



Road to DMM

Bootcamp Go

Summary: This document is the subject for the Bootcamp Go of the Road to DMM @ 42 Tokyo.

Contents

I	Mission	2
II	Instruction	3
III	Vocabulary	4
IV	Development Environment	5
IV.1	Requirements	5
IV.2	Start	5
IV.3	Shutdown	5
IV.4	Log	5
IV.5	Hot Reload	5
IV.6	Swagger UI	6
IV.7	Test	6
IV.8	Authentication	6
IV.9	DB	6
V	Code	7
V.1	Architecture	7
V.1.1	app	7
V.1.2	config	7
V.1.3	domain	7
V.1.4	handler	8
V.1.5	dao	8
V.2	Module Dependency	8
V.3	Library	8
V.4	Utilities	8
V.4.1	app/handler/request	9
V.4.2	app/handler/httperror	9
V.4.3	app/handler/auth	9
VI	Exercise 00 : リポジトリの準備	10
VII	Exercise 01 : アカウント作成APIの実装	11
VIII	Exercise 02 : アカウント情報取得APIの実装	12
IX	Exercise 03 : ステータス投稿APIの実装	13
X	Exercise 04 : ステータス取得APIの実装	14
XI	Exercise 05 : ステータス削除APIの実装	15

XII	Exercise 06 : パブリックタイムライン取得APIの実装	16
XIII	Bonus 01: ユニットテストの追加	17
XIV	Bonus 02: フォロー関連機能の実装	18
XV	Bonus 03: media機能の実装	19

Chapter I

Mission

In this MISSION, you will commit to an exercise for DMM's Go new graduate training.

you will implement Yatter, a virtual Twitter/Mastodon-like service.

You will implement Yatter's backend API in this MISSION.

<https://github.com/dmm-com/mission-dmm-bootcamp-go>

Chapter II

Instruction

- Only this page will serve as reference; do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We **will not** take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet /`
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- If no other explicit information is displayed, you must use this versions of languages : `Go - 1.16.X`.

Chapter III

Vocabulary

- status : post to a sns
 - follow : Subscribing to another account's status
 - following : Accounts that an account follows
 - follower : Accounts that are following an account
- timeline : A collection of statuses arranged in chronological order
 - public timeline : 全アカウントのstatusが集まるtimeline
 - home timeline : followしているアカウントのstatusが集まるtimeline

Chapter IV

Development Environment

開発環境をdocker-composeで構築しています。

IV.1 Requirements

- Go
- docker / docker-compose

IV.2 Start

```
docker-compose up -d
```

IV.3 Shutdown

```
docker-compose down
```

IV.4 Log

ログの確認

```
docker-compose logs
```

ストリーミング

```
docker-compose logs -f
```

webサーバonly

```
docker-compose logs web  
docker-compose logs -f web
```

IV.5 Hot Reload

[air](#)によるホットリロードをサポートしており、コードを編集・保存すると自動で反映されます。読み込まれない場合は`docker-compose restart`を実行してください。

IV.6 Swagger UI

API仕様をSwagger UIで確認できます。開発環境を立ち上げ、Webブラウザでlocalhost:8081にアクセスしてください。

IV.7 Test

各API定義の"Try it out"からAPIの動作確認を行うことができます。

IV.8 Authentication

鍵マークのついたエンドポイントは認証付きエンドポイントです。AuthenticationというHTTPヘッダにusername \$ユーザー名を指定する単純な仕様です。動作確認の際には画面上部の"Authorize"からヘッダの値の設定を行ってください。

IV.9 DB

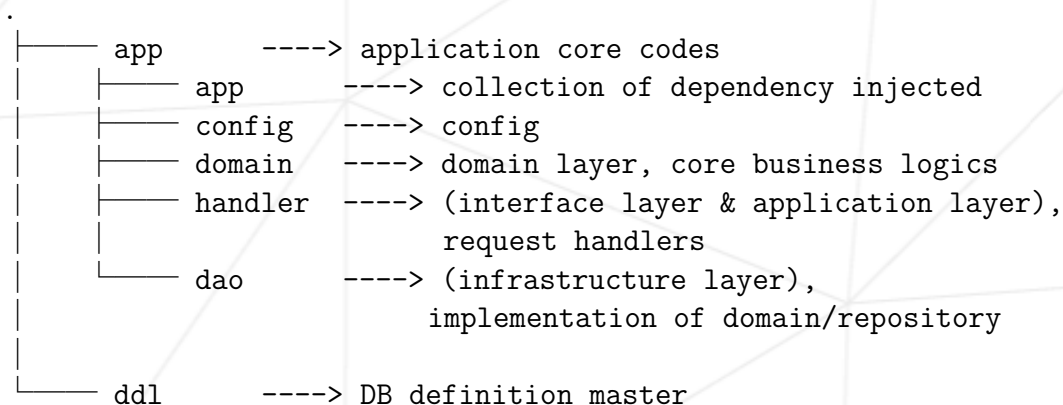
マイグレーションツールの用意はありません。初回起動時にddl/以下にあるSQLファイルが実行されます。再読み込みの際は.data/mysql/を削除し、DBを初期化してください。

```
docker-compose down
rm -rfd .data/mysql
docker-compose up -d
```


