OpenAPI and Service Integration

Vijay Paul Nayar

2023-08-29 Tue

Outline

Who am I?

Vijay Paul Nayar

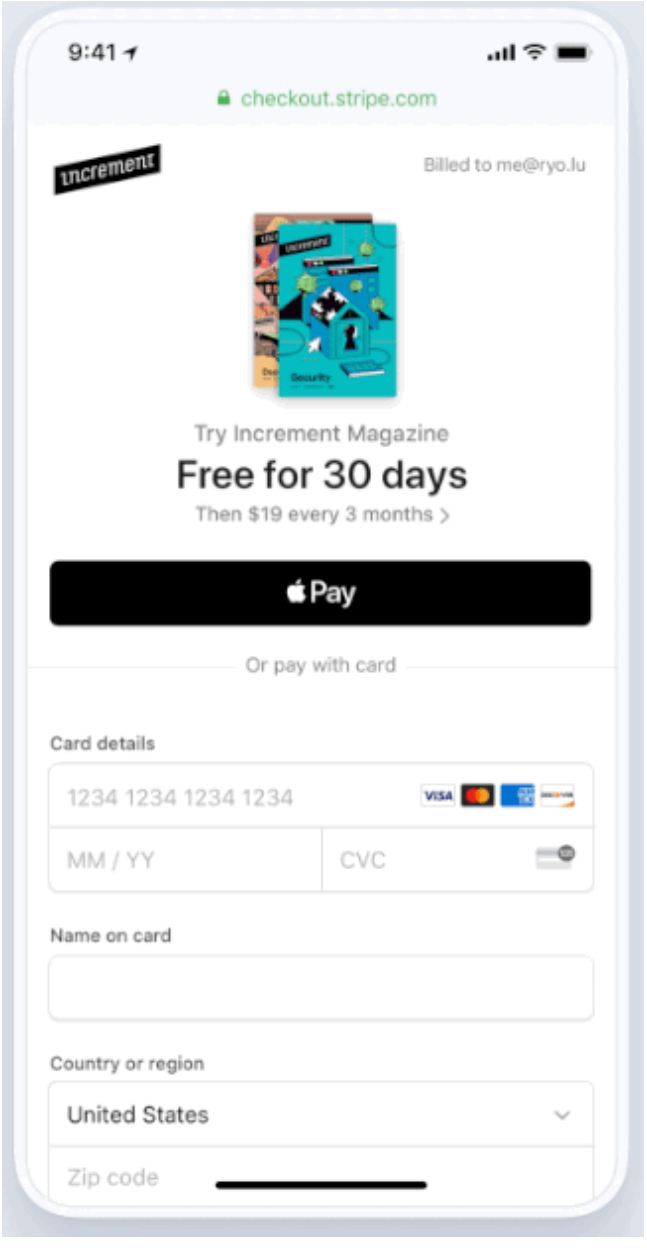
- Java developer and CTO of a FinTech
- Left CTO role to found own company

Funnel-Labs.io: Performant D apps.

- Funnel: High performance data storage system for ride-hailing and micro-mobility companies.
- Fiveum: Office chat and video built to minimize interruptions and improve focus.

How did OpenAPI Come Up?

- Built Funnel Service MVP...
  - How do customers pay for the service?
    - Most services use credit-cards
  - How to easily add credit-card support?
    - Stripe is popular and common
  - How to use Stripe?
    - Stripe has a REST API, but it's huge
  - How do Java/Python do this?
    - Generated OpenAPI client
  - Do such tools exist in D?
    - No, but they could.



Presentation Agenda

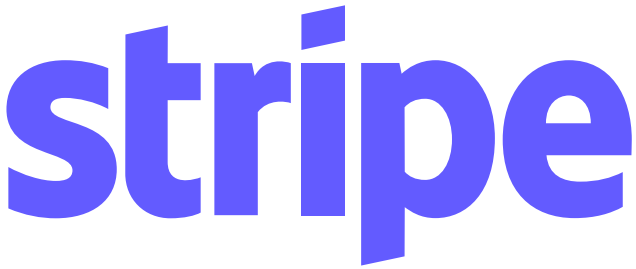
Introduction

External Service Interoperability

Companies often depend on useful external services.
For example:

- Stripe (financial transactions)
- OpenAI (categorize sentiment, question/answer, content generation)
- Slack (real-time communication)

