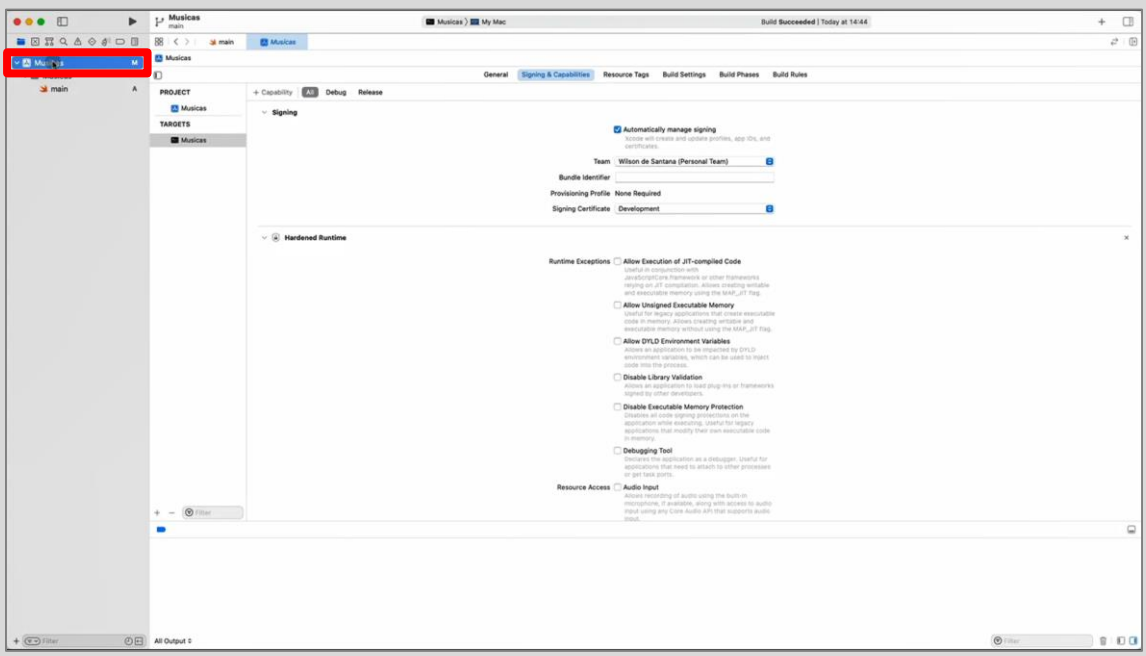
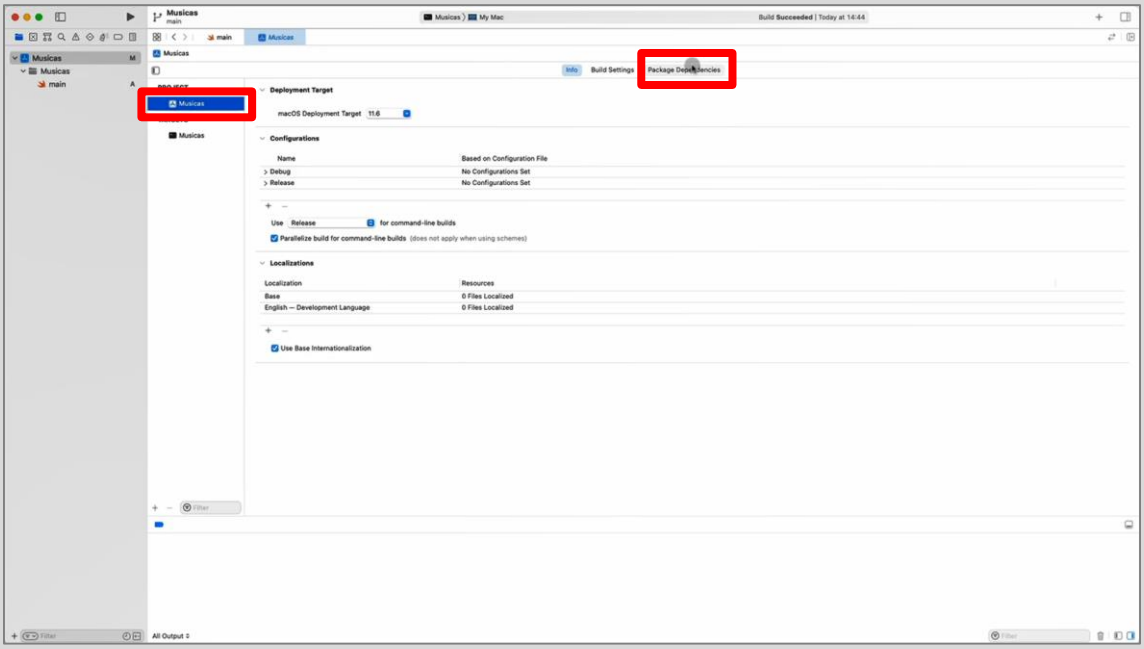
API de conexão e acesso

Vamos criar uma interface de conexão com o banco de dados, ou seja, uma API de conexão e acesso. Antes de iniciar a API propriamente dita, vamos acrescentar um pacote ao projeto.

