

MEMORY BOOK

Um projeto que transforma lembranças em
pontos interativos no mapa








Autores:
Alberto Pontieri
Diogo Nascimento
Guilherme Pança

(link do vídeo da aplicação).

OBJETIVO:

Registrar momentos especiais (texto, fotos, descrições e localizações) e guardá-los em um espaço visual e afetivo.

FUNCIONALIDADES PRINCIPAIS:

-  Marcar lugares importantes no mapa interativo (React-Leaflet)
-  CRUD de memórias (criar, visualizar, editar, excluir)
-  Upload de fotos e cores personalizadas
-  Integração com Spotify para adicionar trilha sonora
-  Sistema de autenticação e perfis de usuário
-  Temas e gradientes personalizáveis
-  Interface responsiva para desktop e mobile

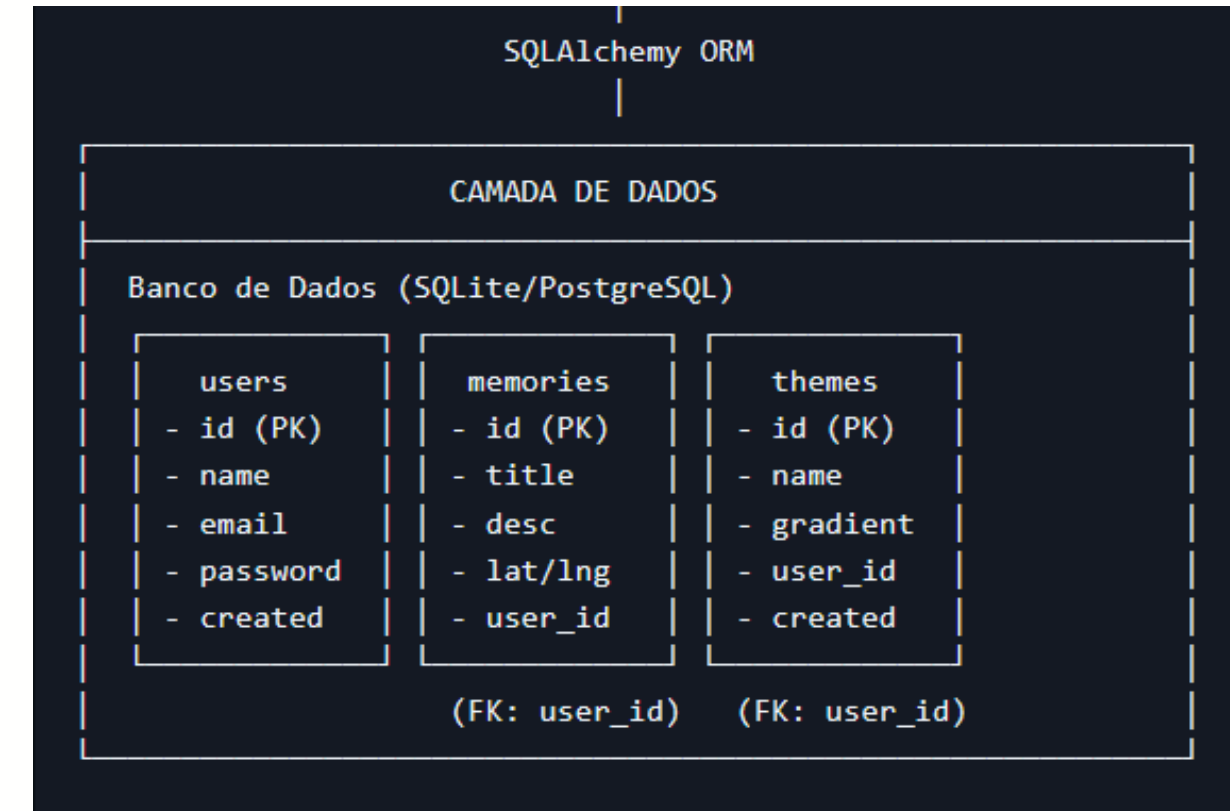
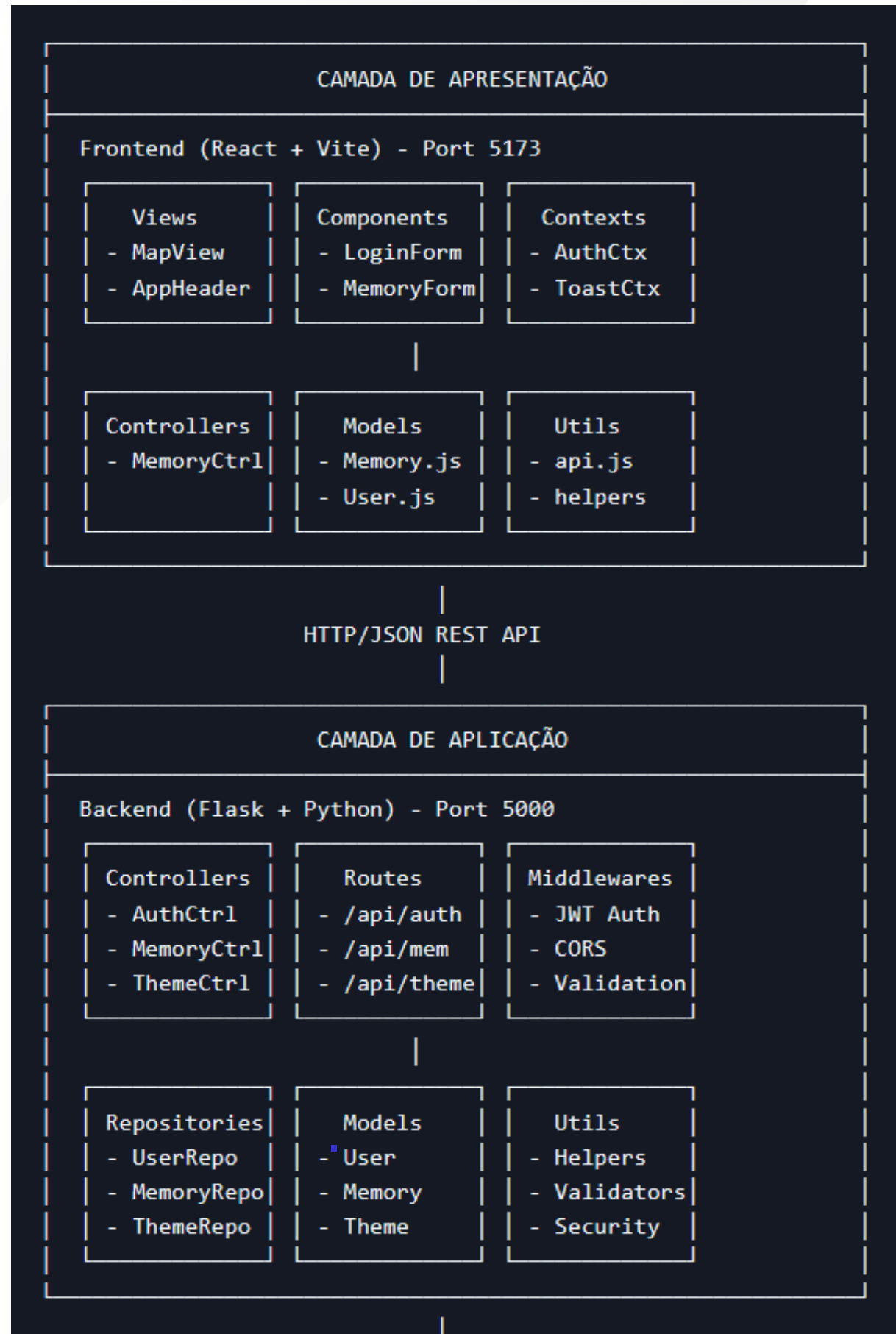
PADRÃO DE ARQUITETURA ADOTADO:

Arquitetura Cliente-Servidor em Camadas com o padrão MVC
(Model-View-Controller).

Justificativa:

Equilibra simplicidade e extensibilidade, isolando interface, regras de negócio e persistência. Garante robustez, testabilidade e escalabilidade.

DIAGRAMA DE ARQUITETURA:



PADRÃO DE PROJETO OBSERVER

O QUE É?

é um padrão que permite que várias partes do sistema sejam avisadas automaticamente quando algo muda.

