Construção de Sistemas de Software
Tecnologias de Informação
Semestre 2025/2026

Projeto Prático #1 - WeatherWise: Serviço de Meteorologia

Grupo 13
Vitória Correia - 62211
Guilherme Soares - 62372
Duarte Soares - 62371

Diagrama Entidade-Relação (ER) da base de dados:

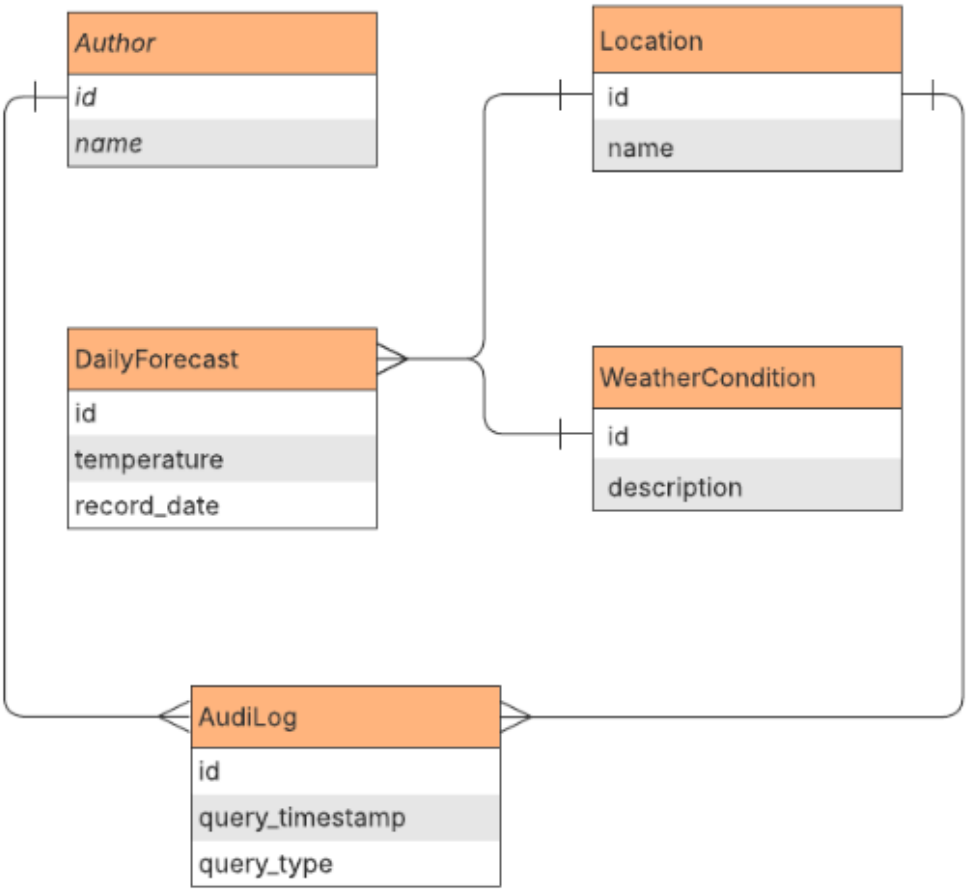


Diagrama de Sequência de Sistema (SSD) do caso de uso D:

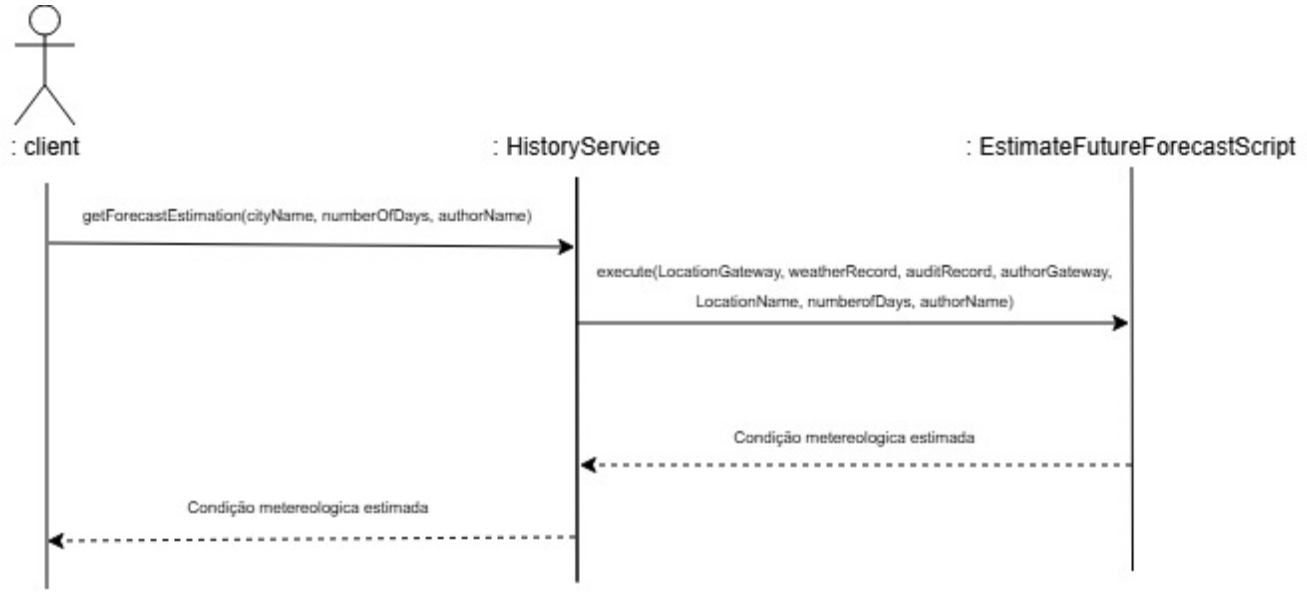
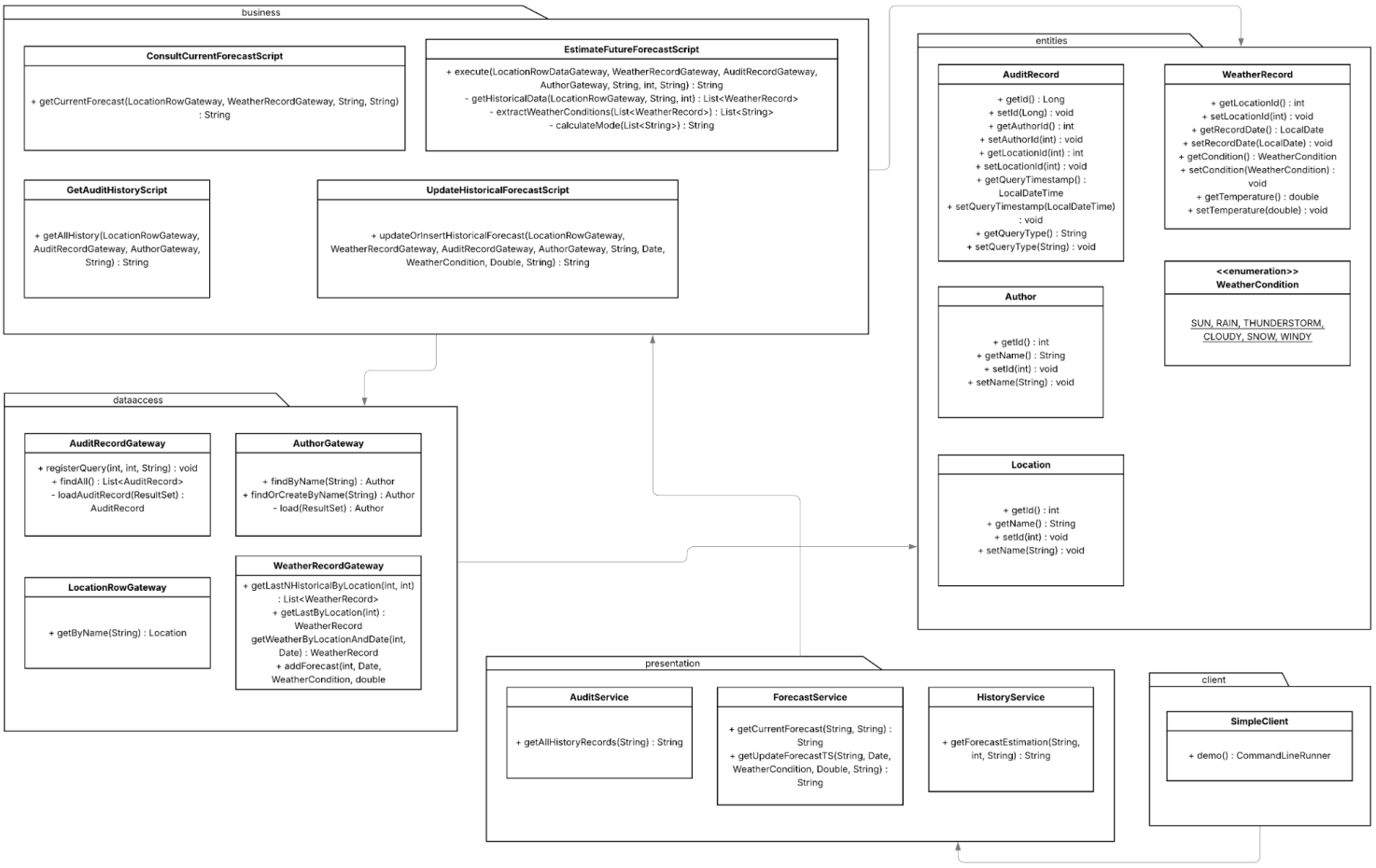


