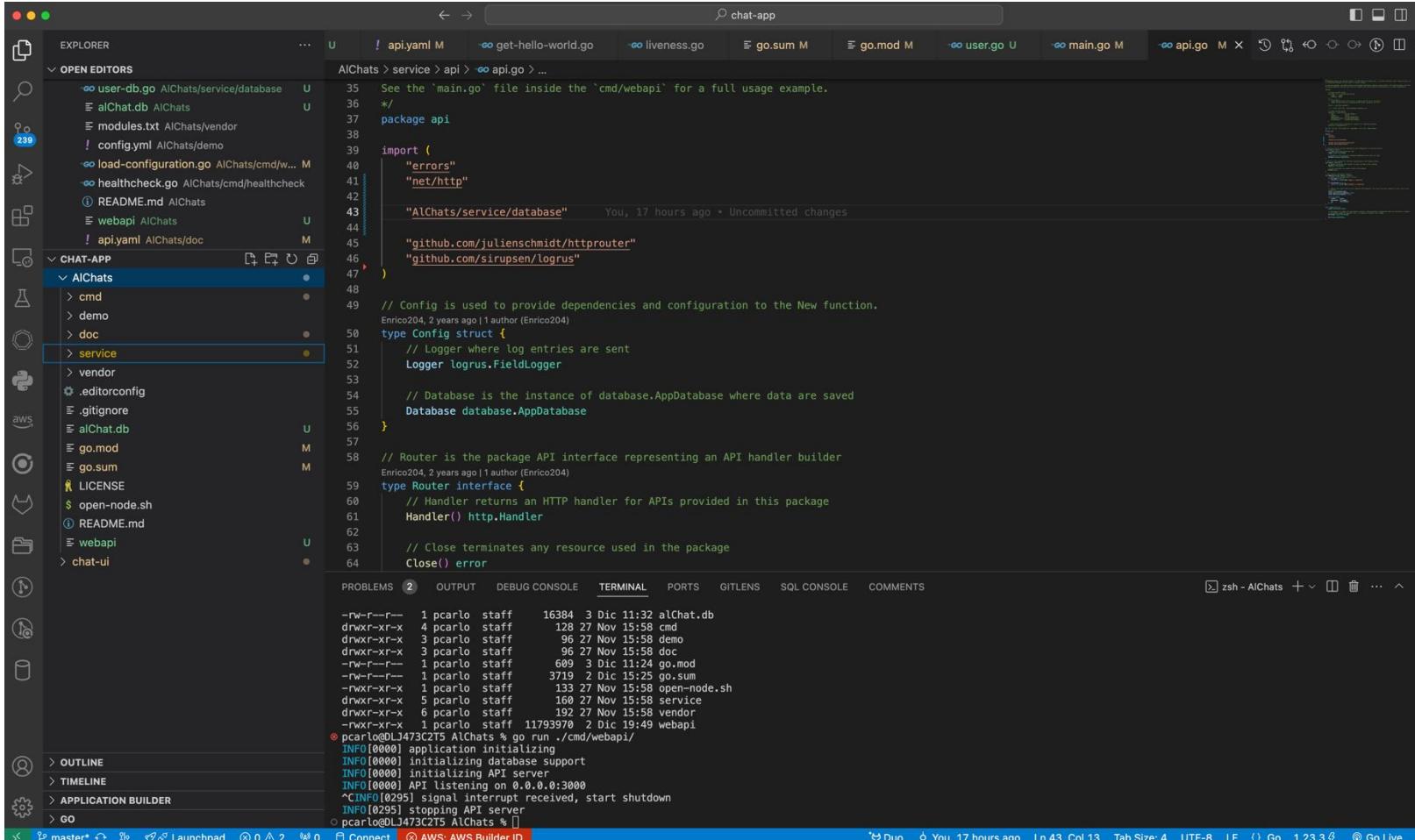


Come aggiungere un nuovo modello, le sue query Db e i suoi endpoint http

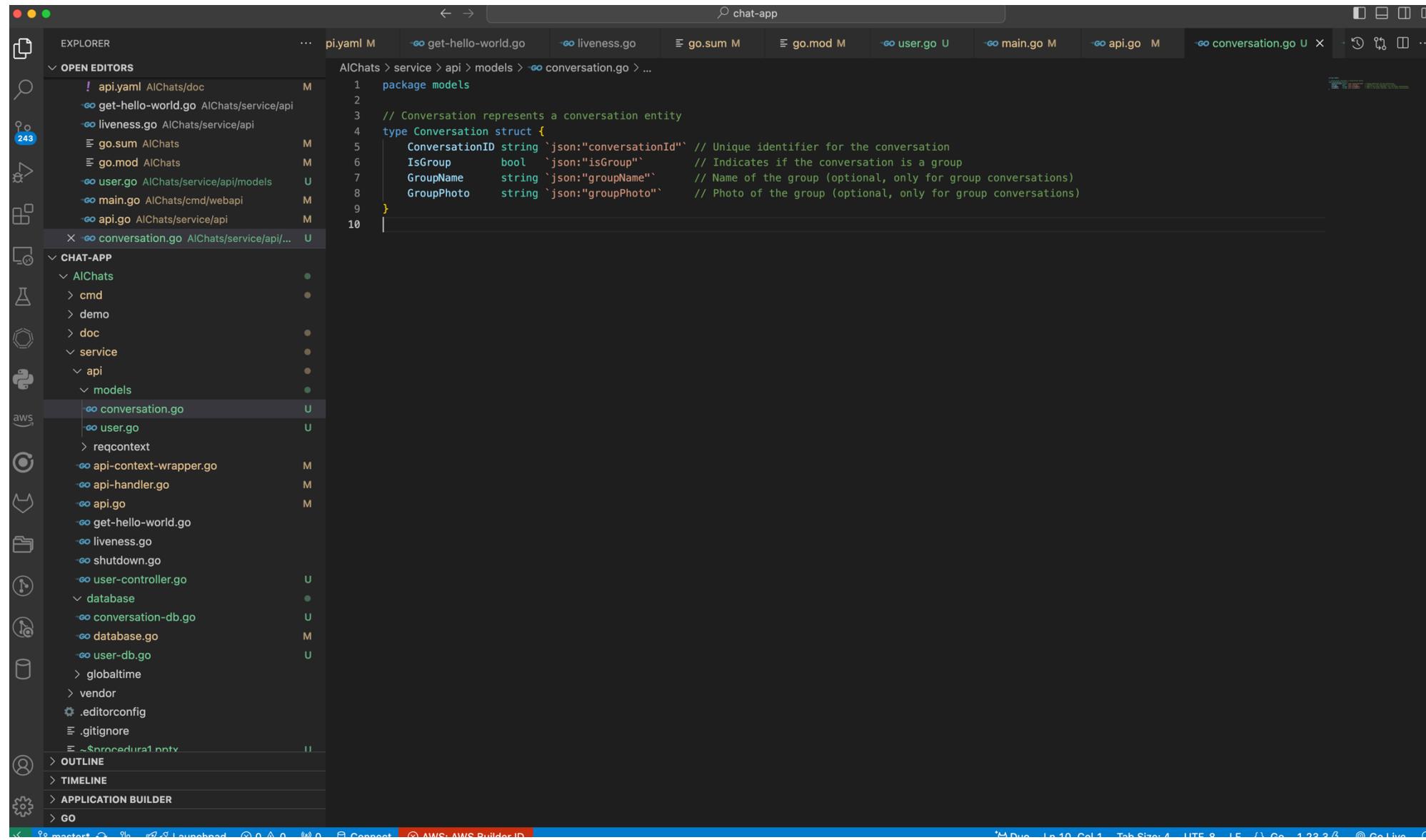


```
pcarlo@DLJ473C2T5:~/AlChats$ cd cmd/webapi
pcarlo@DLJ473C2T5:~/AlChats/cmd/webapi$ go run .
INFO[0000] application initializing
INFO[0000] initializing database support
INFO[0000] initializing APT server
INFO[0000] API listening on 0.0.0.0:3000
^CINFO[0295] signal interrupt received, start shutdown
INFO[0295] stopping APT server
pcarlo@DLJ473C2T5:~/AlChats%
```

