

DOCUMENTAÇÃO

SYNERGY

Gustavo Barra Felizardo

Documentação Completa do Backend em C++

27 de novembro de 2025

Sumário

1	Introdução	5
1.1	Objetivo	5
1.2	Escopo	5
1.3	Arquitetura Geral	5
1.4	Tecnologias Utilizadas	6
2	Classes Base	7
2.1	Entity	7
2.1.1	Descrição	7
2.1.2	Localização	7
2.1.3	Atributos	7
2.1.4	Métodos Principais	7
2.1.5	Características	9
3	Classes de Usuários e Empresas	10
3.1	User	10
3.1.1	Descrição	10
3.1.2	Localização	10
3.1.3	Atributos	10
3.1.4	Métodos Principais	10
3.1.5	Características	11
3.2	Company	12
3.2.1	Descrição	12
3.2.2	Localização	12
3.2.3	Atributos	12
3.2.4	Métodos Principais	12
3.2.5	Características	13
4	Sistema de Casos	14
4.1	Case	14
4.1.1	Descrição	14
4.1.2	Localização	14
4.1.3	Atributos	14
4.1.4	Métodos Principais	14
4.1.5	Características	15
4.2	CaseSystem	16
4.2.1	Descrição	16
4.2.2	Localização	16

4.2.3	Atributos	16
4.2.4	Métodos Principais	16
4.2.5	Características	17
5	Sistema de Ranking	18
5.1	Ranking	18
5.1.1	Descrição	18
5.1.2	Localização	18
5.1.3	Atributos	18
5.1.4	Métodos Principais	18
5.1.5	Características	18
5.2	CompanyRanking	19
5.2.1	Descrição	19
5.2.2	Localização	19
5.2.3	Atributos	19
5.2.4	Métodos Principais	19
5.2.5	Características	19
6	Rede Social	20
6.1	Post	20
6.1.1	Descrição	20
6.1.2	Localização	20
6.1.3	Estruturas Auxiliares	20
6.1.4	Atributos	20
6.1.5	Métodos Principais	20
6.1.6	Características	21
6.2	Feed	22
6.2.1	Descrição	22
6.2.2	Localização	22
6.2.3	Atributos	22
6.2.4	Métodos Principais	22
6.2.5	Características	22
6.3	SocialNetwork	23
6.3.1	Descrição	23
6.3.2	Localização	23
6.3.3	Atributos	23
6.3.4	Métodos Principais	23
6.3.5	Características	24
7	Sistema Central	25

7.1	System	25
7.1.1	Descrição	25
7.1.2	Localização	25
7.1.3	Atributos	25
7.1.4	Métodos Principais	25
7.1.5	Características	27
8	Gerenciamento de Banco de Dados	28
8.1	DbManager	28
8.1.1	Descrição	28
8.1.2	Localização	28
8.1.3	Atributos	28
8.1.4	Métodos Principais	28
8.1.5	Métodos Privados	29
8.1.6	Características	30
9	Utilitários	31
9.1	Utils	31
9.1.1	Descrição	31
9.1.2	Localização	31
9.1.3	Funções	31
9.1.4	Características	31
10	Relacionamentos entre Classes	33
10.1	Diagrama de Herança	33
10.2	Relacionamentos de Composição	33
10.3	Dependências	33
11	Padrões de Projeto	35
11.1	Facade Pattern	35
11.2	Template Method Pattern	35
11.3	Strategy Pattern	35
11.4	Smart Pointers	35
12	Fluxo de Dados	36
12.1	Ciclo de Vida do Sistema	36
12.2	Fluxo de Resolução de Casos	36
12.3	Fluxo de Feed	37
13	Banco de Dados	38
13.1	Estrutura de Tabelas	38

13.2 Persistência	38
14 Considerações de Segurança	39
14.1 Autenticação	39
14.2 Validação de Dados	39
14.3 Integridade Referencial	39
15 Gerenciamento de Memória	40
15.1 Smart Pointers	40
15.2 Ownership	40
15.3 Containers STL	40
16 Extensibilidade	41
16.1 Novos Tipos de Entidades	41
16.2 Customização de Rankings	41
16.3 Novos Tipos de Cases	41
17 Conclusão	42
17.1 Resumo	42
17.2 Pontos Fortes	42
17.3 Melhorias Futuras	42
17.4 Manutenibilidade	43

1 Introdução

Este documento apresenta a documentação técnica completa do módulo do Sistema Synergy. O sistema implementa uma rede social que conecta usuários e empresas através de desafios e questões (cases), permitindo que usuários demonstrem suas habilidades e aprimorem suas habilidades e sejam ranqueados de acordo com seu desempenho.