Diagrama de Classes :



Nota: Todos os diagramas encontram-se no final do relatório em maiores dimensões.

Arquitetura em camadas:

Camada: dataaccess
Padrão Arquitetural: Row Data Gateway (RDWG)
Responsabilidade Principal: *Persistência de Dados.* É o único código que interage com o JDBC/SQL. Responsável por mapear o ResultSet para objetos de Domínio (POJOs) e vice-versa. *Não contém lógica de negócio.*

Camada: business
Padrão Arquitetural: Transaction Script
Responsabilidade Principal: *Lógica de Negócio e Coordenação.* Contém a lógica sequencial para executar um Caso de Uso. Coordena a ação, chamando Gateways para obter ou guardar dados. *Implementa os 4 Casos de Uso*.

Camada: entities
Padrão Arquitetural: POJOs / Entidades de Domínio
Responsabilidade Principal: *Modelo de Dados.* Representa a estrutura e o estado dos dados de negócio, sem qualquer lógica de persistência ou de aplicação. Serve como contrato de dados entre as camadas.

Camada: presentation (services)
Padrão Arquitetural: Service Layer
Responsabilidade Principal: *Ponto de entrada da aplicação (API).* Expõe a lógica de negócio ao cliente. Cada "Serviço" "embrulha"(wraps) os Transaction Scripts correspondentes da camada business, simplificando a interface e desacoplando o cliente da implementação interna.

Justificação do Desenho da base de dados:

O desenho da base de dados relacional foi concebido para satisfazer os requisitos funcionais (Casos de Uso A-D) e não funcionais (auditoria, consistência) do projeto.

A estrutura segue uma abordagem normalizada (Terceira Forma Normal - 3FN) para garantir a integridade dos dados, reduzir a redundância e otimizar o desempenho das consultas.

Justificação das Tabelas (Normalização)

O esquema é composto por cinco tabelas principais:

Location, **WeatherCondition**, **Author**: Estas são tabelas de "dimensão" ou de normalização. Em vez de armazenar repetidamente *strings* (como "Lisboa", "Sol", ou "Ana Silva") nas tabelas de factos, armazenamos apenas as suas chaves primárias (IDs).

