INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

INDOOR INF

Arthur Alves, Arthur Faria e Rafael Estanislau

2024

INSTITUTO DE INFORMÁTICA
UFG

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Agenda

- Introdução e Funcionalidades
- Arquitetura de Software
- Arquitetura de Sistema
- Aspectos de Computação Ubíqua
- Indoor INF e Gêmeos Digitais
- Implementação

Introdução e Funcionalidades

Introdução

- Sistema de geolocalização interno do Instituto de Informática da UFG.
- Através do mapeamento do prédio do Instituto de Informática, o algoritmo do Indoor INF calcula a rota mais curta até o destino.
- O projeto Indoor INF visa proporcionar assistência aos estudantes novatos para se orientarem dentro do edifício do Instituto de Informática da UFG.

Funcionalidades

- O sistema Indoor INF utiliza tecnologia de NFC para fornecer assistência de localização.
- O mapeamento detalhado dos prédios, combinado com o avançado algoritmo de *Dijkstra*, permite calcular rotas eficientes com base no contexto do usuário.
- A visão geral abrange a integração harmoniosa do sistema com as necessidades de orientação dos estudantes.
- Proporciona uma solução eficaz para facilitar a navegação dentro dos edifícios acadêmicos.

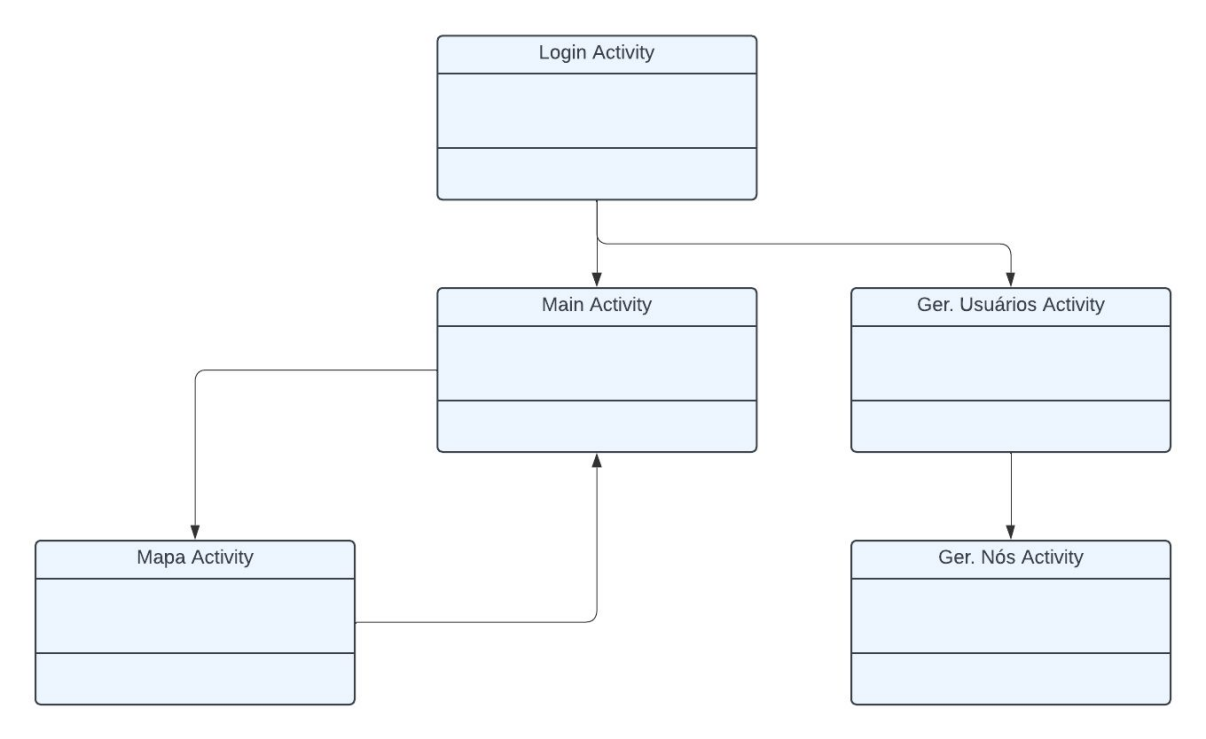
Arquitetura de Software

Arquitetura de Software

- Baseado no escopo, nos requisitos, e nos casos de uso mapeados, foram definidos os seguintes atributos de qualidade prioritários para o software:

    - **Segurança**: serão manipuladas credenciais de login dos usuários;
    - **Manutenibilidade**: os ambientes internos mapeados, as designações deles, e os mapas e identificadores correspondentes poderão sofrer alterações;
    - **Usabilidade**: se pedir direções para alguém parecer mais fácil do que utilizar a aplicação, ela não é usável o suficiente;
    - **Portabilidade**: o usuário precisa de uma forma de se localizar em movimento, e de continuar seu uso em outro dispositivo.

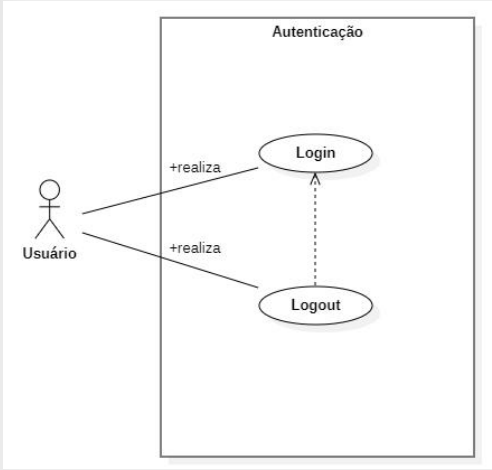
Diagramas de Classes

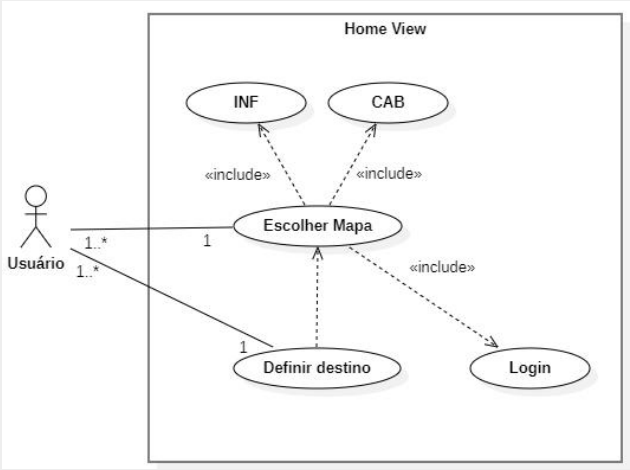


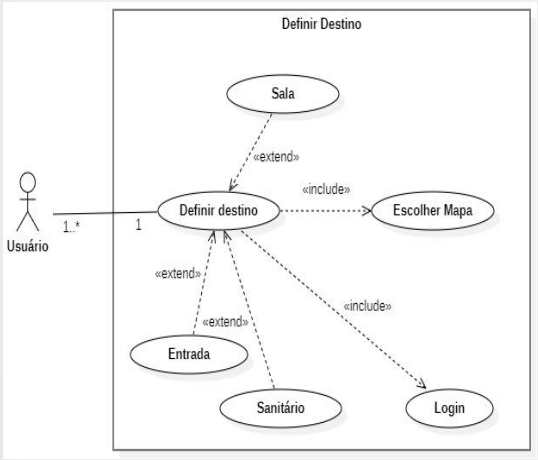
Arquitetura de Software

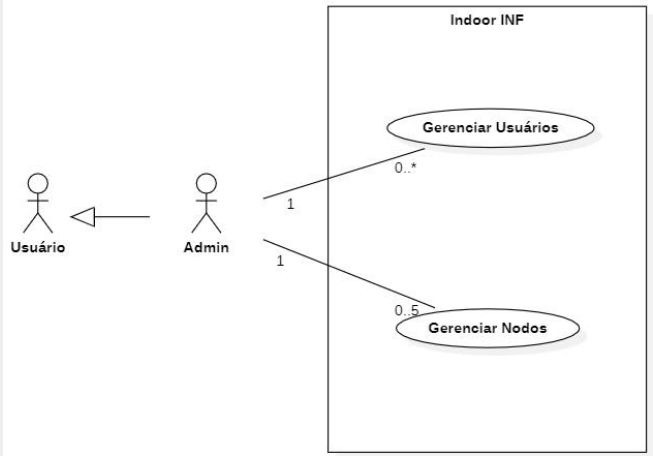
- **Requisitos Funcionais**: 15 requisitos.
- **Requisitos Não Funcionais:** 10 requisitos.
- **Casos de Uso**: 6 casos de uso.
- **Cenários de Uso**: 9 cenários de uso.

