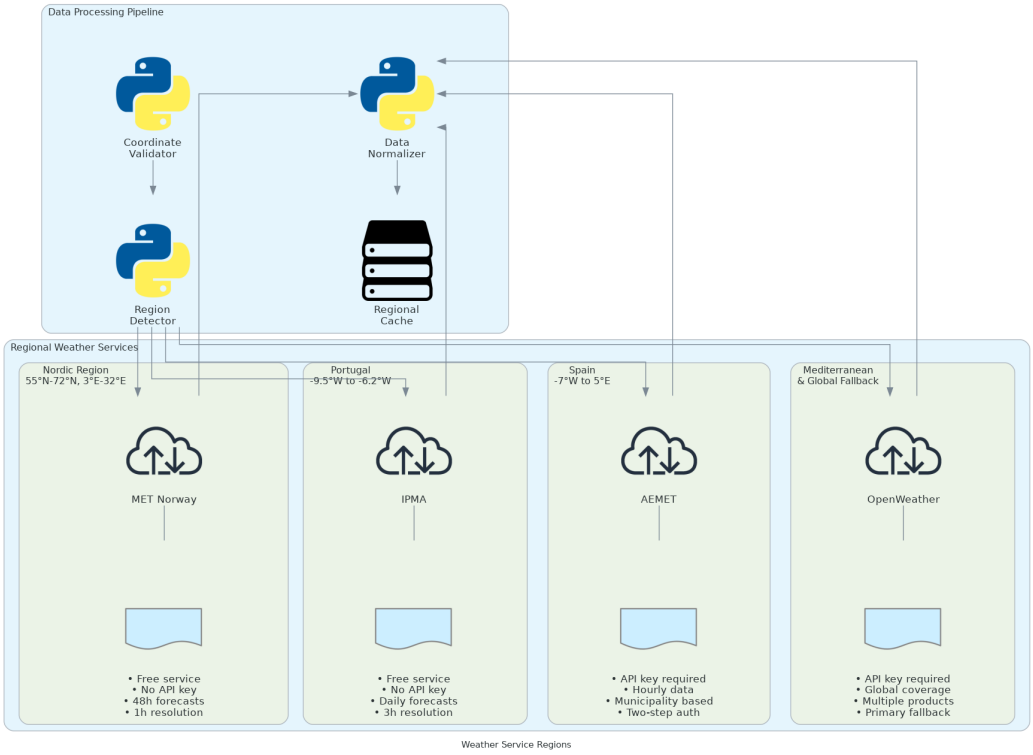
1. Data Retrieval:
   - Select appropriate service based on coordinates
   - Fetch data with error handling and retries
   - Cache responses to minimize API calls

2. Data Normalization:
   - Convert all temperatures to Celsius
   - Standardize wind speeds to m/s
   - Normalize precipitation to mm
   - Convert timestamps to local timezone

3. Forecast Assembly:
   - Combine data points for event duration

- Calculate averages and extremes
- Generate human-readable summary
- Add weather alerts if applicable

4. Cache Management:
   - Store processed data
   - Update based on configured intervals
   - Separate caches per region
   - Automatic cache invalidation



## Weather Service Error Handling

The application implements comprehensive error handling and fallback strategies for weather data retrieval:

### Error Handling by Service

1. MET Norway (met.no):
   - Rate Limiting: Exponential backoff with max 3 retries
   - Invalid Coordinates: Fallback to nearest valid point
   - Service Outage: Switch to OpenWeather API
   - Response Validation:
     • Missing data points: Interpolate from surrounding times
     • Invalid values: Use reasonable defaults
     • Timezone issues: Convert all to UTC then local

2. IPMA (Portugal):
   - City ID Not Found: Use nearest city based on coordinates
   - Missing Daily Data: Fall back to 3-hour forecasts
   - Authentication Issues: Retry with delay
   - Data Quality:
     • Temperature range validation
     • Wind speed class conversion
     • Precipitation probability normalization

3. AEMET (Spain):
   - Two-Step Request Handling:
     • Step 1: Get data URL with retry on failure
     • Step 2: Fetch actual data with separate retry logic
   - API Key Issues:
     • Validate key before requests
     • Auto-refresh if expired
     • Fall back to OpenWeather if key invalid
   - Municipality Lookup:
     • Cache municipality IDs
     • Use nearest if exact match not found
     • Fall back to provincial forecast

4. OpenWeather (Mediterranean/Fallback):
   - Primary Fallback Service:
     • Used when regional services fail
     • Provides consistent data format
     • Global coverage
   - API Key Management:
     • Rotate between multiple keys if available
     • Monitor usage limits
     • Implement rate limiting