The screenshot shows a terminal window with the following command history:

```
pcarlo@DLJ473C2T5:~/AlChats$ cd cmd/webapi
pcarlo@DLJ473C2T5:~/AlChats/cmd/webapi$ go run .
INFO[0000] application initializing
INFO[0000] initializing database support
INFO[0000] initializing APT server
INFO[0000] API listening on 0.0.0.0:3000
^CINFO[0295] signal interrupt received, start shutdown
INFO[0295] stopping APT server
pcarlo@DLJ473C2T5:~/AlChats%
```

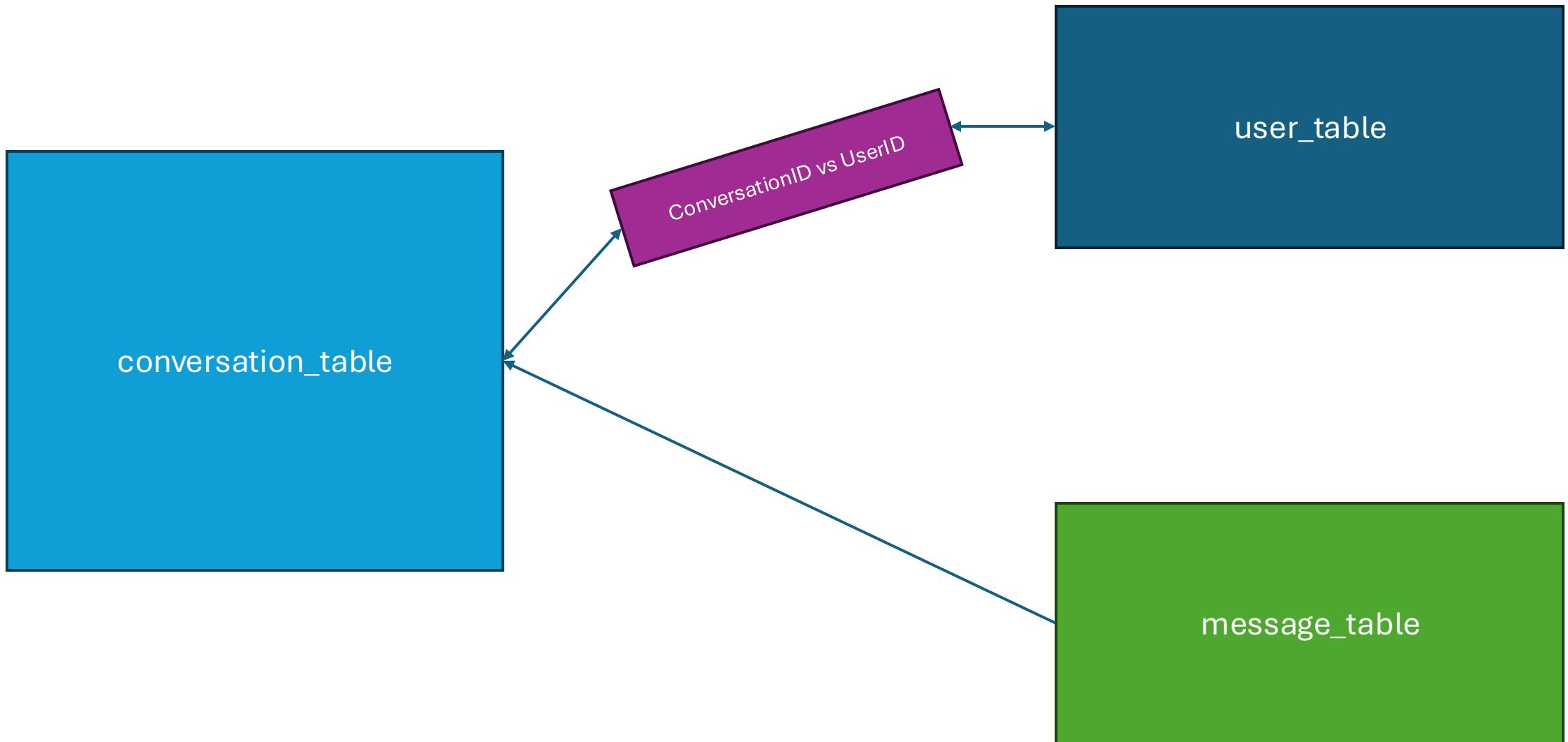
Create a Model Conversation



The screenshot shows a dark-themed code editor interface with multiple tabs open. The active tab is `conversation.go` under the `models` directory of the `AIChats/service/api` package. The code defines a `Conversation` struct with fields for `ConversationID`, `IsGroup`, `GroupName`, and `GroupPhoto`. The code editor's sidebar shows the project structure, including `cmd`, `demo`, `doc`, `service`, `api`, `models`, `aws`, `reqcontext`, `api-context-wrapper.go`, `api-handler.go`, `api.go`, `get-hello-world.go`, `liveness.go`, `shutdown.go`, `user-controller.go`, `database`, `conversation-db.go`, `database.go`, `user-db.go`, `globaltime`, `vendor`, `.editorconfig`, and `.gitignore`.

```
1 package models
2
3 // Conversation represents a conversation entity
4 type Conversation struct {
5     ConversationID string `json:"conversationId"` // Unique identifier for the conversation
6     IsGroup        bool   `json:"isGroup"`       // Indicates if the conversation is a group
7     GroupName      string `json:"groupName"`    // Name of the group (optional, only for group conversations)
8     GroupPhoto     string `json:"groupPhoto"`   // Photo of the group (optional, only for group conversations)
9 }
10
```

Design Database Tables for Conversation



DDL and Initialization for Conversation

The screenshot shows a code editor interface with multiple tabs open. The main tab displays a Go file named `database.go`. The code defines an interface `AppDatabase` and its implementation `appdbimpl`. The implementation includes logic to check if a table exists and to create it if not. It also includes a `Ping()` method. The code editor has a dark theme, and the code is color-coded. The left sidebar shows a project structure for a service named `AIChats`, which includes cmd, demo, doc, and service directories. The `service` directory contains api, models, conversation.go, user.go, recontext, api-context-wrapper.go, api-handler.go, api.go, get-hello-world.go, liveness.go, shutdown.go, user-controller.go, database, and database.go. The `database` directory contains database.go and user-db.go. The `database.go` file is currently selected in the sidebar.

```
42 type AppDatabase interface {
43     Ping() error
44 }
45
46 // New returns a new instance of AppDatabase based on the SQLite connection `db`.
47 // `db` is required - an error will be returned if `db` is `nil`.
48 func New(db *sql.DB) (AppDatabase, error) {
49     if db == nil {
50         return nil, errors.New("database is required when building a AppDatabase")
51     }
52
53     var tableName string
54     err := db.QueryRow(`SELECT name FROM sqlite_master WHERE type='table' AND name='user_table';`).Scan(&tableName)
55     if err != nil {
56         sqlStmt := `CREATE TABLE user_table (
57             UserID TEXT PRIMARY KEY DEFAULT (lower(hex(randomblob(16)))), -- Generate a UUID
58             Username TEXT NOT NULL UNIQUE, -- Not null and unique constraint
59             Photo TEXT -- Optional field (can be NULL)
60         );`
61         _, err = db.Exec(sqlStmt)
62         if err != nil {
63             return nil, fmt.Errorf("error creating database structure: %w", err)
64         }
65     }
66
67     return &appdbimpl{
68         db,
69     }, nil
70 }
71
72 func (db *appdbimpl) Ping() error {
73     return db.c.Ping()
74 }
```

in questa sezione va aggiunta la verifica per la nuova tabella

Ma conviene usilizzare delle fuzioni ausiliarie per gestire molte tabelle

vediamo come nelle prossime slides

DDL and Initialization for Conversation

```
func New(db *sql.DB) (AppDatabase, error) {
    _, err := db.Exec(sqlStmt)
    if err != nil {
        return nil, fmt.Errorf("error creating database structure: %w", err)
    }

    return &appdbimpl{
        c: db,
    }, nil
}

func (db *appdbimpl) Ping() error {
    return db.c.Ping()
}

func ensureTableExists(db *sql.DB, tableName, createStmt string) error {
    var existingTable string
    err := db.QueryRow("SELECT name FROM sqlite_master WHERE type='table' AND name=?", tableName).Scan(&existingTable)
    if errors.Is(err, sql.ErrNoRows) {
        _, err := db.Exec(createStmt)
        if err != nil {
            return fmt.Errorf("error creating table %s: %w", tableName, err)
        }
    } else if err != nil {
        return fmt.Errorf("error checking for table %s: %w", tableName, err)
    }
    return nil
}
```