Casos de Uso


Autenticação
Usuário
+realiza
Login
+realiza
Logout


Home View
INF
CAB
«include»
«include»
Usuário
1..*
1..*
1
Escolher Mapa
«include»
1
Definir destino
Login


Definir Destino
Sala
«extend»
Usuário
1..*
1
Definir destino
«include»
Escolher Mapa
«extend»
«extend»
«include»
Entrada
Sanitário
Login


Indoor INF
Gerenciar Usuários
0..*
Usuário
Admin
1
1
0..5
Gerenciar Nodos

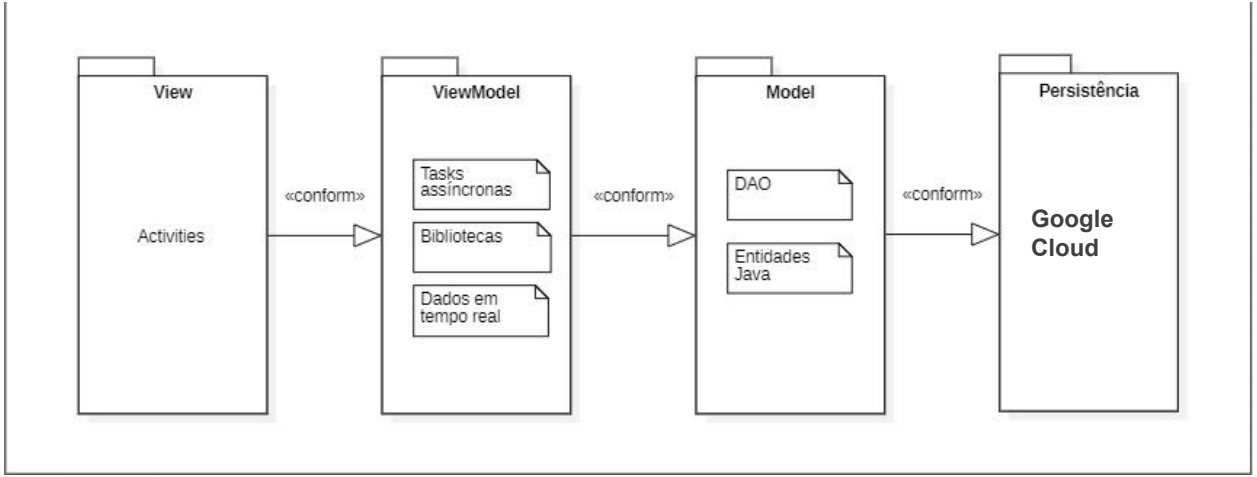
Arquitetura de Sistema

Arquitetura de Sistema

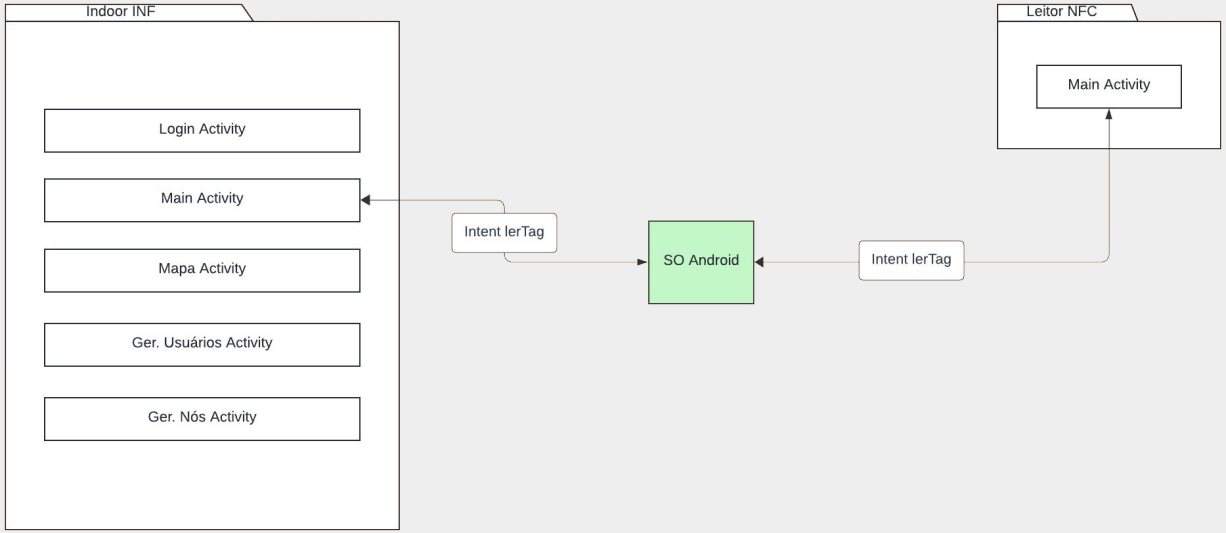
- Com relação à arquitetura de sistema, o Indoor INF está estruturado da seguinte forma:
    - **Camada de Apresentação**: mapa, telas de informações, interface de navegação, elementos visuais;
    - **Camada de Lógica de Apresentação**: tradutores de dados, controladores de eventos, atualizadores da IU, validadores de entrada;
    - **Camada de Modelo**: estruturas de dados, algoritmo de Dijkstra, gestor de contexto situacional;
    - **Segurança**: módulo de autenticação Firebase, do Google Sign-in, e criptografia de dados;
    - **Integrações Externas**: Near Field Communication (NFC) - cada tag representa a uma sala do INF, o usuário, ao abrir o aplicativo lê a tag presente em sua sala atual.
    - **Tecnologias**: front-end e back-end em Kotlin; persistência em nuvem.

Arquitetura de Sistema

- Definida através do diagrama arquitetural seguinte. Segue o padrão MVVM (Model-View-View/Model):