- **Benefícios:**

    - **Consistência:** Garante que "Lisboa" é sempre escrito da mesma forma. Se o nome precisar de ser corrigido (ex: "Lisboa" para "Lisboa Capital"), a alteração é feita num único local.
    - **Eficiência:** Armazenar um INTEGER (4 bytes) é significativamente mais eficiente (em espaço e em velocidade de JOIN) do que armazenar um VARCHAR(100) (até 101 bytes) em milhões de registos de AuditLog ou DailyForecast.
  - A tabela Author foi criada para normalizar o requisito de "registar o autor de cada consulta".

DailyForecast: Esta é a tabela de "factos" que armazena o histórico meteorológico, servindo de base para o **Caso de Uso B** (Atualizar) e **Caso de Uso D** (Estimar).

AuditLog: Esta é a tabela de "factos" que armazena o registo de todas as consultas, satisfazendo o **Caso de Uso C**.

<u>Justificação de Chaves e Restrições (Integridade e Consistência)</u>

A integridade dos dados é garantida através de um conjunto rigoroso de chaves e restrições:

- **Chaves Primárias (SERIAL PRIMARY KEY):**
  - Todas as tabelas utilizam um SERIAL (um inteiro auto-incrementado) como Chave Primária (PK).
  - **Justificação:** A utilização de um campo **SERIAL** como chave primária garante que os dados sejam únicos e evita problemas futuros, como ter que alterar a chave primária (por exemplo, o **NIF**) caso surjam novos registros.

- **Restrições UNIQUE:**
  - Location(name), WeatherCondition(description), Author(name): Garantem que não podem existir duas localizações, condições ou autores com o mesmo nome. O nome é a "chave de negócio".
  - DailyForecast(location_id, record_date): Esta é a restrição de negócio mais importante. Garante que só pode existir **um** registo meteorológico por localidade, por dia.

- **Chaves Estrangeiras (FK) e Ações ON DELETE:**
  - DailyForecast.location_id usa **ON DELETE CASCADE**: Se uma Location for apagada da base de dados, todos os seus registos meteorológicos históricos são automaticamente apagados. Esta é uma decisão de negócio que assume que os registos históricos de uma localização inexistente são irrelevantes.

○ AuditLog.author_id e AuditLog.location_id usam **ON DELETE RESTRICT**: Esta é uma decisão crítica de auditoria. O sistema **proíbe** a eliminação de um Author ou Location se existirem registos de auditoria associados a eles. Isto garante a integridade e a não-repudiação do histórico de consultas.

Justificação de Tipos de Dados

A escolha dos tipos de dados foi feita para otimizar o armazenamento e garantir a precisão:

- **NUMERIC(4, 1) (Tabela DailyForecast):**
  - ○ FLOAT ou REAL são tipos de dados de precisão variável e podem causar erros de arredondamento. INTEGER não permite casas decimais.
  - ○ NUMERIC(4, 1) foi escolhido por ser um tipo de precisão fixa, ideal para temperaturas. Permite 4 dígitos no total, com 1 casa decimal (intervalo de -999.9 a 999.9), o que é mais do que suficiente para temperaturas, garantindo exatidão.
- **DATE vs. TIMESTAMP:**
  - ○ DailyForecast.record_date usa **DATE**, pois um registo meteorológico refere-se a um dia inteiro, não a uma hora específica.
  - ○ AuditLog.query_timestamp usa **TIMESTAMP**, pois a auditoria exige saber o momento exato da consulta (data *e* hora). O uso de DEFAULT CURRENT_TIMESTAMP automatiza esta captura.
- **SERIAL (Tabela AuditLog):**
  - ○ SERIAL é um INTEGER (32-bit), permitindo cerca de 2.1 mil milhões de registos. Se o sistema antecipar um volume de consultas superior (ex: milhares de queries por segundo), BIGSERIAL (64-bit) seria uma escolha mais robusta, mas SERIAL é suficiente para este protótipo.