1.1 Objetivo

O objetivo deste documento é fornecer uma descrição detalhada da arquitetura, classes, métodos e funcionalidades implementadas no backend do sistema.

1.2 Escopo

Esta documentação abrange exclusivamente as classes implementadas no diretório , incluindo:

- Classes de entidades base (Entity, User, Company)
- Sistema de casos e desafios (Case, CaseSystem)
- Sistema de ranking (Ranking, CompanyRanking)
- Rede social e feed de postagens (Post, Feed, SocialNetwork)
- Sistema central de orquestração (System)
- Gerenciamento de banco de dados (DbManager)
- Utilitários do sistema (Utils)

1.3 Arquitetura Geral

O sistema utiliza o padrão de orientação a objetos com herança, polimorfismo e encapsulamento. A arquitetura é organizada em camadas:

- **Backend:** Lógica de negócio e entidades do sistema (src/backend)
- **Database:** Camada de persistência com SQLite (src/database)
- **Headers:** Interfaces públicas das classes (include)

1.4 Tecnologias Utilizadas

- **Linguagem:** C++11
- **Banco de Dados:** SQLite3
- **Build System:** Makefile
- **Bibliotecas STL:** vector, map, memory, iostream, algorithm

2 Classes Base

2.1 Entity

2.1.1 Descrição

A classe `Entity` é a classe abstrata base que representa qualquer entidade do sistema (usuários ou empresas). Implementa funcionalidades comuns como autenticação, perfil, seguidores e seguindo.

2.1.2 Localização

- Header: `include/backend/Entity.hpp`
- Implementação: `src/backend/Entity.cpp`

2.1.3 Atributos

Tabela 1: Atributos da classe `Entity`

Atributo	Tipo	Descrição
<code>_id</code>	<code>int</code>	Identificador único da entidade
<code>_name</code>	<code>std::string</code>	Nome da entidade
<code>_email</code>	<code>std::string</code>	Endereço de e-mail
<code>_phone_number</code>	<code>std::string</code>	Número de telefone
<code>_password</code>	<code>std::string</code>	Senha para autenticação
<code>_description</code>	<code>std::string</code>	Descrição/biografia
<code>_state</code>	<code>bool</code>	Estado ativo/inativo
<code>_followers</code>	<code>vector<string></code>	Lista de seguidores
<code>_following</code>	<code>vector<string></code>	Lista de seguindo
<code>_num_entities</code>	<code>int (static)</code>	Contador de entidades
<code>_next_id</code>	<code>int (static)</code>	Próximo ID disponível

2.1.4 Métodos Principais

Construtor

```
1 Entity( const std::string& name ,
2         const std::string& email ,
3         const std::string& phone_number ,
4         const std::string& password ,
5         bool state ,
6         const std::string& description );
```

Getters

- `int getId() const`
- `std::string getName() const`
- `std::string getEmail() const`
- `std::string getPhoneNumber() const`
- `std::string getDescription() const`
- `std::string getPassword() const`
- `bool getState() const`
- `vector<string>& getFollowers()`
- `vector<string>& getFollowing()`

Setters

- `void setName(const std::string& name)`
- `void setEmail(const std::string& email)`
- `void setPhoneNumber(const std::string& phone_number)`
- `void setState(bool state)`
- `void setDescription(const std::string& description)`
- `void setPassword(const std::string& password)`

Outros Métodos

- `bool checkPassword(const string& password_to_verify) const`
- `void addFollower(const string& username)`
- `void addFollowing(const string& username)`
- `virtual void displayProfile() = 0` - Método abstrato

2.1.5 Características

- Classe abstrata (não pode ser instanciada diretamente)
- Base para User e Company através de herança
- Destrutor virtual para polimorfismo adequado
- Sobrecarga do operador « para saída formatada

3 Classes de Usuários e Empresas

3.1 User

3.1.1 Descrição

A classe `User` representa um usuário do sistema, herdando de `Entity`. Gerencia informações como CPF, pontuações por empresa e histórico de casos resolvidos.

