バックエンド第3回 更新・削除・登録の実装

更新機能仕様

RESTful API(Controller)仕様

Class名：ItemController

機能	メソッド	URL	method名	レスポンスボディ
1件更新	PUT	/api/items/{id}	updateByKey(id)	void
1件削除	DELETE	/api/items/{id}	deleteByKey(id)	void
新規登録	POST	/api/items/	createNew(param)	int id

REPOSITORY仕様

Interface名：ItemDao

No.	戻り値型	メソッド名	引数
3	void	updateByKey	Item item

No.	戻り値型	メソッド名	引数
4	void	deleteByKey	int id
5	void	createNew	Item item

Mapper名：ItemDao

No.	ディレクティブ	id	パラメータタイプ	SQL
3	update	updateByKey	com.example.webapi.entity.Item	insert into items (id, title, content, status, created, updated) values (#{id}, #{title}, #{content}, #{status}, now(), now()) on duplicate key update title = #{title}, content = #{content}, status = #{status}, updated = now()
4	delete	deleteByKey	int	delete from items where id = #{id}
5	insert	createNew	com.example.webapi.entity.Item	insert into items (title, content, status, created, updated) values (#{title}, #{content}, #{status}, now(), now())

SERVICE仕様

Class名：ItemService

No.	戻り値型	メソッド名	引数
3	void	updateByKey	Item item
4	void	deleteByKey	int id
5	int	createNew	Item item

REPOSITORY(java)実装

`src/main/java/com/example/webapi/repository`の`ItemDao.java`に追加します。

`ItemDao.java: public interface ItemDao {}` に以下を追加します。

```
    void updateByKey(Item item);

    void deleteByKey(int id);

    void createNew(Item item);
```

REPOSITORY(xml)実装

`src/main/resources/com/example/webapi/repository`の`ItemDao.xml`に追加します。

`ItemDao.xml: <mappaer> ~ </mapper>`の`selectByKey`の下に以下を追加します。

```
    <update id="updateByKey" parameterType="com.example.webapi.entity.Item">
        insert into items (id, title, content, status, created, updated)
        values ( #{id}, #{title}, #{content}, #{status}, now(), now() )
        on duplicate key update
        title = #{title}, content = #{content}, status = #{status}, updated =
now()
    </update>

    <delete id="deleteByKey" parameterType="int">
        delete from items where id = #{id}
    </delete>

    <insert id="createNew" useGeneratedKeys="true" keyProperty="id"
parameterType="com.example.webapi.entity.Item">
        insert into items (title, content, status, created, updated)
        values ( #{title}, #{content}, #{status}, now(), now() )
    </insert>
```

updateに関して

PUTの仕様として、更新KEYが存在すればUPDATE、存在しなければINSERTですので、MySQLでは `insert ~ on duplicate key update ~` を使います。

insertに関して

INSERT後に自動生成したIDを取得してフロントエンド側に返す必要があるので、以下の設定をします。
`useGeneratedKeys="true" keyProperty="id"`

この設定をしておくと、service層で`item.getId()`することで生成されたIDを取得できます。

SERVICE実装

`src/main/java/com/example/webapi/service`の`ItemService.java`に追加します。

`ItemService.java:`の`selectByKey`の下に以下を追加します。

```
    @Transactional(rollbackFor = Exception.class, propagation =
Propagation.REQUIRED, readOnly = false)
    public void updateByKey(Item item) {
        itemMapper.updateByKey( item );
    }

    @Transactional(rollbackFor = Exception.class, propagation =
Propagation.REQUIRED, readOnly = false)
```

```
    public void deleteByKey(int id) {
        itemMapper.deleteByKey( id );
    }

    @Transactional(rollbackFor = Exception.class, propagation =
Propagation.REQUIRED, readOnly = false)
    public int createNew(Item item) {
        itemMapper.createNew( item );
        return item.getId();
    }
```

@Transactionalアノテーション

トランザクションを書き込み可能(`readOnly = false`)に設定し、
全ての例外を補足し、発生した場合ROLLBACKする(`rollbackFor = Exception.class`)よう設定します。
`propagation`に関しては、`REQUIRED:` 「トランザクションが開始されていなければ新規に開始し、すでに開始されていればそのトランザクションをそのまま利用する」に設定します。