aggiungiamo la funzione
ensureTableExists

che verifica solo se
una tabella è già
esistente

DDL and Initialization for Conversation

The screenshot shows a code editor with multiple tabs open. The active tab is `database.go` under the `service/database` package. The code defines functions for pinging the database and initializing it. It includes SQL schema definitions for three tables: `user_table`, `conversation_table`, and `user_conversation_table`. The `user_table` has columns for UserID (primary key), Username (unique), and Photo. The `conversation_table` has columns for ConversationID (primary key), IsGroup (boolean), GroupName, and GroupPhoto. The `user_conversation_table` is a many-to-many table with composite primary keys (UserID, ConversationID) and foreign keys linking to both user and conversation tables.

```
func (db *appdbimpl) Ping() error {
    return db.c.Ping()
}

func initializeDatabase(db *sql.DB) error {
    // User table schema
    user_table_schema := `CREATE TABLE user_table (
        UserID TEXT PRIMARY KEY DEFAULT (lower(hex(randomblob(16)))) -- Generate a UUID
        Username TEXT NOT NULL UNIQUE, -- Not null and unique constraint
        Photo TEXT, -- Optional field (can be NULL)
    );`
```

```
// Conversation table schema
conversation_table_schema := `CREATE TABLE conversation_table (
    ConversationID TEXT PRIMARY KEY DEFAULT (lower(hex(randomblob(16)))) -- Generate a UUID
    IsGroup BOOLEAN NOT NULL CHECK (IsGroup IN (0, 1)), -- Boolean value for group indicator
    GroupName TEXT, -- Optional group name
    GroupPhoto TEXT -- Optional group photo
);`
```

```
user_conversation_table_schema := `CREATE TABLE user_conversation_table (
    UserID TEXT NOT NULL, -- User ID
    ConversationID TEXT NOT NULL, -- Conversation ID
    PRIMARY KEY (UserID, ConversationID), -- Composite primary key
    FOREIGN KEY (UserID) REFERENCES user_table(UserID) ON DELETE CASCADE, -- Link to user_table
    FOREIGN KEY (ConversationID) REFERENCES conversation_table(ConversationID) ON DELETE CASCADE -- Link to conversation_table
);`
```

```
// Initialize TABLES
if err := ensureTableExists(db, "user_table", user_table_schema); err != nil {
    return err
}

if err := ensureTableExists(db, "conversation_table", conversation_table_schema); err != nil {  
    You, 1 second ago  
    return err
}

if err := ensureTableExists(db, "user_conversation_table", user_conversation_table_schema); err != nil {
    return err
}
```

aggiungiamo la funzione
initializeDatabase

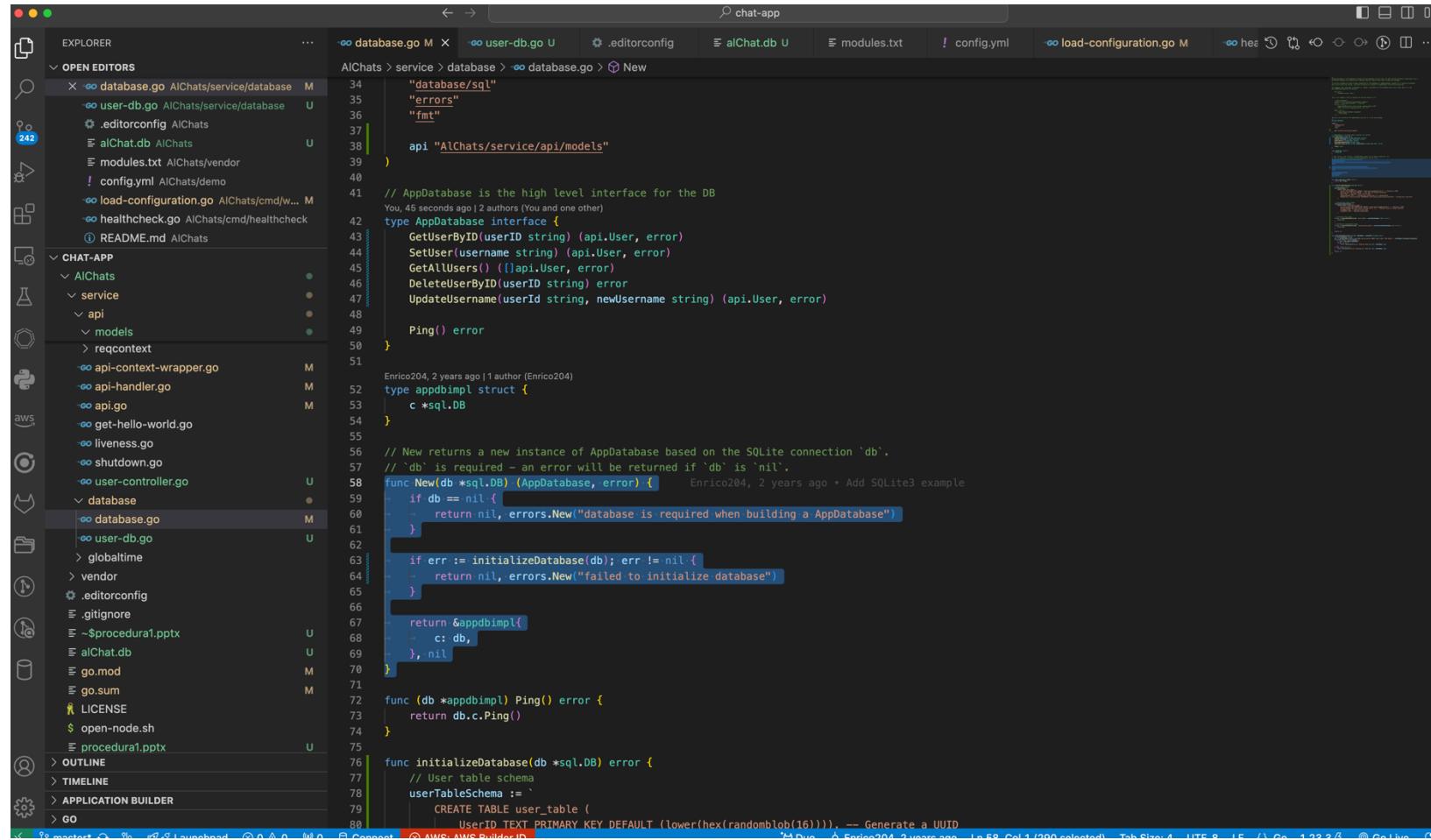
che crea le tabelle se non
esistono

aggiungo una tabella per testire
la relazione molti a molti

DDL and Initialization for Conversation

modifichiamo la funzione New
in modo da chiamare
le nostre funzioni

testiamo la nuova struttura



```
database.go
package AlChats

import (
    "database/sql"
    "errors"
    "fmt"
    "github.com/Enrico204/api"
    "github.com/Enrico204/appdbimpl"
    "github.com/Enrico204/config"
    "github.com/Enrico204/load-configuration"
    "github.com/Enrico204/healthcheck"
    "github.com/Enrico204/vendor"
    "github.com/Enrico204/reqcontext"
    "github.com/Enrico204/api-context-wrapper"
    "github.com/Enrico204/api-handler"
    "github.com/Enrico204/api"
    "github.com/Enrico204/get-hello-world"
    "github.com/Enrico204/liveness"
    "github.com/Enrico204/shutdown"
    "github.com/Enrico204/user-controller"
    "github.com/Enrico204/database"
    "github.com/Enrico204/user-db"
    "github.com/Enrico204/globaltime"
    "github.com/Enrico204/vendor"
    "github.com/Enrico204/.editorconfig"
    "github.com/Enrico204/.gitignore"
    "github.com/Enrico204/~$procedura1.pptx"
    "github.com/Enrico204/alChat.db"
    "github.com/Enrico204/go.mod"
    "github.com/Enrico204/go.sum"
    "github.com/Enrico204/LICENSE"
    "github.com/Enrico204/open-node.sh"
    "github.com/Enrico204/~$procedura1.pptx"
)

// AppDatabase is the high level interface for the DB
type AppDatabase interface {
    GetUserByID(userID string) (api.User, error)
    SetUser(username string) (api.User, error)
    GetAllUsers() ([]api.User, error)
    DeleteUserID(userID string) error
    UpdateUsername(userID string, newUsername string) (api.User, error)
    Ping() error
}

// New returns a new instance of AppDatabase based on the SQLite connection `db`.
// `db` is required - an error will be returned if `db` is `nil`.
func New(db *sql.DB) (AppDatabase, error) {
    if db == nil {
        return nil, errors.New("database is required when building a AppDatabase")
    }

    if err := initializeDatabase(db); err != nil {
        return nil, errors.New("failed to initialize database")
    }

    return &appdbimpl{
        db,
        nil,
    }
}

func (db *appdbimpl) Ping() error {
    return db.c.Ping()
}

func initializeDatabase(db *sql.DB) error {
    // User table schema
    userTableSchema := `
        CREATE TABLE user_table (
            UserID TEXT PRIMARY KEY DEFAULT (lower(hex(randomblob(16)))) -- Generate a UUID
    `

    _, err := db.Exec(userTableSchema)
    if err != nil {
        return err
    }

    return nil
}
```