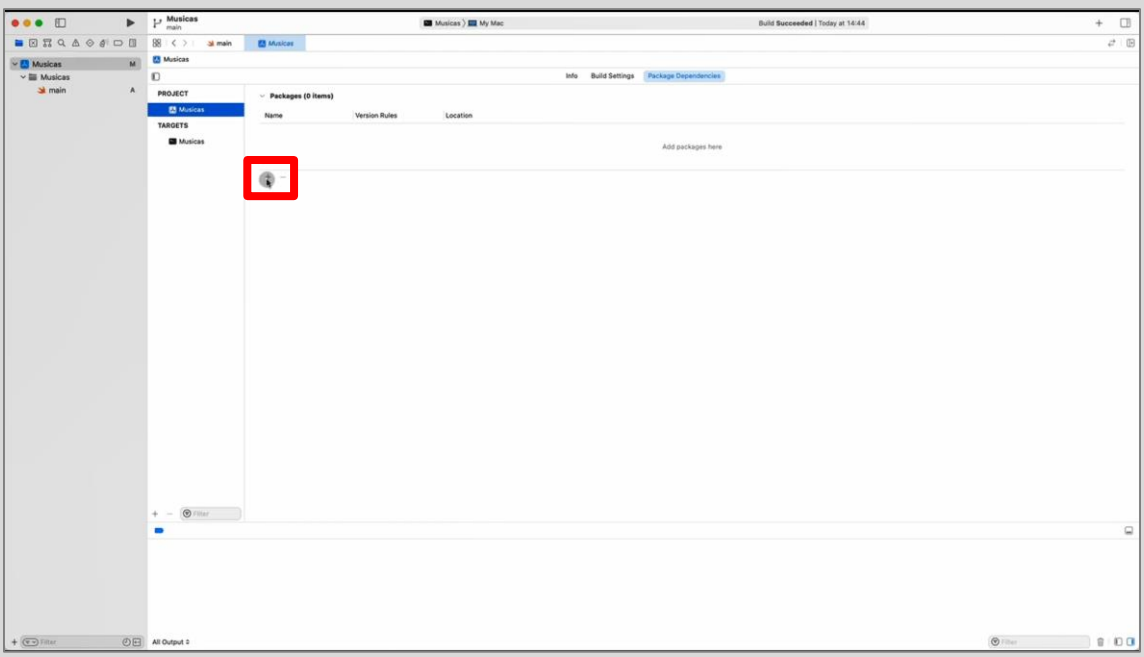
1. Clique no projeto.



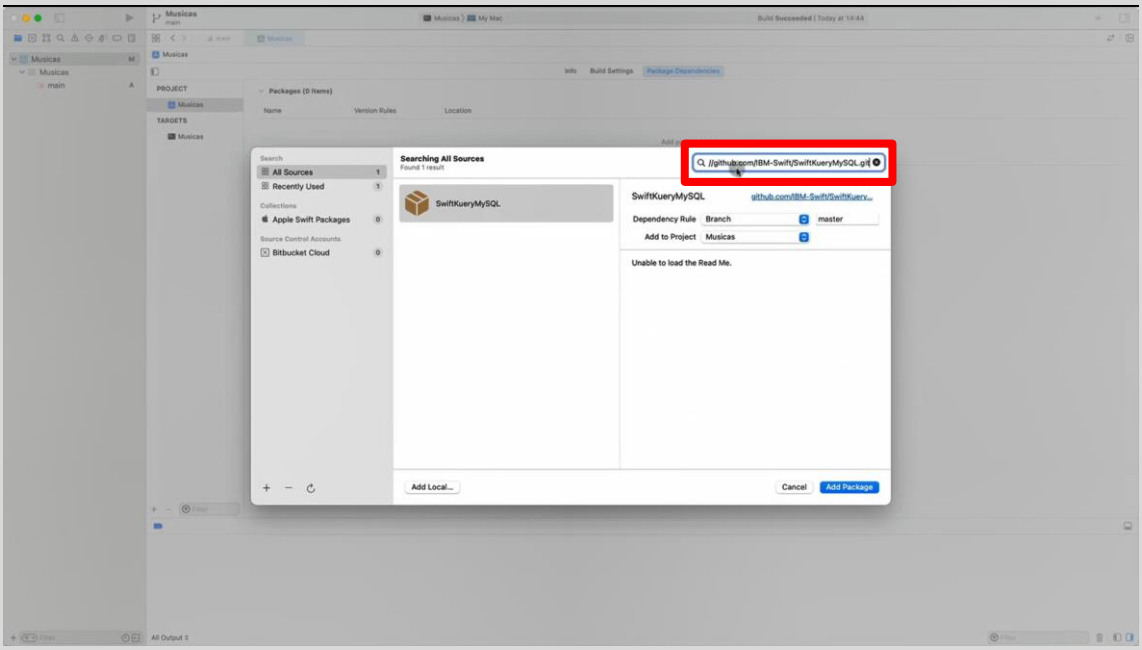
2. Selecione o **projeto** e clique em **pacotes**.



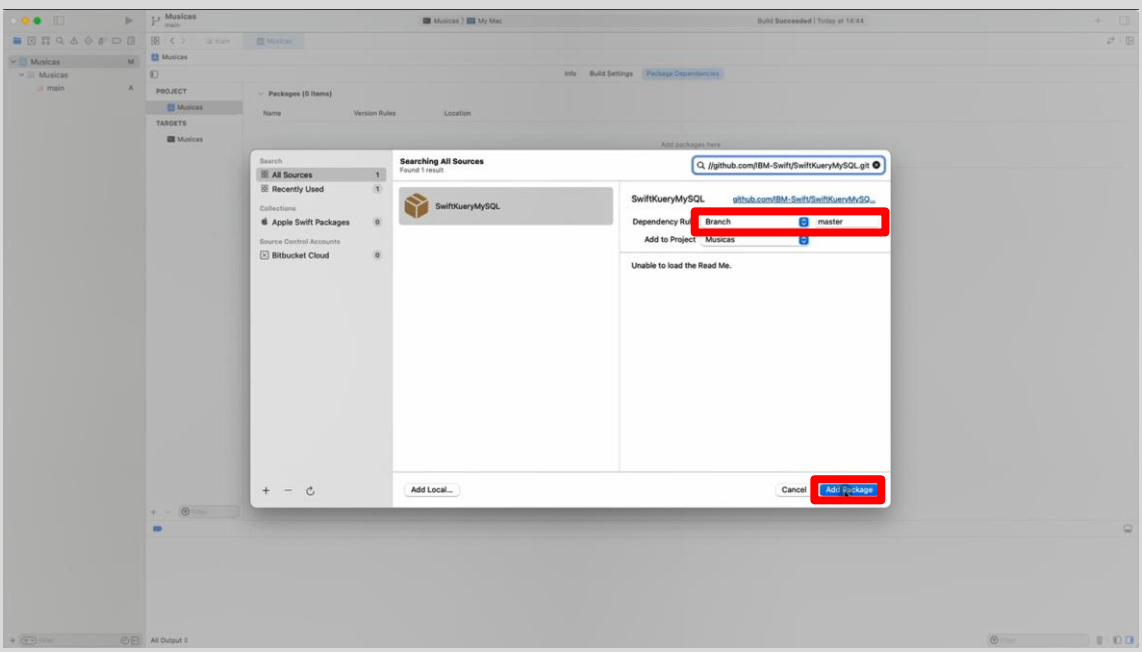
Agora, clique no sinal de **+**.