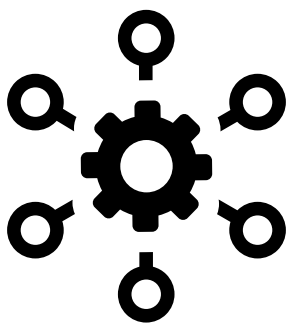
Hand written clients are time-consuming and error-prone



Internal Service Interoperability

Even internal services face interoperability challenges:

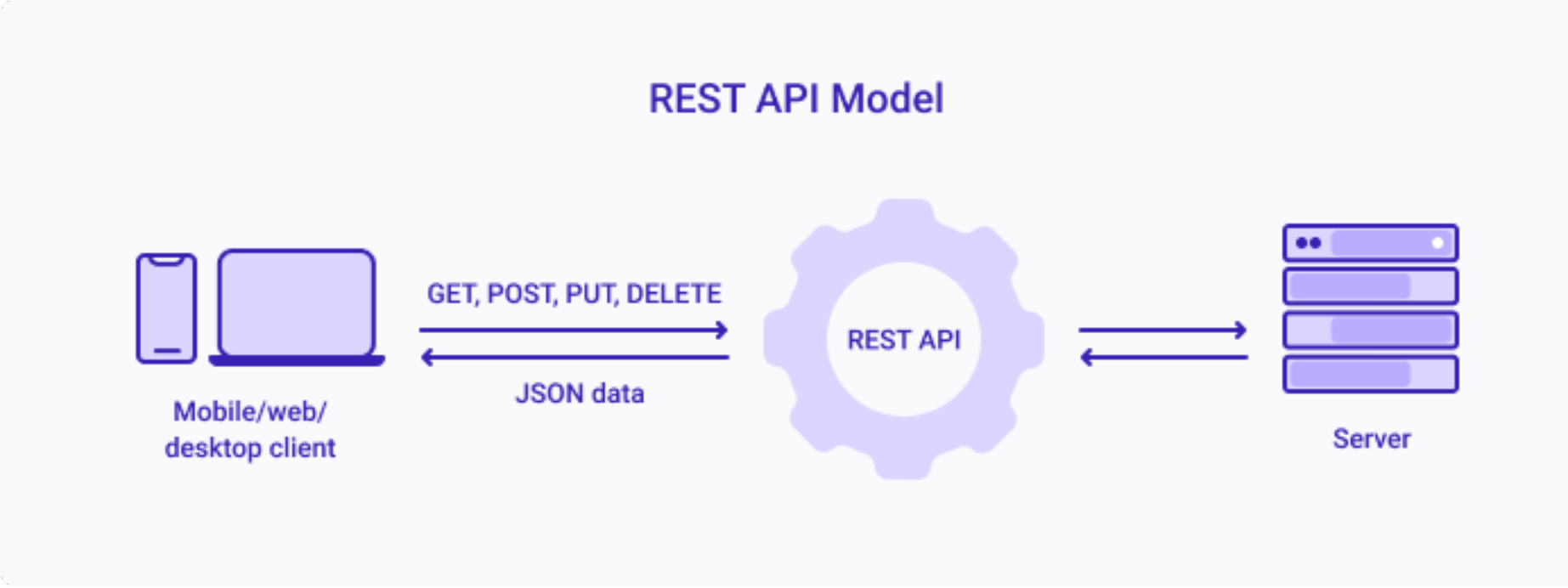
- Communication must be secure
- Interfaces should be understandable and standardized
- Multiple programming languages must be supported (companies change technologies, different employees have different skills, etc.)



REST Interfaces

(Re)presentational (S)tate (T)ransfer is an architectural style designed for the web

- Many forms, typically JSON/Avro/Protobuf over HTTPS
- URLs arranged into "nouns" with HTTP Methods representing "verbs"
- By itself, too vague to be uniform
- Minor performance penalty for increased clarity



Presentation Agenda

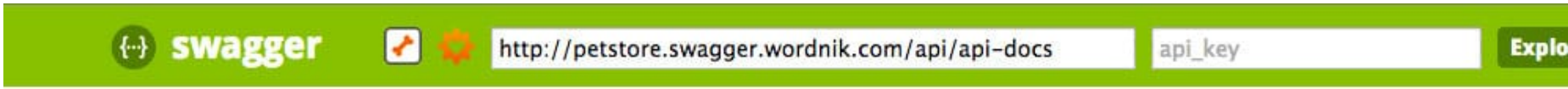
What is OpenAPI?

