VOYAGE GROUP
1Dayインターン

三浦 裕典
ムラタ ガイア ソウイチロウ
さっさー
ひむ
ともかつ

タイムスケジュール

11:00 - 11:15 講師/サポータ紹介 インターン生自己紹介
11:15 - 11:30 アイスブレイク
11:30 - 12:10 インターン概要 GitHubチュートリアル

12:10 - 13:10 休憩

13:10 - 14:10 Mission1（一人で）
- 13:10 - 13:15 Mission 1-1, 1-2 説明
- 13:15 - 13:35 開発
- 13:35 - 14:00 開発
- 14:00 - 14:10 答え合わせ

14:10 - 15:30 Mission2（一人で）
- 14:10 - 14:50 Mission 2-1
- 15:00 - 15:30 Mission 2-2

15:40 - 18:20 Mission3（チームで開発）
- 15:40 - 15:45 チーム発表、席移動
- 15:45 - 16:15 前半開発 - やること決め
- 16:15 - 16:55 とりあえずやってみる
  WIPのPR出してね
- 16:55 - 17:05 前半振り返り / 後半やること
- 17:05 - 18:20 後半開発

18:20 - 18:30 発表、サポータ講評

宣伝とお願い

ブログ　http://techlog.voyagegroup.com/

Twitter　https://twitter.com/tech_voyage

社外に情報発信する一環で後ろから写真を撮らせてください。

インターンテーマ

Git、GitHubによる
ソース管理と開発フロー

インターンテーマ

golang!golang!golang!

インターンテーマ

チーム開発

インターンテーマ

1日の限られた時間の中で、
この3つのテーマを本日は学んでいきます！

インターン概要

今回のインターンでは夏のエンジニアインターンTreasureの
エッセンスを伝えることを目的としています。

エッセンスを伝える上でフロントエンド(HTML&JavaScript)、
サーバサイド(golang)を連携させた開発を用いミッション形式で
実践しながら**Git、GitHubによるソースコードの管理と開発フロー、チーム開発**について学びます。

Tutorial

Git, GitHubチュートリアル

Git, GitHub Tutorial

本日は下記のリポジトリを用いて学びます。リポジトリを操作するための準備をします。
https://github.com/VG-Tech-Dojo/vg-1day-2018-06-10

1.Collaborators
皆さんの GitHub ID を事前に今回のリポジトリに招待してあるので確認しましょう

2. go get しましょう
GOPATH 以下に今日作業するディレクトリとサンプルリポジトリが作成されます

```
$ go get github.com/VG-Tech-Dojo/example/hello
$ hello
Hello, Go examples!
$ ls -la $GOPATH/src/github.com/VG-Tech-Dojo/
```

```
# helloが動かなかったらこのへんの設定(~/.bashrc とかに書いとくと良い)
$ export PATH=$PATH:$(go env GOPATH)/bin
```

Git, GitHub Tutorial

3. git clone をしましょう (gopath配下に持ってくる必要がある)

```
$ cd $GOPATH/src/github.com/VG-Tech-Dojo/
$ git clone git@github.com:VG-Tech-Dojo/vg-1day-2018-06-10.git
```

$GOPATH/src/github.com/VG-Tech-Dojo/vg-1day-2018-06-10 が今日の作業するディレクトリで

リポジトリが存在するか確認し、リポジトリのディレクトリに遷移しましょう

```
$ cd $GOPATH/src/github.com/VG-Tech-Dojo/vg-1day-2018-06-10
$ ls -la
```

おまけ
go get でリポジトリを取得してくることもできます

```
$ go get github.com/VG-Tech-Dojo/vg-1day-2018-06-10
```

Tutorial1−1の内容

Tutorial1-1:本日作業するディレクトリを作りましょう

issue -> branch -> 開発 -> push -> pull request

Tutorial1-1:本日作業するディレクトリを作りましょう

1. issueの作成
https://github.com/VG-Tech-Dojo/vg-1day-2018-06-10/issues から
「NEW issue」ボタンを押しissueを作成しましょう
titleは「tutorial-1−1-xxx」としましょう(xxxはニックネーム)

2. branchの作成
issueに対応したbranchを作成し遷移しましょう

```
$ git branch tutoria-l-1-xxx
$ git checkout tutorial-1-1-xxx
```

3. 確認
checkoutで指定したbranchにマークがされていればOKです

```
$ git branch
* tutorial_1-1_xxx
  master
```

Tutorial1-1:本日作業するディレクトリを作りましょう

4. セットアップ
originalのコピーとimport pathの修正

```
$ make setup/mac nickname=xxx
```

linuxの人は以下どちらか

```
$ make setup/bsd nickname=xxx
$ make setup/gnu nickname=xxx
```

5. 各自の作業用ディレクトリの確認
コピーしたディレクトリを確認して移動しましょう

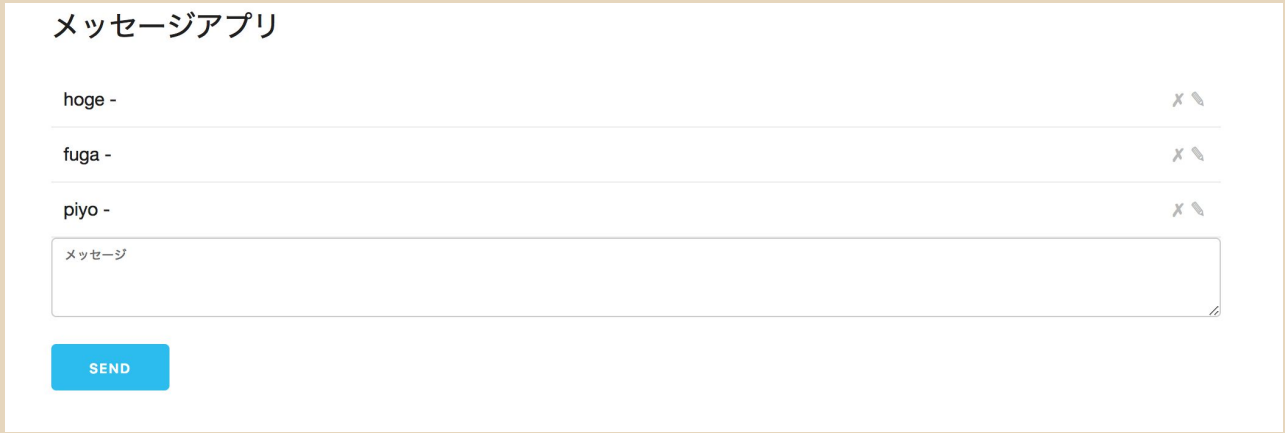
```
$ ls xxx
$ cd xxx
```

Tutorial1-1:本日作業するディレクトリを作りましょう

6. セットアップと動作確認 (初回はちょっと時間がかかります)

```
$ make deps
$ make run
```

ブラウザで http://localhost:8080 を開きメッセージアプリが表示されることを確認しましょう
(Listenし続けるため以降は別ターミナルで Tutorialを行います)



Tutorial1-1:本日作業するディレクトリを作りましょう

表示が確認できたら、pull requestを出して、自分でマージしましょう。

マージ後リモートリポジトリをローカルのmasterに取り込みましょう

よくあるエラー

```
vendor/gopkg.in/yaml.v2/encode_test.go:486: struct field tag `_` not compatible with reflect.StructTag.Get: bad syntax for struct tag pair
```