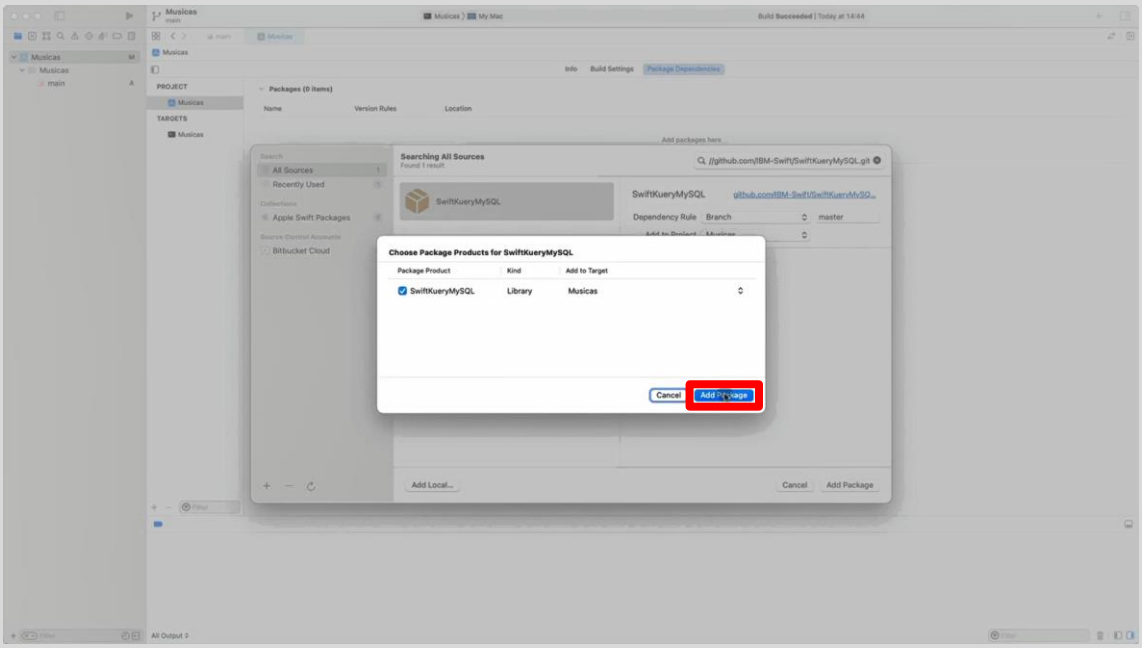
3. Acrescente o **endereço git** do projeto na barra de pesquisa.



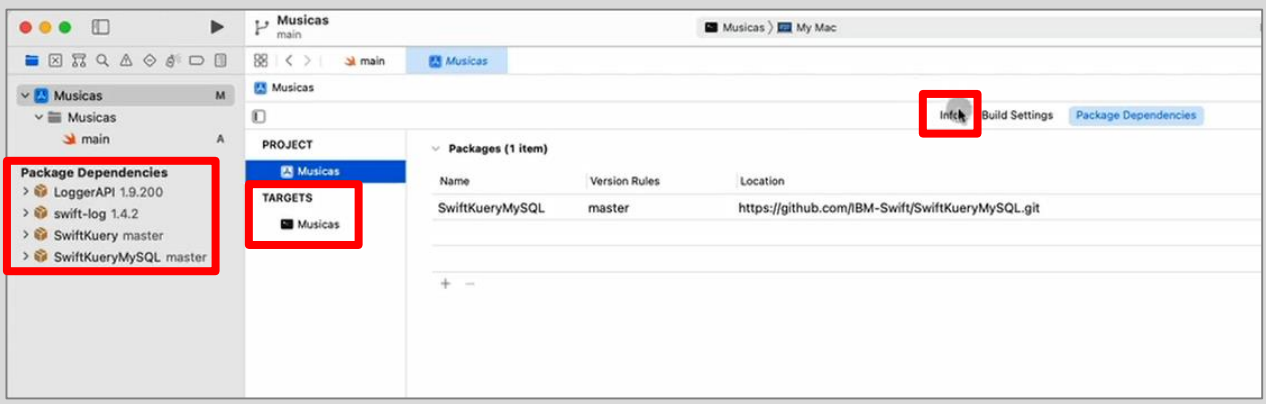
Verifique se é a **branch master** e adicione o pacote.



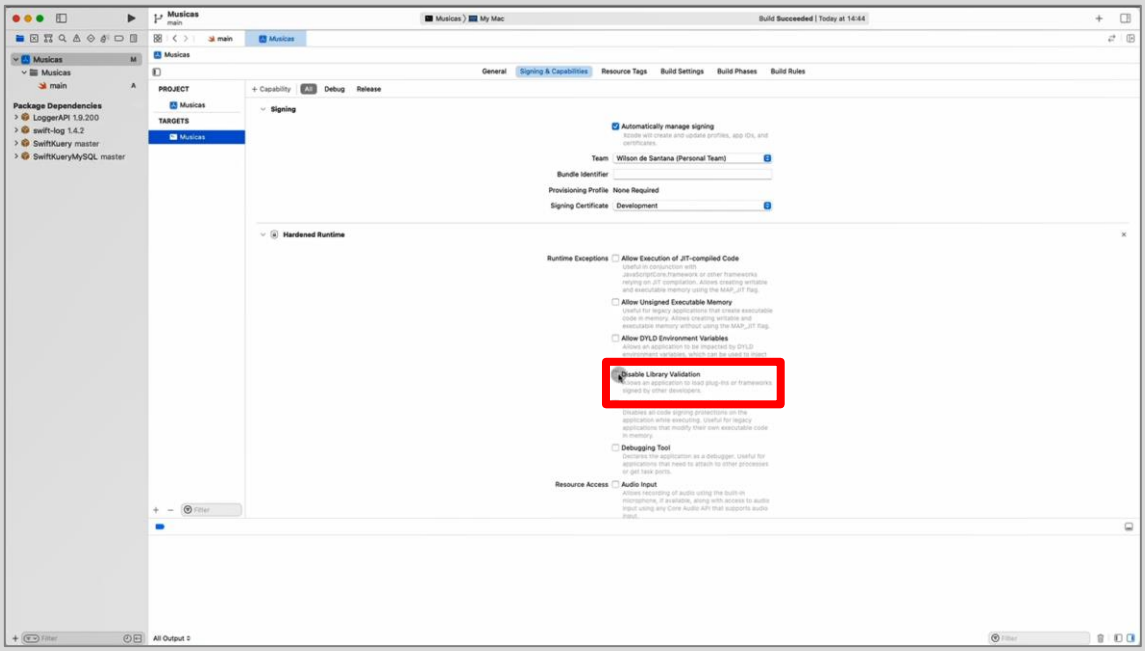
4. Na janela seguinte, confirme a adição do pacote.



