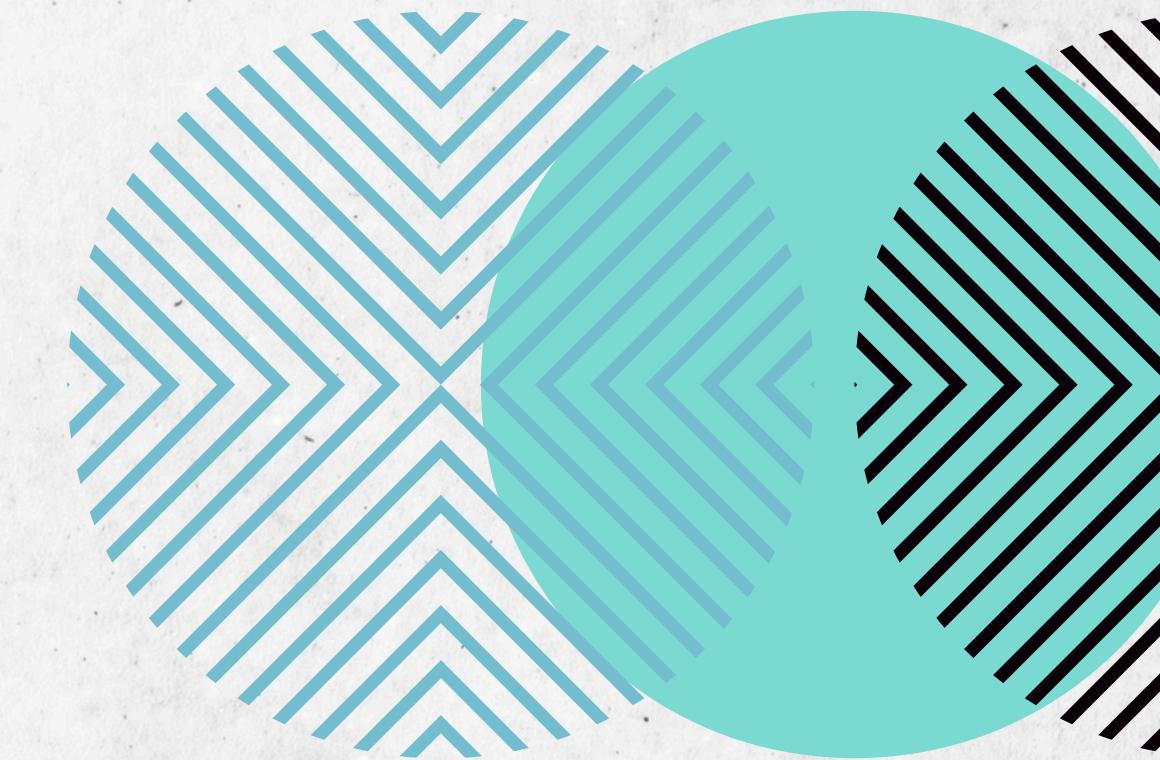


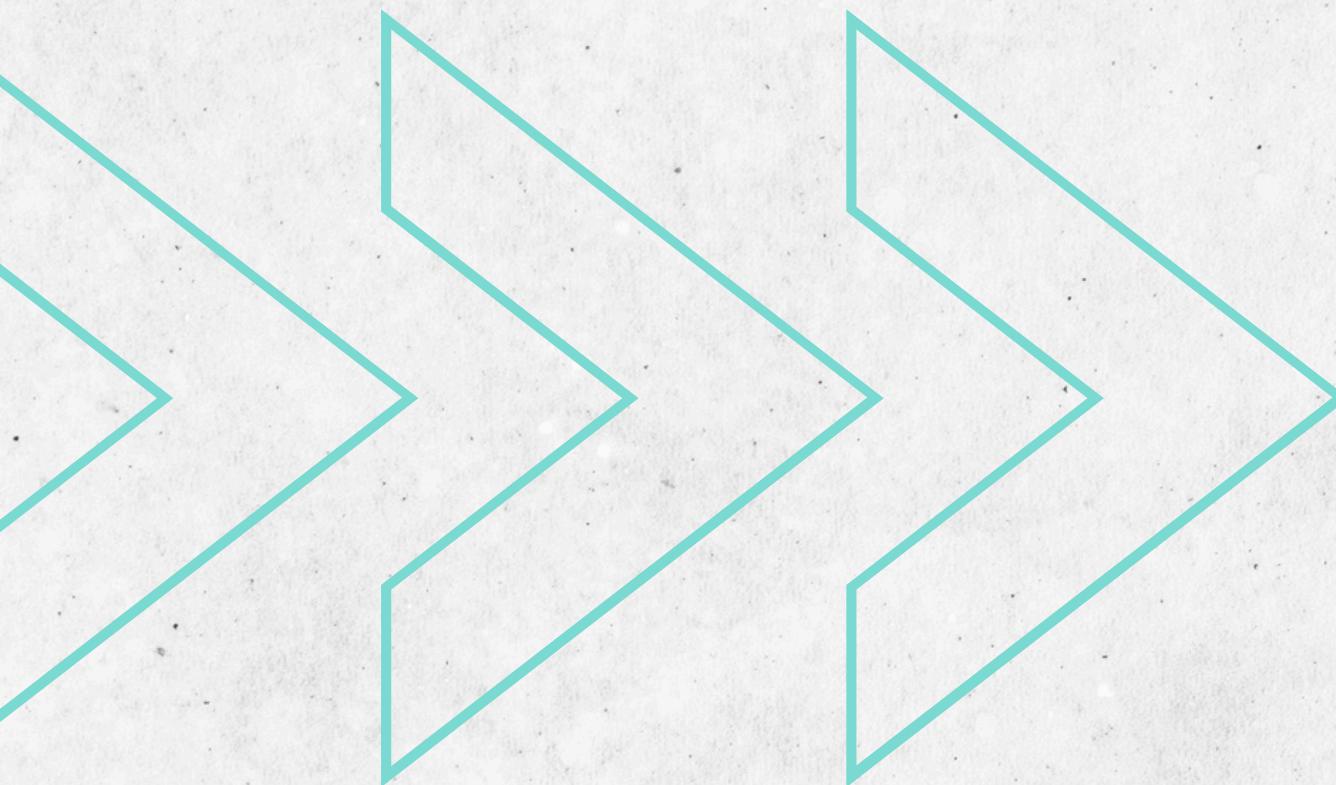
Sistema de Empréstimo de Livros via RPC



Autores: Herbert de Souza Mariano
Gabriel Perez Vargas de Vasconcelos
Merhi Osolins Daychoum



índice



- 01
- 02
- 03
- 05
- 06

- Linguagem escolhida
- Estratégia de Desenvolvimento
- Sobre o Projeto
- Características do Sistema
- Conclusão



Linguagem escolhida



Por que Go (Golang)?

1. Sintaxe simples e eficiente para sistemas concorrentes.
2. Suporte nativo à concorrência com goroutines.
3. Biblioteca padrão com suporte a RPC via net/rpc.
4. Excelente desempenho para aplicações de rede.
5. Fácil integração com SQLite e outras tecnologias.

Estratégia de Desenvolvimento Arquitetura e Estratégia

**ADOÇÃO DE UM MODELO DIVIDIDO
ENTRE CLIENTE E SERVIDOR**

**COMUNICAÇÃO VIA RPC (REMOTE
PROCEDURE CALL).**

**ARMAZENAMENTO PERSISTENTE
COM SQLITE.**

**ORGANIZAÇÃO MODULAR DO
CÓDIGO**





Estratégia de Desenvolvimento

Organização modular do código

CLIENT.GO

responsável por requisições.

SERVER.GO

onde a lógica do servidor e funções
RPC estão implementadas.

DATASTORE.GO

gerenciamento do banco de
dados e regras de negócio.