参考URL： https://qiita.com/NagaokaKenichi/items/a279857cc2d22a35d0dd

CONTROLLER実装

`src/main/java/com/example/webapi/controller`の`ItemController.java`に追加します。

`ItemController.java:`の`selectByKey`の下に以下を追加します。

```
    @PutMapping(value = "/items/{id}")
    public void updateByKey(@PathVariable int id, @RequestBody Item item) {
        item.setId(id);
        itemService.updateByKey(item);
    }

    @DeleteMapping(value = "/items/{id}")
    public void deleteByKey(@PathVariable int id) {
        itemService.deleteByKey(id);
    }

    @PostMapping(value = "/items")
    public int createNew(@RequestBody Item item) {
        return itemService.createNew(item);
    }
```

@PutMappingアノテーション

メソッドが`PUT`の場合に評価される。
`value = "xxx"`で/api以降のURLを取り出し、合致したものが実行される。

- /api/items/1 -> updateByKey(1, item)がコールされる

`@PathVariable`はPathに含まれる変数を取得する。
サンプルの場合URLの`{id}`に設定された値を`int id`として取り出す。

`@RequestBody`はフロントエンドから渡されるリクエストボディ部を取り出します。サンプルの場合、id, title, content, statusの4つの項目を持つJsonデータとなります。

データ例

```
{
    "id": 10,
    "title": "PUT更新テスト",
    "content": "存在しないIDは新規に登録される",
    "status": "PROGRESS"
}
```

@DeleteMappingアノテーション

メソッドが`DELETE`の場合に評価される。
`value = "xxx"`で/api以降のURLを取り出し、合致したものが実行される。

- /api/items/1 -> deleteByKey(1)がコールされる

@PostMappingアノテーション

メソッドが`POST`の場合に評価される。

- /api/items/ -> createNew(item)がコールされる

`@RequestBody`でフロントエンドから渡されるリクエストボディ部を取り出します。

ビルド・動作確認

ビルド

gradlew buildでビルドします。

```
PS C:\home\webapi> ./gradlew build

> Task :test
2020-07-11 22:49:08.994 DEBUG 20096 --- [extShutdownHook]
o.s.w.c.s.GenericWebApplicationContext   : Closing
org.springframework.web.context.support.GenericWebApplicationContext@39401536,
started on Sat Jul 11 22:49:05 JST 2020
2020-07-11 22:49:09.000  INFO 20096 --- [extShutdownHook]
o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down ExecutorService
'applicationTaskExecutor'

BUILD SUCCESSFUL in 17s
5 actionable tasks: 5 executed
```