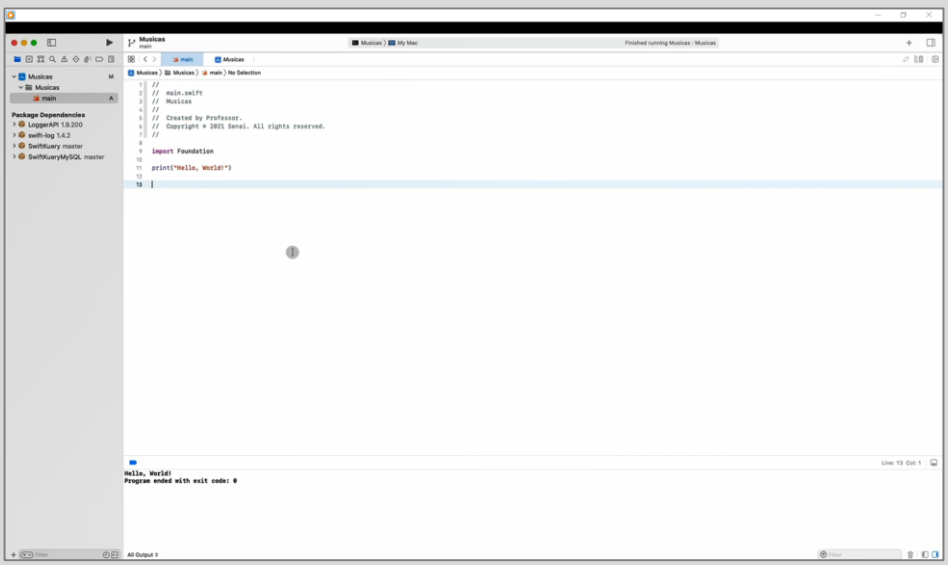
5. Note que a estrutura do projeto é montada no lado esquerdo. Clique na aba **info** e, depois, em **Targets/Musicas**.