client.go



```
func realizarEmpréstimo(usuario, livro, dataEmpréstimo string) {
    client, err := rpc.Dial("tcp", ":1234")
    if err != nil {
        log.Fatal("Failed to connect to server:", err)
    }

    var reply string
    err = client.Call("Server.RealizarEmpréstimo", struct{ Usuario, Livro, DataEmpréstimo string }{Usuario: usuario, Livro: livro, DataEmpréstimo: dataEmpréstimo}, &reply)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(reply)
}

func devolverLivro(usuario, livro string) {
    client, err := rpc.Dial("tcp", ":1234")
    if err != nil {
        log.Fatal("Failed to connect to server:", err)
    }

    var reply string
    err = client.Call("Server.DevolverLivro", struct{ Usuario, Livro string }{Usuario: usuario, Livro: livro}, &reply)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(reply)
}
```

client.go



```
func consultarLivro(livro string) {
    client, err := rpc.Dial("tcp", ":1234")
    if err != nil {
        log.Fatal("Failed to connect to server:", err)
    }

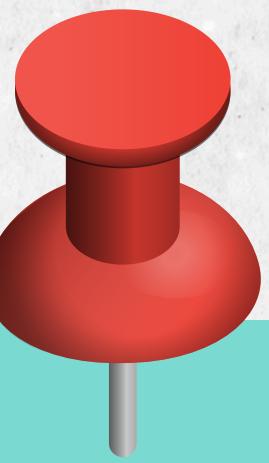
    var reply string
    err = client.Call("Server.ConsultarLivro", livro, &reply)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(reply)
}

func consultarEmpréstimosUsuario(usuario string) {
    client, err := rpc.Dial("tcp", ":1234")
    if err != nil {
        log.Fatal("Failed to connect to server:", err)
    }

    var reply string
    err = client.Call("Server.ConsultarEmpréstimosUsuario", usuario, &reply)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(reply)
}
```

```
func main() {
    // Exemplos de chamada das funções RPC
    realizarEmpréstimo("João", "Golang Programming", "2023-10-01")
    devolverLivro("João", "Golang Programming")
    consultarLivro("Golang Programming")
    consultarEmpréstimosUsuario("João")
    consultarEmpréstimosLivro("Golang Programming")
}
```

client.go



```
func consultarLivro(livro string) {
    client, err := rpc.Dial("tcp", ":1234")
    if err != nil {
        log.Fatal("Failed to connect to server:", err)
    }

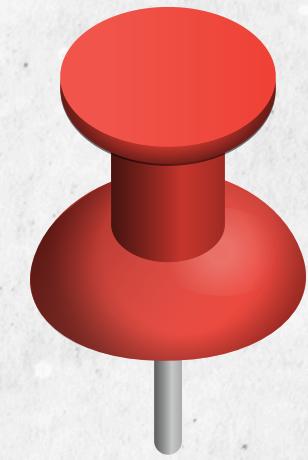
    var reply string
    err = client.Call("Server.ConsultarLivro", livro, &reply)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(reply)
}

func consultarEmpréstimosUsuario(usuario string) {
    client, err := rpc.Dial("tcp", ":1234")
    if err != nil {
        log.Fatal("Failed to connect to server:", err)
    }

    var reply string
    err = client.Call("Server.ConsultarEmpréstimosUsuario", usuario, &reply)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(reply)
}
```

```
func main() {
    // Exemplos de chamada das funções RPC
    realizarEmpréstimo("João", "Golang Programming", "2023-10-01")
    devolverLivro("João", "Golang Programming")
    consultarLivro("Golang Programming")
    consultarEmpréstimosUsuario("João")
    consultarEmpréstimosLivro("Golang Programming")
}
```

server.go



```
func getLivroID(titulo string) (int, error) {
    row := datastore.DB().QueryRow("SELECT id FROM livros WHERE titulo = ?", titulo)

    var id int
    err := row.Scan(&id)
    if err != nil {
        return 0, fmt.Errorf("livro não encontrado")
    }

    return id, nil
}
```



```
func (s *Server) RealizarEmpréstimo(args struct {
    Usuario      string
    Livro       string
    DataEmpréstimo string
}, reply *string) error {
    livroID, err := getLivroID(args.Livro)
    if err != nil {
        *reply = err.Error()
        return nil
    }

    err = datastore.RealizarEmprestimo(args.Usuario, livroID, args.DataEmpréstimo)
    if err != nil {
        *reply = err.Error()
        return nil
    }

    *reply = "Sucesso"
    return nil
}
```

server.go

```
func (s *Server) DevolverLivro(args struct {
    Usuario string
    Livro   string
}, reply *string) error {
    livroID, err := getLivroID(args.Livro)
    if err != nil {
        *reply = err.Error()
        return nil
    }

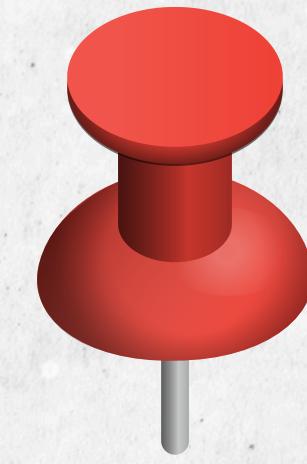
    err = datastore.DevolverLivro(args.Usuario, livroID)
    if err != nil {
        *reply = err.Error()
        return nil
    }

    *reply = "Sucesso"
    return nil
}
```