POR QUE UTILIZAMOS?

Para sincronizar automaticamente a interface quando dados importantes mudam (como login/logout do usuário).

QUE PROBLEMA RESOLVE?

Quando o usuário faz login, TODOS os componentes da tela precisam saber disso para se atualizar. Solução: Um "observador central" avisa todos os componentes interessados automaticamente.

****Situação:**** Usuário faz login → AppHeader mostra contador de memórias

```
``javascript
// AuthContext.jsx (SUBJECT)
const [user, setUser] = useState(null);
const [isAuthenticated, setIsAuthenticated] = useState(false);

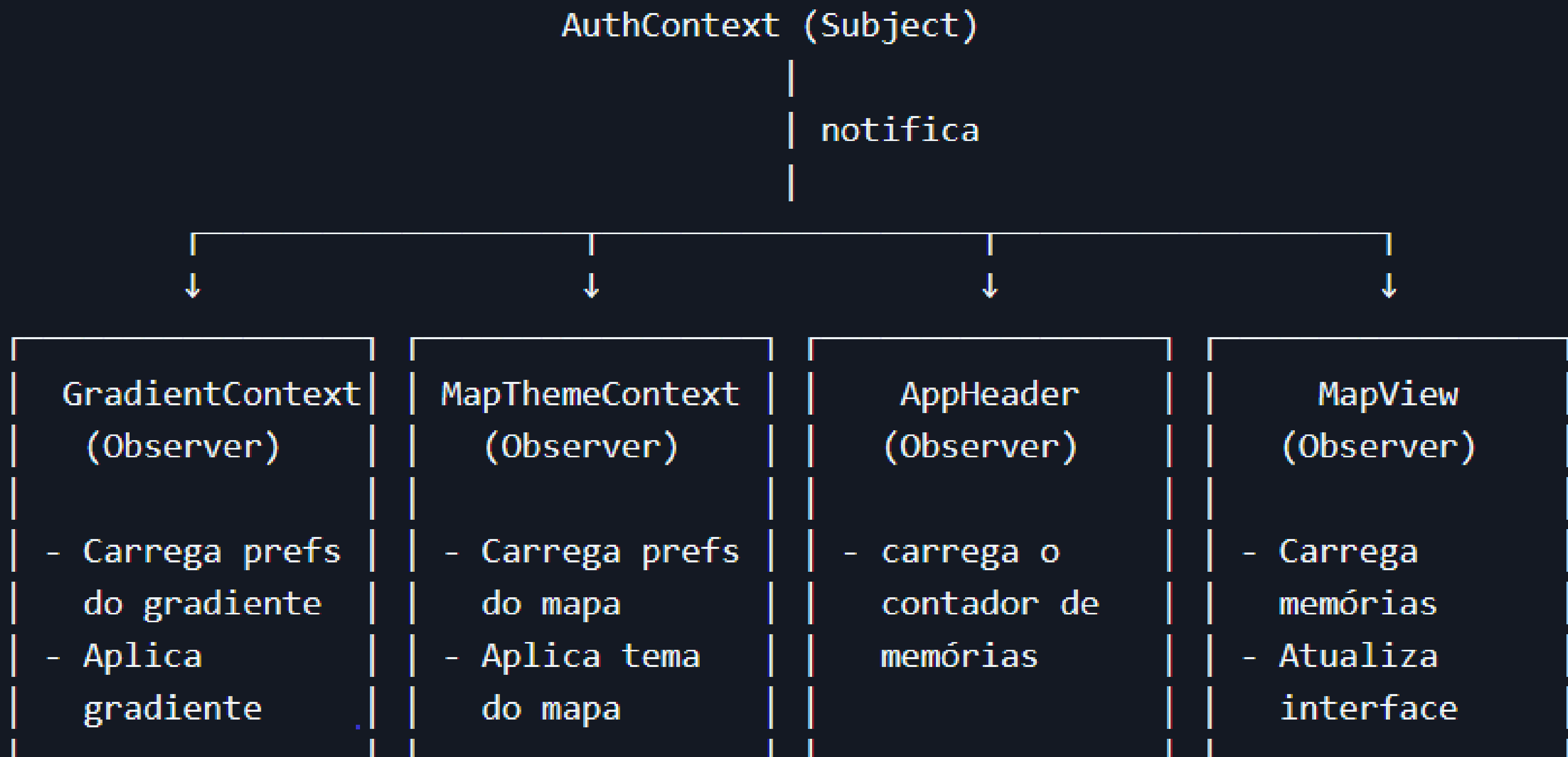
// Quando login acontece:
setUser(userData);
setIsAuthenticated(true); // ← NOTIFICA TODOS!

// AppHeader.jsx (OBSERVER)
const { isAuthenticated } = useAuth(); // ← ESCUTA mudanças
const { memoriesCount } = useMemories();

// Automaticamente mostra contador quando autenticado:
{isAuthenticated && (
  <span>📌 {memoriesCount} memórias</span>
)}
...