6. Na janela seguinte, marque a opção Disable Library Validation para rodar a aplicação.

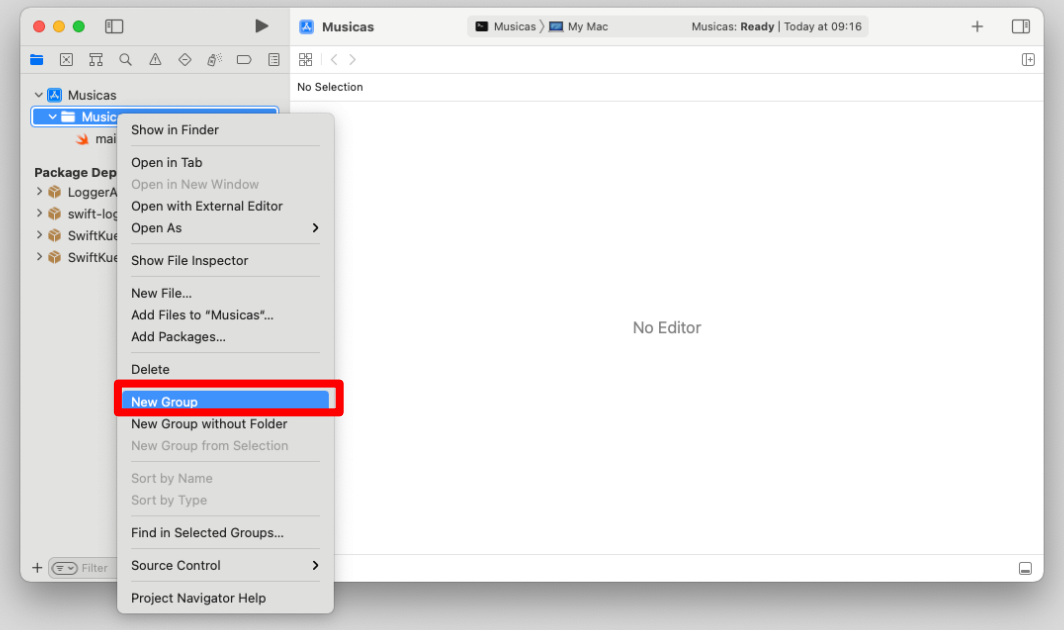


7. Agora está tudo pronto para iniciar o projeto propriamente dito.

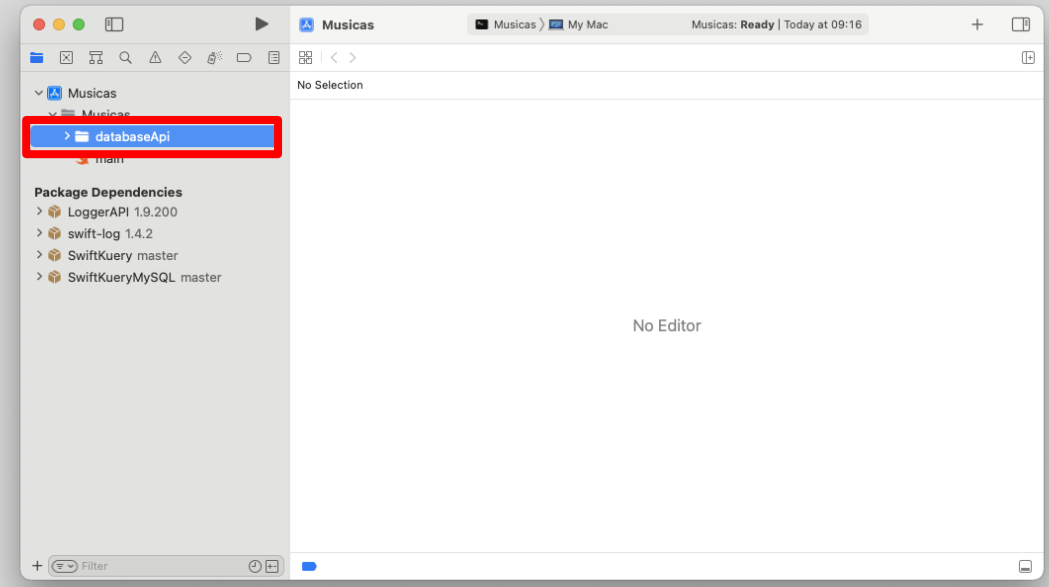


Com o objetivo de organizar o desenvolvimento do projeto, vamos criar uma pasta para manter o arquivo de parâmetros e os programas que implementam a API. Confira os passos a seguir.

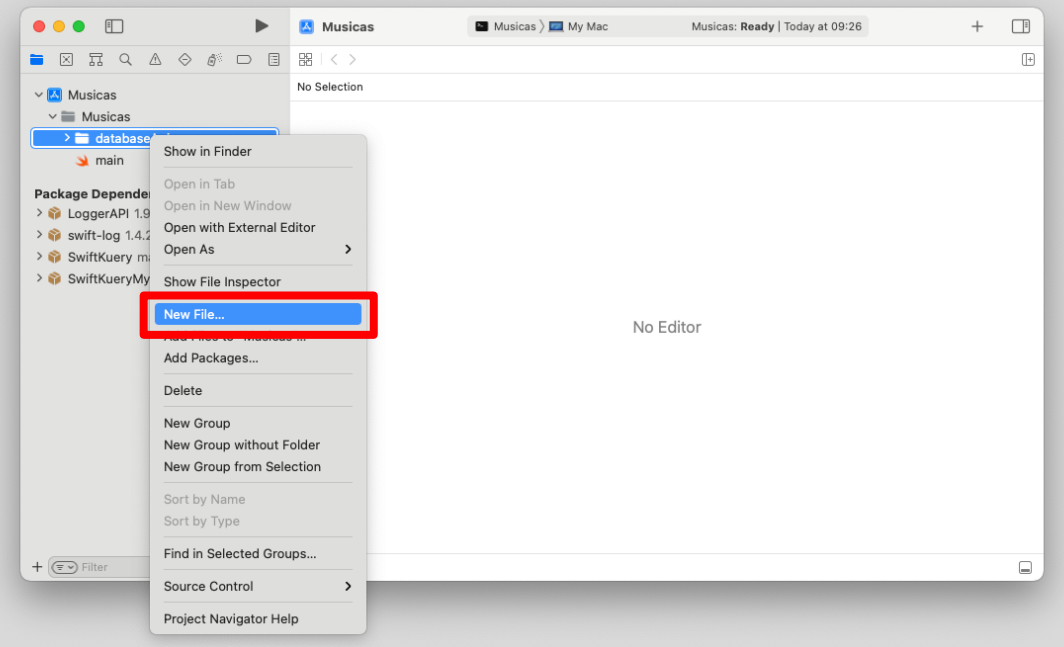
1. Clique com o botão direito do mouse sobre o **nome do projeto** e selecione a opção “**New Group**”.



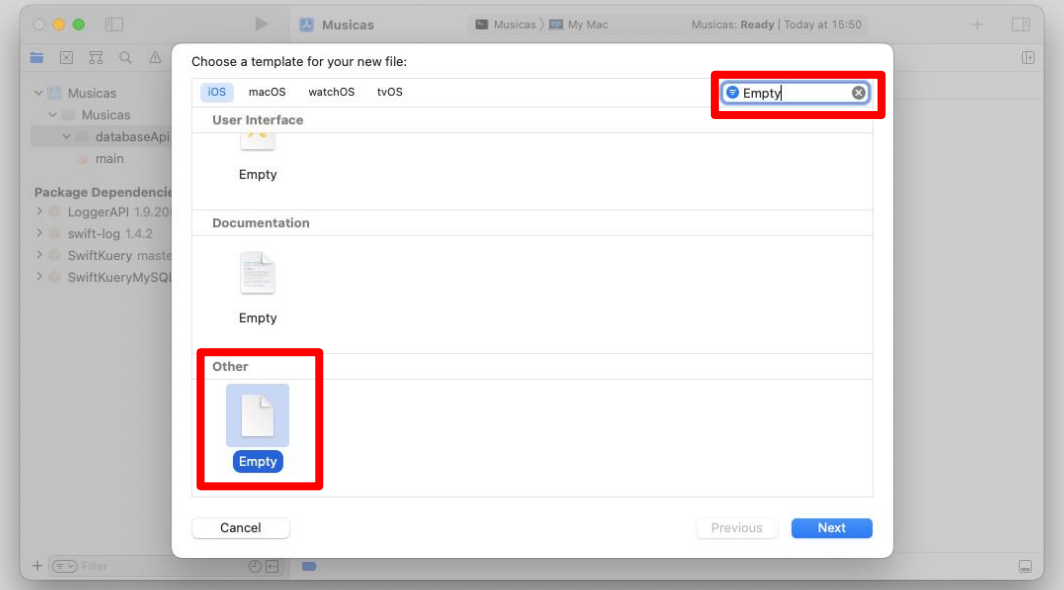
2. Defina o nome para o grupo criado. Um grupo equivale a uma pasta para o Xcode.