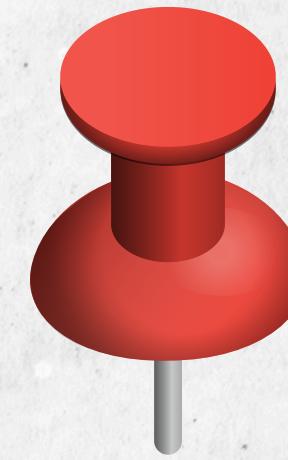
```
func (s *Server) ConsultarLivro(titulo string, reply *string) error {
    livroID, err := getLivroID(titulo)
    if err != nil {
        *reply = err.Error()
        return nil
    }

    resultado, err := datastore.ConsultarLivro(livroID)
    if err != nil {
        *reply = err.Error()
        return nil
    }

    *reply = resultado
    return nil
}
```



server.go



```
func (s *Server) ConsultarEmpréstimosUsuario(usuario string, reply *string) error {
    resultados, err := datastore.ConsultarEmpréstimosUsuario(usuario)
    if err != nil {
        *reply = err.Error()
        return nil
    }

    *reply = fmt.Sprintf("%v", resultados)
    return nil
}

func (s *Server) ConsultarEmpréstimosLivro(titulo string, reply *string) error {
    livroID, err := getLivroID(titulo)
    if err != nil {
        *reply = err.Error()
        return nil
    }

    resultados, err := datastore.ConsultarEmpréstimosLivro(livroID)
    if err != nil {
        *reply = err.Error()
        return nil
    }

    *reply = fmt.Sprintf("%v", resultados)
    return nil
}
```

```
func main() {
    // Inicializa o banco de dados SQLite
    if err := datastore.InitDB(); err != nil {
        log.Fatalf("Erro ao inicializar banco: %v", err)
    }

    // Registra o servidor RPC
    rpc.Register(new(Server))
    listener, err := net.Listen("tcp", ":1234")
    if err != nil {
        log.Fatal("Erro ao escutar na porta 1234:", err)
    }

    fmt.Println("Servidor escutando na porta 1234")
    for {
        conn, err := listener.Accept()
        if err != nil {
            log.Print("Erro ao aceitar conexão:", err)
            continue
        }
        go rpc.ServeConn(conn)
    }
}
```

datastore.go

```
var db *sql.DB

type Livro struct {
    ID      int
    Titulo  string
    Disponivel bool
}

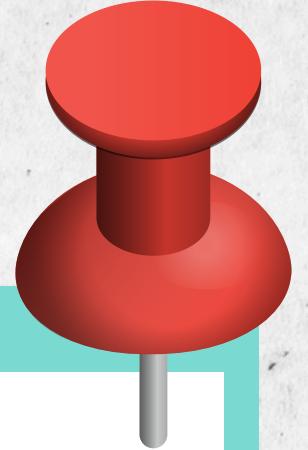
type Emprestimo struct {
    ID          int
    Usuario     string
    LivroID     int
    DataEmprestimo string
    DataDevolucao string
}
```

```
func InitDB() error {
    var err error
    db, err = sql.Open("sqlite", "./biblioteca.db")
    if err != nil {
        return err
    }

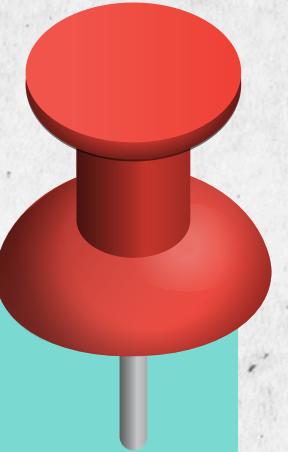
    createTables := `
CREATE TABLE IF NOT EXISTS livros (
    id INTEGER PRIMARY KEY,
    titulo TEXT NOT NULL,
    disponivel BOOLEAN NOT NULL
);

CREATE TABLE IF NOT EXISTS emprestimos (
    id INTEGER PRIMARY KEY,
    usuario TEXT NOT NULL,
    livro_id INTEGER NOT NULL,
    data_emprestimo TEXT,
    data_devolucao TEXT,
    FOREIGN KEY(livro_id) REFERENCES livros(id)
);
_, err = db.Exec(createTables)
if err != nil {
    return err
}

_, err = db.Exec(`INSERT OR IGNORE INTO livros (id, titulo, disponivel) VALUES (1, 'Golang Programming', 1)`)
return err
}
```



