

Soft Arch. Assignment :: 1 four guys one cup

Team : four guys one cup

6031052421 วรวิทย์พล ศรีพรม

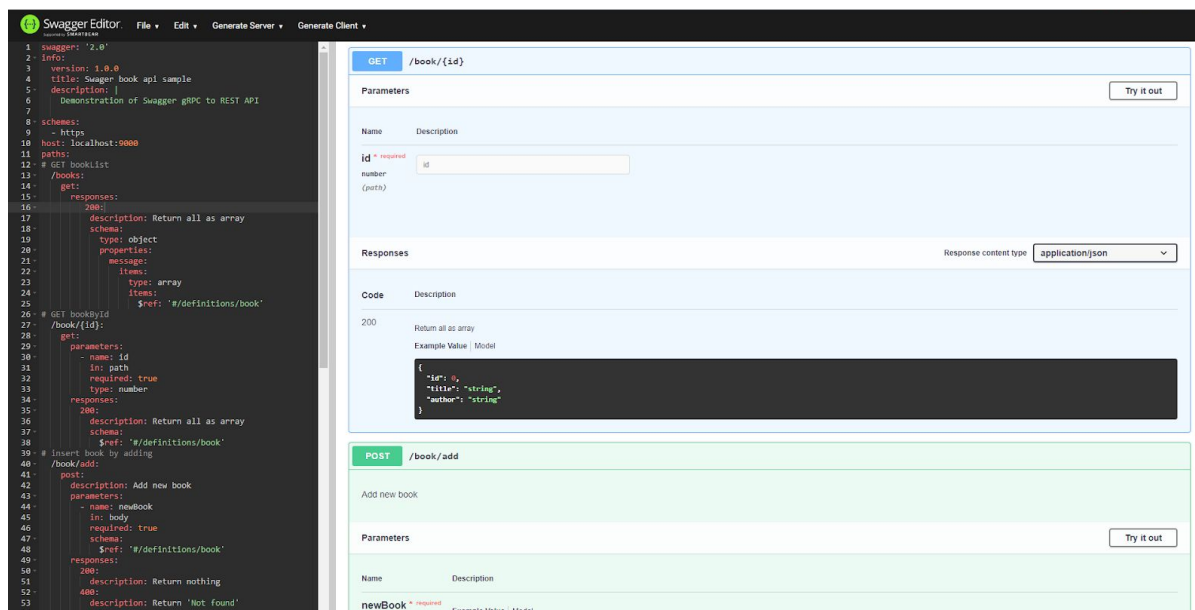
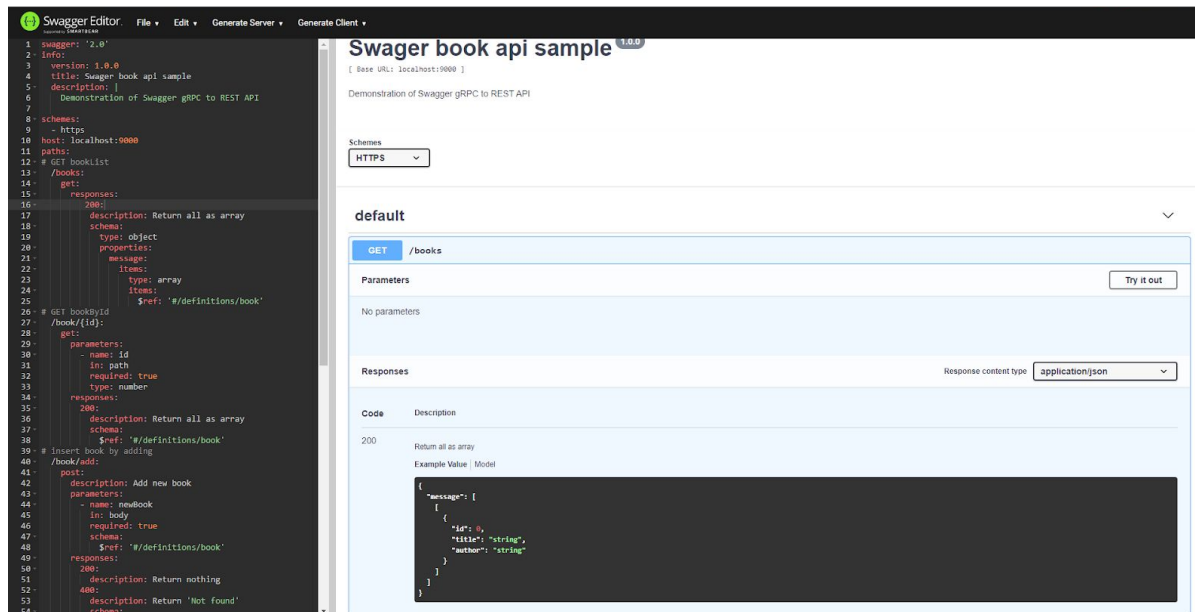
6030515521 วรธนัย ทิพย์สถาพร

6030285121 สาม ยองโย

6030200821 ณัฐนัย กิจวิวัฒน์การ

Things to be delivered:

1. Screenshots of Swagger for your APIs in 2.



Soft Arch. Assignment :: 1 four guys one cup

The image shows the Swagger Editor interface for a REST API. The left pane displays the OpenAPI specification in YAML format. The right pane shows the visual representation of the **POST /book/add** endpoint.