3.1.2 Localização

- **Header:** `/include/backend/User.hpp`
- **Implementação:** `/src/backend/User.cpp`

3.1.3 Atributos

Tabela 2: Atributos da classe `User`

Atributo	Tipo	Descrição
<code>_cpf</code>	<code>std::string</code>	CPF do usuário
<code>_companyScores</code>	<code>map<shared_ptr<Company>, double></code>	Pontuação por empresa
<code>_totalScore</code>	<code>double</code>	Pontuação total geral
<code>_solvedCases</code>	<code>vector<int></code>	IDs de casos resolvidos

3.1.4 Métodos Principais

Construtor

```
1 User( const std::string& name,
2       const std::string& email,
3       const std::string& phone_number,
4       const std::string& password,
5       bool state,
6       const std::string& description,
7       double totalScore,
8       const std::string& cpf);
```

Getters

- `std::string getCpf() const`
- `double getScoreByCompany(shared_ptr<Company> company) const`
- `double getTotalScore() const`
- `vector<int> getSolvedCases() const`

Setters

- `void setCpf(const std::string& cpf)`
- `void setTotalScore(double score)`

Outros Métodos

- `void addScore(shared_ptr<Company> company, double score)`
- `void registerSolvedCase(int caseId)`
- `void displayProfile() override`

3.1.5 Características

- Herda de Entity
- Sistema de pontuação segmentado por empresa
- Mantém histórico de casos resolvidos
- Suporta ranking global e por empresa

3.2 Company

3.2.1 Descrição

A classe `Company` representa uma empresa no sistema, herdando de `Entity`. Gerencia seus próprios casos (desafios) e mantém um ranking específico de usuários.

3.2.2 Localização

- **Header:** `/include/backend/Company.hpp`
- **Implementação:** `/src/backend/Company.cpp`

3.2.3 Atributos

Tabela 3: Atributos da classe `Company`

Atributo	Tipo	Descrição
<code>_cnpj</code>	<code>std::string</code>	CNPJ da empresa
<code>_ranking</code>	<code>CompanyRanking*</code>	Ranking específico da empresa
<code>_companyCaseSystem</code>	<code>CaseSystem*</code>	Sistema de casos da empresa

3.2.4 Métodos Principais

Construtor e Destrutor

```
1 Company( const std::string& name,
2         const std::string& email,
3         const std::string& phone_number,
4         const std::string& password,
5         bool state,
6         const std::string& description,
7         const std::string& cnpj);
8
9 ~Company();
```

Getters

- `std::string getCnpj() const`
- `shared_ptr<Case> getCase(int caseId) const`
- `CompanyRanking* getRanking()`
- `const vector<shared_ptr<Case>& getAllCases()`
- `vector<shared_ptr<Case> getActiveCases() const`

- `vector<shared_ptr<Case>> getInactiveCases() const`
- `CaseSystem& getCaseSystem()`

Gerenciamento de Casos

- `void addCase(const shared_ptr<Case>& c)`
- `void removeCase(int caseId)`
- `void activateCase(int caseId)`
- `void deactivateCase(int caseId)`

Visualização

- `void displayProfile() override`
- `void displayRanking() override`

3.2.5 Características

- Herda de Entity
- Gerencia sistema próprio de casos
- Mantém ranking específico de usuários
- Permite ativação/desativação de casos

4 Sistema de Casos

4.1 Case

4.1.1 Descrição

A classe `Case` representa um desafio técnico que pode ser resolvido por usuários. Possui descrição, solução esperada, pontuação e controle de estado.