起動

SPRING-BOOT DASHBOARDでwebapiを起動します。

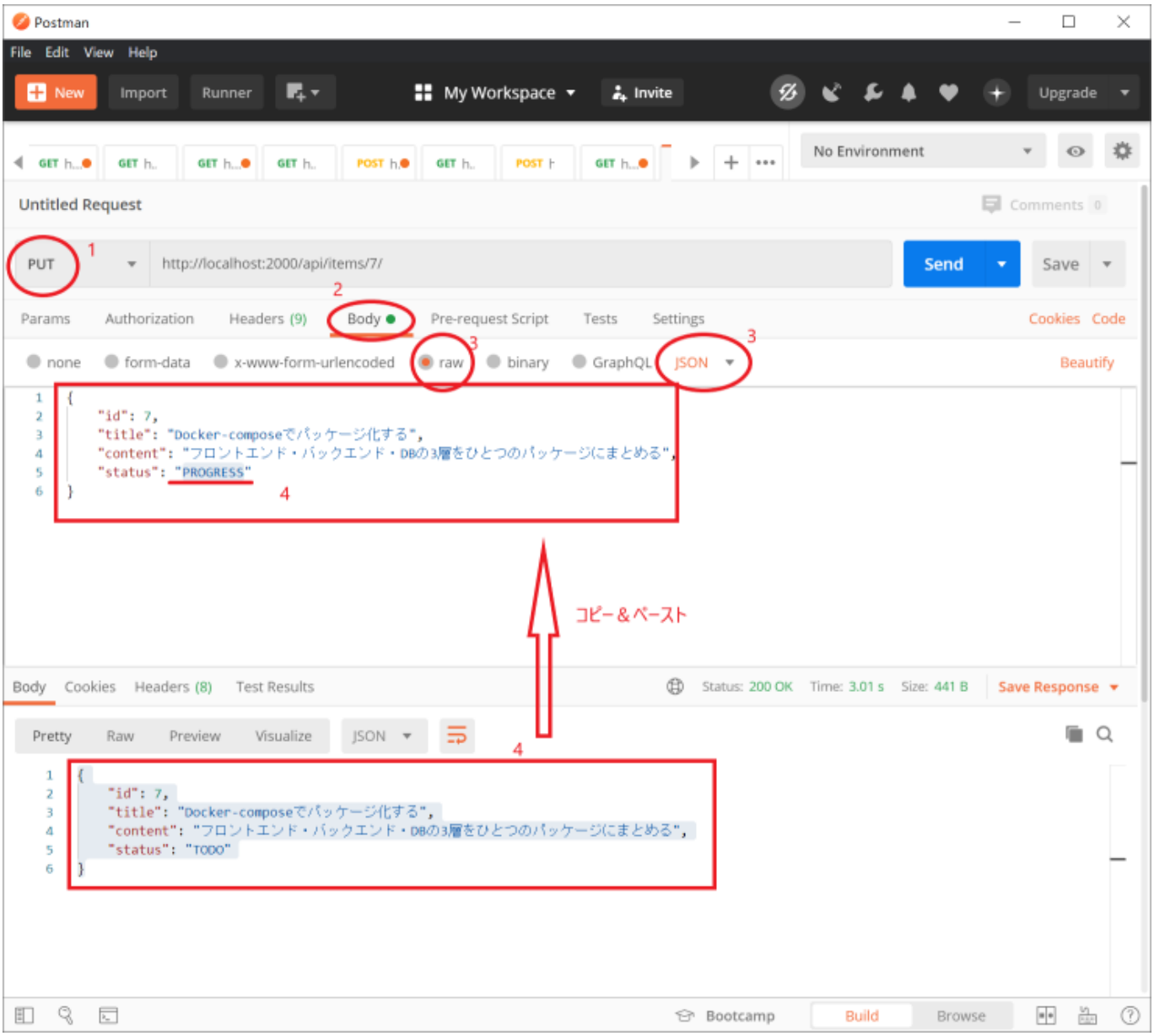
更新動作確認

Postmanを起動し、下記URLにアクセス(GET)して更新対象データを取得します。

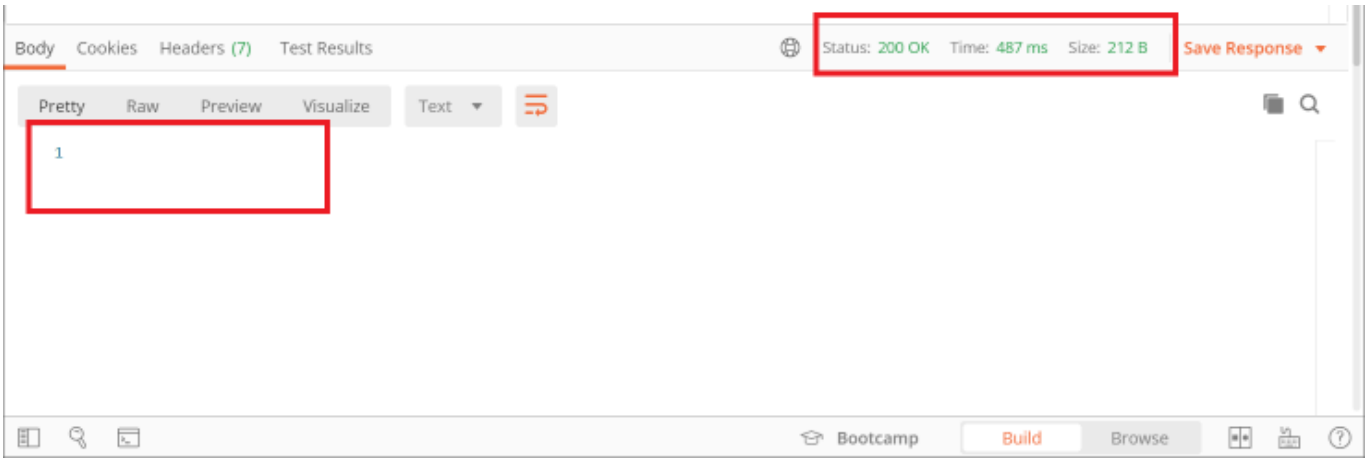
```
http://localhost:2000/api/items/7/
```

1. メソッドを`PUT`に変えます
2. 内容を`Body`を選択します
3. 形式を`raw`を選択し、フォーマットを`Text`から`JSON`に変更します
4. 結果をコピーして、body部分にペーストした後に、statusを"PROGRESS"に変更します

```
{
    "id": 7,
    "title": "Docker-composeでパッケージ化する",
    "content": "フロントエンド・バックエンド・DBの3層をひとつのパッケージにまとめる",
    "status": "PROGRESS"
}
```