Comunicação NFC

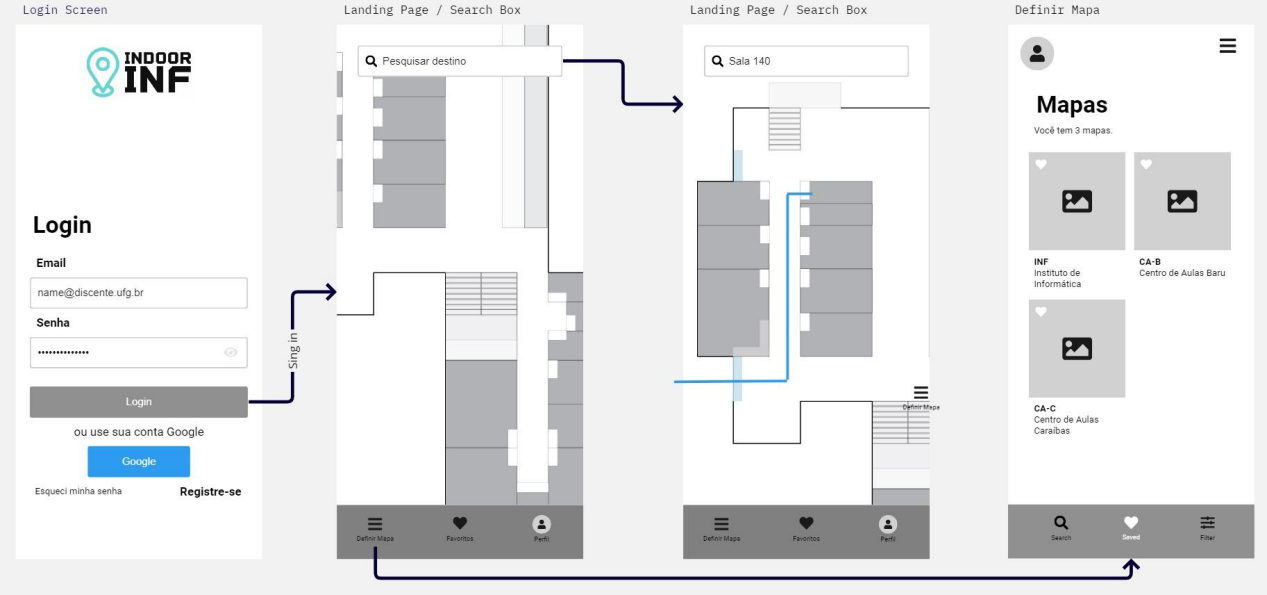


Aspectos de Computação Ubíqua

Aspectos de Computação Ubíqua

- **Dispositivos móveis**: interação direta com o usuário para fornecer informações em tempo real. Utilização de sensores do dispositivo, como o NFC.
- **Gêmeos digitais**: integrado na aplicação para criar uma representação virtual do edifício, salas e informações relacionadas. Isso inclui modelos 2D do edifício, permitindo a consciência situacional.
- **Offloading na nuvem**: transferência do estado de uso da aplicação via Firebase para a nuvem, visando garantir a continuidade de uso do Indoor INF de um dispositivo para outro.
- **Continuidade**: restauração do estado salvo pelo offloading ao se logar em outro dispositivo.