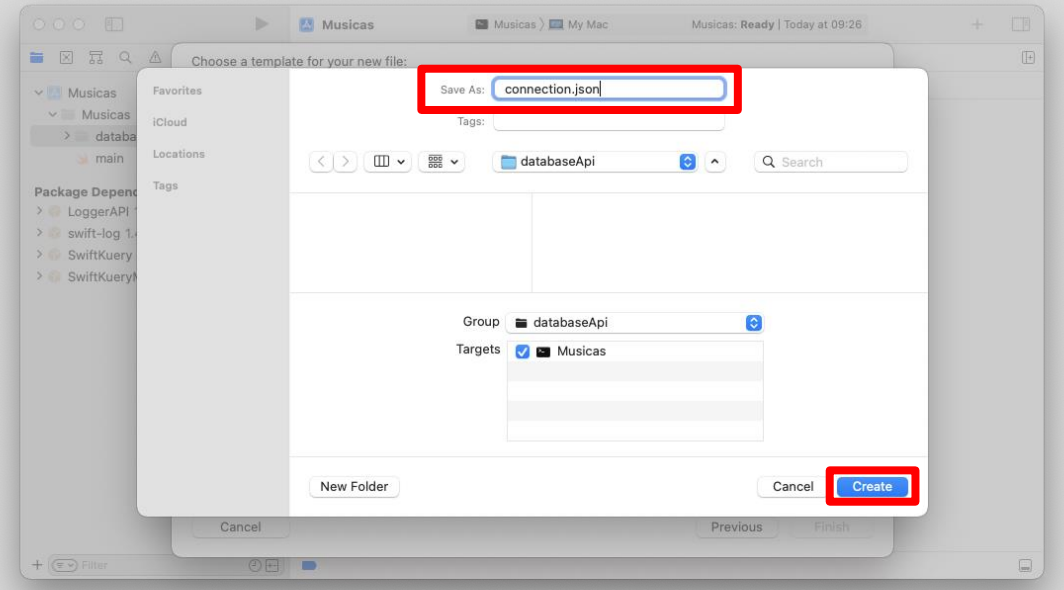
3. Clique com o botão direito do mouse sobre o nome da pasta criada e selecione a opção "**New File**...".



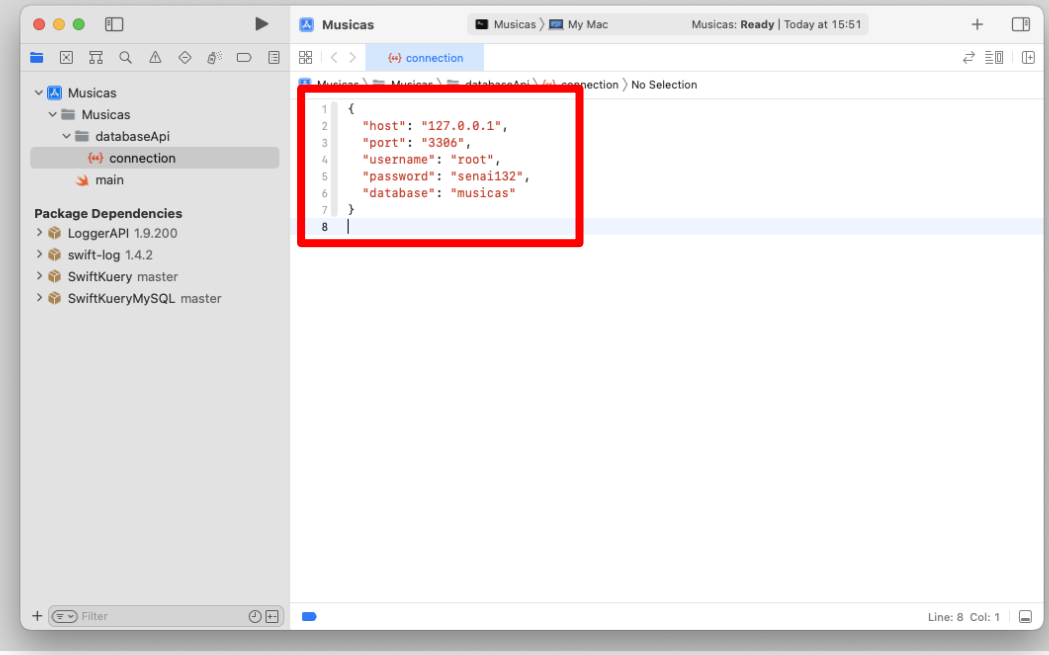
4. Digite **Empty** na caixa de pesquisa e selecione o item do grupo Other, que aparece destacado na imagem, e clique no botão “**Next**”.



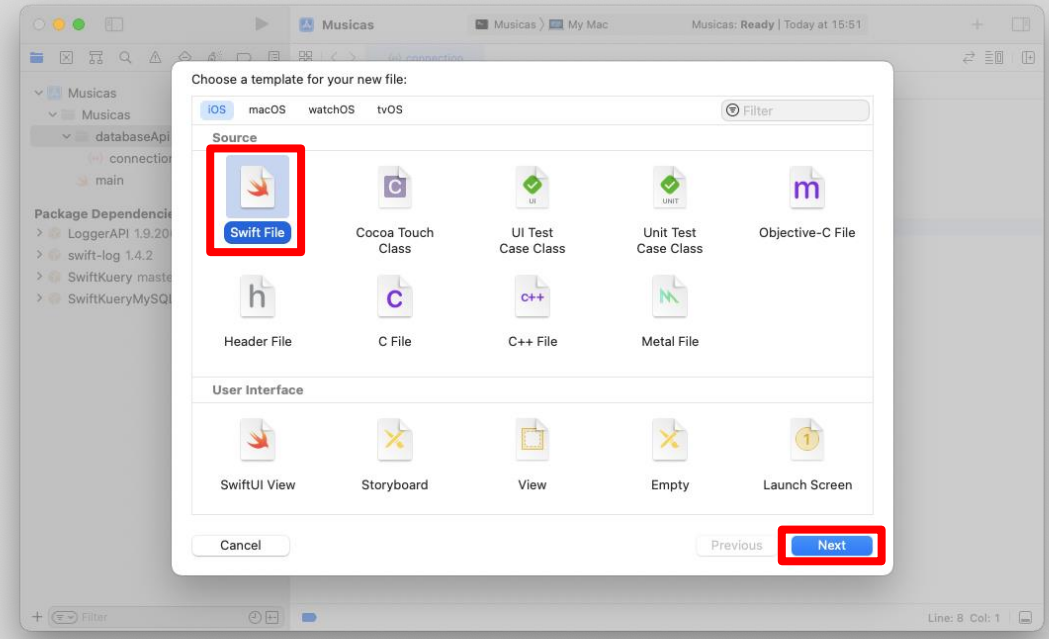
5. Preencha o nome do arquivo e clique no botão **Create**.