```

DIAGRAMA DO PADRÃO OBSERVER:



PADRÃO DE PROJETO FACTORY

O QUE É?

Um padrão que cria objetos de forma padronizada, como uma fábrica que produz produtos seguindo sempre o mesmo processo.

POR QUE UTILIZAMOS?

Para garantir que usuários, memórias e temas sejam sempre criados corretamente, com validações automáticas e campos obrigatórios preenchidos.

QUE PROBLEMA RESOLVE?

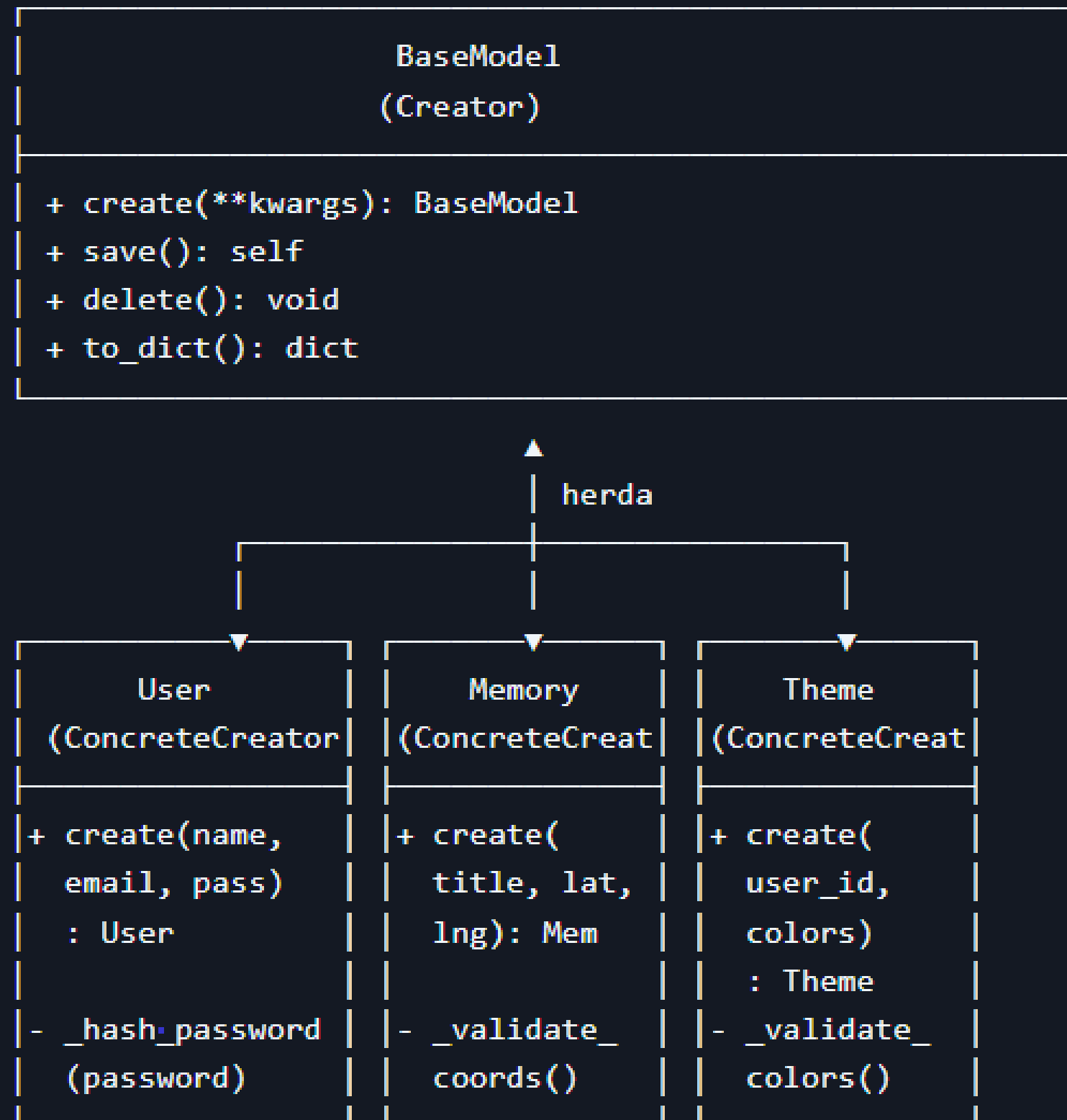
Evita erros humanos na criação de dados - senhas sempre criptografadas, IDs únicos gerados automaticamente, datas preenchidas corretamente.

```
def create(self, **kwargs) -> Any:
    """
    Cria uma nova instância do modelo

    Args:
        **kwargs: Dados para criação

    Returns:
        Instância criada do modelo
    """
    instance = self.model_class.create(**kwargs)
    db.session.commit()
    return instance
```