上記のようなエラーが（たくさん）出る

→ **go version を確認しましょう。 1.10.x より古い場合は新しい go をインストールしましょう。**

make deps に失敗する！

→ **GOPATH, PATH を確認しましょう**

→ **それでも解決できない場合は、Dockerを使ってみてください**

Tutorial1−1でやったこと

Tutorial1-1:本日作業するディレクトリを作りましょう

issue -> branch -> 開発 -> push -> pull request

Tutorial 1-2 ユーザ名を表示・追加しよう

1-2: 取得したユーザ名を画面で表示できるようにしましょう

1-2: メッセージの投稿時にユーザ名を DBに追加出来るようにしましょう

issueを立てて開発しましょう (tutorial-1-2-xxx)！

masterから今回のbranchを作成しましょう！

Server

ヒント表示: model/message.goのMessageAll を見てみよう

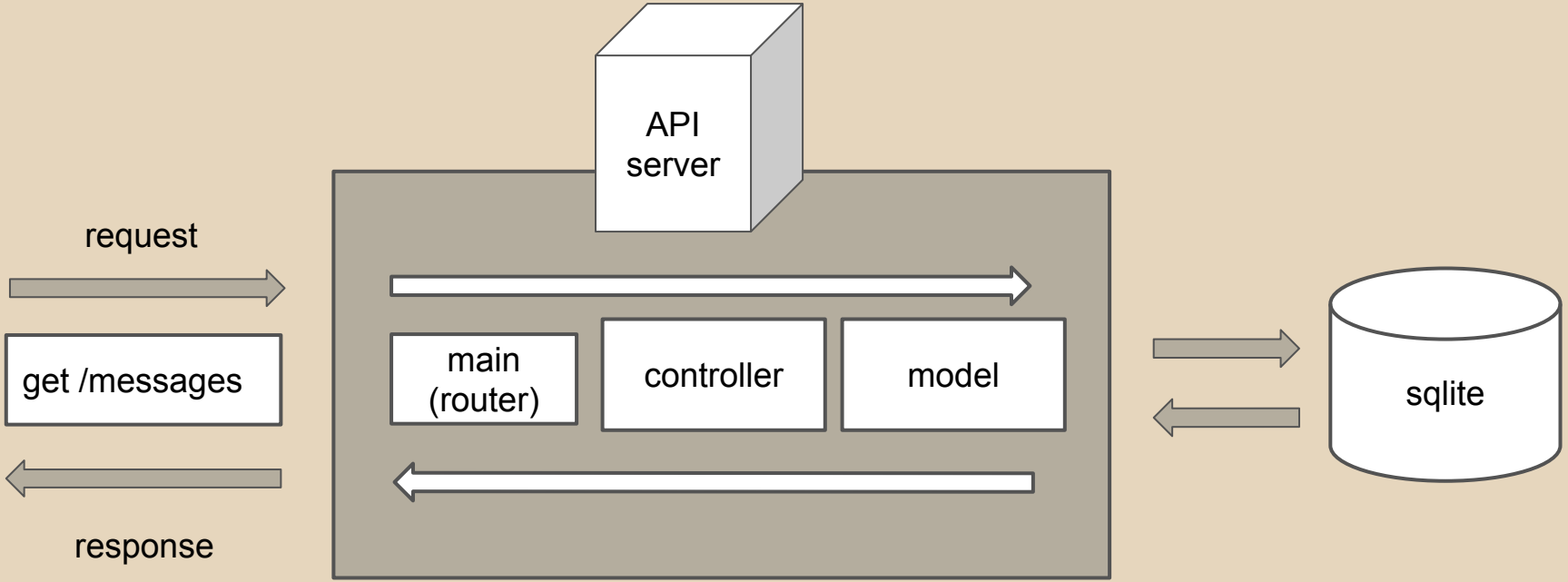
ヒント追加: model/message.goのInsertを見てみよう

Front

ヒント表示 : assets/js/app.jsのtemplateの部分を見てみよう

ヒント追加 : assets/js/app.jsのsendMessageを見てみよう

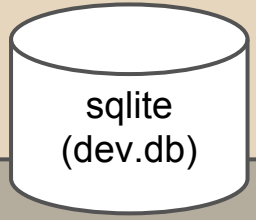
サンプルアプリの説明



API Server 内での一連の動作

Tutorial 1-2 ユーザ名を表示・追加しよう

サーバサイド
DBの内容



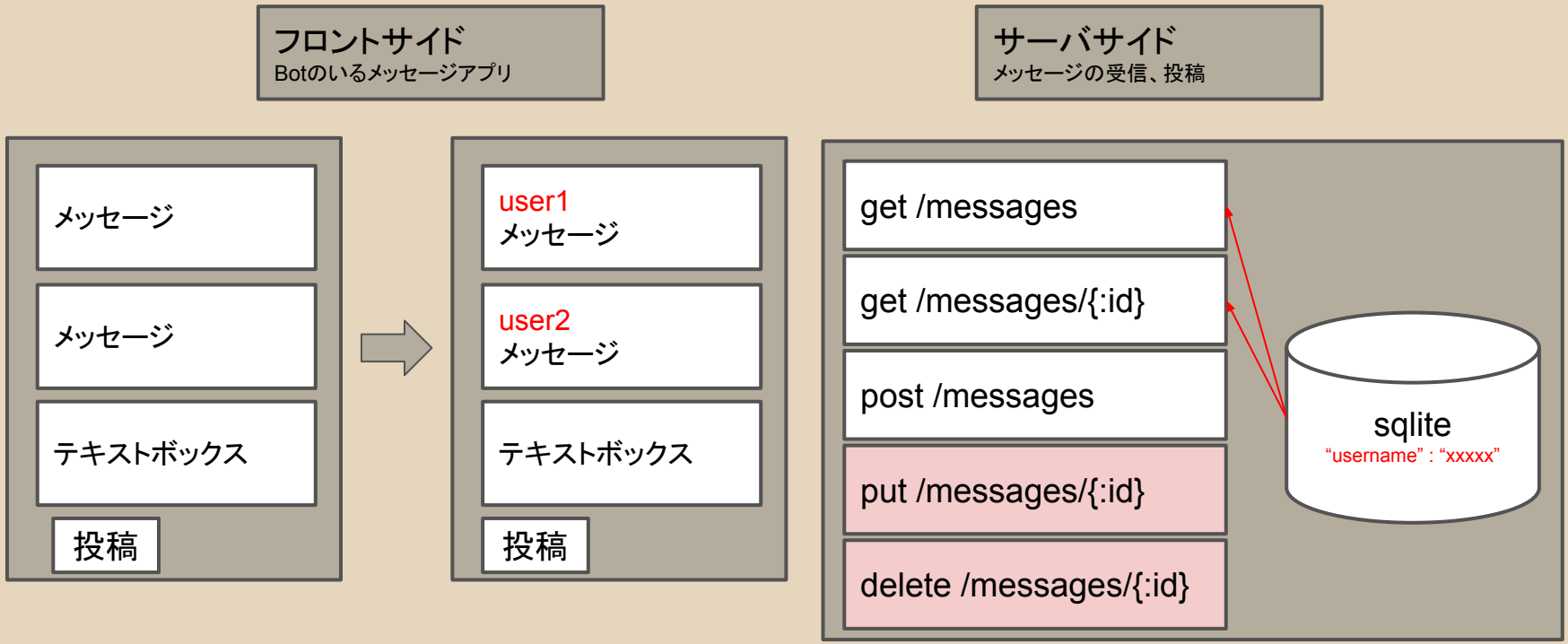
message テーブルの構造

```
CREATE TABLE message (
  id INTEGER NOT NULL PRIMARY KEY,
  body TEXT NOT NULL DEFAULT "",
  username TEXT NOT NULL DEFAULT "",
  created TIMESTAMP NOT NULL DEFAULT
            (DATETIME('now', 'localtime')),
  updated TIMESTAMP NOT NULL DEFAULT
            (DATETIME('now', 'localtime'));
```