OpenAPI

- OpenAPI Specification is open standard to define HTTP APIs for external consumers
  - Builds upon JSON Schemas `https://json-schema.org/`
  - Builds upon Swagger API description and documentation `https://swagger.io/`
  - Split from Swagger in 2016 to become the OpenAPI Initiative, a Linux Foundation project

Benefits of OpenAPI Usage

- Commonly used by major services, e.g. Stripe, Slack, OpenAI, and 2500+ more: `https://apis.guru/`
- Standard formats mean tools can be used to generate client code with:
  - request and responses
  - documentation
  - success and error codes
- Creating an OpenAPI Specification enables low-effort cross-compatibility



Swagger Sample App

This is a sample server Petstore server. You can find out more about Swagger at http://swagger.wordnik.com or on irc.freenode.net, #swagger. For this sample, you can use the api key "special-key" to test the authorization filters

Terms of service
Contact the developer
Apache 2.0

pet : Operations about pets

Show/Hide | List Operations | Expand Operations

GET /pet/{petId} Find pet b

DELETE /pet/{petId} Deletes a

PATCH /pet/{petId} partial updates to a

POST /pet/{petId} Updates a pet in the store with form d

OFF

Parameters

Parameter	Value	Description	Parameter Type	Data Type
petId	(required)	**ID of pet that needs to be updated**	path	string
name		Updated name of the pet	form	string
status		Updated status of the pet	form	string

Response Messages

HTTP Status Code	Reason	Response Model
405	Invalid input	

Structure of an OpenAPI Specification

- OpenAPI Specification is itself a JSON/YAML document

OpenAPI Major Top-Level Attributes

Field Name	Type	Description
servers	[Server Object]	Connection info. for servers offering the API.
paths	Paths Object	Method-specific actions by URL path.
components	Components Object	Re-usable schemas for data by name.
security	[Security Object]	Lists security mechanisms to access the API.

Defining API Endpoints - #/paths

- Mapping from endpoint URL to details

```
{
  "paths": {
    "/files/{file_id}": {  // URL => Path Item
      "delete": {          // Method => Operation
        "operationId": "deleteFile",  // API-unique identifier
        "tags": [          // Tags for grouping documentation
          "OpenAI"
        ],
        "summary": "Delete a file.",  // A 1-liner for documentation.
        "parameters": [  // Request parameters in path/query/header/cookie
          {
            "in": "path",
            "name": "file_id",
            "required": true,
            "schema": {  // JSON Schema format is used.
              "type": "string"
            },
            "description": "The ID of the file to use for this request"
          }
        ],
        "responses": {  // Response data format by HTTP status
          "200": {
            "description": "OK",
            "content": {
```

JSON Schemas

- All data represented in JSON can be described using JSON Schemas.
- Assertions are used to validate if data matches the schema:

type	Primitive valies like null, boolean, object, array, number, string
format	How a type is used, e.g. date-time, email, uri, ipv4, etc.
enum	Limit value to a predefined list.
allOf	All validations must be satisfied.
anyOf	One or more validation must be satisfied.
oneOf	Exactly one validation must be satisfied.

JSON Schema Example

An example schema:

```
{
  "components": {
    "schemas": {
      "CreateChatCompletionResponse": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string"
          },
          "model": {
            "type": "string"
          },
          "choices": {
            "type": "array",
            "items": {
              "type": "object",
              "required": [
                "index",
                "message",
                "finish_reason"
              ],
              "properties": {
                ...
```

An example instance complying with the schema:

```
{
  "id": "3d5e3472-3057-11ee-89d4-c3a0bb88
  "model": "gpt-3.5-turbo",
  "choices": [
    {
      "index": 3,
      "finish_reason": "length",
      "message": { ... }
    },
    ...
```

Schema Primitive Types

- Data Types:
  - The "type" field corresponds broadly to a JSON type.
  - The "format" field clarifies details and usage.

type	format	Description
integer	int32	signed 32 bits
integer	int64	signed 64 bits
number	float	
number	double	
string	password	A hint to UIs to obscure input

Defining Common Components - #/components

APIs commonly have shared data types between paths.

- Error and Created responses
- Query Parameters
- Security headers

Field Name	Type	Description
schemas	string => SchemaObj	Common schemas by name.
responses	string => ResponseObj	Path responses, e.g. errors.
parameters	string => ParameterObj	Request parameter types.
requestBodies	string => RequestBodyObj	Request bodies for POST,PUT.
headers	string => HeaderObj	Common data in HTTP headers.
securitySchemes	string => SecuritySchemeObj	E.g. OAuth, Basic Auth, etc.

