Mastering API Design
with practical use cases

Introdução à Engenharia de Software

Luís Bastião Silva

bastiao@ua.pt

universidade de aveiro
theoria poiesis praxis

ento de eletrónica,
nicações e informática

Agenda

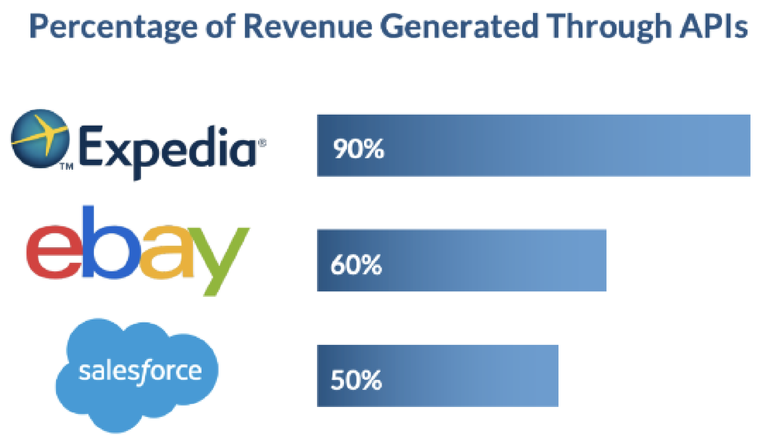
Design an API REST

- Authentication
- Synchronous vs Asynchronous APIs
- Health reports
- Documentation & versions

From the books to the practical and real use cases.

Strategic Value of APIs

Why APIs are so important?



Source: Harvard Business Review, The Strategic Value of APIs, 2015.

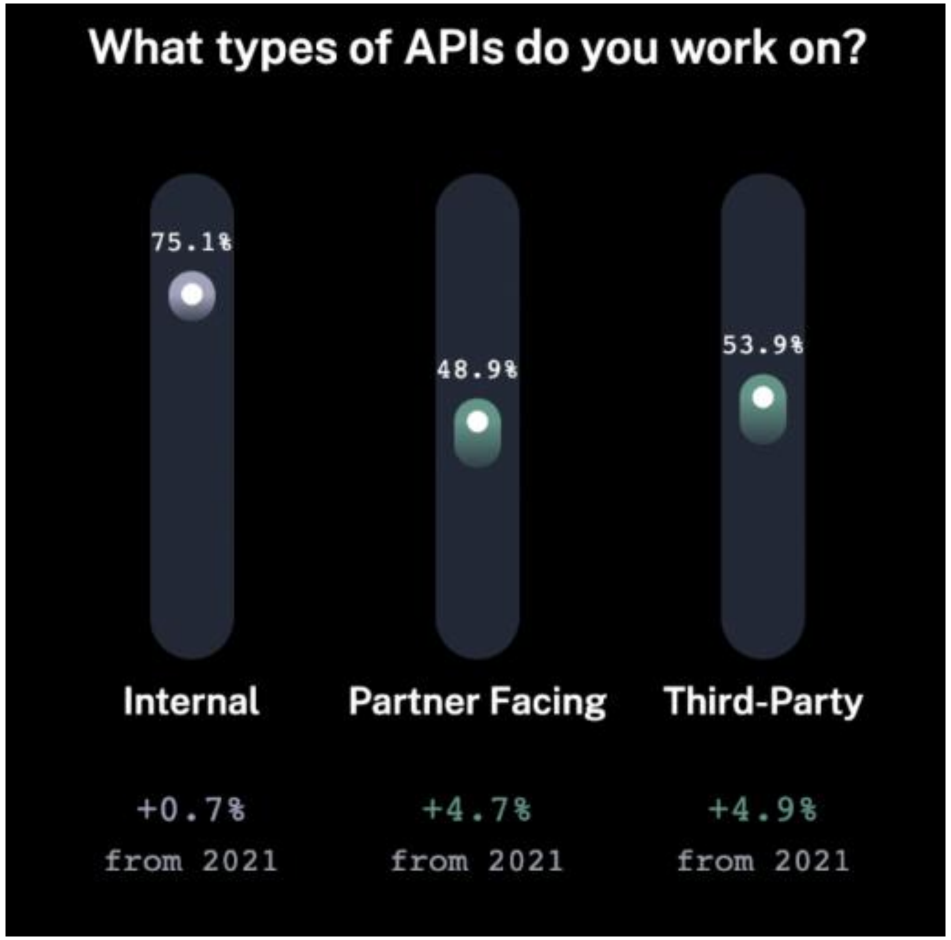
Strategic Value of APIs

Why APIs are so important?