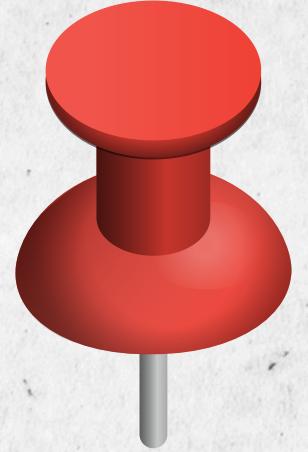
datastore.go



```
func DB() *sql.DB {
    return db
}

func ObterLivro(id int) (Livro, error) {
    var l Livro
    err := db.QueryRow("SELECT id, titulo, disponivel FROM livros WHERE id = ?", id).
        Scan(&l.ID, &l.Titulo, &l.Disponivel)
    if err != nil {
        return Livro{}, fmt.Errorf("livro não encontrado")
    }
    return l, nil
}
```

datastore.go



```
func RealizarEmprestimo(usuario string, livroID int, dataEmprestimo string) error {
    livro, err := ObterLivro(livroID)
    if err != nil {
        return err
    }
    if !livro.Disponivel {
        return fmt.Errorf("livro indisponível")
    }

    tx, err := db.Begin()
    if err != nil {
        return err
    }

    _, err = tx.Exec("INSERT INTO emprestimos (usuario, livro_id, data_emprestimo) VALUES (?, ?, ?)",
                    usuario, livroID, dataEmprestimo)
    if err != nil {
        tx.Rollback()
        return err
    }

    _, err = tx.Exec("UPDATE livros SET disponivel = 0 WHERE id = ?", livroID)
    if err != nil {
        tx.Rollback()
        return err
    }

    return tx.Commit()
}
```

datastore.go

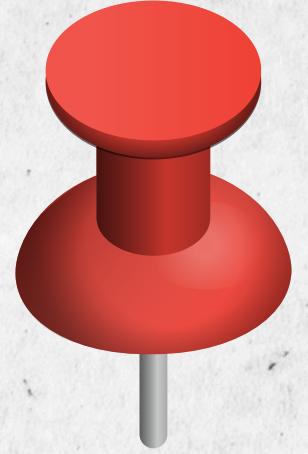
```
func DevolverLivro(usuario string, livroID int) error {
    tx, err := db.Begin()
    if err != nil {
        return err
    }

    var emprestimoID int
    row := tx.QueryRow("SELECT id FROM emprestimos WHERE usuario = ? AND livro_id = ?", usuario, livroID)
    err = row.Scan(&emprestimoID)
    if err != nil {
        tx.Rollback()
        return fmt.Errorf("emprestimo não encontrado")
    }

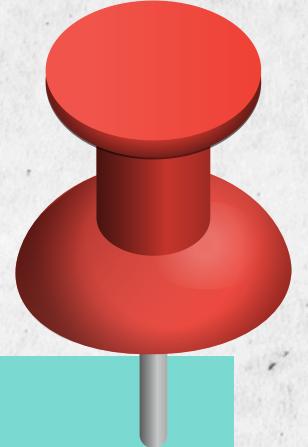
    _, err = tx.Exec("DELETE FROM emprestimos WHERE id = ?", emprestimoID)
    if err != nil {
        tx.Rollback()
        return err
    }

    _, err = tx.Exec("UPDATE livros SET disponivel = 1 WHERE id = ?", livroID)
    if err != nil {
        tx.Rollback()
        return err
    }

    return tx.Commit()
}
```