Reusing Components

- Once defined, components can be referenced by their location in the OpenAPI Schema.
- Substitute type definition with a "$ref" to a component.

```
"properties": {
  "index": {
    "type": "integer"
  },
  "message": {
    "$ref": "#/components/schemas/ChatCompletionRespons
  },
```

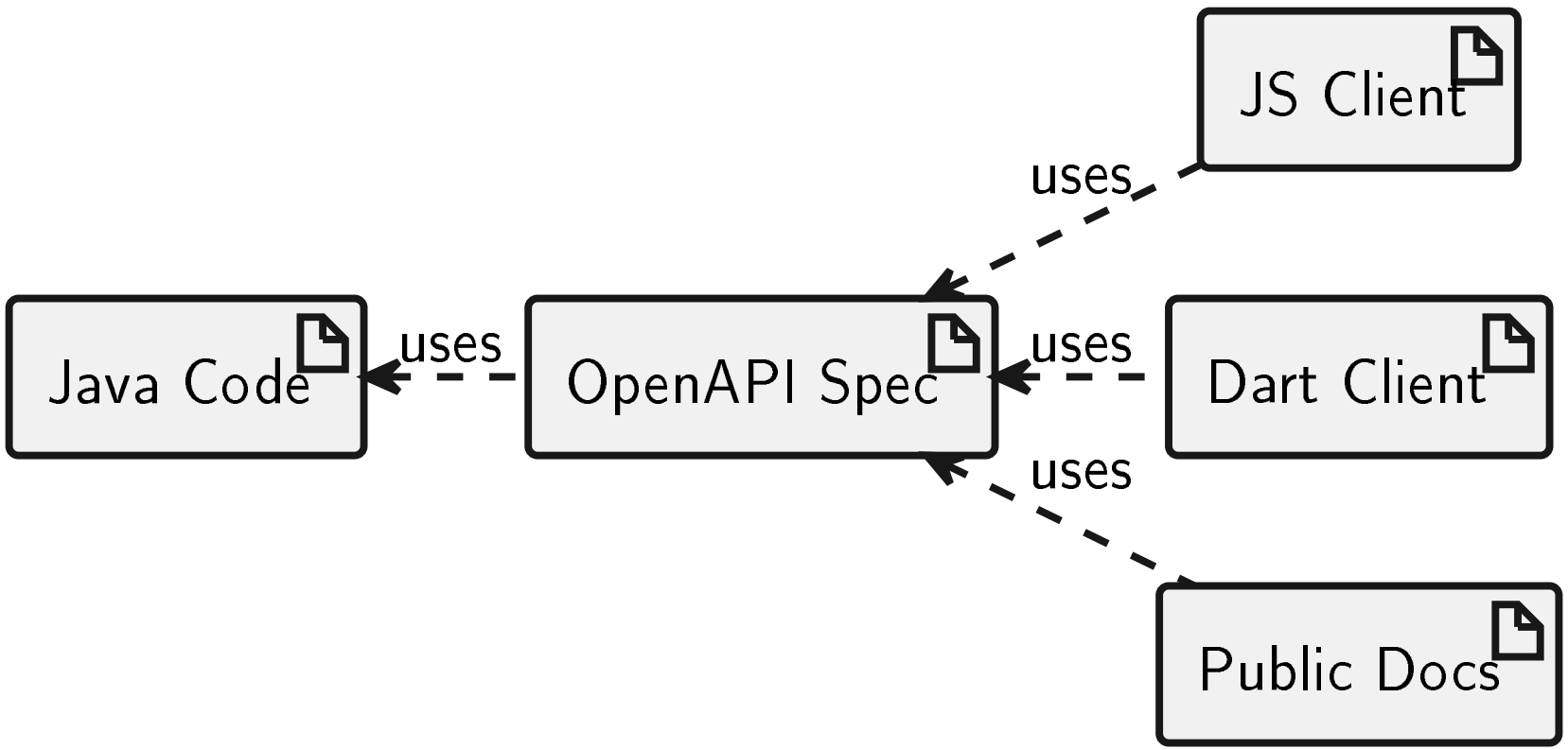
Managing OpenAPI Specs

Creating OpenAPI Specifications

- D currently lacks tools to extract specification from code.
- Open question whether it is better to:
  - Generate specification from code
    - Easier to keep specification up to date
    - Language/Framework-specific projects like SpringDoc
  - Generate interfaces from specification
    - Easier tool integration and multi-language support
    - Projects like openapi-generator