Justificação de Otimizações (Índices)

Para além das chaves primárias (que são indexadas automaticamente), foi criado um índice manual:

- **CREATE INDEX idx_audit_location ON AuditLog (location_id):**
  - ○ **Justificação:** O projeto exige "perguntas de exemplo" como "Quantas vezes a previsão para o 'Porto' já foi consultada?". Esta query irá filtrar a tabela AuditLog pela location_id.
  - ○ Sem este índice, a base de dados teria de ler a tabela AuditLog inteira (um *Full Table Scan*) para encontrar os registos do "Porto".

- Com o índice, a base de dados encontra esses registos de forma quase instantânea. Esta é uma otimização de desempenho crítica, visto que a AuditLog será, previsivelmente, a maior tabela do sistema.

Garantias de Consistência dos Dados:

A consistência dos dados é garantida em duas camadas: ao nível da Base de Dados (o mais forte) e ao nível da Aplicação.

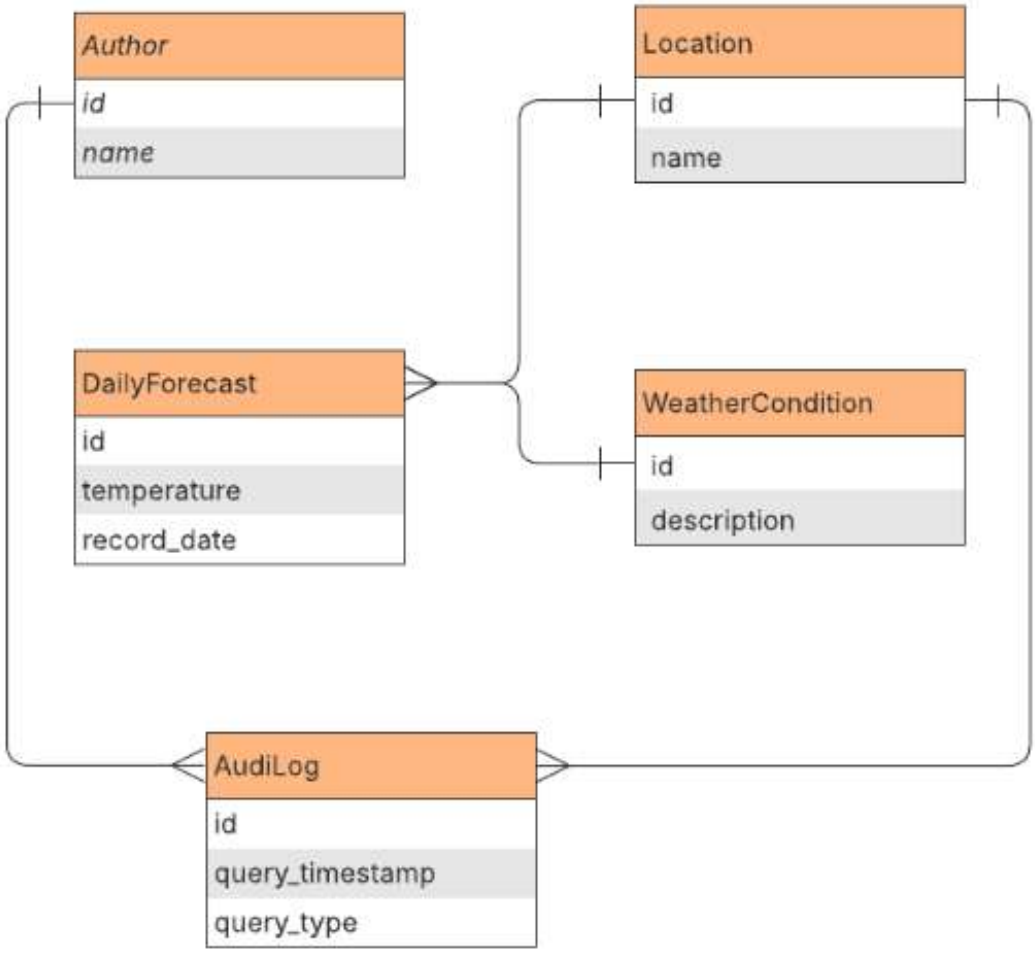
- **Nível da Base de Dados (Integridade Referencial)**:
  - O uso de FOREIGN KEY (chaves estrangeiras) garante que é impossível criar um AuditLog ou DailyForecast que aponte para um Author ou Location inexistente.
  - O uso de ON DELETE RESTRICT no AuditLog é a garantia mais forte: proíbe que um Author ou Location seja apagado se existirem registos de auditoria associados, assegurando que o histórico de auditoria (AuditLog) permanece sempre intacto e consistente.