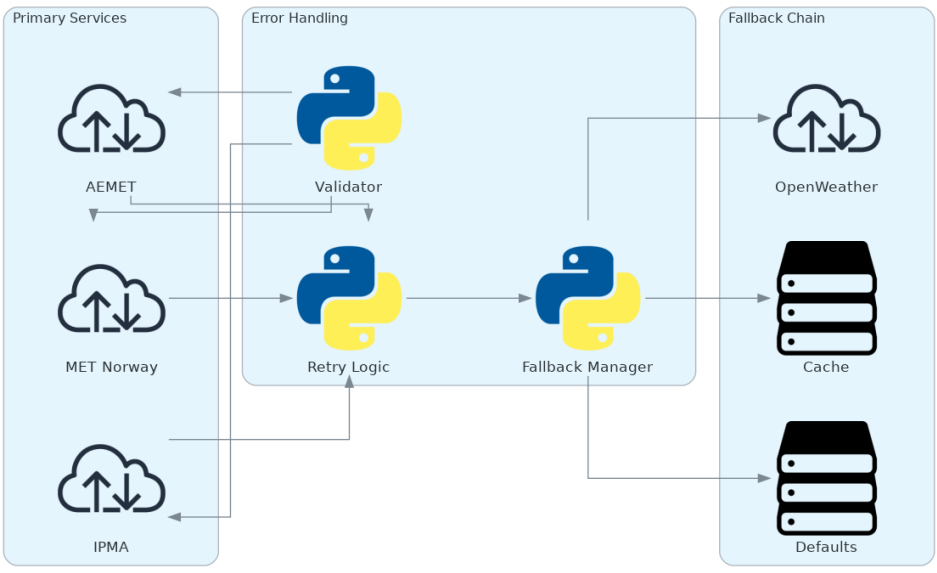
### Fallback Strategy Implementation

1. Service Selection Fallbacks:
   - Primary: Region-specific service (MET/IPMA/AEMET)
   - Secondary: OpenWeather API
   - Tertiary: Cached historical data
   - Last Resort: Default weather parameters

2. Data Quality Assurance:
   - Validate all incoming data
   - Check value ranges and units
   - Ensure timestamp consistency
   - Verify coordinate boundaries

3. Cache Management:
   - Store successful responses
   - Implement sliding expiration
   - Separate caches per region
   - Progressive data refresh

4. Recovery Mechanisms:
   - Automatic service switching
   - Graceful degradation
   - Partial data handling
   - User notification




Weather Service Fallbacks

### Weather Service Error Codes and Handling

| Attribute | Type | Description |
|---|---|---|
| Rate Limit Exceeded | HTTP 429 | Implement exponential backoff, retry after specified delay, switch to fallback if persistent. |
| Invalid API Key | HTTP 401/403 | Validate key, attempt refresh if possible, switch to fallback service if unresolvable. |
| Service Unavailable | HTTP 503 | Retry with backoff, switch to fallback service after max retries, use cached data if available. |
| Invalid Coordinates | HTTP 400 | Validate coordinates before request, use nearest valid point, fall back to regional defaults. |
| Timeout | Network | Implement request timeout, retry with increased timeout, switch to fallback after max attempts. |
| Malformed Response | Parse Error | Validate response schema, attempt partial data extraction, use cached data if parsing fails. |
| Missing Data Points | Data Quality | Interpolate from available data, use historical averages, fall back to conservative estimates. |
| Invalid Values | Data Quality | Apply range validation, use nearest valid value, fall back to regional averages if necessary. |

## CRM API Documentation

The Golf Calendar system integrates with multiple CRM systems for golf club booking management. Each system has its own API structure, authentication methods, and data formats.

### WiseGolf API

Base URL: https://api.wisegolf.club/v2

Authentication:
- Type: Bearer Token
- Header: Authorization: Bearer <token>
- Token Validity: 24 hours
- Refresh: POST /auth/refresh with refresh_token

Key Endpoints:

1. Authentication
   POST /auth/login
   {
    "username": "user@example.com",
    "password": "encrypted_password"
   }
   Response:
   {
    "access_token": "jwt_token",
    "refresh_token": "refresh_token",
    "expires_in": 86400
   }

2. Tee Time Search
   GET /tee-times/search
   Parameters:
   - date: YYYY-MM-DD
   - course_id: integer
   - players: integer (1-4)
   - start_time: HH:MM
   - end_time: HH:MM

   Response:
   {
    "available_times": [
     {
      "id": "12345",
      "course_id": 1,
      "datetime": "2024-01-26T10:00:00Z",
      "available_slots": 4,
      "price_category": "member",

```
      "duration_minutes": 120,
      "booking_restrictions": {
        "min_players": 1,
        "max_players": 4,
        "member_only": true
      }
    }
  ],
  "course_info": {
    "name": "Main Course",
    "holes": 18,
    "walking_time": 120
  }
}
```


3. Reservation Creation

POST /reservations

Body:


```
{
  "tee_time_id": "12345",
  "players": [
    {
      "member_id": "M123",
      "name": "John Doe",
      "handicap": 15.4,
      "guest": false
    }
  ],
  "special_requests": {
    "cart": true,
    "rental_clubs": false
  }
}
```


Response:


```
{
  "reservation_id": "R789",
  "confirmation_code": "WG2024123",
  "status": "confirmed",
  "payment_required": false,
  "calendar_entry": {
    "start": "2024-01-26T10:00:00Z",
    "end": "2024-01-26T12:00:00Z",
    "location": {
      "name": "Golf Club Name",
      "address": "Club Address",
      "coordinates": {
        "lat": 60.2859,
        "lon": 24.8427
      }
```

```
    }
   }
  }
```

4. Member Profile
   GET /clubs/{club_id}/members/{member_id}
   Response:

```
  {
   "member_id": "M123",
   "status": "active",
   "membership_type": "full",
   "handicap": 15.4,
   "booking_privileges": {
    "max_advance_days": 14,
    "max_active_bookings": 3,
    "guest_allowed": true
   },
   "preferences": {
    "preferred_tee_times": ["morning", "afternoon"],
    "notifications": {
     "email": true,
     "sms": false
    }
   }
  }
```

Key Attributes:
- member_id: Unique member identifier
- handicap: Current playing handicap
- booking_privileges: Member-specific booking rules
- tee_time_id: Unique identifier for available time slot
- confirmation_code: Booking reference number
- status: Reservation status (confirmed, pending, cancelled)