DDL and Initialization for Conversation

testiamo la nuova struttura

cancellazione db

```
pcarlo@DLJ473C2T5 chat-app % cd AlChats
pcarlo@DLJ473C2T5 AlChats % ls -al al*
-rw-r--r-- 1 pcarlo staff 16384 3 Dic 12:58 alChat.db
pcarlo@DLJ473C2T5 AlChats % rm alChat.db
pcarlo@DLJ473C2T5 AlChats % ls -al al*
zsh: no matches found: al*
pcarlo@DLJ473C2T5 AlChats %
```

avvio programma

```
pcarlo@DLJ473C2T5 AlChats % go run ./cmd/webapi/
INFO[0000] application initializing
INFO[0000] initializing database support
INFO[0000] initializing API server
INFO[0000] API listening on 0.0.0.0:3000
```

check database

```
pcarlo@DLJ473C2T5 AlChats % ls -al al*
-rw-r--r-- 1 pcarlo staff 24576 3 Dic 13:40 alChat.db
pcarlo@DLJ473C2T5 AlChats % sqlite3 alChat.db
SQLite version 3.43.2 2023-10-10 13:08:14
Enter ".help" for usage hints.
sqlite> SELECT name FROM sqlite_master WHERE type='table';
user_table
conversation_table
user_conversation_table
sqlite>
```

Creazione funzioni DB per Conversation

```
AIChats > service > api > models > conversation.go > ...
1 package models
2
3 // Conversation represents a conversation entity
4 type Conversation struct {
5     ConversationID string `json:"conversationId" // Unique identifier for the conversation
6     IsGroup         bool   `json:"isGroup"      // Indicates if the conversation is a group
7     GroupName       string `json:"groupName"    // Name of the group (optional, only for group conversations)
8     GroupPhoto      string `json:"groupPhoto"   // Photo of the group (optional, only for group conversations)
9 }
10
```

creazione
modello per conversation

Creazione funzioni DB per Conversation

The screenshot shows a web browser window with the URL chatgpt.com/c/674dc258-b0d4-8005-98fe-4389fc1ad643. The page title is "ChatGPT". On the left, there's a sidebar with a "Today" section containing links like "Go import troubleshooting", "Create User Table SQL", and "RabbitMQ vs STOMP Flutter". Below that is a "Previous 7 Days" section with links such as "500 Error CORS Debugging", "Real-time Notifications Scaling", "Maven Lettuce-core Missing Ar...", "BCC Tanzu Spring Subscription", "CSS Not Loading Vue", "Vue SPA Example", "What is Hibernate", "Login Handler Improvement", "Felt Crafty Image", "Spostare call pomeriggio", "Career Opportunity Discussion", "Meeting Delay Explanation", "Hai un'idea", "Cloud Competencies Summary", "INPS call e saluti", and "Java to Go Translation". At the bottom of the sidebar is an "Upgrade plan" button.

The main content area contains a prompt from ChatGPT asking to "crea una funzione simile alla seguente" (create a function similar to the following). It provides a Go code snippet:

```
func (db *appdbimpl) SetUser(username string) (api.User, error) {
    var user api.User

    // SQL to insert a new user and retrieve the generated UserID and other fields
    query :=
        `INSERT INTO user_table (Username)
         VALUES (?)`  
        RETURNING UserID, Username, COALESCE(Photo, "")
```

Below this, ChatGPT asks "per il seguente modello" (according to the following model) and provides a "Conversation struct":

```
type Conversation struct {  
    ConversationID string  
    RecentConversationID string  
    UserID string  
}
```

At the bottom of the main content area, it says "Message ChatGPT" and "ChatGPT can make mistakes. Check important info."

chatGPT

Creazione funzioni DB per Conversation

```
SetConversation(isGroup bool, groupName, groupPhoto string) (models.Conversation, error)
```

The screenshot shows a code editor with two tabs open:

- conversation-db.go**: This tab contains Go code for creating a conversation database. It includes imports for `database`, `api`, and `fmt`. The main function `SetConversation` takes parameters `isGroup`, `groupName`, and `groupPhoto`, returning a `Conversation` object and an `error`. It performs validation checks on user IDs and then executes an SQL query to insert the conversation details into the `conversation_table`.
- AppDatabase.go**: This tab contains Go code for initializing database support. It imports `database/sql`, `errors`, and `fmt`. It defines an interface `AppDatabase` with methods like `GetUserByUserID`, `SetUser`, `GetAllUsers`, `DeleteUserByUserID`, `UpdateUserName`, and `SetConversation`. It also defines a struct `appdbimpl` which implements the `AppDatabase` interface.

The code editor interface includes an Explorer sidebar, a search bar, and various status indicators at the bottom.

Creazione funzioni DB per Conversation

codice completo

```
package database

import (
    "github.com/AIChats/service/api/models"
    "fmt"
)

func (db *appdb) SetConversation(userIDs []string, isGroup bool, groupName, groupPhoto string) (*models.Conversation, error) {
    var conversation api.Conversation

    // Check for invalid userIDs length
    if len(userIDs) == 1 {
        return &conversation, fmt.Errorf("cannot create a conversation with only one user")
    }

    // Check if userIDs is greater than 2 and isGroup is false
    if len(userIDs) > 2 && !isGroup {
        return &conversation, fmt.Errorf("cannot create a group conversation with more than two users without setting isGroup to true")
    }

    // SQL to insert a new conversation and retrieve the generated ConversationID and other fields
    query := `

        INSERT INTO conversation_table (IsGroup, GroupName, GroupPhoto)
        VALUES (?, ?, ?)
        RETURNING
            ConversationID,
            IsGroup,
            COALESCE(GroupName, ''),
            COALESCE(GroupPhoto, '')

    `

    // Insert the conversation and fetch the generated fields
    err := db.c.QueryRow(query, isGroup, groupName, groupPhoto).Scan(&conversation.ConversationID, &conversation.IsGroup, &conversation.GroupName, &conversation.GroupPhoto)

    if err != nil {
        // Handle any database error
        return &conversation, fmt.Errorf("failed to create conversation: %w", err)
    }

    // Insert user-conversation relationships into the user_conversation_table
    for _, userID := range userIDs {
        // Check if the UserID exists in the user_table
        var exists bool
        checkUserQuery := `SELECT EXISTS(SELECT 1 FROM user_table WHERE UserID = ?)`
        err := db.c.QueryRow(checkUserQuery, userID).Scan(&exists)
        if err != nil {
            return &conversation, fmt.Errorf("failed to check if user exists: %w", err)
        }

        // If the user does not exist, return an error
        if !exists {
            return &conversation, fmt.Errorf("user with UserID %s does not exist", userID)
        }

        // SQL to insert the relationship between user and conversation
        relationshipQuery := `

            INSERT INTO user_conversation_table (UserID, ConversationID)
            VALUES (?, ?)

        `

        // Insert the user-conversation relationship
        _, err = db.c.Exec(relationshipQuery, userID, conversation.ConversationID)
        if err != nil {
            // If an error occurs, return the error
            return &conversation, fmt.Errorf("failed to create user-conversation relationship: %w", err)
        }
    }
}

return &conversation, nil
}
```

verifica se gli userid passati sono giusti
(considerando group o conversazione)

aggiorna tabella relazione multi-molti
dopo aver verificato che ogni utente esiste

Creazione funzioni DB per Conversation

```
 GetAllConversations() ([]api.Conversation, error)
```