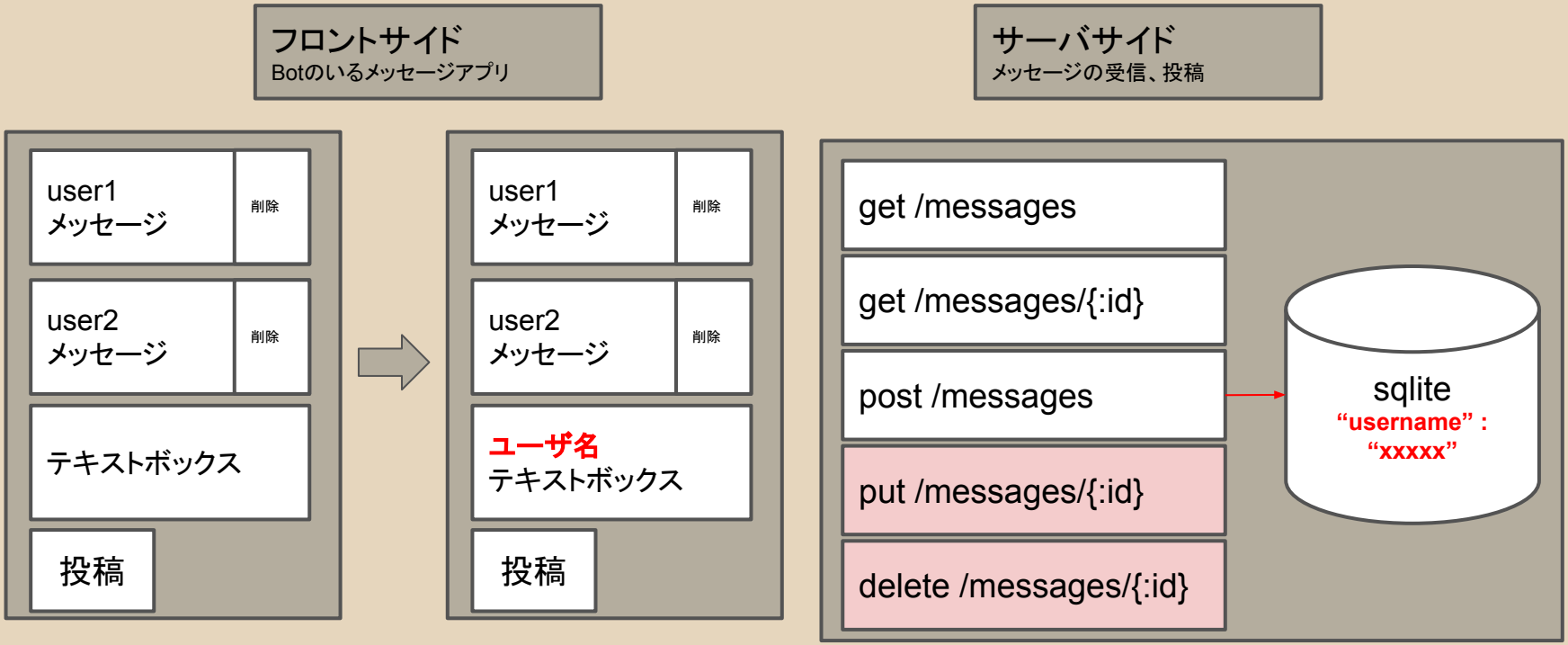
現在dev.dbに入っているデータ

```
sqlite> select id, body, username from message;

id|body|username
1 |hoge|sampleuser
2 |fuga |sampleuser
3 |piyo |sampleuser
```

Tutorial 1-2 ユーザ名を表示・追加しよう



Tutorial 1-2 ユーザ名を表示・追加しよう



Tutorial 1-2 ユーザ名を表示・追加しよう

1-2: 取得したユーザ名を画面で表示できるようにしましょう

1-2: メッセージの投稿時にユーザ名を DBに追加出来るようにしましょう

issueを立てて開発しましょう (tutorial-1-2-xxx)！

masterから今回のbranchを作成しましょう！

Server

ヒント表示: model/message.goのMessageAll を見てみよう

ヒント追加: model/message.goのInsertを見てみよう

Front

ヒント表示 : assets/js/app.jsのtemplateの部分を見てみよう

ヒント追加 : assets/js/app.jsのsendMessageを見てみよう

Tutorial 1-2 ユーザ名を表示・追加しよう

pull requestを出したら今回も自分でマージまで行いましょう

マージ後リモートリポジトリをローカルのmasterに取り込みましょう

```
$ git checkout master
$ git branch
$ git fetch origin master
$ git merge FETCH_HEAD
```

エラーとならず取り込めたことを確認

チートシート

Git＆GitHubオペレーション

Git & GitHubオペレーション

1. gitでの確認

```
$ git status
```

2. add（後ろのドット(.) を忘れないように）

```
$ git add .
```

3. commit

```
$ git commit -m "メッセージ"
```

4. push

リモートリポジトリにpushしましょう

```
$ git push origin tutorial_1-1_xxx
```

5. ブラウザで自分のpushしたbranchを確認しましょう

Git＆GitHubオペレーション

Git & GitHubオペレーション

6. pull request

https://github.com/VG-Tech-Dojo/vg-1day-2018-06-10/pulls

「New pull request」を押して pull request を作成しましょう

7.セルフマージ

pull request が作成されたら、自分で merge をしましょう

8. merge後のローカルリポジトリへの取り込み

```
$ git checkout master
$ git fetch origin master
$ git merge FETCH_HEAD
```

9. master branchでの自己紹介の動作確認

ブラウザで http://localhost:8080/xxx を表示してみましょう

他の人の自己紹介を表示するにはどうしたら良いか考えてみましょう

サンプルアプリの簡単な補足

大前提 vg-1day-2018-05-13/xxx にいること

サービスの起動 (停止する場合はctrl + C)

```
$ make run
```

APIリクエスト

```
$ make curl_ping
$ make curl_messages_get_all
$ make curl_messages_get ID=
$ make curl_message_post BODY=
$ make curl_message_put ID= BODY=
$ make curl_message_delete ID=
```

DB操作

```
$ sqlite3 dev.db
sqlite> .help
```

Mission1

golangでAPIを実装する

午後のMissionに入る前に

本日は各Missionを行う際は
午前中に習った
issue -> branch -> 開発 -> push -> pull request
のサイクルで開発します