5. Sendボタンをクリックします -> 応答は空になりHTTP STAUSは200で返ってくれば成功です



6. メソッドをGETに戻して再度データを取得します　-> statusが"PROGRESS"に変わっています

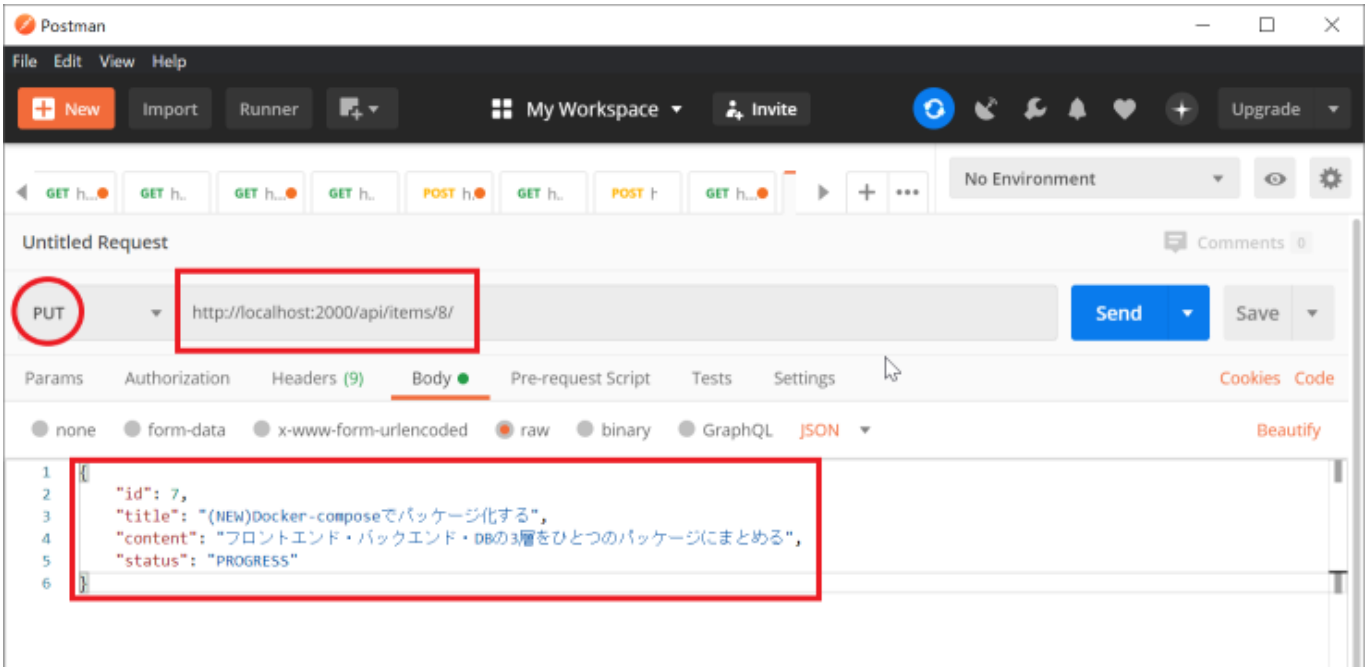
更新(新規登録)動作確認

PUT で存在しないIDを更新しようとすれば新規に登録できることを確認します。

1. 先ほどの画面の続きで、送信データのtitleを下記のように変更します

```
{
    "id": 7,
    "title": "(NEW)Docker-composeでパッケージ化する",
    "content": "フロントエンド・バックエンド・DBの3層をひとつのパッケージにまとめる",
    "status": "PROGRESS"
}
```

2. メソッドをPUTに変更します
3. URLをhttp://localhost:2000/api/items/8/ に変更します **(重要)**
4. 他設定を確認しSendボタンをクリックします。



5. メソッドをGETに戻して再度データを取得します　-> id=8で新規データが登録されます

削除機能確認

DELETEでid指定したデータの削除を確認します。

1. メソッドをDELETEに変更します
2. URLはhttp://localhost:2000/api/items/8/ になっていることを確認します
3. Send ボタンをクリックします
4. メソッドをGET に変更します
5. URLをhttp://localhost:2000/api/items/ に変更します
6. Send ボタンをクリックします　->　id=8 のレコードが削除されています