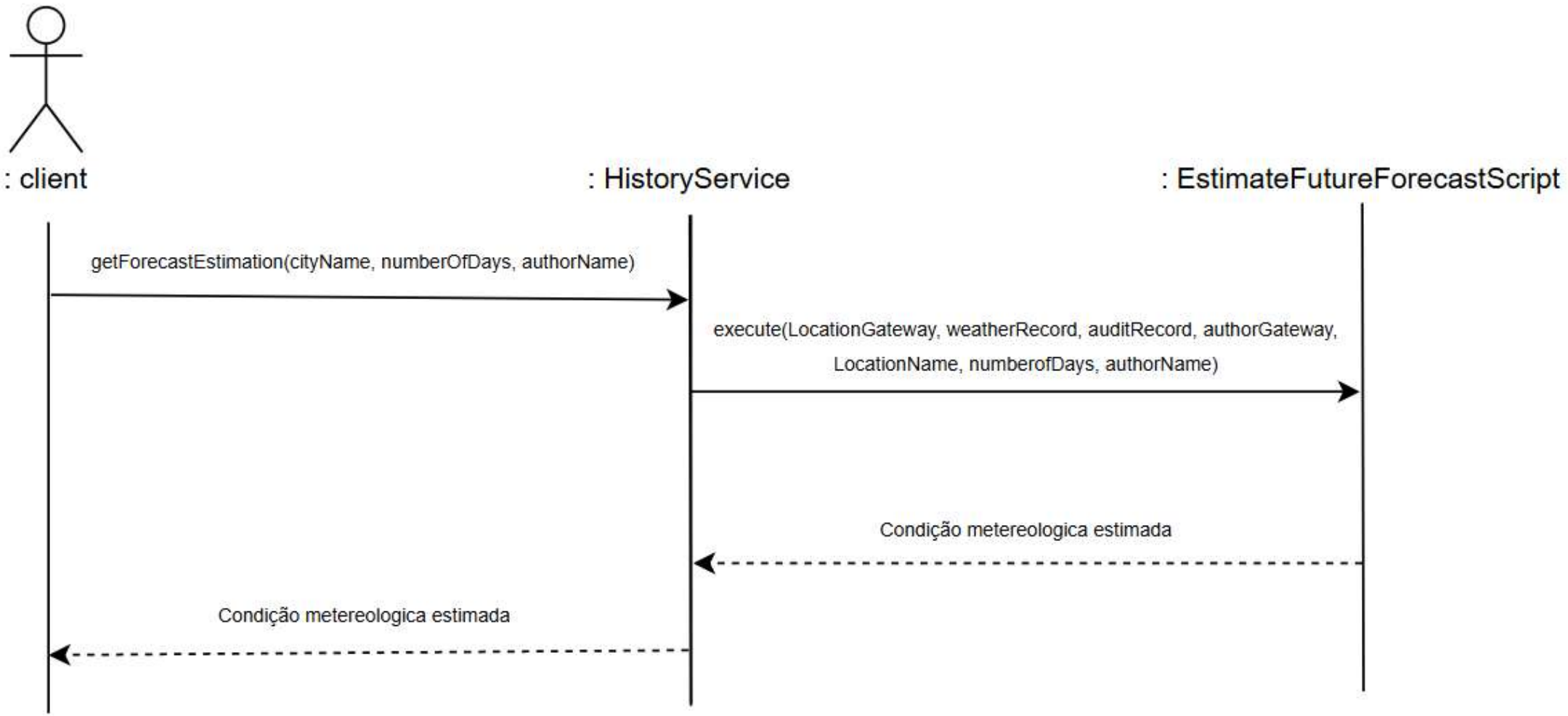
- **Nível da Base de Dados (Unicidade dos Dados):**
  - As restrições UNIQUE nas colunas name das tabelas Location e Author garantem que não podem existir duas localidades ou autores com o mesmo nome.