4.1.2 Localização

- **Header:** `/include/backend/Case.hpp`
- **Implementação:** `/src/backend/Case.cpp`

4.1.3 Atributos

Tabela 4: Atributos da classe `Case`

Atributo	Tipo	Descrição
<code>_id</code>	<code>int</code>	Identificador único do caso
<code>_name</code>	<code>std::string</code>	Nome/título do caso
<code>_description</code>	<code>std::string</code>	Descrição detalhada
<code>_solution</code>	<code>std::string</code>	Solução esperada
<code>_state</code>	<code>bool</code>	Estado ativo/inativo
<code>_points</code>	<code>double</code>	Pontuação que vale

4.1.4 Métodos Principais

Construtor

```
1 Case( const std::string& name ,
2       const std::string& description);
```

Getters

- `int getId() const`
- `const std::string& getName() const`
- `const std::string& getDescription() const`
- `const std::string& getSolution() const`
- `bool getState() const`
- `double getPoints() const`

Setters

- `void setName(const std::string& name)`
- `void setDescription(const std::string& description)`
- `void setSolution(const std::string& solution)`
- `void setState(bool state)`
- `void setPoints(double points)`

Outros Métodos

- `void attemptSolve(User* user, Company* company, const string& solution)`
- `void displayCase() const`

4.1.5 Características

- Sistema de validação de solução
- Pontuação configurável
- Controle de disponibilidade via estado
- Integração com sistema de pontuação

4.2 CaseSystem

4.2.1 Descrição

A classe `CaseSystem` gerencia coleções de casos, tanto globais quanto específicos de empresas.

4.2.2 Localização

- **Header:** `/include/backend/CaseSystem.hpp`
- **Implementação:** `/src/backend/CaseSystem.cpp`

4.2.3 Atributos

Tabela 5: Atributos da classe `CaseSystem`

Atributo	Tipo	Descrição
<code>_cases</code>	<code>vector<shared_ptr<Case></code>	Coleção de casos gerenciados

4.2.4 Métodos Principais

Getters

- `shared_ptr<Case> getCase(int caseId) const`
- `const vector<shared_ptr<Case>& getAllCases() const`
- `vector<shared_ptr<Case> getActiveCases() const`
- `vector<shared_ptr<Case> getInactiveCases() const`

Gerenciamento

- `void addCase(shared_ptr<Case> newCase)`
- `void removeCaseByName(std::string Casename)`
- `void activateCase(int caseId)`
- `void deactivateCase(int caseId)`

4.2.5 Características

- Centraliza gerenciamento de casos
- Filtragem por estado (ativo/inativo)
- Usa smart pointers para segurança
- Suporta busca por ID

5 Sistema de Ranking

5.1 Ranking

5.1.1 Descrição

A classe `Ranking` gerencia coleções de usuários e fornece ordenação por pontuação. Serve como base para rankings globais e específicos.

5.1.2 Localização

- **Header:** `/include/backend/Ranking.hpp`
- **Implementação:** `/src/backend/Ranking.cpp`

5.1.3 Atributos

Tabela 6: Atributos da classe `Ranking`

Atributo	Tipo	Descrição
<code>_users</code>	<code>vector<shared_ptr<User>></code>	Usuários no ranking

5.1.4 Métodos Principais

Getters

- `const vector<shared_ptr<User>& getUsers()`
- `const int getSize()`

Outros Métodos

- `virtual void addUser(shared_ptr<User> user)`
- `void sort()`
- `void display() const`

5.1.5 Características

- Classe concreta instanciável
- Ordenação automática por pontuação
- Usado para ranking global
- Base para `CompanyRanking`

5.2 CompanyRanking

5.2.1 Descrição

A classe `CompanyRanking` herda de `Ranking` e representa o ranking específico de uma empresa, baseado nas pontuações dos usuários naquela empresa.

5.2.2 Localização

- **Header:** `/include/backend/CompanyRanking.hpp`
- **Implementação:** `/src/backend/CompanyRanking.cpp`

5.2.3 Atributos

Tabela 7: Atributos da classe `CompanyRanking`

Atributo	Tipo	Descrição
<code>_company</code>	<code>Company*</code>	Referência à empresa dona do ranking

5.2.4 Métodos Principais

Construtor

```
1 CompanyRanking(Company* _company);
```

5.2.5 Características

- Herda de `Ranking`
- Ranking específico por empresa
- Ordena por pontuação na empresa
- Gerenciado pela classe `Company`

6 Rede Social

6.1 Post

6.1.1 Descrição

A classe `Post` representa uma postagem na rede social, contendo autor, conteúdo, timestamp, comentários e curtidas.