### NexGolf API

Base URL: https://nexgolf.fi/api/v3

Authentication:
- Type: Cookie-based Session
- Login Endpoint: POST /auth/session
- Session Duration: 12 hours
- CSRF Token Required: X-CSRF-Token header

Key Endpoints:

1. Session Creation
   POST /auth/session

```
  {
```

```
  "club_id": "NGF123",
  "username": "member_id",
  "password": "encrypted_password"
}
Response:
{
  "session_id": "sess_12345",
  "csrf_token": "csrf_token_value",
  "valid_until": "2024-01-27T12:00:00Z"
}
```

2. Available Times
   GET /clubs/{club_id}/times
   Parameters:
   - date: YYYY-MM-DD
   - course: integer
   - group_size: integer

```
Response:
{
  "times": [
    {
      "slot_id": "NG789",
      "time": "2024-01-26T09:00:00+02:00",
      "course": {
        "id": 1,
        "name": "Pääkenttä",
        "type": "18-hole"
      },
      "availability": {
        "total": 4,
        "booked": 1,
        "minimum_players": 2
      },
      "restrictions": {
        "members_only": true,
        "competition": false
      }
    }
  ],
  "day_info": {
    "sunrise": "08:15",
    "sunset": "16:45",
    "maintenance": []
  }
}
```

3. Booking Creation
   POST /clubs/{club_id}/bookings
   Body:

```
{
 "slot_id": "NG789",
 "players": [
  {
   "id": "NGM456",
   "type": "member",
   "extras": {
    "cart": true
   }
  }
 ],
 "notes": "Cart requested"
}
```

Response:

```
{
 "booking_id": "NGB123",
 "reference": "NG20240126-123",
 "status": "confirmed",
 "details": {
  "start_time": "2024-01-26T09:00:00+02:00",
  "course": "Pääkenttä",
  "player_count": 1,
  "extras": {
   "cart": {
    "confirmed": true,
    "number": "Cart-7"
   }
  }
 }
}
```

4. Member Details

GET /clubs/{club_id}/members/{member_id}

Response:

```
{
 "id": "NGM456",
 "membership": {
  "type": "full",
  "valid_until": "2024-12-31",
  "home_club": true
 },
 "playing_rights": {
  "advance_booking_days": 7,
  "booking_quota": {
   "active_limit": 3,
   "current_count": 1
  }
 },
 "handicap_info": {
```

```
    "exact": 12.4,
    "playing": 12,
    "last_updated": "2024-01-20"
   }
  }
```

Key Attributes:
- slot_id: Unique time slot identifier
- booking_id: Unique reservation identifier
- reference: Human-readable booking reference
- player_count: Number of players in booking
- handicap_info: Current handicap details

### TeeTime API

Base URL: https://teetimeapi.golf/v1

Authentication:
- Type: API Key
- Header: X-Api-Key: <key>
- Additional: Club-specific credentials in request body

Key Endpoints:

1. Club Authentication

```
  POST /clubs/auth
  {
   "club_id": "TT123",
   "api_key": "club_specific_key",
   "user_credentials": {
    "member_number": "12345",
    "pin": "encrypted_pin"
   }
  }
  Response:
  {
   "auth_token": "tt_session_token",
   "permissions": ["view", "book", "modify"],
   "expires_at": "2024-01-27T00:00:00Z"
  }
```

2. Time Slots

```
  GET /clubs/{club_id}/slots
  Parameters:
  - date: YYYY-MM-DD
  - players: integer
  - time_range: morning|afternoon|evening

  Response:
```

```
{
 "date": "2024-01-26",
 "slots": [
  {
   "id": "TTS456",
   "start": "2024-01-26T08:30:00+02:00",
   "product": {
    "id": "18H",
    "name": "18 Holes",
    "duration": 120
   },
   "capacity": {
    "total": 4,
    "available": 3
   },
   "pricing": {
    "member": 0,
    "guest": 65
   },
   "booking_window": {
    "opens": "2024-01-12T00:00:00Z",
    "closes": "2024-01-26T07:30:00Z"
   }
  }
 ],
 "weather_advisory": null
}
```


3. Create Booking
   POST /clubs/{club_id}/bookings
   Body:


```
{
 "slot_id": "TTS456",
 "booking": {
  "players": [
   {
    "member_number": "12345",
    "type": "member",
    "rental_set": null
   }
  ],
  "preferences": {
   "starting_tee": 1,
   "cart_required": false
  }
 }
}
```