YAML Specification (Left Pane):

```
1 swagger: '2.0'
2 info:
3   version: 1.0.0
4   title: Swagger book api sample
5   description: |
6     Demonstration of Swagger gRPC to REST API
7
8 schemes:
9   - https
10 host: localhost:9000
11 paths:
12   /books:
13     get:
14       responses:
15         200:
16           description: Return all as array
17           schema:
18             type: object
19             properties:
20               message:
21                 type: array
22                 items:
23                   type: object
24                   $ref: '#/definitions/book'
25
26 # GET bookById
27 /book/{id}:
28   get:
29     parameters:
30       - name: id
31         in: path
32         required: true
33         type: number
34     responses:
35       200:
36         description: Return all as array
37         schema:
38           $ref: '#/definitions/book'
39
40 # insert book by adding
41 /book/add:
42   post:
43     description: Add new book
44     parameters:
45       - name: newbook
46         in: body
47         required: true
48         schema:
49           $ref: '#/definitions/book'
50     responses:
51       200:
52         description: Return nothing
53       400:
54         description: Return 'Not found'
55         schema:
```

Visual Representation (Right Pane):

- Endpoint:** **POST /book/add**
- Description:** Add new book
- Parameters:** None
- Body:** **newbook** (required, object). Example Value / Model:

```
{ "id": 0, "title": "string", "author": "string" }
```
- Responses:**

Code	Description
200	Return nothing
400	Return 'Not found'

The image shows the Swagger Editor interface for a REST API, specifically the **DELETE /book/del/{id}** endpoint.

YAML Specification (Left Pane):

```
1 swagger: '2.0'
2 info:
3   version: 1.0.0
4   title: Swagger book api sample
5   description: |
6     Demonstration of Swagger gRPC to REST API
7
8 schemes:
9   - https
10 host: localhost:9000
11 paths:
12   /books:
13     get:
14       responses:
15         200:
16           description: Return all as array
17           schema:
18             type: object
19             properties:
20               message:
21                 type: array
22                 items:
23                   type: object
24                   $ref: '#/definitions/book'
25
26 # GET bookById
27 /book/{id}:
28   get:
29     parameters:
30       - name: id
31         in: path
32         required: true
33         type: number
34     responses:
35       200:
36         description: Return all as array
37         schema:
38           $ref: '#/definitions/book'
39
40 # insert book by adding
41 /book/add:
42   post:
43     description: Add new book
44     parameters:
45       - name: newbook
46         in: body
47         required: true
48         schema:
49           $ref: '#/definitions/book'
50     responses:
51       200:
52         description: Return nothing
53       400:
54         description: Return 'Not found'
55         schema:
```

Visual Representation (Right Pane):

- Endpoint:** **DELETE /book/del/{id}**
- Description:** Delete book by id
- Parameters:** **id** (required, number, path). Example Value / Model:

```
id
```
- Responses:**

Code	Description
200	Return deleted book
400	Return 'Not found'

2. Source codes of 2 and 3.

```
const express = require('express');

const app = express();

var books = [
  {
    id: 123, title: 'A Tale of Two Cities', author: 'Charles Dickens'
  },
]

app.get('/books', (req, res)=>{
  return res.send(
    {
      "books": books
    }
  )
});

app.get('/book/:id', (req, res)=>{
  for (var i = 0; i < books.length; i++){
    if (books[i].id == req.params.id){
      return res.send(
        books[i]
      );
    }
  }
  return res.status(404).json({
    "message": "Not found"
  })
});

app.post('/book/add', (req, res)=>{
  let {_id, _title, _author} = req.body

  if (_id && _title && _author){
    books.push({
      id: _id,
      title: _title,
      author: _author
    })
    return res.status(200).json({})
  }
  else return res.sendStatus(404)
});

app.get('/book/del/:id', (req, res)=>{
  if (req.params.id){
    for (var i = 0; i < books.length; i++){
      if (books[i].id == req.params.id){
        books.splice(i, 1);
        return res.status(200).json({})
      }
    }
    return res.status(404).json({
      "message": "Not found"
    })
  }
});

app.listen(50050, () =>
  console.log(`App listening on port ${50050}!`),
);
```

3. Compare how to call the methods based on gRPC and REST API side-by-side, e.g. in a Table format as shown below.

Function	gRPC	REST API
List books	execute client.go list command in command line to call service list in books.proto	GET http:domain/books
insert book	execute client.go insert with id in command line to call service Insert in books.proto	POST http:domain/book/add
Get book by id	execute client.go get with id in command line to call service Get in books.proto	GET http:domain/book/{id}
Delete book by id	execute client.go delete with id in command line to call service Delete in books.proto	GET http:domain/book/del/{id}

4. What are the main differences between REST API and gRPC?

gRPC is strongly typed messages. so, REST API is more flexible to send requests with JSON format.

5. What are the benefits of introducing interfaces in front of the gRPC and REST API of the book services.

Easy to migrate or change the backend functions while not interfering with the frontend calling to service

6. Based on the introduced interface, compare how to call the methods based on gRPC and REST API side-by-side, e.g. in a

Function	gRPC	REST API
List books	call from interface client.js to access function and message in books.proto	GET http:domain/books
insert book	call from interface client.js to access function and message in books.proto	POST http:domain/book/add
Get book by id	call from interface client.js to access function and message in books.proto	GET http:domain/book/{id}
Delete book by id	call from interface client.js to access function and message in books.proto	GET http:domain/book/del/{id}

7. Draw a component diagram representing the book services with and without interfaces.