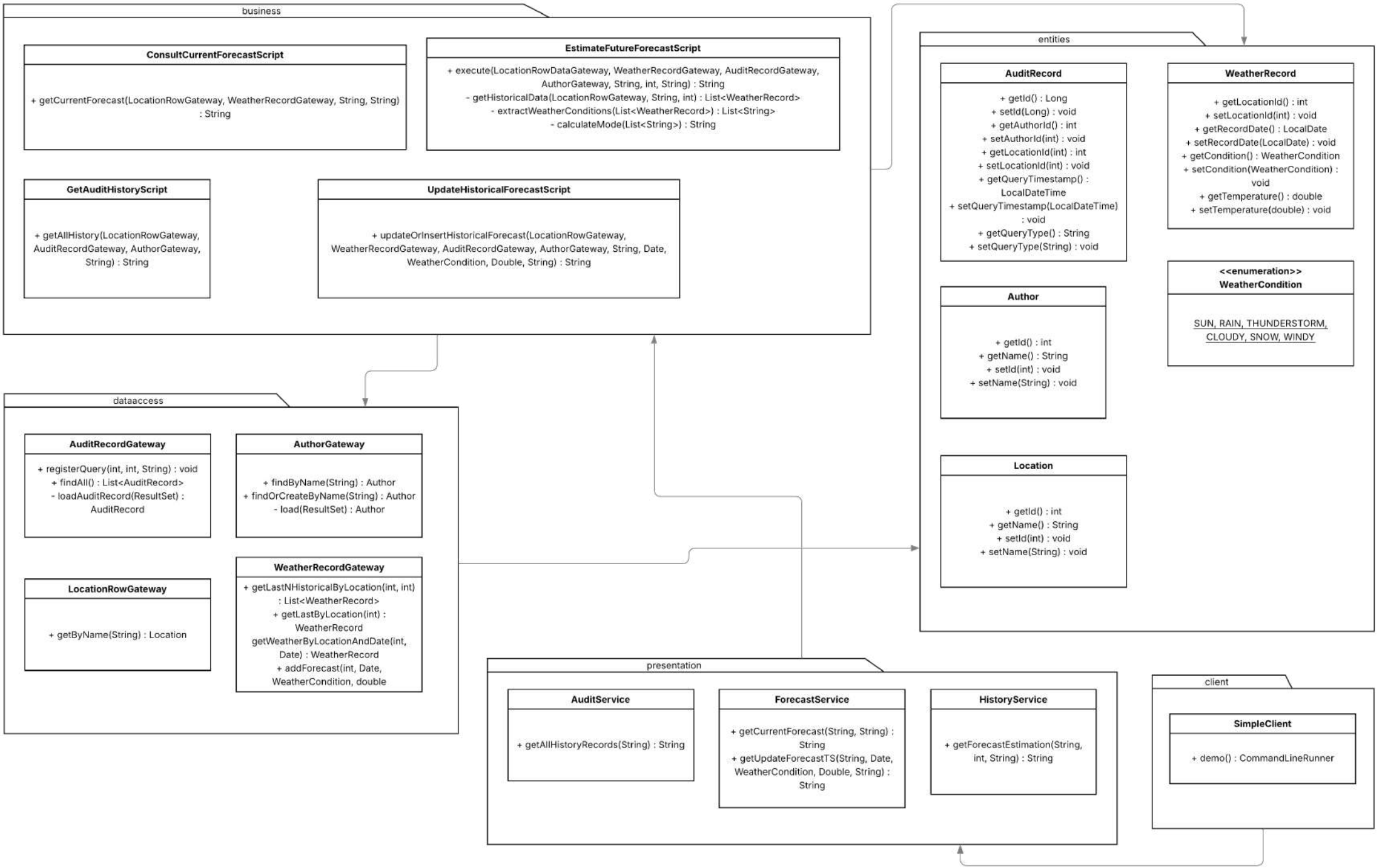
- **Nível da Aplicação (Consistência Lógica):**
  - O padrão findOrCreateByName implementado no AuthorGateway garante a consistência lógica. Antes de registar uma auditoria, a aplicação verifica ativamente se o autor já existe. Se não existir, cria-o. Isto assegura que a aplicação funciona em harmonia com a restrição UNIQUE da base de dados, evitando erros e duplicados.