   Response:


```
{
```

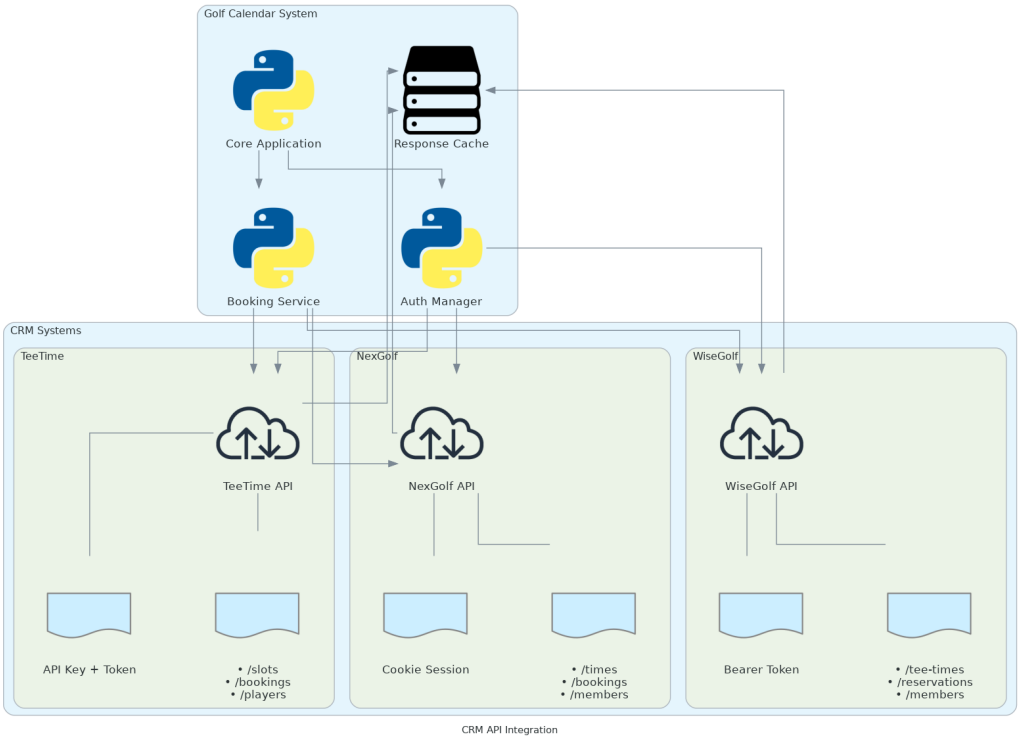
```
  "booking": {
    "id": "TTB789",
    "reference": "TT-20240126-789",
    "status": "confirmed",
    "slot": {
      "date": "2024-01-26",
      "time": "08:30",
      "course": "Main Course"
    },
    "players": [
      {
        "member_number": "12345",
        "checked_in": false,
        "rental_equipment": []
      }
    ],
    "payment_status": "not_required",
    "cancellation_policy": {
      "deadline": "2024-01-25T16:00:00Z",
      "fee_applies": true
    }
  }
}
```


4. Player Profile
   GET /clubs/{club_id}/players/{member_number}
   Response:

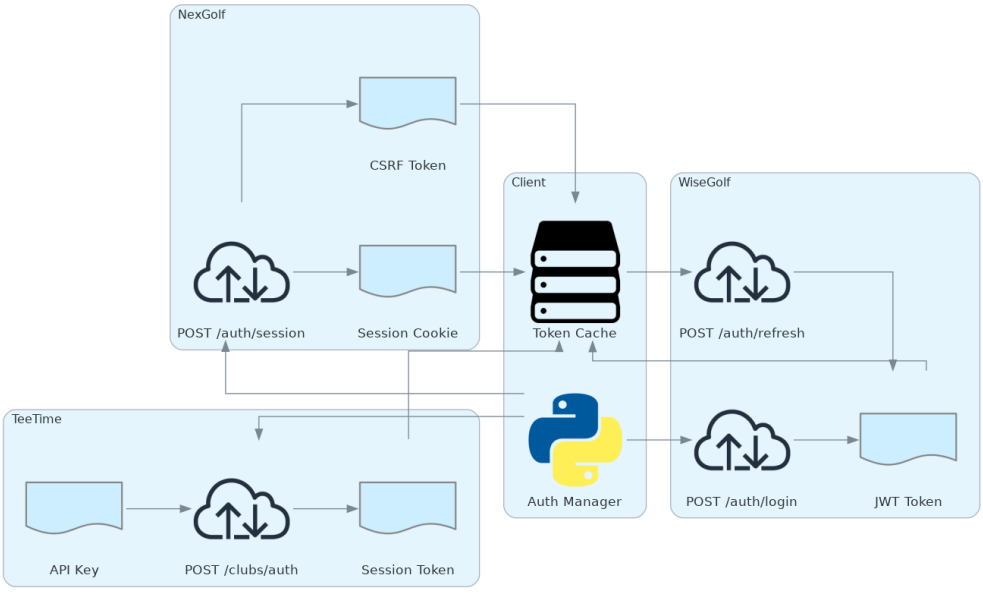

```
{
  "member": {
    "number": "12345",
    "category": "full_member",
    "status": "active"
  },
  "playing_rights": {
    "booking_horizon": 14,
    "concurrent_bookings": 3,
    "guest_privileges": true
  },
  "statistics": {
    "rounds_played": 45,
    "no_shows": 0,
    "average_pace": 118
  },
  "equipment": {
    "own_cart": false,
    "rental_preferences": {
      "club_set": "right-handed",
      "cart": "single"
    }
  }
```

}

Key Attributes:
- slot_id: Unique time slot identifier
- booking.id: Unique booking reference
- member_number: Player identification
- status: Booking confirmation status
- cancellation_policy: Rules for cancellation



CRM API Integration




CRM Authentication Flows

## CRM API Response Handling

The Golf Calendar system implements comprehensive response handling for each CRM API:

### Response Processing by System

1. WiseGolf:
   - Response Format: JSON with snake_case
   - Date Format: ISO 8601 with UTC
   - Error Handling:
     • HTTP 401: Token refresh required
     • HTTP 429: Rate limiting (exponential backoff)
     • HTTP 409: Booking conflict resolution
   - Data Validation:
     • Schema validation for all responses
     • Handicap range checks
     • Time slot availability confirmation