DIAGRAMA DO PADRÃO FACTORY:



PADRÃO DE PROJETO REPOSITORY

O QUE É?

Um padrão que funciona como um "bibliotecário" - sabe onde encontrar e como organizar os dados, sem que você precise saber os detalhes.

POR QUE UTILIZAMOS?

Para separar a lógica de negócio do acesso ao banco de dados, mantendo o código organizado e fácil de manter.

QUE PROBLEMA RESOLVE?

Se mudarmos o banco de dados, só precisamos alterar os repositórios. Os controllers continuam funcionando normalmente, sem quebrar nada.

```
# Criar instância do Flask
app = Flask(__name__)

# Aplicar configurações
app.config.from_object(config_class)

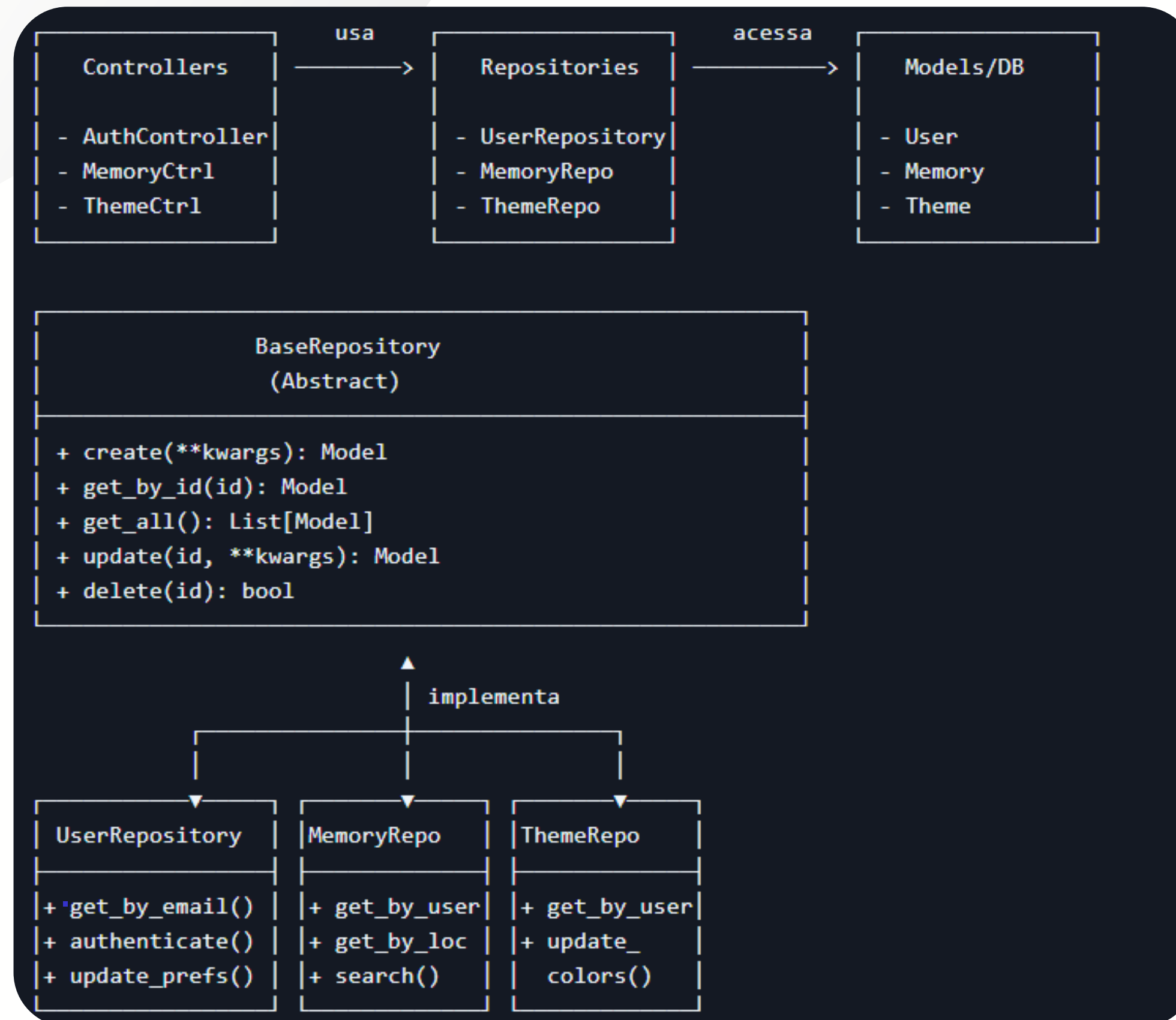
# Inicializar extensões
db.init_app(app)
jwt.init_app(app)

# Configurar CORS
CORS(app, origins=app.config['CORS_ORIGINS'])

# Registrar blueprints (controllers)
from src.controllers.auth_controller import auth_bp
from src.controllers.memory_controller import memory_bp
from src.controllers.theme_controller import theme_bp

app.register_blueprint(auth_bp, url_prefix='/api/auth')
app.register_blueprint(memory_bp, url_prefix='/api/memories')
app.register_blueprint(theme_bp, url_prefix='/api/themes')
```