The screenshot shows a code editor interface with a dark theme. The left sidebar contains a file tree with project structure. The main editor area displays Go code for a function named `GetAllConversations`.

```
func (db *appdbimpl) GetAllConversations() ([]api.Conversation, error) {
    var conversations []api.Conversation
    query := `SELECT
        ConversationID,
        IsGroup,
        COALESCE(GroupName, ''),
        COALESCE(GroupPhoto, '')
    FROM conversation_table`

    rows, err := db.c.Query(query)
    if err != nil {
        return nil, fmt.Errorf("failed to retrieve conversations: %w", err)
    }
    defer rows.Close()

    for rows.Next() {
        var conversation api.Conversation
        err := rows.Scan(&conversation.ConversationID, &conversation.IsGroup, &conversation.GroupName, &conversation.GroupPhoto)
        if err != nil {
            return nil, fmt.Errorf("failed to scan conversation row: %w", err)
        }
        conversations = append(conversations, conversation)
    }

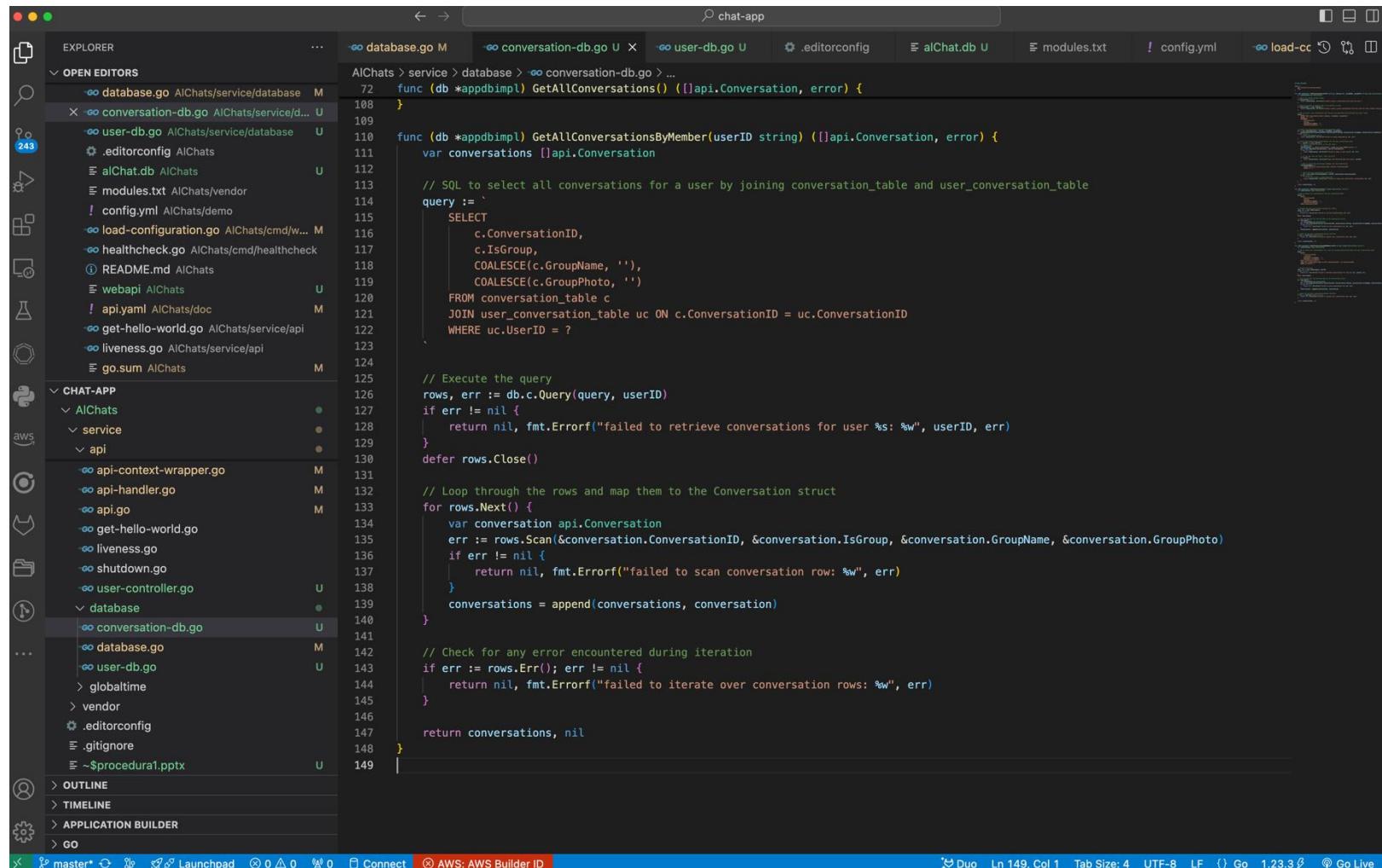
    // Check for any error encountered during iteration
    if err := rows.Err(); err != nil {
        return nil, fmt.Errorf("failed to iterate over conversation rows: %w", err)
    }

    return conversations, nil
}
```

The code implements a database query to select all conversations from the `conversation_table`. It uses the `COALESCE` function to handle null values for `GroupName` and `GroupPhoto`. The results are mapped to a `Conversation` struct and returned as a slice of `Conversation` objects.

Creazione funzioni DB per Conversation

```
 GetAllConversationsByMember(userID string) ([]api.Conversation, error)
```



```
func (db *appdbimpl) GetAllConversations() ([]api.Conversation, error) {
    var conversations []api.Conversation
    query := `
        SELECT
            c.ConversationID,
            c.IsGroup,
            COALESCE(c.GroupName, ''),
            COALESCE(c.GroupPhoto, '')
        FROM conversation_table c
        JOIN user_conversation_table uc ON c.ConversationID = uc.ConversationID
        WHERE uc.UserID = ?`

    // Execute the query
    rows, err := db.c.Query(query, userID)
    if err != nil {
        return nil, fmt.Errorf("failed to retrieve conversations for user %s: %w", userID, err)
    }
    defer rows.Close()

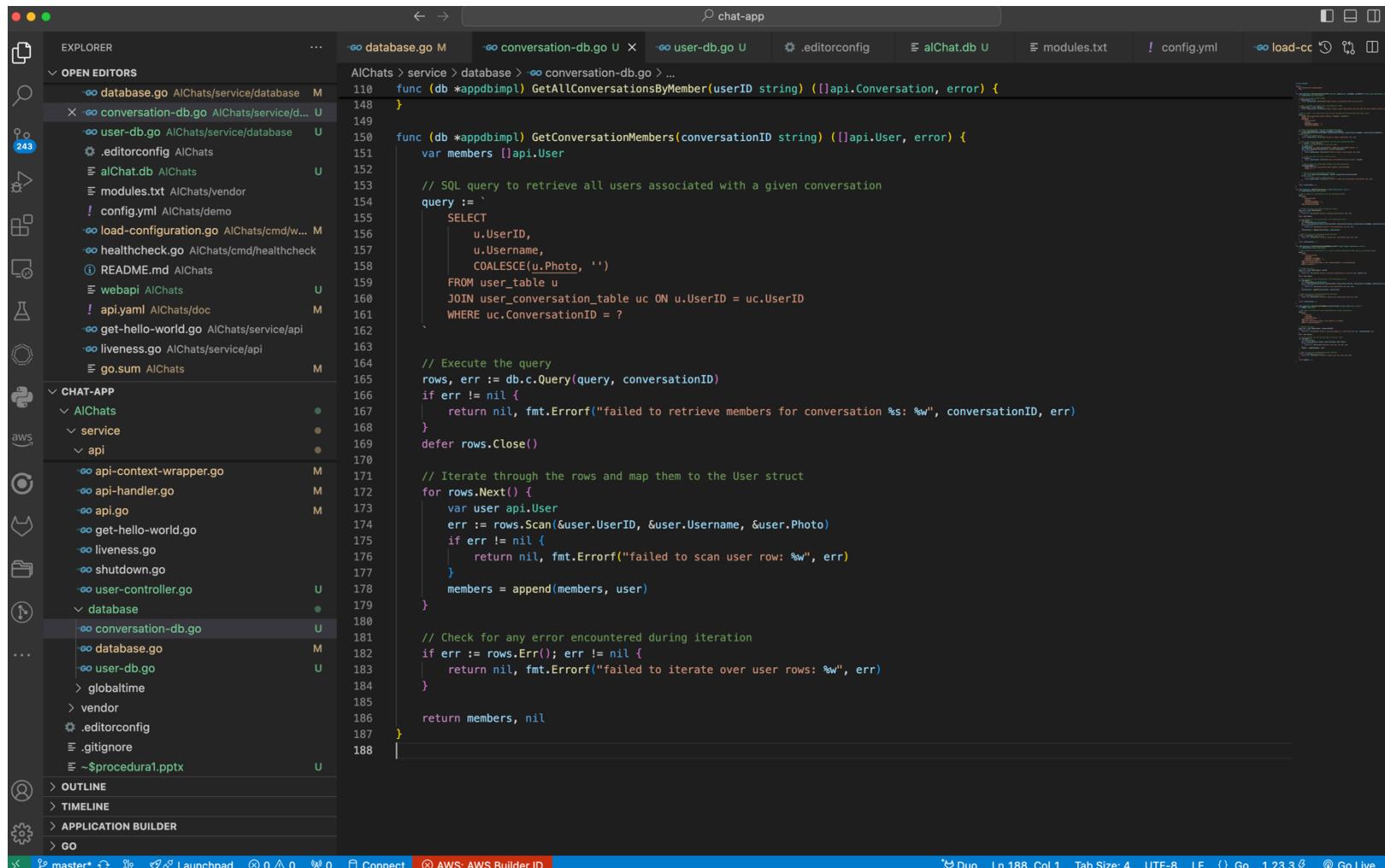
    // Loop through the rows and map them to the Conversation struct
    for rows.Next() {
        var conversation api.Conversation
        err := rows.Scan(&conversation.ConversationID, &conversation.IsGroup, &conversation.GroupName, &conversation.GroupPhoto)
        if err != nil {
            return nil, fmt.Errorf("failed to scan conversation row: %w", err)
        }
        conversations = append(conversations, conversation)
    }

    // Check for any error encountered during iteration
    if err := rows.Err(); err != nil {
        return nil, fmt.Errorf("failed to iterate over conversation rows: %w", err)
    }

    return conversations, nil
}
```