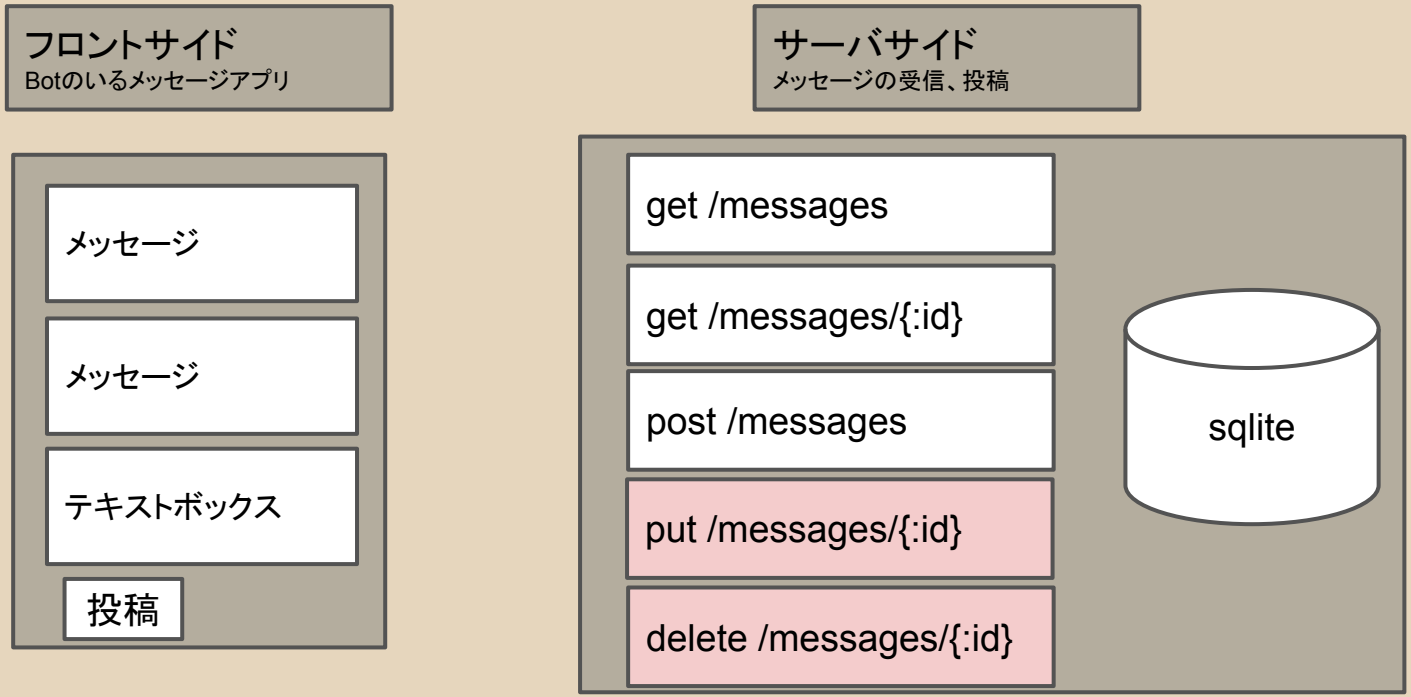
Mission 1 の概要

- Web API とは
- サンプルアプリの説明
- APIの実装（Mission1 Start!!）

Web API とは

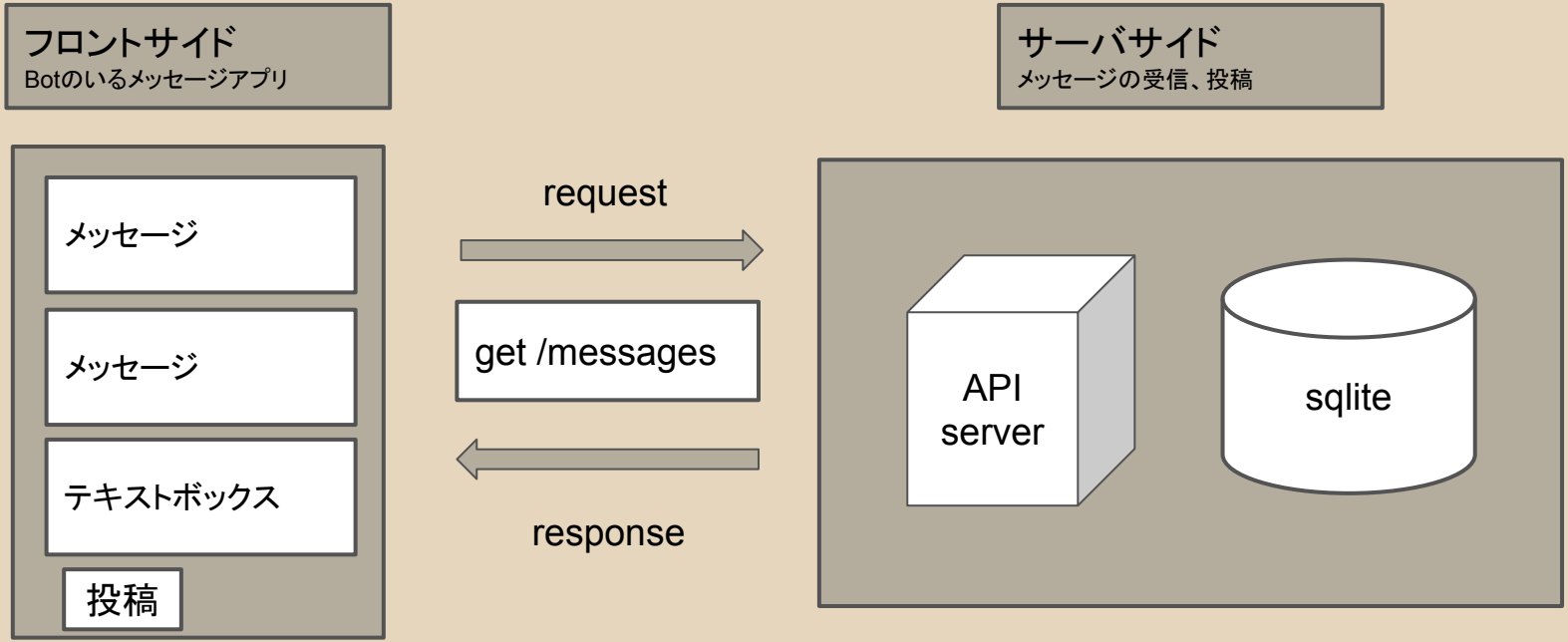
- API
  - Application Programing Interface
  - アプリケーションが何らかの処理を行うための手段や決まりごと
- Web API
  - HTTPプロトコルを用いたAPI

サンプルアプリの説明

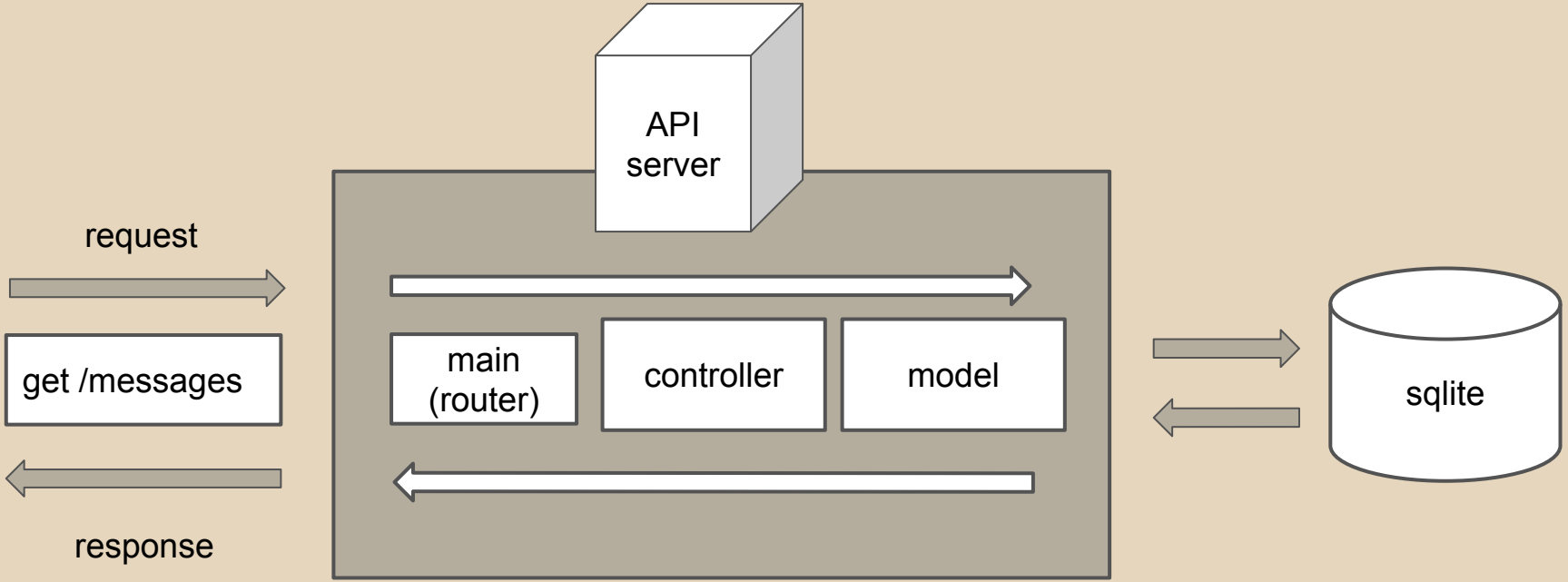

フロントサイド
Botのいるメッセージアプリ
メッセージ
メッセージ
テキストボックス
投稿
サーバサイド
メッセージの受信、投稿
get /messages
get /messages/{:id}
post /messages
put /messages/{:id}
delete /messages/{:id}
sqlite