Protótipos de Telas

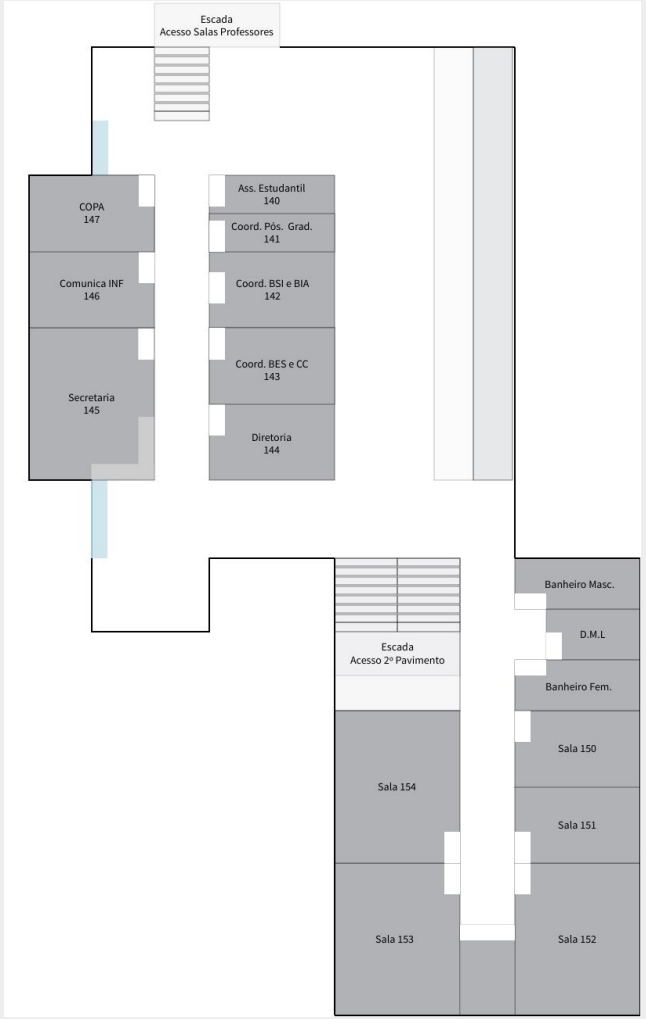


Indoor INF e Gêmeos Digitais

Indoor INF e Gêmeos Digitais

- O conceito de gêmeo digital pode foi integrado na aplicação para criar uma representação virtual do edifício, salas e informações relacionadas. Isso inclui modelo 2D0 do edifício, permitindo uma visualização mais imersiva. Além disso, o gêmeo digital pode ser atualizado em tempo real com informações dinâmicas, como eventos especiais, mudanças na disposição das salas ou obras em andamento nos edifícios. Isso proporciona aos usuários uma representação digital precisa do ambiente físico, melhorando a orientação e a experiência de navegação. A integração será realizada usando tecnologias de modelagem 2D e APIs para atualizações em tempo real.