Diagrama ER com entidades coloridas (notação UML)

Vitória Correia | November 2, 2025




: client
: HistoryService
: EstimateFutureForecastScript
getForecastEstimation(cityName, numberOfDays, authorName)
execute(LocationGateway, weatherRecord, auditRecord, authorGateway,
LocationName, numberofDays, authorName)
Condição metereologica estimada
Condição metereologica estimada


business
ConsultCurrentForecastScript
+ getCurrentForecast(LocationRowGateway, WeatherRecordGateway, String, String) : String
EstimateFutureForecastScript
+ execute(LocationRowDataGateway, WeatherRecordGateway, AuditRecordGateway, AuthorGateway, String, int, String) : String
- getHistoricalData(LocationRowGateway, String, int) : List<WeatherRecord>
- extractWeatherConditions(List<WeatherRecord>) : List<String>
- calculateMode(List<String>) : String
GetAuditHistoryScript
+ getAllHistory(LocationRowGateway, AuditRecordGateway, AuthorGateway, String) : String
UpdateHistoricalForecastScript
+ updateOrInsertHistoricalForecast(LocationRowGateway, WeatherRecordGateway, AuditRecordGateway, AuthorGateway, String, Date, WeatherCondition, Double, String) : String
entities
AuditRecord
+ getId() : Long
+ setId(Long) : void
+ getAuthorId() : int
+ setAuthorId(int) : void
+ getLocationId(int) : int
+ setLocationId(int) : void
+ getQueryTimestamp() : LocalDateTime
+ setQueryTimestamp(LocalDateTime) : void
+ getQueryType() : String
+ setQueryType(String) : void
WeatherRecord
+ getLocationId() : int
+ setLocationId(int) : void
+ getRecordDate() : LocalDate
+ setRecordDate(LocalDate) : void
+ getCondition() : WeatherCondition
+ setCondition(WeatherCondition) : void
+ getTemperature() : double
+ setTemperature(double) : void
Author
+ getId() : int
+ getName() : String
+ setId(int) : void
+ setName(String) : void
<<enumeration>>
WeatherCondition
SUN, RAIN, THUNDERSTORM, CLOUDY, SNOW, WINDY
Location
+ getId() : int
+ getName() : String
+ setId(int) : void
+ setName(String) : void
dataaccess
AuditRecordGateway
+ registerQuery(int, int, String) : void
+ findAll() : List<AuditRecord>
- loadAuditRecord(ResultSet) : AuditRecord
AuthorGateway
+ findByName(String) : Author
+ findOrCreateByName(String) : Author
- load(ResultSet) : Author
LocationRowGateway
+ getByName(String) : Location
WeatherRecordGateway
+ getLastNHistoricalByLocation(int, int) : List<WeatherRecord>
+ getLastByLocation(int) : WeatherRecord
getWeatherByLocationAndDate(int, Date) : WeatherRecord
+ addForecast(int, Date, WeatherCondition, double
presentation
AuditService
+ getAllHistoryRecords(String) : String
ForecastService
+ getCurrentForecast(String, String) : String
+ getUpdateForecastTS(String, Date, WeatherCondition, Double, String) : String
HistoryService
+ getForecastEstimation(String, int, String) : String
client
SimpleClient
+ demo() : CommandLineRunner