Backend Server

Server Code, the heart of the system.

Index

Scope / Purpose

Backend has the following tasks:

- Handles Game State
- Connects to the hybrid die
- future: Connects to the Hue Lights

The backend is in flow-control when the game is running.

The api and communication flow with the client web app is described in the api folder.

Development

Directory structure as seen in "Golang Standard Project Layout"

Requirements

- Golang 1.20 or higher
- npm to generate GO code from api spec

Setup

Download dependencies and generate GO code from api spec.

```
go mod download
npm i
go generate cmd/quizzit/quizzit.go (generates golang types from the
./api/websocket-asyncapi.yaml)
```

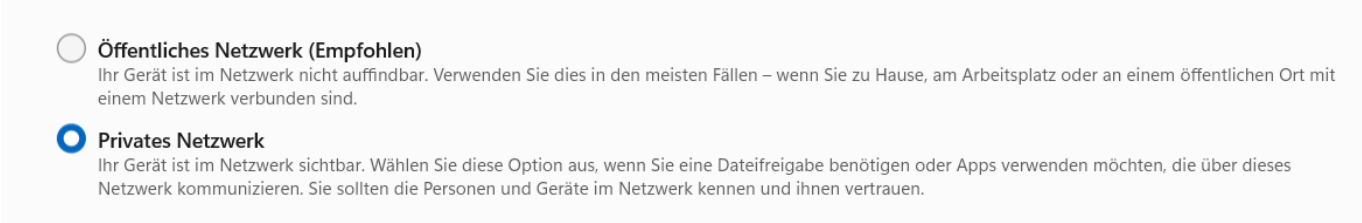
Run

To start the server:

```
go run cmd/quizzit/quizzit.go
```

The server will start listening on http://localhost:8080.

If you are using Windows and want to use the Hybrid Die make sure you are in the same `private` Network as the die. `Private` referes to your system's security settings of this Network. For windows, see the screenshot below.



Debug in VS-Code

Possible VS-Code `.vscode/launch.json` configuration:

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Quizzit GO",
      "type": "go",
      "request": "launch",
      "mode": "auto",
      "program": "${workspaceFolder}/backend/cmd/quizzit/quizzit.go",
      "cwd": "${workspaceFolder}/backend",
      "trace": "trace"
    }
```

```
  ]
}
```

Test

```
go test ./test
```

Build

To build the binary:

```
go build cmd/quizzit/quizzit.go
```

This will create a binary named quizzit in the project directory.

Code format

Golang has a built-in command-line tool called go fmt that automatically formats Go source code. The go fmt command formats your code according to a set of rules defined in the Go code style guidelines.

Could be implemented in CI/CD in the future.

Testing Websocket

Can use this lovely page here: https://websocketking.com/ and connect to `ws://localhost:8080/ws`.
Some examples for valid JSONs are listed below, for all possible event, view the api specification.

Submit a generic confirm

```
{
  "messageType": "player/generic/Confirm"
}
```

Submit player count

```
{
  "messageType": "player/setup/SubmitPlayerCount",
  "body": 3
}
```

Roll a category

```
{
  "messageType": "player/die/DigitalCategoryRollRequest"
}
```

Submit an Answer

```
{
  "messageType": "player/question/SubmitAnswer",
  "body": {
    "questionId": "question-1",
    "answerId": "C"
  }
}
```

Deployment

Create deployment

To create a deployable gitlab release, go to the tag section of the backend repository in gitlab.

Create a new tag with a specified new version number (e.g. v1.0.0).

A new release will be created, which can be found in the deploy section.

In production the backend runs as binary on a RaspberryPi 4B along with the client web app. For more information regarding deploying to production, look into the RaspberryPi Installation Guide.

System requirenments:

- tested on RaspberryPi 4B
- should run on most systems with:
  - Windows or Linux as operating system
  - Wifi card

Related Content

- API Specification
- Backend Wiki
- Game Loop
- RaspberryPi Installation Guide
- Go Generate