DIAGRAMA DO PADRÃO REPOSITORY:



PADRÃO DE PROJETO FACADE

O QUE É?

É um padrão que fornece uma interface única e simplificada para um conjunto de subsistemas ou operações complexas. Ele "esconde" a complexidade.

POR QUE UTILIZAMOS?

Para desacoplar os componentes da interface (React) da lógica complexa de comunicação com a API, como autenticação e tratamento de erros.

QUE PROBLEMA RESOLVE?

Um componente (ex: LoginForm) não deveria precisar saber como gerenciar tokens JWT, montar headers de autenticação ou tratar códigos de erro da API. Solução: O Facade fornece uma "fachada" única (ApiFacade) que o componente chama com um método simples (ex: login()), e o Facade esconde e gerencia toda essa complexidade internamente.

```
const LoginForm = ({ onSwitchToRegister }) => {
  const handleSubmit = async (e) => {

    try {
      const result = await login(formData.email, formData.password);

      if (!result || !result.success) {
        const errorMessage = result?.error || 'Erro ao fazer login. Tente novamente.';
        const errorSuggestion = result?.suggestion || null;
        const errorType = result?.errorType || null;
      }
    }
  }
}
```

```
class ApiFacade {
  static async login(credentials) {
    const response = await this.#makeRequest('/auth/login', {
      method: 'POST',
      body: JSON.stringify(credentials),
    });

    if (response.access_token) {
      TokenManager.setToken(response.access_token);
    }

    return response;
  }
}
```

```
class ApiFacade {
  static async #makeRequest(endpoint, options = {}) {
    const url = `${API_BASE_URL}${endpoint}`;
    const token = TokenManager.getToken();

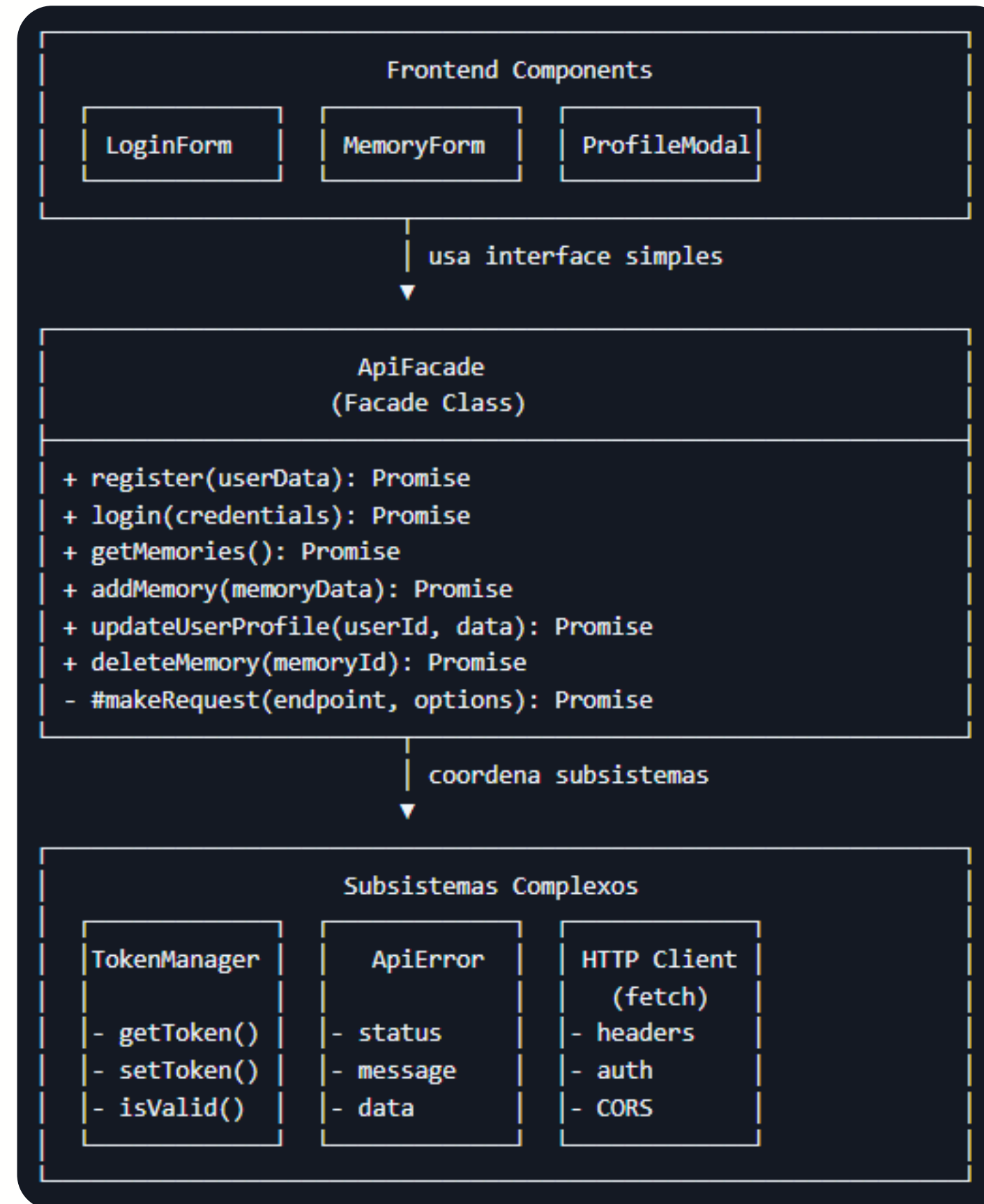
    const defaultOptions = {
      // ...
    };

    const finalOptions = {
      // ...
    };

    try {
      const response = await fetch(url, finalOptions);
      const data = await response.json();

      if (!response.ok) {
        const errorMsg = (data && (data.message || data.error || data.detail)) || 'Erro na requisição';
        throw new ApiError(
          errorMsg,
          response.status,
          data
        );
      }
    }
  }
}
```

DIAGRAMA DO PADRÃO FACADE:



MUITO
OBRIGADO!