Chapter V

Code

V.1 Architecture



V.1.1 app

モジュールの依存関係を整理するパッケージで、DIコンテナを扱います。今回は簡素なものになっていて、DAOの組み立てとhandlerのDAO（が提供するdomain/repository）への依存の管理のみ行っています。

V.1.2 config

サーバーの設定をまとめたパッケージです。DBやlistenするポートなどの設定を取得するAPIがまとめてあります。

V.1.3 domain

アプリケーションのモデルを扱うdomain層のパッケージです。

domain/object

ドメインに登場するデータ・モノの表現やその振る舞いを記述するパッケージです。今回は簡単のためDTOの役割も兼ねています。

domain/repository

domain/objectで扱うEntityの永続化に関する処理を抽象化し、インターフェースとして定義するパッケージです。具体的な実装はdaoパッケージで行います。

V.1.4 handler

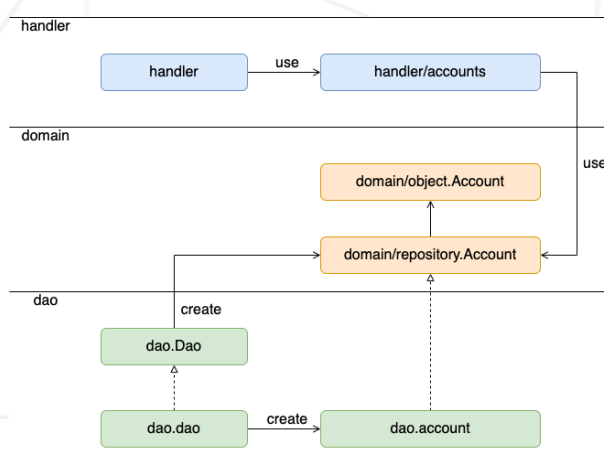
HTTPリクエストのハンドラを実装するパッケージです。リクエストからパラメータを読み取り、エンドポイントに応じた処理を行ってレスポンスを返します。機能の提供のために必要になる各種処理の実装は別のパッケージに切り分け、handlerは入出力に注力するケースも多いですが、今回は簡単のため統合しています。

V.1.5 dao

domain/repositoryに対する実装を提供するパッケージです。DBなど外部モジュールへアクセスし、データの保存・取得・更新などの処理を実装します。

V.2 Module Dependency

モジュールの依存関係



V.3 Library

- HTTP
 - chi ([ドキュメント](#))
- DB
 - sqlx ([ドキュメント](#))

V.4 Utilities

このテンプレートでは実装をサポートするユーティリティ関数を提供しています。

V.4.1 app/handler/request

リクエストの扱いに関するユーティリティをまとめています。テンプレートにはパスパラメータidの読み取りを補助する関数IDOfを用意しています。

```
// var r *http.Mux
r.Get("/{id}", func(w http.ResponseWriter, r *http.Request){
    id, err := request.IDOf(r)
    ...
})
```

V.4.2 app/handler/httperror

エラーレスポンスを返すためのユーティリティをまとめています。

```
func SomeHandler(w http.ResponseWriter, r *http.Request) {
    ...
    if err != nil {
        httperror.InternalServerError(w, err)
    }
    return
}
...
}
```

V.4.3 app/handler/auth

認証付きエンドポイントの実装のためのミドルウェア関数を提供しています。
chi.Mux#Useやchi.Mux#Withを用いて利用できます。


- [chiドキュメント](#)

ミドルウェアを埋め込んだエンドポイントでは*http.RequestからAccountOfでリクエストと紐づくアカウントを取得できます。

```
// var r *http.Request
account := auth.AccountOf(r)
```