6.1.2 Localização

- **Header:** `/include/backend/Post.hpp`
- **Implementação:** `/src/backend/Post.cpp`

6.1.3 Estruturas Auxiliares

Comment

```
1 struct Comment {  
2     int index;  
3     std::string author;  
4     std::string content;  
5     Comment(int i, std::string a,  
6             const std::string& c);  
7 };
```

6.1.4 Atributos

Tabela 8: Atributos da classe `Post`

Atributo	Tipo	Descrição
<code>_author</code>	<code>shared_ptr<Entity></code>	Autor da postagem
<code>_content</code>	<code>std::string</code>	Conteúdo
<code>_timestamp</code>	<code>std::time_t</code>	Data e hora
<code>_comments</code>	<code>vector<Comment></code>	Comentários
<code>_liked_by</code>	<code>vector<string></code>	Quem curtiu

6.1.5 Métodos Principais

Getters

- `shared_ptr<Entity> get_author() const`
- `std::string get_content() const`
- `std::time_t get_timestamp() const`

- `vector<Comment> get_comments() const`
- `vector<string> get_liked_by() const`
- `int get_num_likes() const`
- `int get_num_comments() const`

Interações

- `void addComment(const string& author, const string& content)`
- `void removeComment(int index)`
- `void addLiker(const string& liker)`
- `void removeLiker(const string& liker)`

6.1.6 Características

- Sistema de comentários indexados
- Sistema de curtidas
- Timestamp automático
- Sobrecarga de operadores

6.2 Feed

6.2.1 Descrição

A classe `Feed` representa um feed de postagens, com ordenação por data.

6.2.2 Localização

- **Header:** `/include/backend/Feed.hpp`
- **Implementação:** `/src/backend/Feed.cpp`

6.2.3 Atributos

Tabela 9: Atributos da classe `Feed`

Atributo	Tipo	Descrição
<code>_posts</code>	<code>vector<shared_ptr<Post>></code>	Coleção de postagens

6.2.4 Métodos Principais

Gerenciamento

- `void addPost(const shared_ptr<Post>& post)`
- `void removePost(const shared_ptr<Post>& post)`
- `void addPosts(const vector<shared_ptr<Post>>& posts)`
- `void removePosts(const vector<shared_ptr<Post>>& posts)`

Utilidades

- `void clear()`
- `void sortByDate()`

6.2.5 Características

- Ordenação cronológica
- Operações em lote
- Gerenciamento eficiente

6.3 SocialNetwork

6.3.1 Descrição

A classe `SocialNetwork` gerencia todas as funcionalidades de rede social: feed personalizado, postagens, seguidores e interações.

6.3.2 Localização

- **Header:** `/include/backend/SocialNetwork.hpp`
- **Implementação:** `/src/backend/SocialNetwork.cpp`

6.3.3 Atributos

Tabela 10: Atributos da classe `SocialNetwork`

Atributo	Tipo	Descrição
<code>feed</code>	<code>Feed</code>	Feed do usuário
<code>current_user</code>	<code>const shared_ptr<Entity>*</code>	Usuário atual
<code>all_users</code>	<code>const vector<shared_ptr<Entity>*>*</code>	Todos usuários
<code>all_companies</code>	<code>const vector<shared_ptr<Entity>*>*</code>	Todas empresas
<code>all_posts</code>	<code>vector<shared_ptr<Post>*>*</code>	Todas postagens
<code>_rankingGlobal</code>	<code>Ranking*</code>	Ranking global

6.3.4 Métodos Principais

Visualização

- `void viewMyFeed()`
- `void viewMyPosts()`
- `void viewProfile(string profile_name)`

Postagens

- `void createPost(const string& content)`
- `void deletePost(const string& content)`

Relacionamentos

- `void followUser(const string& followed_name)`
- `void unfollowUser(const string& unfollowed_name)`

Interações

- `void commentPost(const int which_post)`
- `void removeCommentPost(const int which_post)`
- `void likePost(const int which_post)`
- `void unlikePost(const int which_post)`

Rankings

- `void displayGlobalRanking()`
- `void displayUserRanking()`

6.3.5 Características

- Gerenciamento completo da rede social
- Feed personalizado por seguidos
- Sistema de interações completo
- Usa apenas referências (sem ownership)

7 Sistema Central

7.1 System

7.1.1 Descrição

A classe `System` é o orquestrador central que integra todas as funcionalidades: autenticação, rede social, casos, rankings e persistência.