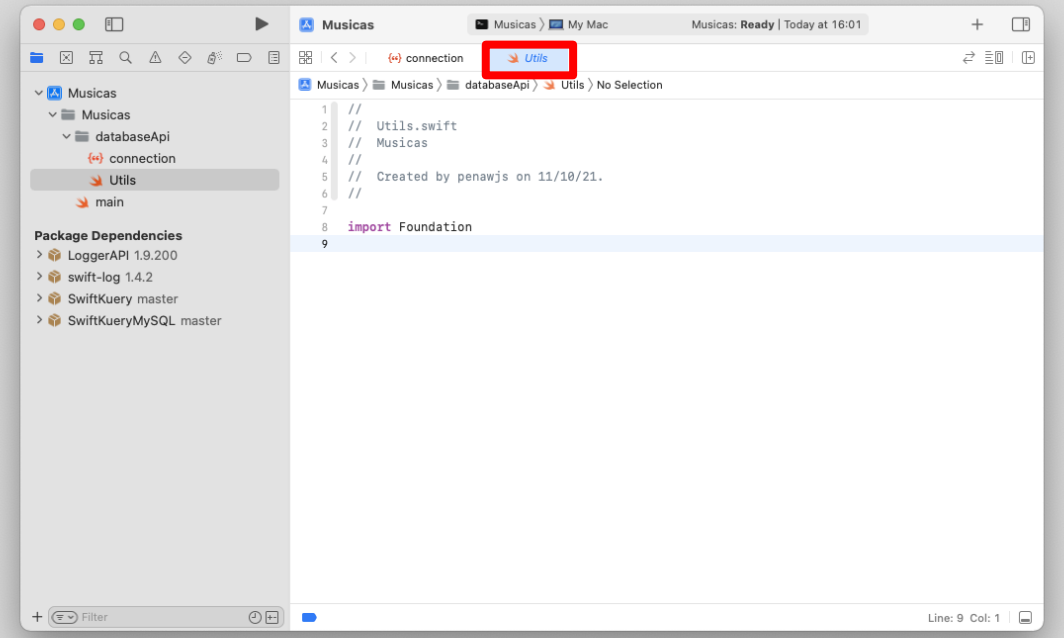
6. Preencha o arquivo criado com os parâmetros de conexão.



7. Clique com o botão direito do mouse na pasta definida para conter os programas da API de conexão ao MySQL, clique em "**New File**", selecione o item "**Swift File**" e clique no botão "**Next**".



8. Nomeie este arquivo como **Utils.swift**.



Agora que já temos a base de nossas configurações criada e sabemos como criar um novo arquivo, seguiremos com a codificação.

Para que possamos utilizar a plataforma de conexão SwiftKueryMySQL no início do nosso programa, devemos importar o seguinte conjunto de referências:

```
import SwiftKuery
import SwiftKueryMySQL
```

Todo o conjunto de métodos que criaremos ficará na classe CommonUtils. Esta classe conterá as referências do pool de conexões, da conexão em uso e uma referência estática da própria instância da classe CommonUtils:

```
class CommonUtils {
 private var pool: ConnectionPool?
 private var connection: Connection?
 static let sharedInstance = CommonUtils()
 private init() {}

 // abaixo serão construídos os métodos para a API de
aa//acesso MySQL
}
```

O atributo pool guardará a referência do pool de conexões ao banco de dados para garantir eficiência no estabelecimento da conexão; o atributo connection armazenará a referência em uma conexão e permitirá o reaproveitamento desta para várias transações.

O atributo estático sharedInstance guarda a referência da única instância criada para a aplicação; o método privado **init** garante que nenhuma outra rotina crie uma instância da aplicação. Essa construção é um padrão de desenvolvimento chamado **Singleton** e é utilizado quando necessitamos que não haja mais que uma instância de uma classe ativa ao mesmo tempo durante a execução de uma aplicação, com o objetivo de garantir a reutilização de determinadas funcionalidades por toda a aplicação.

O pool de conexões será inicializado com o método getConnectionPool, que executará muitos passos. Portanto, vamos analisá-lo por partes, iniciando com a leitura do arquivo de parâmetros de conexão ao banco de dados MySQL. Antes, contudo, vejamos sua estrutura inicial:

```
private func getConnectionPool(
    characterSet: String? = nil) -> ConnectionPool {

 if let pool = pool {
    return pool
 }

 return pool!
}
```

Esse método aceita opcionalmente a definição do tipo de conjunto de caracteres utilizado nos dados armazenados no banco de dados, que podem ser ISSO-88591, UTF-8 etc.

A primeira instrução testa pela existência prévia do pool que, em caso de já existir, tem sua referência retornada. Essa é uma estratégia para otimizar o tempo de criação de novo pool de conexão a cada requisição a esse método. A última instrução é utilizada para retornar a referência do pool após a sua construção (que elaboraremos no próximo passo):

```
  private func getConnectionPool(
     characterSet: String? = nil) -> ConnectionPool {

   if let pool = pool {
      return pool
   }
   do {
      let connectionFile = #file.replacingOccurrences(
              of: "Utils.swift", with:
"connection.json")
      let data = Data(referencing: try NSData(
              contentsOfFile: connectionFile))
      let json = try JSONSerialization.jsonObject(with:
data)

   } catch {
      print("Erro lançado")
      pool = nil
      print(error.localizedDescription)
   }
   return pool!
  }
```

O código em destaque foi acrescentado ao método e declara connectionFile para conter o local no sistema de arquivos onde está o arquivo connection.json para, na sequência, ter seus dados lidos para data e convertido em formato JSON. Como esse processo para a localização e leitura dos dados permite o lançamento de exceções, todo o bloco de código foi envolvido na estrutura do **catch** para tratar possíveis falhas.