Chapter VI

Exercise 00 : リポジトリの準備

	Exercise 00
リポジトリの準備	
提出するディレクトリ : <i>ex00/</i>	
提出するファイル : *	


こちらの[リンク](#)からリポジトリをクローンして、課題のTurn-in Repositoryに提出すること。

リポジトリをプッシュした後に、上記の項目を一読し、以下を確認すること。

- 開発環境の起動の仕方
- Swagger UIの動作
- コードのアーキテクチャ

Chapter VII

Exercise 01 : アカウント作成APIの実装

	Exercise 01
アカウント作成APIの実装	
提出するディレクトリ : <i>ex01/</i>	
提出するファイル : *	

以下のエンドポイントを実装しましょう。

- POST /v1/accounts

エンドポイントの実装には以下が必要になります。

- 必要な MySQL テーブルの定義 (ddl/ddl.sql)
- app/domain/repository,objectの用意
- app/daoの実装
- app/handlerの実装
- app/handler/router.goへの登録

このエンドポイントについてはテンプレートに以下が用意されています。


- MySQLテーブルの定義
- アカウントオブジェクトの定義 (app/domain/object)
- POST /accountsのハンドラ雛形

残りの部分について作業を行なっていきましょう。

なお、imageやfollowに関するフィールド・仕様はここでは無視してOKです。

Chapter VIII

Exercise 02 : アカウント情報取得APIの実装


	Exercise 02
アカウント情報取得APIの実装	
提出するディレクトリ : <i>ex02/</i>	
提出するファイル : *	

STEP2を参考に以下のエンドポイントを実装しましょう。

- GET /v1/accounts/username

Chapter IX

Exercise 03 : ステータス投稿APIの実装

	Exercise 03
ステータス投稿APIの実装	
提出するディレクトリ : <i>ex03/</i>	
提出するファイル : *	


以下のエンドポイントを実装しましょう。

- POST /v1/statuses

mediaに関するフィールド・仕様はここでは無視してOKです。

Chapter X

Exercise 04 : ステータス取得APIの実装


	Exercise 04
ステータス取得APIの実装	
提出するディレクトリ : <i>ex04/</i>	
提出するファイル : *	

以下のエンドポイントを実装しましょう。

- GET /v1/statuses/id

Chapter XI

Exercise 05 : ステータス削除APIの実装


	Exercise 05
ステータス削除APIの実装	
提出するディレクトリ : <i>ex05/</i>	
提出するファイル : *	

以下のエンドポイントを実装しましょう。

- DELETE /statuses/id

Chapter XII

Exercise 06 : パブリックタイムライン 取得APIの実装


	Exercise 06
パブリックタイムライン取得APIの実装	
提出するディレクトリ : <i>ex06/</i>	
提出するファイル : *	

以下のエンドポイントを実装しましょう。

- GET /v1/timelines/public

Chapter XIII


Bonus 01: ユニットテストの追加

	Exercise 07
Bonus 01: ユニットテストの追加	
提出するディレクトリ : <i>ex07/</i>	
提出するファイル : *	

handlerやdaoの実装などにユニットテストを追加してみましょう。 HTTPテストやDB mockの仕組みも考えてみてください。

Chapter XIV

Bonus 02: フォロー関連機能の実装


	Exercise 08
Bonus 02: フォロー関連機能の実装	
提出するディレクトリ : <i>ex08/</i>	
提出するファイル : *	

アカウントのフォローに関連するAPIを実装してみましょう。DBスキーマも設計して拡張してください。

- アカウントのfollow : POST `/accounts/username/follow`
- following一覧取得 : GET `/accounts/username/following`
- follower一覧取得 : GET `/accounts/username/followers`
- アカウントのunfollow : POST `/accounts/username/unfollow`
- アカウントとのrelation取得 : GET `/accounts/relationships`
- home timeline取得 : GET `/timelines/home`
- アカウントのfollowing・follower数の取得

Chapter XV

Bonus 03: media機能の実装

	Exercise 09
Bonus 03: media機能の実装	
提出するディレクトリ : <i>ex09/</i>	
提出するファイル : *	

ステータスやアカウントで画像を扱えるようにしてみましょう。DBスキーマの設計・拡張や画像の管理の仕組みも必要になります。

- media投稿APIの実装 : POST /media
- アカウント情報更新APIの実装 : POST /accounts/update_credentials
- アカウント・status関連APIの編集（必要に応じて）