2. NexGolf:
   - Response Format: JSON with camelCase
   - Date Format: ISO 8601 with local timezone
   - Session Management:
     • Cookie renewal
     • CSRF token validation
     • Session expiry handling
   - Booking Validation:
     • Member status verification
     • Booking quota checks
     • Time slot locking

3. TeeTime:
   - Response Format: JSON with mixed case
   - Date Format: YYYY-MM-DD + HH:mm
   - Authentication:
     • API key validation
     • Club-specific credentials
     • Token refresh management
   - Response Processing:
     • Time zone conversions
     • Price calculation
     • Availability updates

### CRM API Error Codes and Handling

| Attribute | Type | Description |
|---|---|---|
| Authentication Failed | HTTP 401 | Refresh authentication token or prompt for new credentials. |
| Rate Limit Exceeded | HTTP 429 | Implement backoff strategy, cache responses, retry with exponential delay. |
| Booking Conflict | HTTP 409 | Check availability again, offer alternative slots, handle concurrent bookings. |
| Invalid Request | HTTP 400 | Validate request parameters, check time formats, verify member status. |
| Resource Not Found | HTTP 404 | Verify club/course IDs, check member numbers, validate time slot existence. |
| Quota Exceeded | Business Logic | Check booking limits, verify member privileges, handle waiting lists. |
| Time Slot Locked | Business Logic | Implement retry mechanism, check alternative slots, handle concurrent access. |
| Invalid Member Status | Business Logic | Verify membership status, check playing rights, validate handicap requirements. |

## Authentication Strategies

The Golf Calendar system implements multiple authentication strategies to handle different CRM systems:

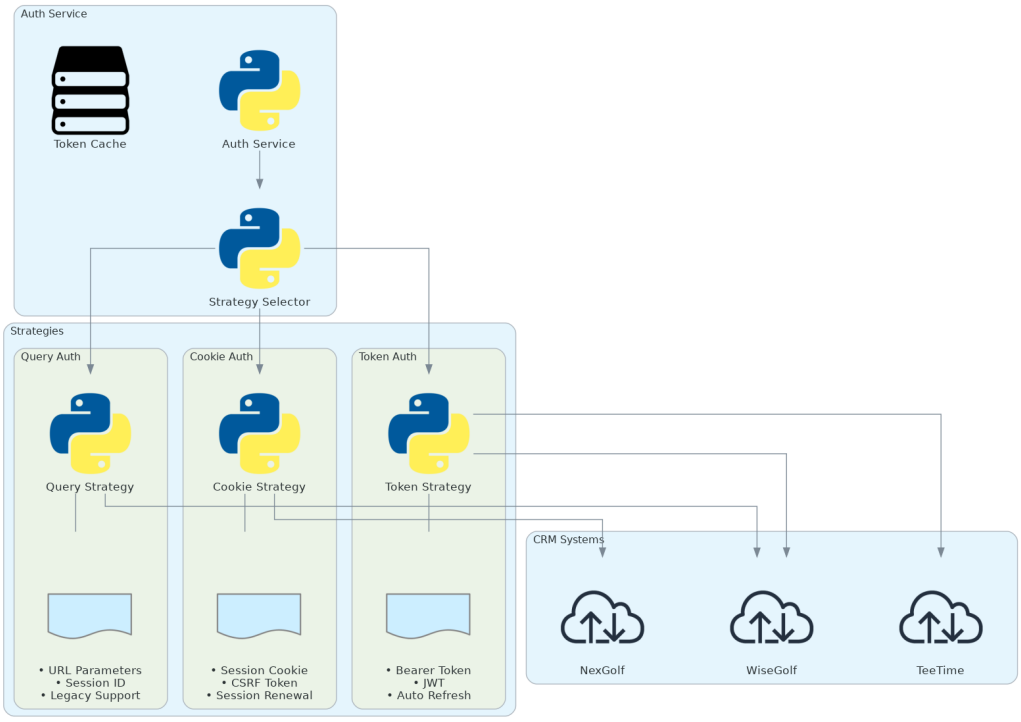
### Authentication Types

1. Token App Authentication (token_appauth):
   - Used by: WiseGolf, TeeTime
   - Flow:
     • Initial token request with credentials
     • Token stored in auth_details
     • Token included in Authorization header
     • Automatic token refresh when expired
   - Headers:
     • Authorization: Bearer <token>
     • Content-Type: application/json

2. Cookie Authentication (cookie):
   - Used by: NexGolf
   - Flow:
     • Session creation with credentials
     • Cookie stored in auth_details
     • Cookie included in subsequent requests
     • CSRF token handling where required
   - Headers:
     • Cookie: <session_cookie>
     • X-CSRF-Token: <csrf_token>

3. Query Authentication (query):
   - Used by: Legacy systems
   - Flow:
     • Credentials included as URL parameters
     • Session maintained through query params
     • Less secure, used only for legacy support
   - URL Format:
     • https://api.example.com/endpoint?token=<token>

4. Authentication Strategy Selection:
   - Based on club configuration
   - Fallback to unsupported strategy
   - Automatic strategy switching on failure
   - Credential refresh handling