Creazione funzioni DB per Conversation

```
GetConversationMembers(conversationID string) ([]api.User, error)
```



The screenshot shows a dark-themed code editor interface with multiple tabs and a sidebar.

EXPLORER: Shows the project structure:

- AIChats > service > database > conversation-db.go
- AIChats > service > database > user-db.go
- AIChats > service > database > .editorconfig
- AIChats > service > database > modules.txt
- AIChats > service > database > config.yml
- AIChats > service > database > load-cc
- AIChats > service > database > healthcheck.go
- AIChats > service > database > README.md
- AIChats > service > database > webapi
- AIChats > service > database > api.yaml
- AIChats > service > database > get-hello-world.go
- AIChats > service > database > liveness.go
- AIChats > service > database > go.sum
- CHAT-APP > AIChats > service > api > api-context-wrapper.go
- CHAT-APP > AIChats > service > api > api-handler.go
- CHAT-APP > AIChats > service > api > api.go
- CHAT-APP > AIChats > service > api > get-hello-world.go
- CHAT-APP > AIChats > service > api > liveness.go
- CHAT-APP > AIChats > service > api > shutdown.go
- CHAT-APP > AIChats > service > database > user-controller.go
- CHAT-APP > AIChats > service > database > conversation-db.go
- CHAT-APP > AIChats > service > database > database.go
- CHAT-APP > AIChats > service > database > user-db.go
- CHAT-APP > AIChats > service > database > globaltime
- CHAT-APP > AIChats > service > database > vendor
- CHAT-APP > AIChats > service > database > .editorconfig
- CHAT-APP > AIChats > service > database > .gitignore
- CHAT-APP > AIChats > service > database > ~\$procedure1.pptx
- OUTLINE
- TIMELINE
- APPLICATION BUILDER
- GO

EDITOR: Displays the content of the `conversation-db.go` file:

```
func (db *appdbimpl) GetAllConversationsByMember(userID string) ([]api.Conversation, error) {
}

func (db *appdbimpl) GetConversationMembers(conversationID string) ([]api.User, error) {
    var members []api.User

    // SQL query to retrieve all users associated with a given conversation
    query := `SELECT
        u.UserID,
        u.Username,
        COALESCE(u.Photo, '')
    FROM user_table u
    JOIN user_conversation_table uc ON u.UserID = uc.UserID
    WHERE uc.ConversationID = ?`

    // Execute the query
    rows, err := db.c.Query(query, conversationID)
    if err != nil {
        return nil, fmt.Errorf("failed to retrieve members for conversation %s: %w", conversationID, err)
    }
    defer rows.Close()

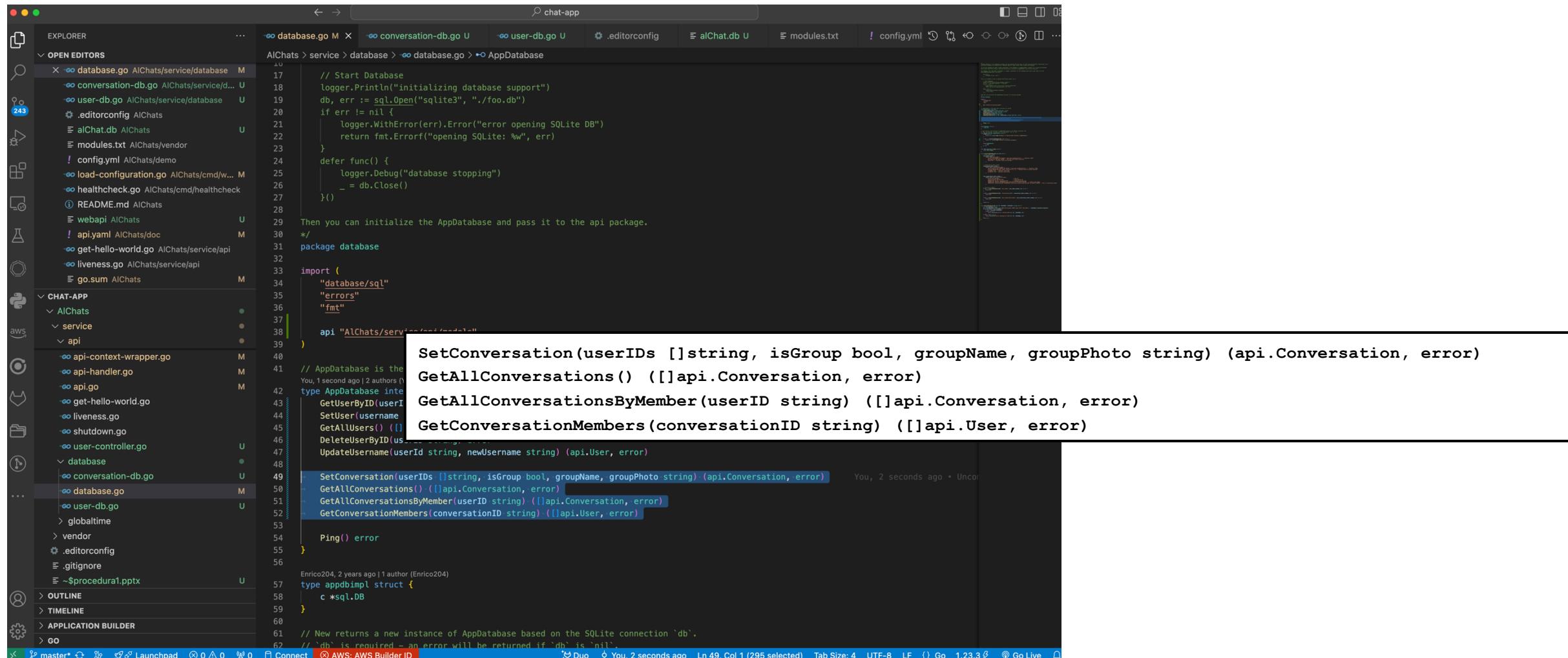
    // Iterate through the rows and map them to the User struct
    for rows.Next() {
        var user api.User
        err := rows.Scan(&user.UserID, &user.Username, &user.Photo)
        if err != nil {
            return nil, fmt.Errorf("failed to scan user row: %w", err)
        }
        members = append(members, user)
    }

    // Check for any error encountered during iteration
    if err := rows.Err(); err != nil {
        return nil, fmt.Errorf("failed to iterate over user rows: %w", err)
    }

    return members, nil
}
```

STATUS BAR: Shows the current file is `master`, line 188, column 1, tab size 4, encoding UTF-8, and other standard developer tools.