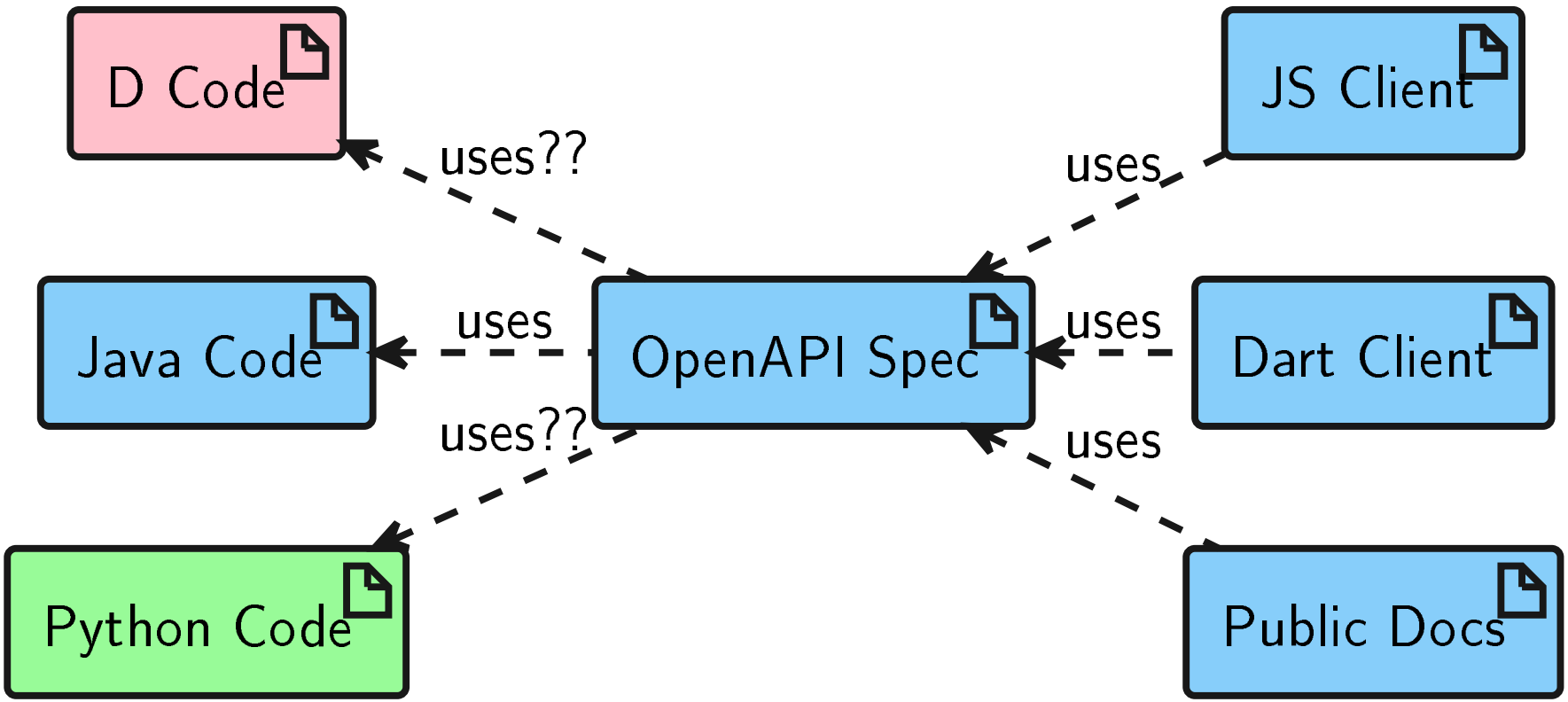
OpenAPI Specification from Code

- Systems like SpringDoc are specific to language (Java) and web framework (Spring)
- OpenAPI Specification is updated with code changes