サンプルアプリの説明

$ ***make run***
$ ***make curl_messages_get_all*** を実行してみよう



サンプルアプリの説明



API Server 内での一連の動作

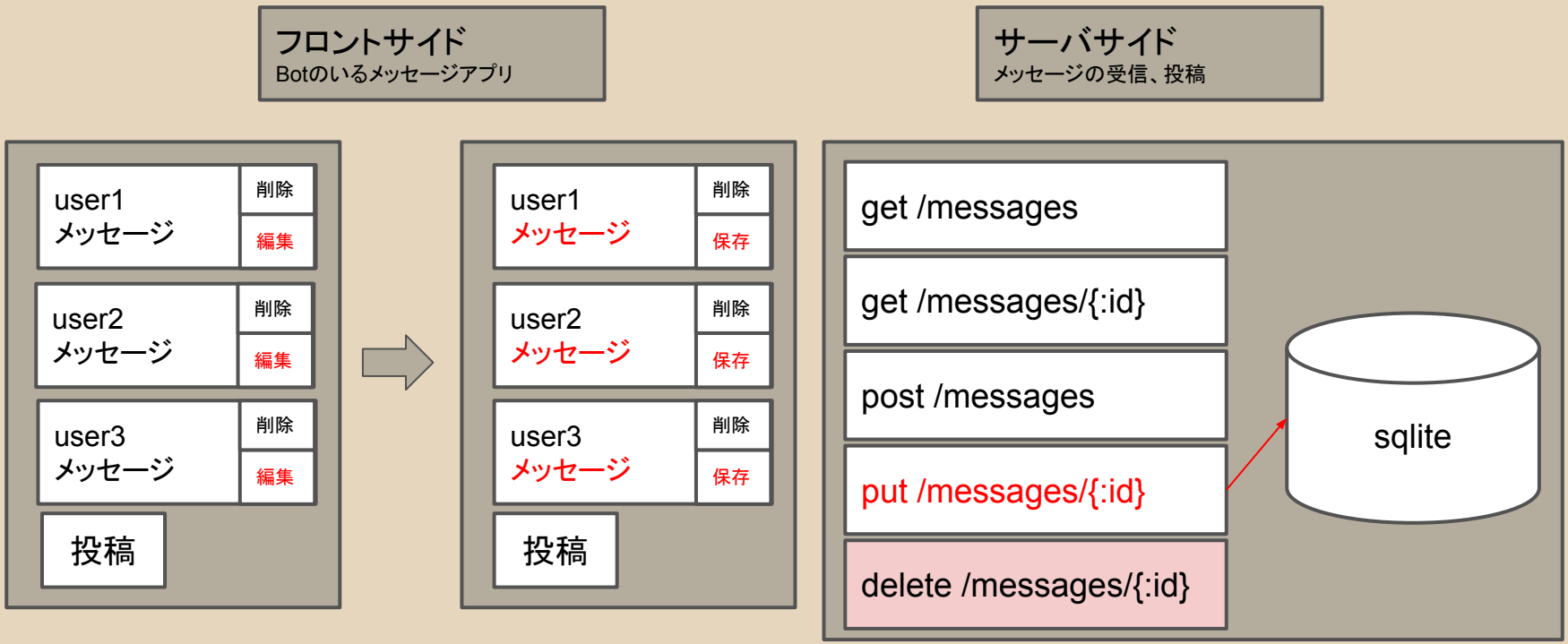
サンプルアプリの構造

```
├── server.go
├── model
│   └── message.go
├── db
│   └── db.go
├── controller
│   └── message.go
├── templates
│   └── index.html
├── assets
│   └── js
│       └── app.js
```

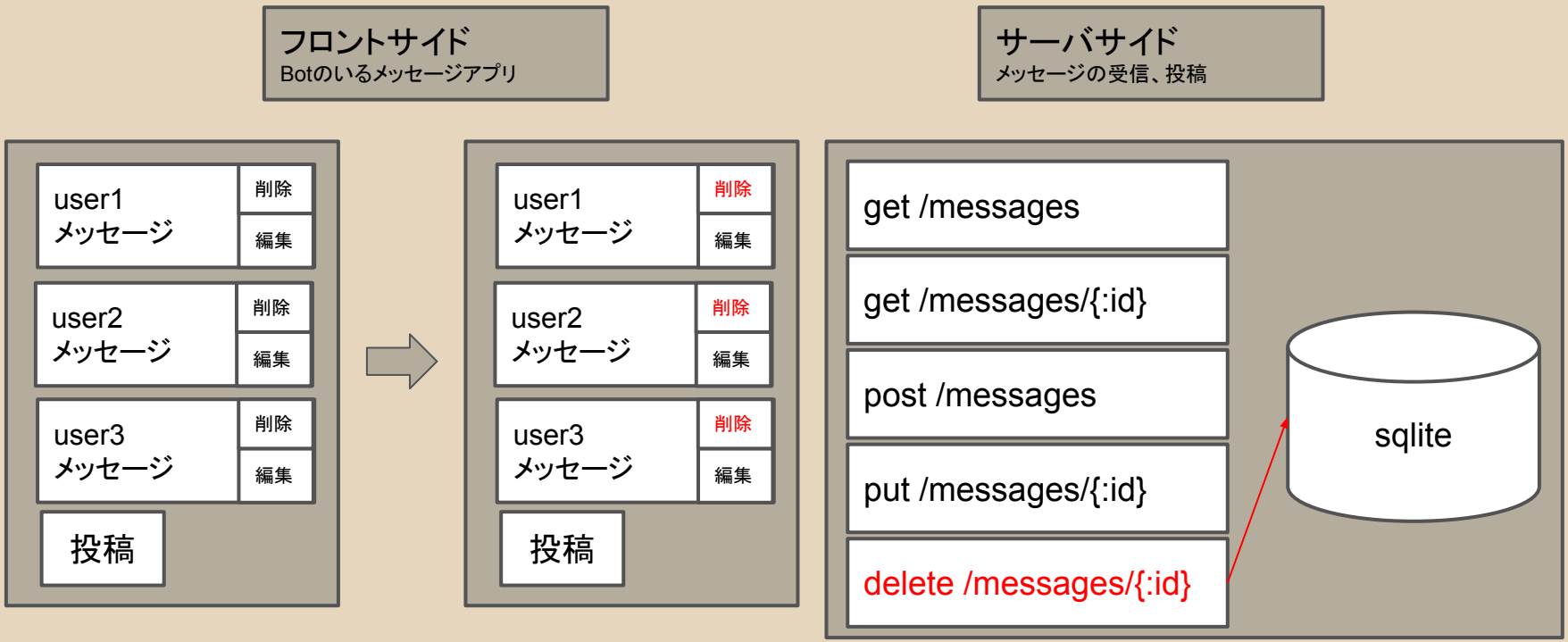
Mission1 の内容

- Mission1-1 : メッセージを編集しよう
- Mission1-2 : メッセージを削除しよう

Mission 1-1 メッセージを編集しよう



Mission 1-2 メッセージを削除しよう



Mission 1-1,1-2 メッセージを編集・削除しよう

1-1: メッセージの編集機能を追加しましょう。

今回はどのユーザの投稿でも編集できることにします。

1-2: メッセージの削除機能を追加しましょう。

今回はどのユーザの投稿でも削除ボタンを押したら削除できることにします。

issueを立てて開発しましょう (mission-1-1-xxx, mission-1-2-xxx)！
masterから今回のbranchを作成しましょう！

Server

ヒント: select, insertの実装内容を参考に実装してみましょう
modelに新しく実装を追加していく必要があります

Mission 1-1,1-2 メッセージを編集・削除しよう

pull requestを出したら今回は自分でマージまで行いましょう

マージ後リモートリポジトリをローカルのmasterに取り込みましょう

```
$ git checkout master
$ git branch
$ git fetch origin master
$ git merge FETCH_HEAD
```

エラーとならず取り込めたことを確認

回答サンプル：PRリンク

Mission1-1，1-2 の答え合わせ

答えのサンプルPRを作成したので
PRを見て下さい

何か指摘事項があればコメント下さい

Mission 1 の振り返り

- サンプルアプリを通してAPIを実装
  - git, githubでの開発を体験
  - golangを用いてAPI開発を体験
    - model, controller

Mission2

Botを作ろう！
goroutine！！！

100分

Mission2 でやること

- テーマ
  - goroutineとchannelを使ってみる
  - 外部APIも叩いてみる
- やること
  - メッセージアプリにbotを追加します

Bot とは

ボット(ボット)とは - コトバンク

- インターネット上の操作を自動でするプログラムのこと。
- 《「ロボット」の略》コンピューターで、人の代わりに自動的に実行するプログラムの総称。
- 特定の命令に従って自動的に作業を行う自動化プログラム。