Creazione funzioni DB per Conversation



The screenshot shows a code editor interface with a dark theme. The left sidebar displays a file tree for a project named "chat-app". The main editor area shows a Go file named "database.go". The code implements a database layer using SQLite. It includes functions for initializing the database, performing conversations, and managing users.

```
// Start Database
logger.Println("initializing database support")
db, err := sql.Open("sqlite3", "./foo.db")
if err != nil {
    logger.WithError(err).Error("error opening SQLite DB")
    return fmt.Errorf("opening SQLite: %w", err)
}
defer func() {
    logger.Debug("database stopping")
    _ = db.Close()
}()

Then you can initialize the AppDatabase and pass it to the api package.

package database

import (
    "database/sql"
    "errors"
    "fmt"
)

api "AlChats/service/api"

SetConversation(userIDs []string, isGroup bool, groupName, groupPhoto string) (api.Conversation, error)
GetAllConversations() ([]api.Conversation, error)
GetAllConversationsByMember(userID string) ([]api.Conversation, error)
GetConversationMembers(conversationID string) ([]api.User, error)

func NewAppDatabase(db *sql.DB) AppDatabase {
    You, 1 second ago | 2 authors ()
    type AppDatabase interface {
        GetUserByID(userID string) (*User, error)
        SetUser(username string)
        GetAllUsers() ([]User, error)
        DeleteUserByID(userID string)
        UpdateUsername(userID string, newUsername string) (User, error)
    }

    SetConversation(userIDs []string, isGroup bool, groupName, groupPhoto string) (Conversation, error)
    GetAllConversations() ([]Conversation, error)
    GetAllConversationsByMember(userID string) ([]Conversation, error)
    GetConversationMembers(conversationID string) ([]User, error)

    Ping() error
}

type appdbimpl struct {
    db *sql.DB
}

// New returns a new instance of AppDatabase based on the SQLite connection `db`.
// `db` is required - an error will be returned if `db` is `nil`.
func New(db *sql.DB) AppDatabase {
    if db == nil {
        logger.Error("db is required - an error will be returned if `db` is `nil`")
    }
    return &appdbimpl{db}
}
```

Creazione endpoint HTTP per Conversation

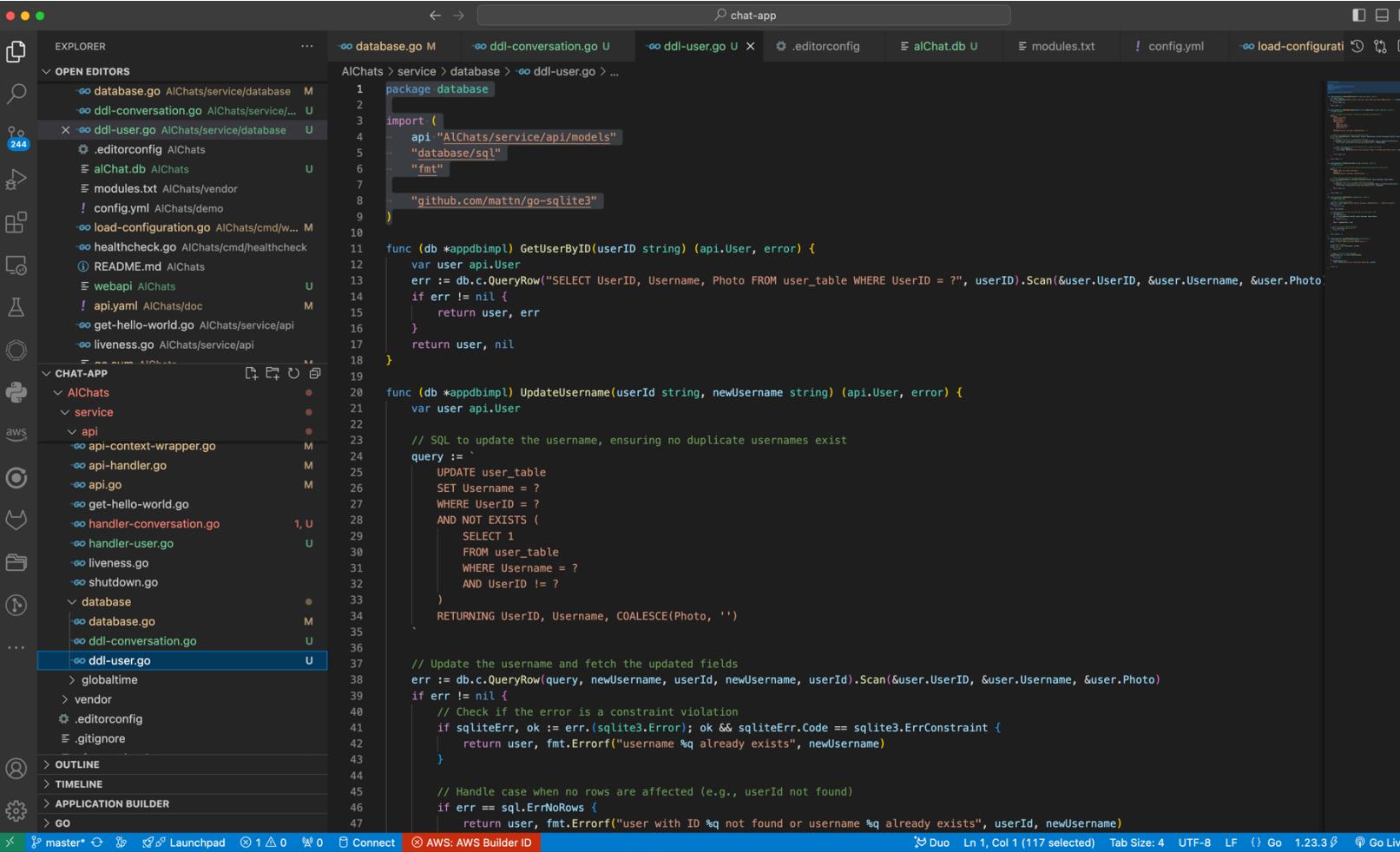
cambio nome dei controller in handler (handler-user ed il nuovo handler-conversation)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CHAT-APP".
 - AIChats:** Contains files like go.sum, go.mod, user.go, main.go, api.go, conversation.go, api-context-wrapper.go, request-context.go, handler-user.go, shutdown.go, and api-handler.go.
 - service:** Contains files like models/user.go.
 - reqcontext:** Contains files like request-context.go, api-context-wrapper.go, api-handler.go, and api.go.
 - database:** Contains files like conversation-db.go, database.go, and user-db.go.
- Open Editors:** Shows multiple tabs open:
 - api.go (M)
 - conversation.go (U)
 - api-context-wrapper.go (M)
 - request-context.go
 - handler-user.go (U)
 - handler-conversation.go (1, U) - The active tab.
 - shutdown.go
 - api-handler.go
- Code Editor:** Displays the content of handler-conversation.go.
- Bottom Status Bar:** Shows tabs for master*, Launchpad, Connect, AWS: AWS Builder ID, and various status indicators like Duo, Ln 1, Col 1, Tab Size: 4, UTF-B, LF, Go, 1.23.3, and Go Live.