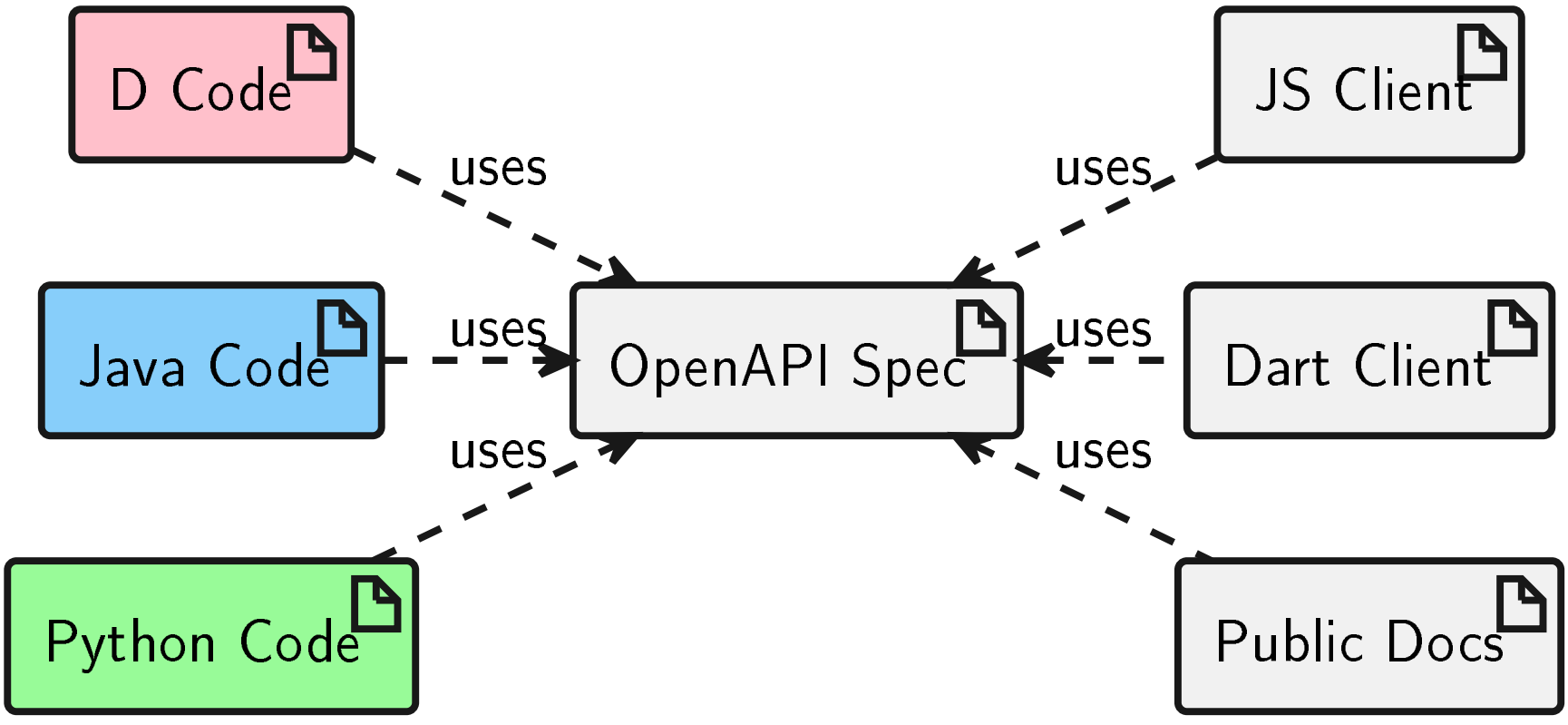
OpenAPI Specification from Code

- What happens when a service is split?
- What if multiple technologies are used?



Code from OpenAPI Specification

- Requires clients/servers to regenerate code after changes



Java SpringDoc OpenAPI Annotations

```
@SecurityScheme(name = "petstore_auth", type = SecuritySchemeType.OAUTH2, flow
                @OAuthScope(name = "write:pets", description = "modify pets in
                @OAuthScope(name = "read:pets", description = "read your pets"
@Tag(name = "pet", description = "the pet API")
public interface PetApi {
        @Operation(summary = "Add a new pet to the store",
            description = "Add a new pet to the store",
            security = { @SecurityRequirement(name = "petstore_auth", scopes =
            tags = { "pet" })
        @ApiResponses(value = {
            @ApiResponse(responseCode = "200",
                description = "Successful operation",
                content = {
                  @Content(mediaType = "application/xml", schema = @Schema(imp
                  @Content(mediaType = "application/json", schema = @Schema(im
            @ApiResponse(responseCode = "405", description = "Invalid input")
        })
        @PostMapping(value = "/pet", consumes = { "application/json", "applica
        default void addPet(
            @Parameter(description = "Create a new pet in the store", required
          // return getDelegate().addPet(pet);
        }
```

Useful D Features

Mixins

The `mixin` expression takes a list of `string` arguments representing a complete D statement and turns them into code.