今日，東京
に反応して

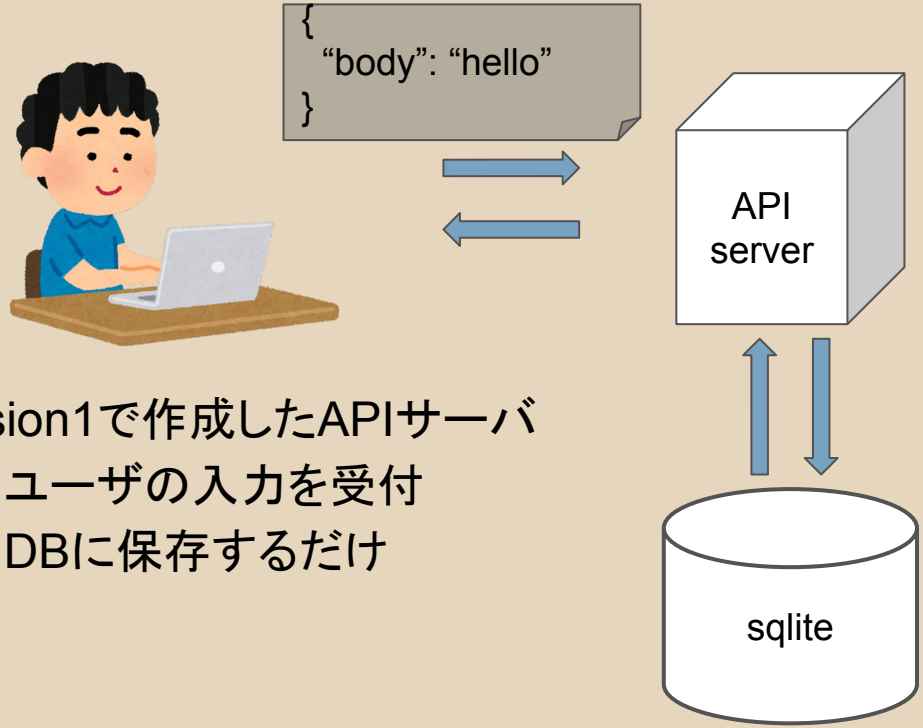
Bot とは

- 今回の文脈では、特定のキーワードに反応して
  何か処理をするプログラム
- メッセージアプリにサンプルが組み込んであるので触ってみよう

Botをさわってみる

「hello」と投稿する

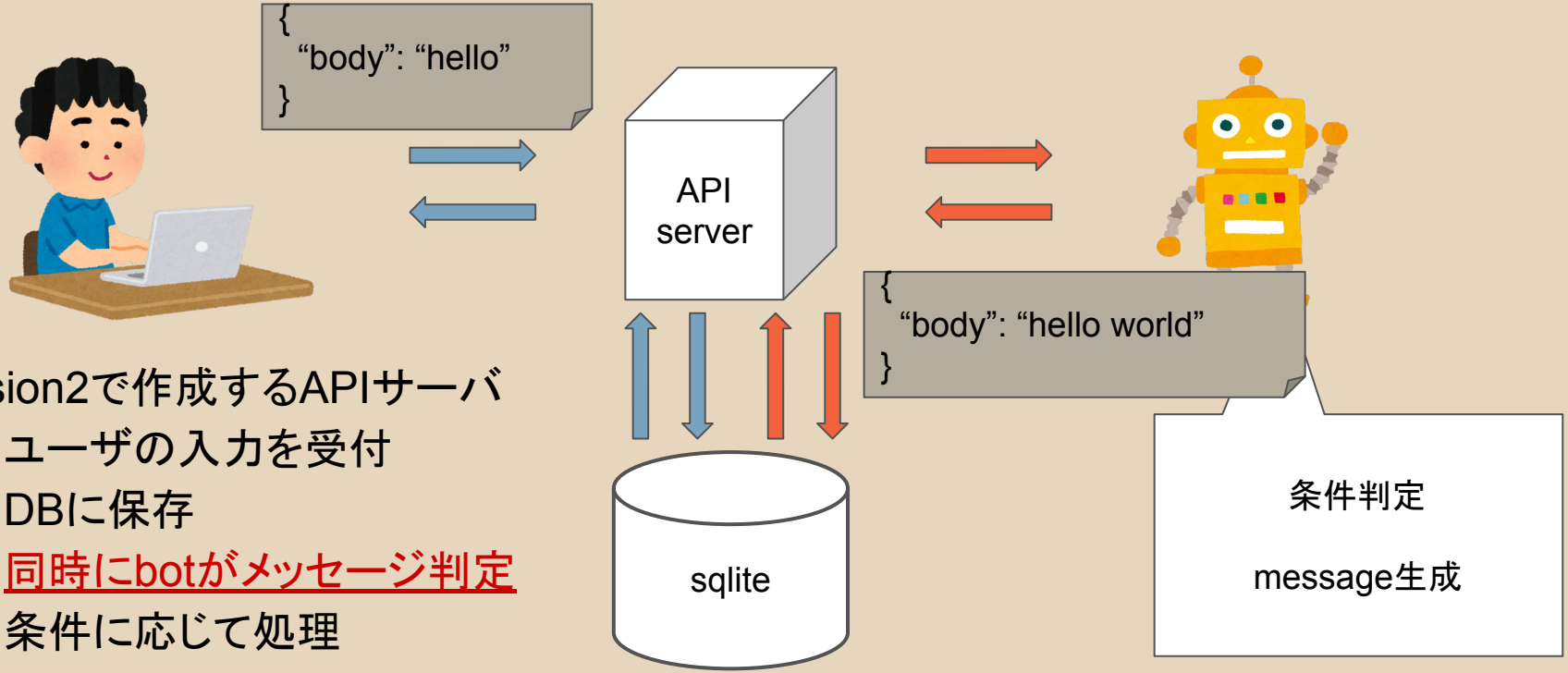
Botシステム構成



Mission1で作成したAPIサーバ

- ユーザの入力を受付
- DBに保存するだけ

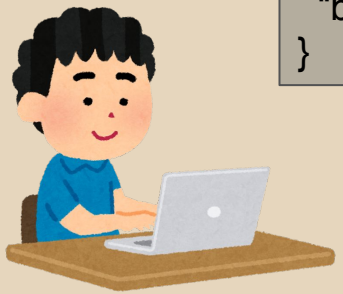
Botシステム構成

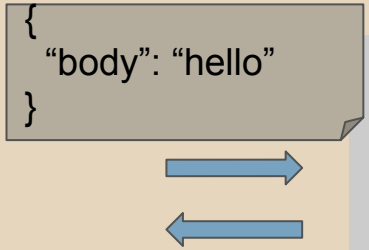


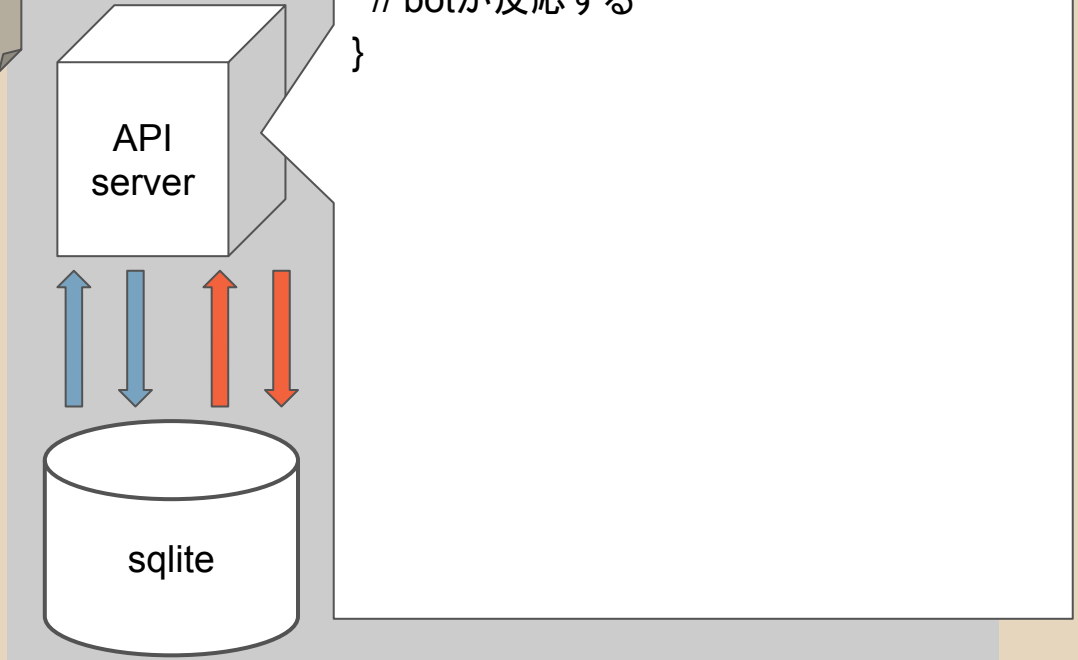
Mission2で作成するAPIサーバ

- ユーザの入力を受付
- DBに保存
- 同時にbotがメッセージ判定
- 条件に応じて処理

どう実現するか

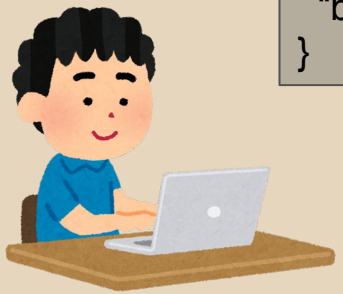






- controller にif文追加すればできそう

どう実現するか



```
{
  “body”: “hello”
}
```

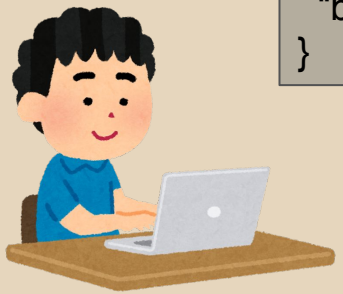
API server

sqlite

```
if message is hoge {
  // botが反応する
}
if message is fuga {
  // 別のbotが反応する
}
if message is piyo {
  // さらに別のbotが反応する
}

...
```

- botもっと増やしたいなー

どう実現するか



```
{
  “body”: “hello”
}
```

API server

sqlite