datastore.go

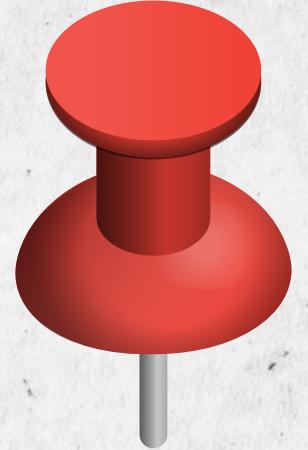


```
func ConsultarLivro(livroID int) (string, error) {
    livro, err := ObterLivro(livroID)
    if err != nil {
        return "", err
    }
    if livro.Disponivel {
        return "Disponível", nil
    }

    var usuario, data string
    err = db.QueryRow("SELECT usuario, data_emprestimo FROM emprestimos WHERE livro_id = ?", livroID).
        Scan(&usuario, &data)
    if err != nil {
        return "", fmt.Errorf("erro ao consultar empréstimo")
    }

    return fmt.Sprintf("Emprestado para %s desde %s", usuario, data), nil
}
```

datastore.go



```
func ConsultarEmprestimosUsuario(usuario string) ([]string, error) {
    rows, err := db.Query("SELECT livro_id, data_emprestimo, data_devolucao FROM emprestimos WHERE usuario = ?", usuario)
    if err != nil {
        return nil, err
    }
    defer rows.Close()

    var result []string
    for rows.Next() {
        var livroID int
        var dataEmprestimo, dataDevolucao string
        rows.Scan(&livroID, &dataEmprestimo, &dataDevolucao)

        status := "no prazo"
        if dataDevolucao != "" {
            dt, _ := time.Parse("2006-01-02", dataDevolucao)
            if time.Now().After(dt) {
                status = "atrasado"
            }
        }

        livro, _ := ObterLivro(livroID)
        result = append(result, fmt.Sprintf("Título: %s, Data: %s, Status: %s", livro.Titulo, dataEmprestimo, status))
    }

    if len(result) == 0 {
        return nil, fmt.Errorf("nenhum empréstimo encontrado para o usuário")
    }
}
```

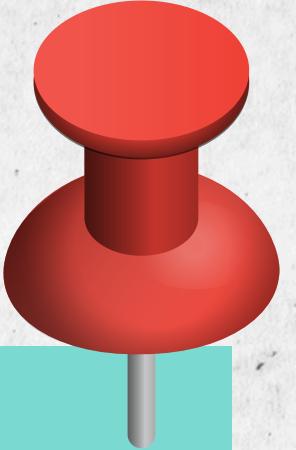
datastore.go

```
func ConsultarEmprestimosLivro(livroID int) ([]string, error) {
    rows, err := db.Query("SELECT usuario, data_emprestimo FROM emprestimos WHERE livro_id = ?", livroID)
    if err != nil {
        return nil, err
    }
    defer rows.Close()

    var result []string
    for rows.Next() {
        var usuario, data string
        rows.Scan(&usuario, &data)
        result = append(result, fmt.Sprintf("Usuário: %s, Data Empréstimo: %s", usuario, data))
    }

    if len(result) == 0 {
        return nil, fmt.Errorf("nenhum histórico de empréstimos encontrado para o livro")
    }

    return result, nil
}
```



Sobre o Projeto

Objetivo

Componentes



Sobre o Projeto

Objetivo

Simular o funcionamento de um sistema de empréstimo de livros:

1. Realizar empréstimos.
2. Devolver livros.
3. Consultar disponibilidade e histórico.



Sobre o Projeto

Componentes

1. Interface cliente (linha de comando).
2. Servidor remoto que executa lógica de empréstimo.
3. Banco de dados SQLite que armazena livros e registros.



Características do Sistema

Por que é um Sistema Distribuído?

1. Cliente e servidor independentes: executam em processos separados.
2. Comunicação por rede: mesmo que local, simula ambiente distribuído.
3. RPC: permite abstrair a comunicação como se fossem chamadas locais.
4. Persistência de dados: banco de dados integrado ao servidor, acessado via chamadas remotas.

Conclusão



Projeto cumpre os princípios básicos de sistemas distribuídos.

Aplicação funcional e extensível.

Abordagem modular facilita manutenção e evolução futura.

Obrigado