- Can make use of variables known at compile-time, e.g. those provided by templates
- Useful for code that declares variables or methods with parameterized identifiers

```
mixin("private bool _myValue;");

string N = "yourVal";
mixin("private bool", "_", N, ";");
```

Mixin Templates

A `mixin template` encloses declarations of fields, functions, classes, structs, etc. When referenced in code with compile-time parameters, it inserts those declarations in the scope in which it was called.

- Mixin Templates: Re-useable code generation

```
import std.traits : isAssignable;
import std.string : capitalize;
import std.typecons : Nullable;

mixin template AddField(C, T, string N) {
  // Declare the variable.
  mixin(T, " ", N, ";");
  mixin(  // Define  setter function.
      C, " set", capitalize(N), "(ST)(ST val) ",
      "if (isAssignable!(T, ST)) {",
      "  this.", N, " = val;",
      "  return this;",
      "}");
}
```

```
// Example usage
class Fish {
  mixin AddField!(typeof(this),
      Nullable!int, "age");
  mixin AddField!(typeof(this),
      Nullable!string, "job");
}

unittest {
  import std.stdio;
  Fish f = new Fish()
      .setAge(42)
      .setJob("Accountant");
  writeln(f.age, " ", f.job);
}
```

Static ForEach

`static foreach` statements generate repeated lines of code in the same scope in which they occur.

- static foreach: Loop over compile-time data, such as class members.

```
import std.traits : Fields, FieldNameTuple,
                    BaseClassesTuple;
// Add setters for a single class.
mixin template AddClassSetters(C) {
  static foreach (
      size_t i; iota(Fields!(C).length)) {
    mixin AddSetter!(
        Fields!(C)[i], FieldNameTuple!(C)[i]);
  }
}
// Add setters for full class hierarchy.
mixin template AddSetters(C) {
  static foreach (B; BaseClassesTuple!(C)) {
    mixin AddSetters!(B);
  }
  mixin AddClassSetters!(C);
}
```

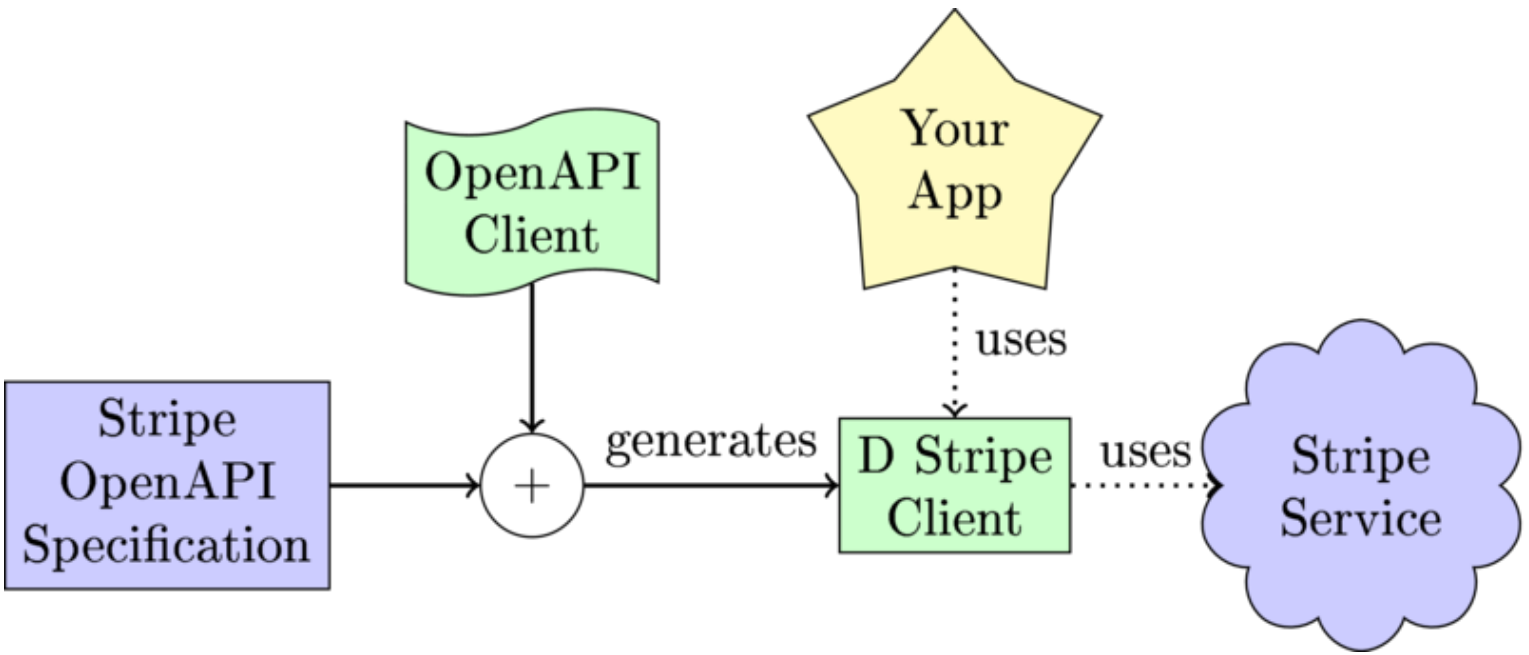
```
class A {
  int a1;
  string a2;
}

class B : A {
  float b1;
  mixin AddSetters!(typeof(this));
}

unittest {
  import std.stdio;
  B b = new B()
    .setA1(3)
    .setA2("ham")
    .setB1(2.9);
}
```