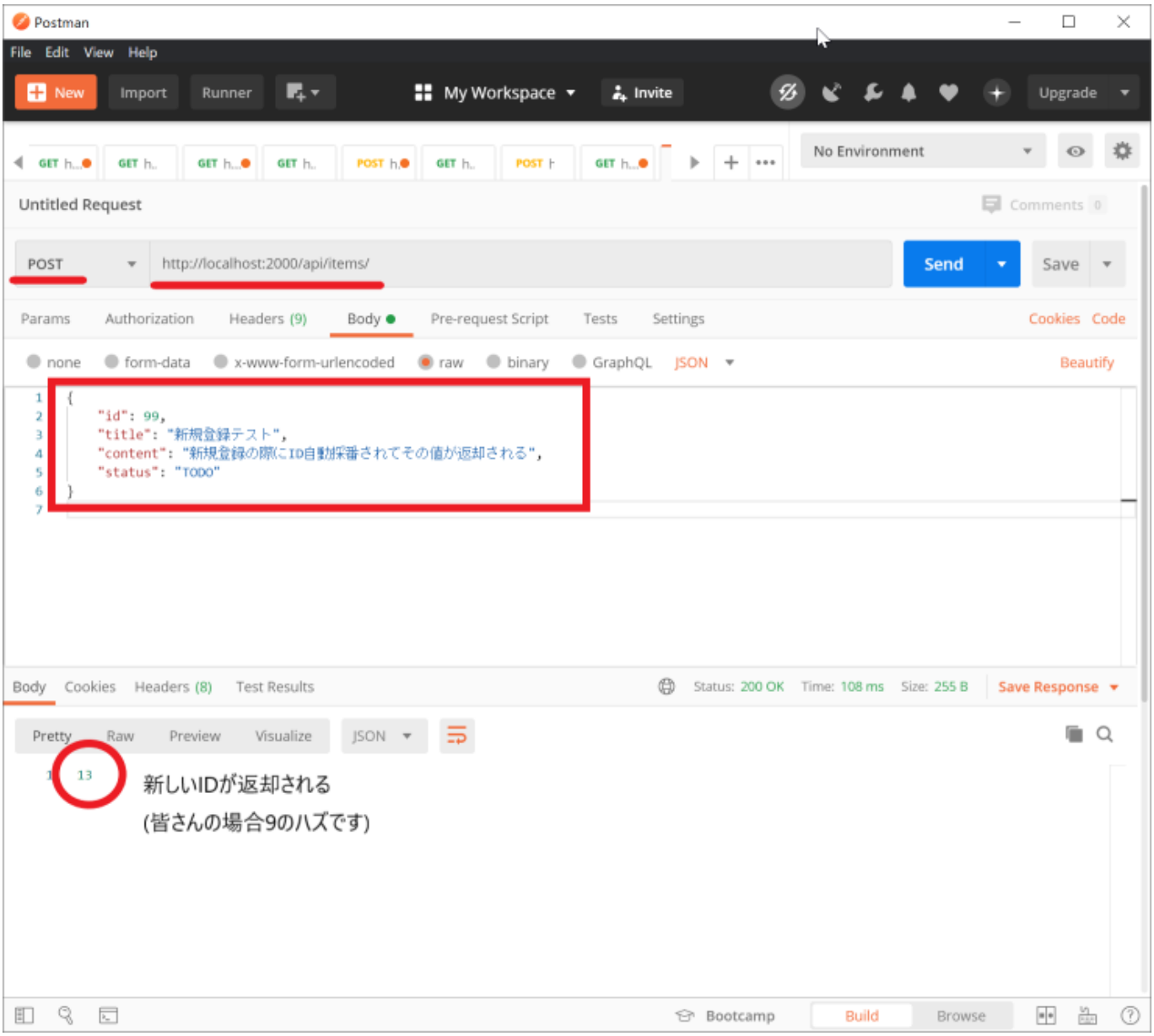
新規登録機能確認

POSTでデータの新規登録を確認します。自動採番されたIDが返却されることも確認します。

1. メソッドをPOSTに変更します
2. URLはhttp://localhost:2000/api/items/になっていることを確認します
3. Body部分に下記内容の新規登録データを作成します。

```
{
    "id": 99,
    "title": "新規登録テスト",
    "content": "新規登録の際にID自動採番されてその値が返却される",
    "status": "TODO"
}
```

4. Sendボタンをクリックします　->　新規IDが返却されます。(皆さんの場合は9)



5. メソッドを`GET`に変更します
6. URLを`http://localhost:2000/api/items/9/` に変更します
7. `Send` ボタンをクリックします　->　先ほど登録した内容が表示されます

まとめ

バックエンド開発では、「Spring Boot」フレームワークを用いたRESTful API開発を学びました。

リクエストの扱いやログ出力、例外(今回は触れていませんが)など、フレームワークが多くの部分をカバーしてくれており、開発者は、データ構造とビジネスロジック実装に集中できることが体験頂けたと思います。

次週はいよいよ、フロントエンドとバックエンドを結合し、アプリとして完成させます。