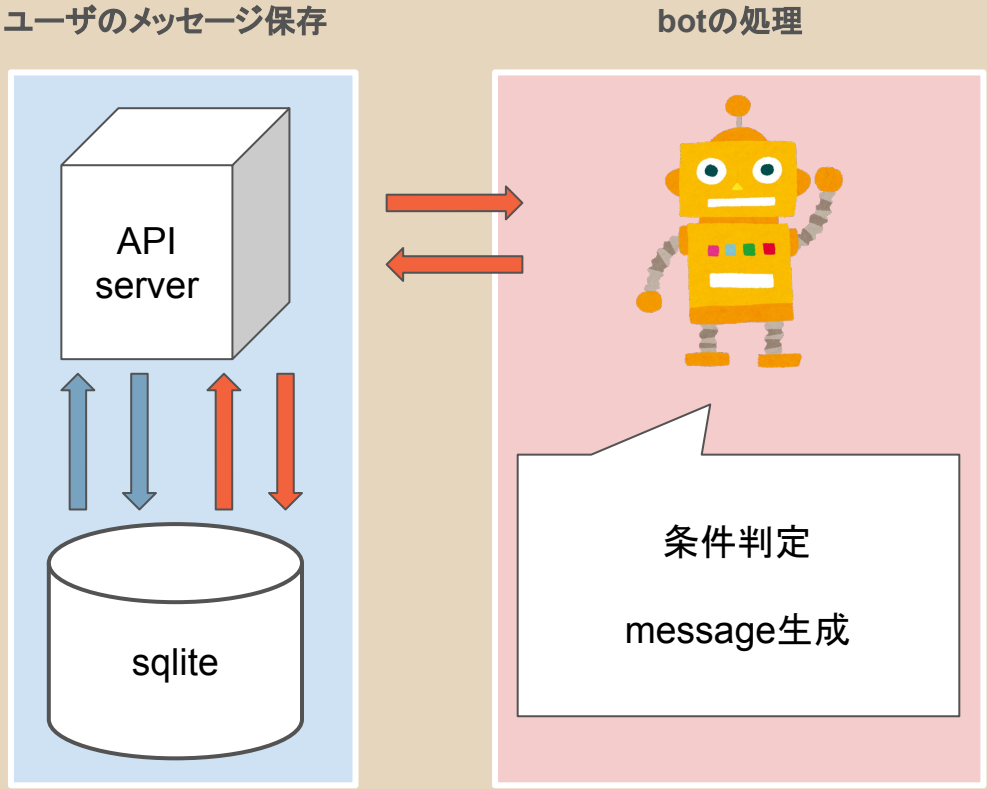
```
if message is hoge {
  // botが反応する
}
if message is fuga {
  // 別のbotが反応する
}
if message is piyo {
  // さらに別のbotが反応する
  // 処理に1秒以上かかる
}

...
```

- botに複雑な処理させたいなー
- botの処理に1秒かかると、
  ユーザーにレスポンスを返すのが
  それだけ遅くなる

どう実現するか

そこで，ユーザのメッセージ保存と
botの処理を別々に走らせます

これによりbot処理の終了を待たずに
ユーザにレスポンスを返せます



goroutine

- goroutineは、Goのランタイムに管理される軽量スレッド
  - https://tour.golang.org/concurrency/1
- Concurrency(並行性)を実現するための仕組み

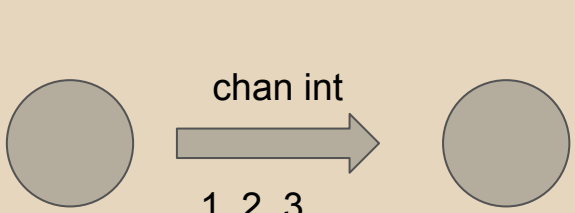
```
func say(s string) {
	for i := 0; i < 5; i++ {
		time.Sleep(100 * time.Millisecond)
		fmt.Println(s)
	}
}

func main() {
	say("world")
	say("hello")
}
```

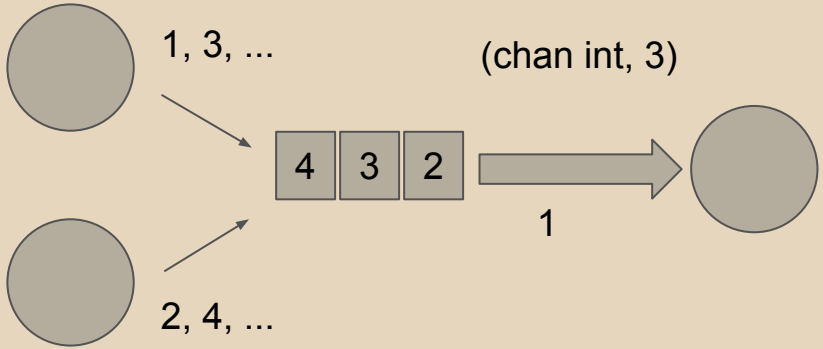
```
func say(s string) {
	for i := 0; i < 5; i++ {
		time.Sleep(100 * time.Millisecond)
		fmt.Println(s)
	}
}

func main() {
	go say("world")
	say("hello")
}
```