7.1.2 Localização

- **Header:** `/include/backend/System.hpp`
- **Implementação:** `/src/backend/System.cpp`

7.1.3 Atributos

Tabela 11: Atributos da classe `System`

Atributo	Tipo	Descrição
<code>dbManager</code>	<code>DbManager</code>	Gerenciador de BD
<code>socialNetwork</code>	<code>SocialNetwork</code>	Rede social
<code>caseSystemGlobal</code>	<code>CaseSystem</code>	Casos globais
<code>rankingGlobal</code>	<code>Ranking</code>	Ranking global
<code>all_users</code>	<code>vector<shared_ptr<Entity>></code>	Todos usuários
<code>current_user</code>	<code>shared_ptr<Entity></code>	Usuário logado
<code>all_companies</code>	<code>vector<shared_ptr<Entity>></code>	Todas empresas
<code>all_posts</code>	<code>vector<shared_ptr<Post>></code>	Todas postagens
<code>modified</code>	<code>bool</code>	Flag de modificações

7.1.4 Métodos Principais

Persistência

- `void saveAllData()`
- `void loadData()`
- `void loadUsers()`
- `void loadCompanies()`
- `void loadGlobalCases()`
- `void loadGlobalRanking()`
- `void loadPosts()`

Autenticação

- `bool login(const string& username, const string& password)`
- `void logout()`
- `bool registerUser(...)`
- `bool registerCompany(...)`

Visualização

- `void viewMyFeed()`
- `void viewMyPosts()`
- `void viewProfile(const string& profile_name)`
- `void viewUserProfile(const string& name)`

Casos

- `void listGlobalCases()`
- `void listCompanyCases(int companyId)`
- `void submitCaseSolution(int caseId, const string& answer)`
- `void searchCases(const string& name)`

Rankings

- `void viewGlobalRanking()`
- `void viewCompanyRanking(int companyId)`
- `void viewMyRanking()`

Rede Social

- `void createPost(const string& content)`
- `void deletePost(const string& content)`
- `void followUser(const string& personName)`
- `void unfollowUser(const string& personName)`
- `void listFollowers()`

- `void listFollowing()`
- `void commentPost(const int which_post)`
- `void likePost(const int which_post)`

Interface

- `void displayMenu()`
- `void updateUserDescription(const string& newDescription)`

7.1.5 Características

- Padrão Facade (fachada)
- Integra todos os módulos
- Gerencia estado global
- Controla persistência
- Gerencia sessão de usuário

8 Gerenciamento de Banco de Dados

8.1 DbManager

8.1.1 Descrição

A classe `DbManager` gerencia todas as operações de persistência usando SQLite, incluindo CRUD completo para todas as entidades.

8.1.2 Localização

- **Header:** `/include/database/DbManager.hpp`
- **Implementação:** `/src/database/DbManager.cpp`

8.1.3 Atributos

Tabela 12: Atributos da classe `DbManager`

Atributo	Tipo	Descrição
db	sqlite3*	Conexão SQLite
dbPath	std::string	Caminho do arquivo de BD

8.1.4 Métodos Principais

Construtor

```
1 DbManager(const std::string& caminhoDb
2           = "rede_social.db");
```

Usuários

- `bool saveUsers(const vector<User>& usuarios)`
- `vector<User> loadUsers()`

Empresas

- `bool saveCompany(const vector<Company>& companies)`
- `vector<Company> loadCompanies()`

Casos

- `bool saveCase(const CaseSystem& caseSystem)`
- `CaseSystem loadGlobalCases()`
- `bool saveCompanyCases(int companyId, const CaseSystem& caseSystem)`
- `CaseSystem loadCompanyCases(int companyId)`