Authentication Strategy Flow

### Authentication Error Handling

| Attribute | Type | Description |
|---|---|---|
| Token Expired | Auth Error | Attempt token refresh, if fails request new credentials, fall back to alternative auth method. |
| Invalid Credentials | Auth Error | Clear stored credentials, prompt for new credentials, verify club configuration. |
| Session Expired | Auth Error | Create new session, handle CSRF token refresh, maintain cookie jar. |
| Missing CSRF Token | Auth Error | Request new CSRF token, update session cookies, retry request with new token. |
| Rate Limited | API Error | Implement exponential backoff, rotate credentials if available, cache successful tokens. |
| Invalid Token Format | Auth Error | Validate token format, check auth strategy compatibility, verify API version. |
| Permission Denied | Auth Error | Verify membership status, check booking privileges, validate club access rights. |
| Connection Failed | Network Error | Retry with backoff, check API endpoint availability, verify network connectivity. |

### Authentication Implementation

1. Strategy Pattern Implementation:
   ```python
   class AuthStrategy:
       def create_headers(self, cookie_name: str, auth_details: Dict[str, str]) -> Dict[str, str]:
           """Create request headers based on auth type."""
           pass

       def build_full_url(self, club_details: Dict[str, Any], membership: Membership) -> str:
           """Build authenticated URL if required."""
           pass
   ```

2. Token Management:
   - Token Storage:
     • Secure storage in auth_details
     • Encrypted when at rest
     • Memory-only during runtime

   - Token Refresh:
     • Automatic refresh before expiry
     • Refresh token rotation
     • Failure recovery

3. Session Management:
   - Cookie Handling:
     • Secure cookie storage
     • Session expiry tracking
     • Automatic session renewal

- CSRF Protection:
  • Token validation
  • Header inclusion
  • Token refresh

4. Security Considerations:
- Credential Encryption:
  • Sensitive data encryption
  • Secure credential storage
  • Memory cleanup

- Rate Limiting:
  • Request throttling
  • Exponential backoff
  • API key rotation

### Authentication Configuration

| Attribute | Type | Description |
|---|---|---|
| auth_type | string | Authentication strategy type (token_appauth, cookie, query). Required. Determines auth flow. |
| auth_details.token | string | Authentication token for token-based auth. Required for token_appauth. |
| auth_details.refresh_token | string | Token for refreshing expired auth tokens. Optional for token_appauth. |
| auth_details.cookie | string | Session cookie for cookie-based auth. Required for cookie auth type. |
| auth_details.csrf_token | string | CSRF token for cookie-based auth. Required if CSRF protection enabled. |
| cookie_name | string | Name of session cookie for cookie-based auth. Required for cookie auth type. |
| token_expiry | integer | Token expiration time in seconds. Optional. Default varies by CRM. |
| retry_limit | integer | Maximum number of auth retry attempts. Optional. Default: 3. |

## Golf Club Factory and Club Types

The Golf Calendar system uses a factory pattern to create and manage different types of golf club integrations. Each club type has its own specific implementation and requirements.

### Golf Club Types

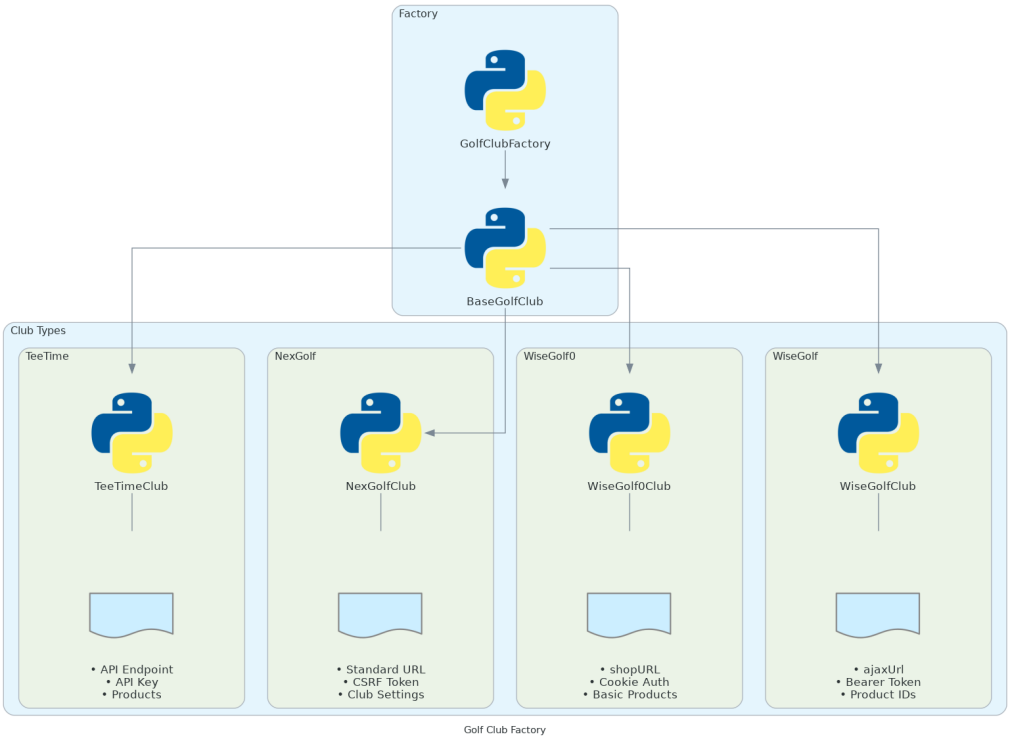
1. WiseGolf Club:
   - Modern REST API implementation
   - JWT-based authentication
   - Features:
     • Real-time availability
     • Member pricing
     • Advanced booking rules
     • Equipment rental
   - Configuration:
     • Requires ajaxUrl
     • Bearer token auth
     • Product IDs for booking types