D Project: openapi-client

Simple OpenAPI Client in D

code.dlang.org project: openapi-client

- Consistent interface created/updated in seconds
- Creates data types from OpenAPI Specification
- Creates client to call endpoints
- Configurable server and security controls



openai-client: Creating an OpenAI Client

1. Download the OpenAPI Specification from GitHub:

```
curl https://raw.githubusercontent.com/openai/
  openai-openapi/master/openapi.yaml
  -o openapi.yaml
```

2. Convert to JSON format:

```
yq openapi.yaml -o json > openapi.json
```

3. Invoke `openapi-client` to generate code:

```
dub run openapi-client@2.0.1 --
  --openApiSpec=json/openapi.json
  --packageRoot=openai
```

4. Done!

openai-client: Generated Models

```d
// File: openapi/model/CreateImageEditRequest.d
class CreateImageEditRequest {
  /**
   * The number of images to generate. Must be between
   */
  @vibeName("n")
  @vibeOptional
  @vibeEmbedNullable
  Nullable!(int) n;

  /**
   * The image to edit. Must be a valid PNG file, less
   * provided, image must have transparency, which wil
   */
  @vibeName("image")
  @vibeOptional
  string image;
```

openai-client: Generated Models

- Optional fields are `Nullable`.
- Nested objects as static inner classes
- Documentation included
- Builder pattern used to ease object creation

openai-client: Generated Services

```
// File: openai/service/image_edits_service.d
/**
 * Service to make REST API calls to paths beginning w
 */
class ImagesEditsService {
  /**
   * Creates an edited or extended image given an orig
   * See_Also: HTTP POST `/images/edits`
   */
  void createImageEdit(
      CreateImageEditRequest requestBody,
      CreateImageEditResponseHandler responseHandler,
      ) {
    ApiRequest requestor = new ApiRequest(
        HTTPMethod.POST,
        Servers.getServerUrl(),
        "/images/edits");
```

openai-client: Using Services

```
// Service classes group API functionality by path, e.g. /comple
auto service = new CompletionsService();
// Invoke an API endpoint, this one is for POST /completions
service.createCompletion(
    // Define the request body with a builder.
    CreateCompletionRequest.builder()
        .model("text-davinci-003")
        .prompt(Json("What is the cutest breed of rabbit? "))
        .echo(true)
        .maxTokens(2048)
        .build(),
    // ResponseHandlers have an attribute for each valid respons
    CompletionsService.CreateCompletionResponseHandler.builder()
        .handleResponse200((CreateCompletionResponse response) {
          logDebug("%s", serializeToJson(response).toString());
        })
        .build());
```

openai-client: Server Response


```
{
  "object": "text_completion",
  "created": 1690899388,
  "usage": {
    "prompt_tokens": 10,
    "total_tokens": 68,
    "completion_tokens": 58
  },
  "id": "cmpl-7ikSiD1IqwHn4XMwg8K04lvx2DnL9",
  "model": "text-davinci-003",
  "choices": [
    {
      "index": 0,
      "text": "What is the cutest breed of rabbit?

               The debate for which rabbit breed is the
               as it will depend on what the individual
```

Future Plans

- Move spec-first efforts to more mature projects like openapi-generator
  - Add D client and server-stub generators
- Consider code-first integration via annotations in frameworks like Vibe.d

Thank You

Thank you for your interest and attention!