Gêmeo Digital do INF



Implementação

Implementação: Login

```
private fun firebaseAuth(idToken: String?) {
    val credential = GoogleAuthProvider.getCredential(idToken, null)
    auth.signInWithCredential(credential)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val user = auth.currentUser
                val map = HashMap<String, Any>()
                map["id"] = user?.uid ?: ""
                map["name"] = user?.displayName ?: ""
                map["profile"] = user?.photoUrl?.toString() ?: ""
                database.getReference().child("users").child(user?.uid ?: "").setValue(map)
                val intent = Intent(this, MainActivity::class.java)
                startActivity(intent)
                finish()
            } else {
                val exception = task.exception
                if (exception != null) {
                    Toast.makeText(this, exception.message, Toast.LENGTH_SHORT).show()
                }
            }
        }
}
```

Implementação: Mapeamento de Rota

```
fun calculateShortestPath(startNodeId: Int, endNodeId: Int): List<Node> {
    val startNode = nodes.find { it.id == startNodeId }
    val endNode = nodes.find { it.id == endNodeId }

    if (startNode != null && endNode != null) {
        // Redefinição da propriedade shortestPath
        shortestPath = emptyList()

        // Verificação da existência de arestas
        fun hasEdge(node1: Node, node2: Node): Boolean {
            val neighbors = node1.neighbors
            for (neighbor in neighbors) {
                if (neighbor.id == node2.id) {
                    return true
                }
            }
            return false
        }
```

Implementação: Algoritmo de Dijkstra

```
// Algoritmo de Dijkstra
val distances = mutableMapOf<Node, Float>()
val previousNodes = mutableMapOf<Node, Node?>()
val unvisitedNodes = nodes.toMutableList()

distances[startNode] = 0f

while (unvisitedNodes.isNotEmpty()) {
    val currentNode = unvisitedNodes.minByOrNull { distances.getOrDefault(it, Float.POSITIVE_INFINITY) }!!

    unvisitedNodes.remove(currentNode)

    for (neighbor in unvisitedNodes) {
        if (hasEdge(currentNode, neighbor)) {
            val tentativeDistance = distances.getOrDefault(currentNode, Float.POSITIVE_INFINITY) +
                    calculateDistance(currentNode, neighbor)

            if (tentativeDistance < distances.getOrDefault(neighbor, Float.POSITIVE_INFINITY)) {
                distances[neighbor] = tentativeDistance
                previousNodes[neighbor] = currentNode
            }
        }
    }
}
```

Implementação: NFC

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    // Inicializa o NfcAdapter
    nfcAdapter = NfcAdapter.getDefaultAdapter( context: this)

    // Verifica se o dispositivo suporta NFC
    if (nfcAdapter == null) {
        Toast.makeText( context: this,  text: "Este dispositivo não suporta NFC", Toast.LENGTH_SHORT).show()
        finish()
        return
    }

    // Verifica se o NFC está ativado
    if (!nfcAdapter!!.isEnabled) {
        Toast.makeText( context: this,  text: "Ative o NFC nas configurações do dispositivo", Toast.LENGTH_SHORT).show()
    }
    if(true){
        Toast.makeText( context: this,  text: "Entrou", Toast.LENGTH_SHORT).show()

    }
}
```

Implementação: NFC

```kotlin
override fun onPause() {
    super.onPause()
    // Desabilita a leitura NFC quando a atividade está em segundo plano
    nfcAdapter?.disableForegroundDispatch( activity: this)
}

arthur.peixoto *
override fun onNewIntent(intent: Intent) {
    super.onNewIntent(intent)
    // Manipula a tag NFC recém-detectada
    if (NfcAdapter.ACTION_NDEF_DISCOVERED == intent.action) {
        val messages = intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES)
        if (messages != null) {
            processNdefMessages(messages)
        }
    }
}
arthur.peixoto *
private fun processNdefMessages(ndefMessages: Array<Parcelable>) {
    for (ndefMessage in ndefMessages) {
        val records = (ndefMessage as NdefMessage).records
        for (record in records) {
            val payload = String(record.payload)
            Toast.makeText( context: this,  text: "Conteúdo da tag NFC: $payload", Toast.LENGTH_SHORT).show()
            // Chama a função para atualizar o nó de saida no MyGraphView
            graphView.updateStartNodeFromNFC(ndefMessage)
        }
    }
}
```

Obrigado


INSTITUTO DE
INFORMÁTICA
UFG
UFG
UNIVERSIDADE
FEDERAL DE GOIÁS