Postagens

- `bool savePost(const vector<shared_ptr<Post>>& posts)`
- `vector<shared_ptr<Post>> loadPosts()`

Rankings

- `bool saveGlobalRanking(const vector<shared_ptr<User>>& users)`
- `vector<int> loadGlobalRanking()`
- `bool saveCompanyRanking(int companyId, const vector<shared_ptr<User>>& users)`
- `vector<int> loadCompanyRanking(int companyId)`

Casos Resolvidos

- `bool saveSolvedCases(int userId, const vector<int>& solvedCaseIds)`
- `vector<int> loadSolvedCases(int userId)`

Utilitários

- `bool loadDB()`
- `void closeDB()`
- `bool verifyConnection()`

8.1.5 Métodos Privados

- `void createTables()`
- `bool executerSQL(const string& sql)`

8.1.6 Características

- Camada de persistência completa
- SQLite para armazenamento local
- Criação automática de tabelas
- Tratamento robusto de erros
- Transações atômicas

9 Utilitários

9.1 Utils

9.1.1 Descrição

O namespace `Utils` fornece funções auxiliares para busca de entidades e postagens.

9.1.2 Localização

- **Header:** `/include/backend/Utils.hpp`
- **Implementação:** `/src/backend/Utils.cpp`

9.1.3 Funções

Busca de Entidades

```
1 std::vector<std::shared_ptr<Entity>>
2 findEntities(
3     const std::vector<std::shared_ptr<Entity>>&
4         all_entities,
5     const std::vector<std::string>& names);
6
7 std::shared_ptr<Entity>
8 findEntity(
9     const std::vector<std::shared_ptr<Entity>>&
10         all_entities,
11     const std::string& name);
```

Busca de Postagens

```
1 std::vector<std::shared_ptr<Post>>
2 findPosts(
3     const std::vector<std::shared_ptr<Post>>&
4         all_posts,
5     const std::vector<std::string>& names);
6
7 std::vector<std::shared_ptr<Post>>
8 findPosts(
9     const std::vector<std::shared_ptr<Post>>&
10         all_posts,
11     const std::string& name);
```

9.1.4 Características

- Funções utilitárias para busca
- Sobrecarga para busca única/múltipla

- Trabalha com ponteiros compartilhados
- Namespace para organização

10 Relacionamentos entre Classes

10.1 Diagrama de Herança

Entity (abstrata)

User

Company

Ranking (concreta)

CompanyRanking

10.2 Relacionamentos de Composição

- **System** contém:
 - DbManager (composição)
 - SocialNetwork (composição)
 - CaseSystem (composição)
 - Ranking (composição)
 - Vetores de entidades e postagens
- **Company** contém:
 - CompanyRanking* (agregação)
 - CaseSystem* (agregação)
- **SocialNetwork** contém:
 - Feed (composição)
 - Referências (não ownership)
- **User** contém:
 - Map de pontuações por empresa
 - Vector de casos resolvidos

10.3 Dependências

- Case depende de User e Company
- CompanyRanking herda de Ranking

- `DbManager` depende de todas as entidades
- `Utils` depende de `Entity` e `Post`
- `System` orquestra todos os módulos

11 Padrões de Projeto

11.1 Facade Pattern

A classe `System` atua como fachada, fornecendo interface simplificada para todos os subsistemas.

11.2 Template Method Pattern

A classe `Entity` define estrutura base, delegando comportamentos específicos através de `displayProfile()`.

11.3 Strategy Pattern

Diferentes rankings (global e por empresa) podem usar diferentes estratégias de ordenação.

11.4 Smart Pointers

Uso extensivo de `std::shared_ptr` para gerenciamento automático de memória.