2. WiseGolf0 Club (Legacy):
   - Original WiseGolf implementation
   - Cookie-based authentication
   - Features:
     • Basic availability checking
     • Simple booking flow
     • Limited member features
   - Configuration:
     • Requires shopURL
     • Session cookie auth
     • Basic product mapping

3. NexGolf Club:
   - Nordic golf club system
   - Session-based authentication
   - Features:
     • Competition support
     • Member management
     • Facility booking
     • Equipment tracking
   - Configuration:
     • Standard URL endpoint
     • CSRF protection
     • Club-specific settings

4. TeeTime Club:
   - Modern booking platform
   - API key authentication

- Features:
  - Dynamic pricing
  - Guest booking
  - Weather integration
  - Mobile check-in
- Configuration:
  - API endpoint URL
  - API key required
  - Product configuration



### Club Configuration Attributes

| Attribute | Type | Description |
|---|---|---|
| type | string | Club system type (wisegolf, wisegolf0, nexgolf, teetime). Required. Determines factory creation. |
| url | string | Base API endpoint URL. Required for nexgolf and teetime. |
| ajaxUrl | string | AJAX endpoint URL. Required for wisegolf type. |
| shopURL | string | Shop system URL. Required for wisegolf0 type. |
| variant | string | System variant for special handling. Optional. |
| product | object | Product configuration for bookings. Required for some club types. |
| address | string | Physical club address. Optional. Used for weather and calendar entries. |
| timezone | string | Club timezone. Optional. Defaults to Europe/Helsinki. |

### Club Implementation

1. Base Golf Club:

```python
class BaseGolfClub(GolfClub, ABC):
    def __init__(self, name: str, auth_details: Dict[str, Any],
                 club_details: Dict[str, Any], membership: Dict[str, Any]):
        self.auth_service = AuthService()
        self.club_details = club_details
        self.membership = membership

    @abstractmethod
    def get_reservations(self, user: User) -> List[Reservation]:
        """Get reservations for user."""
        pass
```

2. Factory Implementation:

```python
class GolfClubFactory:
    @staticmethod
    def create_club(club_details: Dict[str, Any],
                    membership: Membership,
                    auth_service: AuthService) -> Optional[GolfClub]:
        club_type = club_details["type"]
        club_class = club_classes.get(club_type)
        return club_class(
            name=club_details.get("name"),
            url=club_details.get("url"),
            auth_service=auth_service,
            club_details=club_details
        )
```

```

3. Club Type Specifics:
   - WiseGolf Implementation:
     • Modern REST endpoints
     • JWT authentication
     • Real-time availability

   - WiseGolf0 Implementation:
     • Legacy endpoints
     • Cookie-based auth
     • Basic functionality

   - NexGolf Implementation:
     • Nordic system integration
     • Session management
     • Competition support

   - TeeTime Implementation:
     • API key authentication
     • Dynamic pricing
     • Weather integration

4. Common Features:
   - Reservation Management:
     • Fetch user reservations
     • Create new bookings
     • Cancel existing bookings

   - Member Handling:
     • Validate membership
     • Check booking rights
     • Handle guest players

   - Error Management:
     • Connection issues
     • Authentication errors
     • Booking conflicts
```

### Club System Error Handling

| Attribute | Type | Description |
|---|---|---|
| Invalid Club Type | Configuration | Verify club type in configuration, check supported types, ensure factory implementation. |
| Missing URL | Configuration | Check required URL configuration for club type, verify endpoint accessibility. |
| Product Configuration | Configuration | Validate product IDs and mapping, check booking type configuration. |
| Booking Failed | Operation | Verify availability, check member privileges, validate booking parameters. |
| Reservation Fetch Failed | Operation | Check authentication, verify membership status, handle partial data. |
| Invalid Response Format | Data | Validate API response format, handle schema changes, check API version. |
| Member Validation Failed | Authentication | Verify membership details, check club access rights, validate credentials. |
| System Unavailable | Connection | Implement retry logic, check system status, handle maintenance windows. |

## Reservation Processing

The Golf Calendar system implements comprehensive reservation processing to handle bookings from multiple CRM systems and integrate them into user calendars.