Creazione endpoint HTTP per Conversation

cambio nome anche ai file del db che contengono le query DDL (data definition language)
user-db.go diventa ddl.user.go (solo per una questione di leggibilità)



```
package database

import (
    "AlChats/service/api/models"
    "database/sql"
    "fmt"
    "github.com/mattn/go-sqlite3"
)

func (db *appdbimpl) GetUserByID(userID string) (api.User, error) {
    var user api.User
    err := db.c.QueryRow("SELECT UserID, Username, Photo FROM user_table WHERE UserID = ?", userID).Scan(&user.UserID, &user.Username, &user.Photo)
    if err != nil {
        return user, err
    }
    return user, nil
}

func (db *appdbimpl) UpdateUsername(userID string, newUsername string) (api.User, error) {
    var user api.User

    // SQL to update the username, ensuring no duplicate usernames exist
    query := `
        UPDATE user_table
        SET Username = ?
        WHERE UserID = ?
        AND NOT EXISTS (
            SELECT 1
            FROM user_table
            WHERE Username = ?
            AND UserID != ?
        )
        RETURNING UserID, Username, COALESCE(Photo, '')
    `

    // Update the username and fetch the updated fields
    err := db.c.QueryRow(query, newUsername, userID, newUsername, userID).Scan(&user.UserID, &user.Username, &user.Photo)
    if err != nil {
        // Check if the error is a constraint violation
        if sqliteErr, ok := err.(sqlite3.Error); ok && sqliteErr.Code == sqlite3.ErrConstraint {
            return user, fmt.Errorf("username %q already exists", newUsername)
        }
    }

    // Handle case when no rows are affected (e.g., userID not found)
    if err == sql.ErrNoRows {
        return user, fmt.Errorf("user with ID %q not found or username %q already exists", userID, newUsername)
    }
}
```

Creazione endpoint HTTP per Conversation

creo ConversationRequest che mi serve per definire il body della POST
creo il SetConversationhandler e lo dichiaro nel router

The image shows two side-by-side code editors, likely from the AWS Toolkit for Visual Studio Code, displaying Go code for a chat application named "chat-app".

Left Editor (chat-app):

```
10    }
11
12    type ConversationRequest struct {
13        UserIDs []string `json:"user_ids"`
14        IsGroup   bool    `json:"is_group"`
15        GroupName string  `json:"group_name,omitempty"`
16        GroupPhoto string  `json:"group_photo,omitempty"`
17    }
18
19    // SetConversationHandler handles the creation of new conversations.
20    func (h *_router) SetConversationHandler(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {
21        w.Header().Set("Content-Type", "application/json")
22
23        // Parse the JSON request body
24        var req ConversationRequest
25        if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
26            http.Error(w, `{"error":"invalid request body"}`, http.StatusBadRequest)
27            return
28        }
29
30        // Validate the required fields
31        if len(req.UserIDs) == 0 {
32            http.Error(w, `{"error":"user_ids is required"}`, http.StatusBadRequest)
33            return
34        }
35
36        // Call the SetConversation function
37        conversation, err := h.db.SetConversation(req.UserIDs, req.IsGroup, req.GroupName, req.GroupPhoto)
38        if err != nil {
39            if strings.Contains(err.Error(), "cannot create a conversation with only one user") ||
40                strings.Contains(err.Error(), "cannot create a group conversation") {
41                http.Error(w, fmt.Sprintf(`{"error":"%v"}`, err), http.StatusBadRequest)
42            } else if strings.Contains(err.Error(), "user with UserID") {
43                http.Error(w, fmt.Sprintf(`{"error":"%v"}`, err), http.StatusNotFound)
44            } else {
45                http.Error(w, fmt.Sprintf(`{"error":"%v"}`, err), http.StatusInternalServerError)
46            }
47            return
48        }
49
50        // Respond with the created conversation
51        if err := json.NewEncoder(w).Encode(conversation); err != nil {
52            http.Error(w, fmt.Sprintf(`{"error":"failed to encode response: %v"}`, err), http.StatusInternalServerError)
53        }
54    }
55}
```

Right Editor (context-wrapper.go):

```
1 package api
2
3 import (
4     "net/http"
5 )
6
7 // Handler returns an instance of httprouter.Router that handle APIs registered here
8 func (rt *_router) Handler() http.Handler {
9     rt.router.GET("/", rt.getHelloWorld)
10
11    // Special routes
12    rt.router.GET("/liveness", rt.liveness)
13
14    // USER ENDPOINT
15    rt.router.POST("/user/session", rt.createUserHandler)
16    rt.router.GET("/users", rt.getAllUsersHandler)
17    rt.router.PUT("/user", rt.updateUsernameHandler)
18
19    // CONVERSATION ENDPOINT
20    rt.router.POST("/conversation", rt.setConversationHandler)
21
22    return rt.router
23}
24
25
```

Bottom Right Panel:

```
{
  "user_ids": ["5eef00fcfddfa3848b9f850e1015184", "daf495832109b2785bb525055633c4c9"],
  "is_group": false,
  "group_name": "",
  "group_photo": "https://example.com/photo.png"
}
```

Creazione endpoint HTTP per Conversation

test endpoint

cancellazione db

```
pcarlo@DLJ473C2T5 chat-app % cd AlChats
pcarlo@DLJ473C2T5 AlChats % ls -al al*
-rw-r--r-- 1 pcarlo staff 16384 3 Dic 12:58 alChat.db
pcarlo@DLJ473C2T5 AlChats % rm alChat.db
pcarlo@DLJ473C2T5 AlChats % ls -al al*
zsh: no matches found: al*
pcarlo@DLJ473C2T5 AlChats %
```

avvio programma

```
pcarlo@DLJ473C2T5 AlChats % go run ./cmd/webapi/
INFO[0000] application initializing
INFO[0000] initializing database support
INFO[0000] initializing API server
INFO[0000] API listening on 0.0.0.0:3000
```

check database

```
sqlite> SELECT name FROM sqlite_master WHERE type='table';
user_table
conversation_table
user_conversation_table
sqlite> select * from user_table;
sqlite> select * from conversation_table;
sqlite> select * from user_conversation_table;
sqlite>
```

nessun dato

Home Workspaces Explore

Welcome to Lightweight API Client, your new testing ground for APIs. If you're looking for your Scratch Pad data, don't worry; it is safe and secure. Sign up to access it anytime and continue working with collections and environments, or import one from your favorite source.

History New Import GET http://localhost:3000/use POST localhost:3000/conversation + ...

localhost:3000/conversation

POST localhost:3000/conversation

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Body

```
1 "user_ids": ["5eef00fcfbddfa3848b9f850e1015184", "daf495832109b2785bb525055633c4c9"],  
2 "is_group": false,  
3 "group_name": "",  
4 "group_photo": "https://example.com/photo.png"  
5  
6  
7
```

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize JSON

```
1 "conversationId": "95eac945d8be32915c0b8f16bfaa9df3",  
2 "isGroup": false,  
3 "groupName": "",  
4 "groupPhoto": "https://example.com/photo.png"  
5  
6
```

Console Not connected to a Postman account

GET http://localhost:3000/use POST localhost:3000/conversation + ...

HTTP http://localhost:3000/users

GET http://localhost:3000/users

Params Authorization Headers (6) Body Pre-request Script Tests

Query Params

Key
Key

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize JSON

```
1 [  
2 {  
3 "userId": "5eef00fcfbddfa3848b9f850e1015184",  
4 "username": "alberto"  
5 },  
6 {  
7 "userId": "daf495832109b2785bb525055633c4c9",  
8 "username": "carlo"  
9 },  
10 {  
11 "userId": "3c59b996675ea47c2000243bc62870d6",  
12 "username": "savina"  
13 },  
14 {  
15 "userId": "9ff0bdbbf168186944839b7da678905d",  
16 "username": "marialuisa"  
17 }  
18 ]
```

Creazione endpoint HTTP per Conversation

test endpoint

cancellazione db

```
pcarlo@DLJ473C2T5 chat-app % cd AlChats
pcarlo@DLJ473C2T5 AlChats % ls -al al*
-rw-r--r-- 1 pcarlo staff 16384 3 Dic 12:58 alChat.db
pcarlo@DLJ473C2T5 AlChats % rm alChat.db
pcarlo@DLJ473C2T5 AlChats % ls -al al*
zsh: no matches found: al*
pcarlo@DLJ473C2T5 AlChats %
```

avvio programma

```
pcarlo@DLJ473C2T5 AlChats % go run ./cmd/webapi/
INFO[0000] application initializing
INFO[0000] initializing database support
INFO[0000] initializing API server
INFO[0000] API listening on 0.0.0.0:3000
```

check database

```
sqlite> select * from user_table;
5eef00fcfbddfa3848b9f850e1015184|alberto|
daf495832109b2785bb525055633c4c9|carlo|
3c59b996675ea47c2000243bc62870d6|savina|
9ff0bdbbf168186944839b7da678905d|marialuisa|
sqlite> select * from conversation_table;
95eac945d8be32915c0b8f16bfaa9df3|0||https://example.com/photo.png
sqlite> select * from user_conversation_table;
5eef00fcfbddfa3848b9f850e1015184|95eac945d8be32915c0b8f16bfaa9df3
daf495832109b2785bb525055633c4c9|95eac945d8be32915c0b8f16bfaa9df3
sqlite>
```