Agora, necessitamos extrair os parâmetros de conexão:

```
  do {
      let connectionFile = #file.replacingOccurrences(
              of: "Utils.swift", with: "connection.json")
      let data = Data(referencing: try NSData(
              contentsOfFile: connectionFile))
      let json = try JSONSerialization.jsonObject(with:
data)

      if let dictionary = json as? [String: String] {
         let host = dictionary["host"]
         let username = dictionary["username"]
         let password = dictionary["password"]
         let database = dictionary["database"]
         var port: Int? = nil
         if let portString = dictionary["port"] {
               port = Int(portString)
         }

         let randomBinary = arc4random_uniform(2)

         let poolOptions = ConnectionPoolOptions(
                  initialCapacity: 1, maxCapacity: 1)

      } else {
         pool = nil
         print("""
                Formato invalido no
                conteúdo do arquivo connection.json:
\(json)
                """)
      }
   } catch {
      print("Erro lançado")
      pool = nil
      print(error.localizedDescription)
   }
```

O teste condicional em destaque foi acrescentado para verificar se o arquivo de parâmetros tem algum conteúdo e, em caso positivo, extrair cada um dos parâmetros, além de preparar as informações para a criação do pool de conexões. Caso não exista conteúdo no arquivo de parâmetros, uma mensagem de erro é emitida. Agora com os parâmetros, vamos criar o pool de conexões:

```
    if let dictionary = json as? [String: String] {
       let host = dictionary["host"]
       let username = dictionary["username"]
       let password = dictionary["password"]
       let database = dictionary["database"]
       var port: Int? = nil
       if let portString = dictionary["port"] {
            port = Int(portString)
       }

       let randomBinary = arc4random_uniform(2)

       let poolOptions = ConnectionPoolOptions(
              initialCapacity: 1, maxCapacity: 1)

       if characterSet != nil || randomBinary == 0 {
            pool = MySQLConnection.createPool(host:
host,
                              user: username, password:
password,
                              database: database, port:
port,
                              characterSet: characterSet,
                              connectionTimeout: 10000,
                              poolOptions: poolOptions)
```

```
        } else {
            var urlString = "mysql://"
            if let username = username, let password =
password {
               urlString += "\(username):\(password)@"
            }
            urlString += host ?? "localhost"
            if let port = port {
               urlString += ":\(port)"
            }
            if let database = database {
               urlString += "/\(database)"
            }

            if let url = URL(string: urlString) {
               pool = MySQLConnection.createPool(
                        url: url, poolOptions:
poolOptions)
            } else {
               pool = nil
               print("URL com formato inválido:
\(urlString)")
            }
        }
     } else {
        pool = nil
        print("""
              Formato invalido no
              conteúdo do arquivo connection.json:
\(json)
              """)
     }
```

Nesse bloco de código em destaque, os parâmetros são validados para serem utilizados na criação do pool de conexões. Além disso, são verificadas suas possibilidades a depender do número de parâmetros existentes e, caso isso não seja possível, uma mensagem de erro será emitida; do contrário, o pool será criado para ser retornado ao final do método.

A seguir, temos a visão completa do método getConnectionPool:

```
  private func getConnectionPool(
      characterSet: String? = nil) -> ConnectionPool {
   if let pool = pool {
      return pool
   }
   do {
      let connectionFile = #file.replacingOccurrences(
              of: "Utils.swift", with:
"connection.json")
      let data = Data(referencing: try NSData(
              contentsOfFile: connectionFile))
      let json = try JSONSerialization.jsonObject(with:
data)

      if let dictionary = json as? [String: String] {
         let host = dictionary["host"]
         let username = dictionary["username"]
         let password = dictionary["password"]
         let database = dictionary["database"]
         var port: Int? = nil
         if let portString = dictionary["port"] {
              port = Int(portString)
         }
```

```
        let randomBinary = arc4random_uniform(2)

        let poolOptions = ConnectionPoolOptions(
                initialCapacity: 1, maxCapacity: 1)

        if characterSet != nil || randomBinary == 0 {
             pool = MySQLConnection.createPool(host:
host,
                                user: username, password:
password,
                                database: database, port:
port,
                                characterSet: characterSet,
                                connectionTimeout: 10000,
                                poolOptions: poolOptions)
        } else {
             var urlString = "mysql://"
             if let username = username, let password =
password {
                urlString += "\(username):\(password)@"
             }
             urlString += host ?? "localhost"
             if let port = port {
                urlString += ":\(port)"
             }
             if let database = database {
                urlString += "/\(database)"
             }

             if let url = URL(string: urlString) {
                pool = MySQLConnection.createPool(
                                url: url, poolOptions:
poolOptions)
```

```
                } else {
                    pool = nil
                    print("URL com formato inválido:
\(urlString)")
                }
            }
        } else {
            pool = nil
            print("""
                    Formato invalido no
                    conteúdo do arquivo connection.json:
\(json)
                    """)
        }
    } catch {
        print("Erro lançado")
        pool = nil
        print(error.localizedDescription)
    }
    return pool!
  }
```

Temos que o método getConnectionPool é privado e não pode ser chamado externamente à classe CommonUtils; isso garante que somente os métodos internos terão acesso à obtenção do pool de conexões. Portanto, devemos criar um método específico para a obtenção da conexão ao banco de dados sem nos preocuparmos com o pool de conexões, que fica sob responsabilidade dessa classe. O método getConnection a seguir é publico e será utilizado para obter a conexão ao bando de dados:

```
  func getConnection() -> Connection? {
   if let connection = connection {
      return connection
   }

   self.connection = nil
   getConnectionPool().getConnection { connection,
error in
      guard let connection = connection else {
         guard let error = error else {
               return print("""
               Falha ao conectar no Banco de Dados:
               \(error?.localizedDescription ?? "Erro
desconhecido")
               """)
         }
         return print("""
               Falha ao conectar no Banco de Dados:
               \(error.localizedDescription)
               """)
      }
      self.connection = connection
      return
   }
   return connection
  }
```

Este método verifica se a conexão já foi estabelecida; em caso positivo, retorna a instância da conexão; caso contrário, chama o método getConnectionPool para obter a conexão do pool de conexões e o retornando caso não ocorra qualquer erro. Na ocorrência de erros, a mensagem de falha será apresentada.