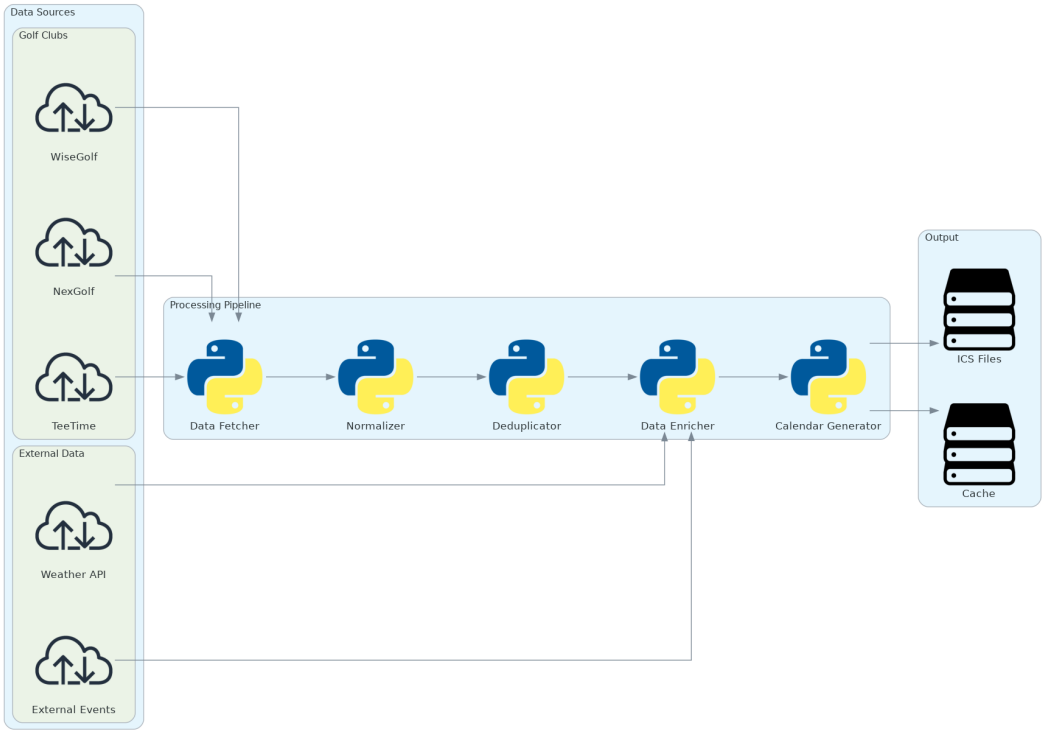
### Reservation Processing Flow

1. Reservation Retrieval:
   - Fetch from multiple clubs
   - Handle different formats
   - Process past/future bookings
   - Deduplication handling

2. Data Normalization:
   - Standardize timestamps
   - Convert time zones
   - Normalize durations
   - Format descriptions

3. Calendar Integration:
   - Create calendar entries
   - Add weather information
   - Include location details
   - Set reminders

4. Optimization:
   - Cache responses
   - Batch processing
   - Incremental updates
   - Memory management




Reservation Processing Flow

### Reservation Processing Implementation

1. Reservation Service:
```python
class ReservationService:
    def process_user(self, user_name: str, user_config: Dict[str, Any],
                     past_days: int = 7) -> Tuple[Calendar, List[Reservation]]:
        """Process reservations for user."""
        cal = Calendar()
        all_reservations = []

        for membership in user_config.memberships:
            club = GolfClubFactory.create_club(club_details, membership)
            raw_reservations = club.fetch_reservations(membership)

            for raw_reservation in raw_reservations:
                reservation = self._create_reservation(raw_reservation, club)
                if self._should_include_reservation(reservation):
                    all_reservations.append(reservation)
                    self._add_to_calendar(reservation, cal)

        return cal, all_reservations
```

2. Reservation Processing:
   - Time Window Handling:
     • Skip past dates beyond window
     • Process future bookings
     • Handle recurring events

   - Deduplication:
     • Track seen UIDs
     • Compare key fields
     • Handle modifications

   - Data Enrichment:
     • Add weather forecasts
     • Include club details
     • Set location info

3. Calendar Integration:
   - Event Creation:
     • Set start/end times
     • Add location details
     • Include weather data

   - File Management:
     • Create ICS files
     • Handle file paths

- Manage backups

4. Error Recovery:
   - Partial Processing:
     - Continue on club errors
     - Skip invalid entries
     - Log issues

   - Data Validation:
     - Check required fields
     - Validate formats
     - Handle missing data

### Reservation Attributes

| Attribute | Type | Description |
|---|---|---|
| start_time | datetime | Reservation start time in local timezone. Required. Must be valid datetime. |
| end_time | datetime | Reservation end time in local timezone. Required. Must be after start_time. |
| club | GolfClub | Reference to golf club. Required. Must be valid club instance. |
| user | User | Reference to booking user. Required. Must be valid user instance. |
| membership | Membership | User's club membership details. Required for member validation. |
| players | List[Player] | List of players in booking. Optional. Includes guests and members. |
| weather | Dict[str, Any] | Weather forecast data. Optional. Added during processing. |
| external_id | string | CRM system booking ID. Required. Used for deduplication. |

### Calendar Integration

1. Calendar Entry Creation:
   - Event Properties:
     • Summary: Club and time
     • Description: Players and weather
     • Location: Club address
     • Duration: Based on holes

   - Special Handling:
     • Timezone conversion
     • All-day events
     • Recurring bookings
     • Competition events

2. File Management:
   - ICS File Handling:
     • Create per user
     • Append new events
     • Remove old events
     • Handle conflicts

   - Path Resolution:
     • Support absolute paths
     • Handle relative paths
     • Create directories
     • Manage permissions

3. Calendar Features:
   - Event Details:
     • Rich descriptions
     • Location mapping
     • Weather forecasts
     • Player information

   - Integration:
     • iCal format
     • Google Calendar
     • Outlook support
     • Mobile sync

4. Update Management:
   - Change Detection:
     • Track modifications
     • Handle cancellations
     • Process updates
     • Maintain history

   - Sync Strategy:

- Incremental updates
- Full refresh option
- Error recovery
- Version control

### Calendar Configuration

| Attribute | Type | Description |
|---|---|---|
| ics_dir | string | Directory for ICS files. Required. Can be absolute or relative path. |
| ics_files | Dict[str, str] | Custom file paths per user. Optional. Overrides default naming. |
| past_days | integer | Days of past reservations to include. Optional. Default: 7. |
| future_days | integer | Days of future reservations to include. Optional. Default: 90. |
| timezone | string | Default timezone for calendar. Required. IANA timezone format. |
| refresh_interval | integer | Minutes between calendar updates. Optional. Default: 60. |
| backup_count | integer | Number of backup files to keep. Optional. Default: 3. |
| file_permissions | string | Unix-style permissions for files. Optional. Default: 0o644. |