channel

goroutine間でデータのやりとりをするしくみ

- goroutine同士をつなぐパイプのようなもの
- キューとして複数データを保持することもできる





bot package 解説

bot package

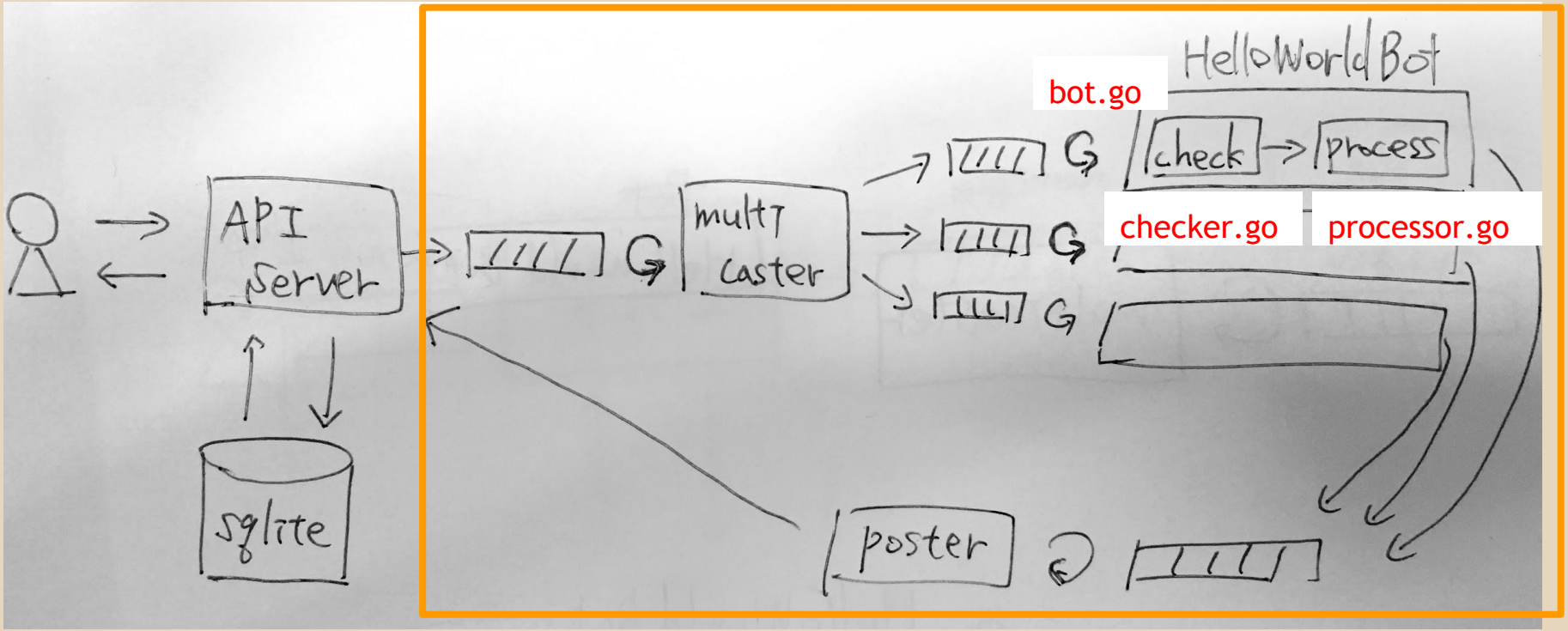
bot

```
├── bot.go：bot生成
├── multicaster.go：multicasterの生成/実装（触らないでok）
├── checker.go：botの反応条件ロジック
├── poster.go：posterの生成/実装（触らないでok）
├── processor.go：botのメッセージ生成ロジック
└── util.go
```

赤字部分を触ればbotを作れます

＊server.go にて botを追加することを忘れずに！

Botが投稿するまでの流れ



bot package（再掲）

bot package

bot

├── bot.go：bot生成
├── broadcaster.go：broadcasterの生成/実装（触らないでok）
├── checker.go：botの反応条件ロジック
├── poster.go：posterの生成/実装（触らないでok）
├── processor.go：botのメッセージ生成ロジック
└── util.go

赤字部分を触ればbotを作れます

＊server.goにて botをbroadcasterに登録することを忘れずに！

Mission2-1: ガチャBotをつくる

「gacha」とpostしたら「SSレア」「Sレア」「レア」「ノーマル」の文字列をランダムで投稿するBotを実装してください。

作業のヒント

bot.go：func NewGachaBot(out chan *model.Message) *Bot 関数を作って
ガチャBotを追加しましょう

processor.go：func (*GachaProcesser) Process() メソッドを作って
ガチャロジックを実装しましょう

Mission2-1: ガチャBotをつくる

issueを立てて開発しましょう(mission2-1)！

masterから今回のbranchを作成しましょう！

Mission2-1の解説

さらにBotらしく

- 外部サービスのAPIを使ってみる
  - 色々なWebサービスがデータや機能をAPIとして公開してくれています。
  - それらを使うことで、複雑な機能を、自分で開発しなくても利用することができます
    - 例：天気の取得、AIとの対話、...

サンプルで用意したBot説明

Yahoo! JAPANのキーフレーズ抽出APIを利用したBot

Yahoo! デベロッパーネットワーク https://developer.yahoo.co.jp/webapi/jlp/keyphrase/v1/extract.html

- 日本語文を解析し、特徴的な表現（キーフレーズ）を抽出
- env/env.goのKeywordAPIAppIDにアプリケーションIDを代入

準備ができたら「keyword hogehoge」を投稿してみましょう

大事なこと

API Key, API Token...等々 は
悪意のある第三者に漏れると悪用される恐れもあります

絶対にパブリックに見える場所に貼ってはいけません

よくあるチョンボ

- 間違えてcommitしてGitHubの公開リポジトリにpush

大事なことなので2回

絶対にパブリックに見える場所に貼ってはいけません

サンプルで用意したBot説明

- APIにリクエスト投げている箇所
  - bot/processor.go の func (p *KeywordProcessor) Process

- APIキーを管理している箇所
  - env/env.go(.gitignore済み）
  - env.goがない場合は
    cp env.go.tmpl env.go

Mission2-2: ChatBotをつくる

「talk 任意のメッセージ」をpostしたら返答するChatBotをつくってください

- やはりBotには意思をもっている風に喋ってほしいですね
- 今回はChatBotの機能を作っている時間もないので、Recruitが提供してくれているTalkAPIを使わせてもらいます。
- **(gmailの場合「+」は使わないようにしましょう)**
  Recruit A3RT TalkAPI https://a3rt.recruit-tech.co.jp/product/talkAPI/

ヒント：なし！がんばりましょう。

（ヒント欲しい～という人は周りのサポーターに相談）

Mission2-2: ChatBotをつくる

issueを立てて開発しましょう(mission2-2)！

masterから今回のbranchを作成しましょう！

Mission2-2で詰まったら

- APIから結果が返ってきてるのか分からない
  - curlで投げてみる
- APIからは取れてるみたいだけどレスポンスに何が入ってるか分からない
  - curlで...
  - fmt.Printf(“%#v”, 変数名) で変数の中身をデバッグできます
- JSONがうまくデコードできない
  - 「golang json struct」とか「golang json decode」とかでggる
- それ以外
  - ggる

Mission2-2の解説

Mission2-3 ex: APIを探す

- 現在、色々な機能、情報をもったAPIを各社提供してくれています
- 駅データ、天気、書誌情報など、データはあるが、そのままだと利用できないものをAPIとしてデータ提供してくれているところも多々あります
- 利用しにくいデータを利用しやすいように提供するのは、新しい価値の創出であり、Webサービスの一歩目とも言えるでしょう
- そのようなAPIを探して、時間があれば、Botに組み込んでみましょう

まとめ

- テーマ
  - goroutineとchannelを使ってみた
  - 外部APIも叩いてみた
- やること
  - メッセージアプリにbotを追加します

まとめ

- goroutine, channelを使った並行処理
- 外部APIを使ったおもしろ機能の開発

Mission3

チーム開発

Mission3 自由課題（チーム開発）

作成したアプリをより充実させ自分らしいアプリにしましょう。

Mission3 自由課題（チーム開発）

- Mission3の流れ
  - 準備
    - ローカルにこれまでの VG-Tech-Dojoのリポジトリの変更の取り込み
      - $ git fetch origin master
      - $ git merge FETCH_HEAD
    - チームごとに作業ディレクトリを用意して開発してください
      - 用意したら、PRを出してください
    - 仕様決め & issueに仕様の記載 20分
  - 開発
    - Mission3のbranchを切り各チーム開発を行う
      - 前半開発 40分
        - masterに対して[WIP]のPR（※WIPとは開発途中と言う意味です）
      - 前半振り返り/後半やること 10分
      - 後半開発 40分

各チーム発表、振り返り

- 各チームごとに制作物の発表
- 講師、サポータから講評