12 Fluxo de Dados

12.1 Ciclo de Vida do Sistema

1. Inicialização:

- System é criado
- DbManager abre conexão
- Dados são carregados

2. Autenticação:

- Login do usuário
- System define `current_user`
- SocialNetwork é configurado

3. Operações:

- Interações do usuário
- Modificações rastreadas
- Dados mantidos em memória

4. Persistência:

- Salvamento via DbManager
- Escrita no SQLite
- Flag modified limpa

5. Finalização:

- Logout/encerramento
- Dados salvos
- Conexão fechada
- Recursos liberados

12.2 Fluxo de Resolução de Casos

1. Usuário visualiza casos disponíveis
2. Seleciona um caso
3. Submete solução via `submitCaseSolution()`

4. Case valida em `attemptSolve()`
5. Se correto:
 - Pontos adicionados ao User
 - Pontuação registrada para Company
 - Caso marcado como resolvido
 - Rankings atualizados
6. Persistência no banco

12.3 Fluxo de Feed

1. Usuário solicita feed
2. SocialNetwork coleta posts dos seguidos
3. Feed populado e ordenado
4. Posts exibidos
5. Interações possíveis (curtir, comentar)
6. Atualizações persistidas

13 Banco de Dados

13.1 Estrutura de Tabelas

O sistema utiliza SQLite com as seguintes tabelas:

- **users:** Dados dos usuários
- **companies:** Dados das empresas
- **cases:** Casos globais
- **company_cases:** Casos por empresa
- **posts:** Postagens
- **global_ranking:** Ordem do ranking global
- **company_ranking:** Rankings por empresa
- **solved_cases:** Histórico de soluções

13.2 Persistência

- Criação automática de tabelas
- Validação de conexões
- Transações para consistência
- Métodos de save/load por entidade
- Tratamento de erros SQL

14 Considerações de Segurança

14.1 Autenticação

- Senhas verificadas via `checkPassword()`
- Validação de credenciais no login
- Operações requerem autenticação

14.2 Validação de Dados

- CPF e CNPJ armazenados para validação futura
- IDs únicos previnem conflitos
- Estado ativo/inativo permite desativação segura

14.3 Integridade Referencial

- `shared_ptr` garante vida dos objetos
- Referências sem ownership em `SocialNetwork`
- Verificação de existência antes de operações

15 Gerenciamento de Memória

15.1 Smart Pointers

Uso extensivo de `std::shared_ptr` para:

- Gerenciamento automático
- Compartilhamento seguro
- Prevenção de vazamentos
- Destruição automática

15.2 Ownership

- **System:** Possui todos os dados principais
- **Company:** Possui `CaseSystem` e `CompanyRanking`
- **CaseSystem:** Compartilha Cases
- **Ranking:** Compartilha Users
- **SocialNetwork:** Apenas referências

15.3 Containers STL

- `std::vector` para coleções
- `std::map` para associações
- `std::string` para texto

16 Extensibilidade

16.1 Novos Tipos de Entidades

Para adicionar novo tipo:

1. Herdar de `Entity`
2. Implementar `displayProfile()`
3. Adicionar atributos específicos
4. Atualizar `DbManager`

16.2 Customização de Rankings

Para rankings customizados:

1. Herdar de `Ranking`
2. Sobrescrever `sort()`
3. Implementar critérios específicos

16.3 Novos Tipos de Cases

1. Herdar de `Case`
2. Implementar validação específica
3. Adicionar campos necessários

17 Conclusão

O sistema Synergy implementa uma arquitetura robusta e extensível para rede social profissional focada em desafios técnicos.

17.1 Resumo

- **12 classes principais** bem organizadas
- **Sistema de herança** com classes abstratas
- **Persistência completa** via SQLite
- **Gerenciamento de memória** com smart pointers
- **Padrões de projeto** bem aplicados
- **Separação de responsabilidades** clara

17.2 Pontos Fortes

- Arquitetura OO bem estruturada
- Herança e polimorfismo adequados
- Gerenciamento automático de memória
- Persistência completa de dados
- Sistema modular e extensível
- Documentação inline com Doxygen

17.3 Melhorias Futuras

- Implementar hash de senhas
- Validação de CPF/CNPJ
- Sistema de notificações
- Testes unitários
- Logging de operações
- Tratamento robusto de exceções

17.4 Manutenibilidade

- Separação headers/implementação
- Comentários Doxygen
- Nomenclatura consistente
- Estrutura lógica de pastas

Esta documentação serve como guia completo para desenvolvedores trabalhando no Sistema Synergy .