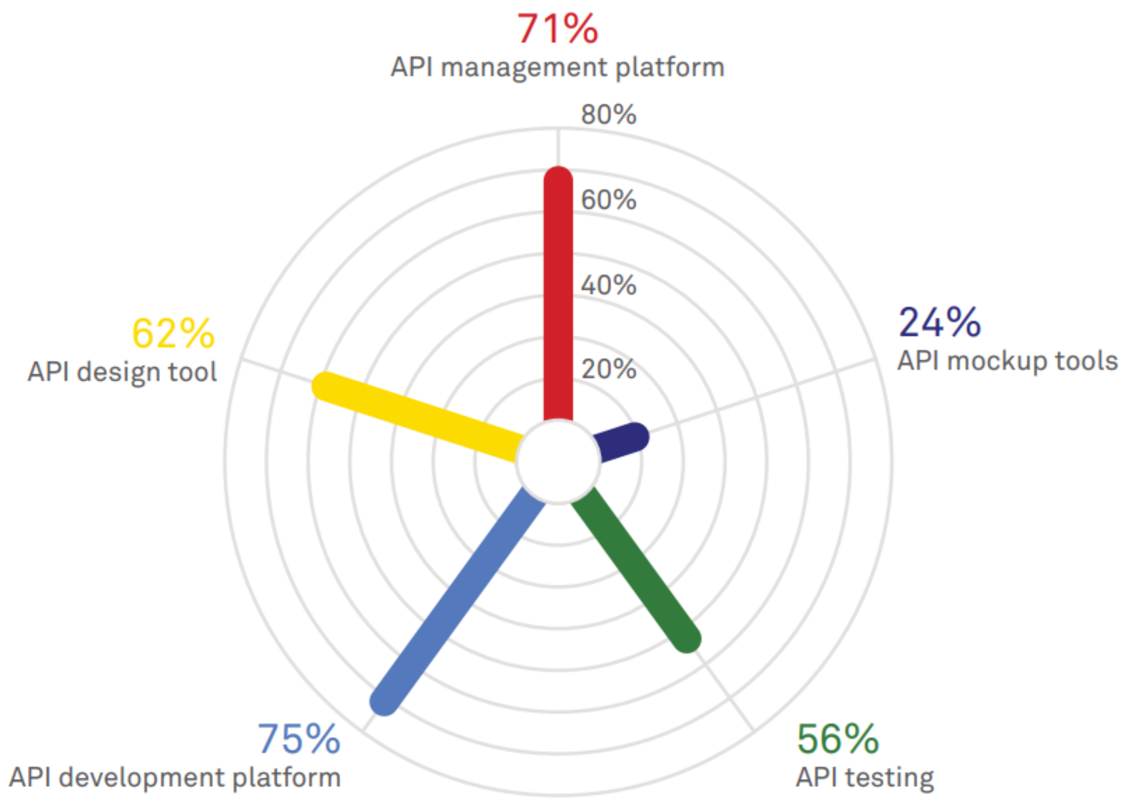
- "75% of developers utilize internal APIs
- 54% of developers are also using third-party APIs (up from 49% in 2021)
- 49% of developers are also using partner-facing APIs (up from 44% in 2021)"

https://rapidapi.com/blog/state-of-apis-growth-and-more-growth-on-tap-for-2023/

The 4th annual State of APIs Report comprised insights from more than 850 global developers, engineers, and leaders from across the technology community spanning over 100 countries including the US, the UK, Germany, and India

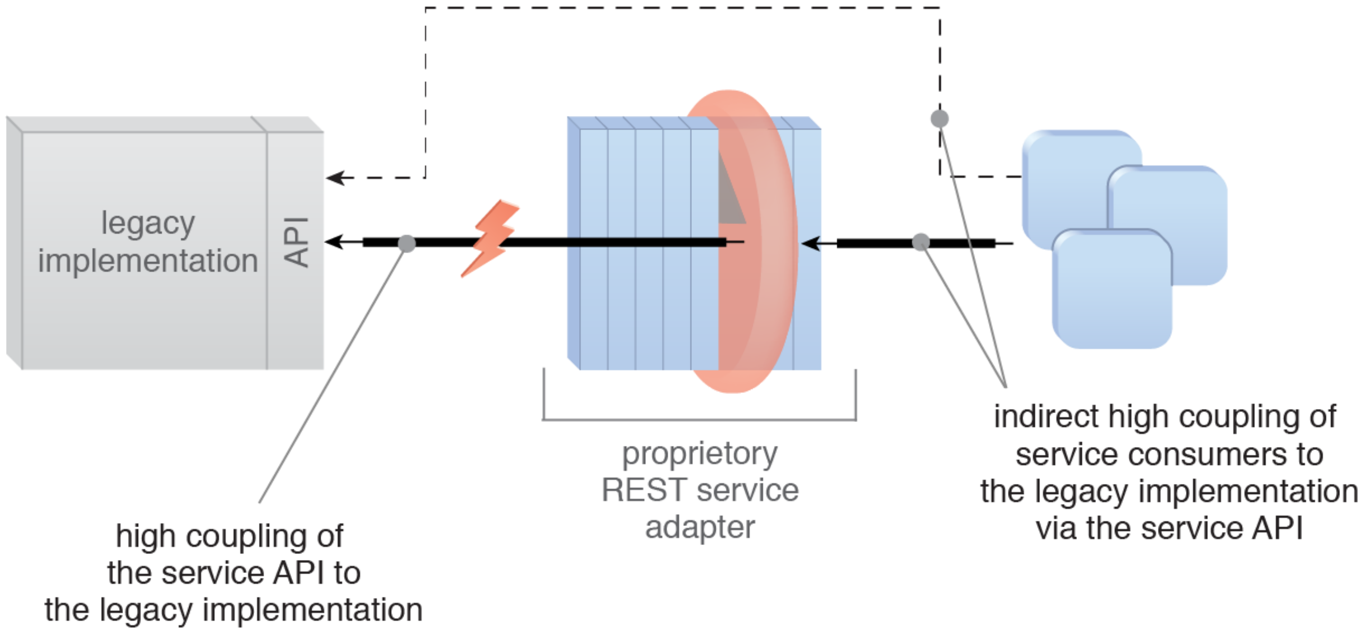


Tools used in your API lifecycle



APIs to legacy or vendor-locked systems

- Unlocking legacy applications using APIs
- Package legacy applications with APIs as an abstraction layer


legacy implementation
API
proprietory REST service adapter
indirect high coupling of service consumers to the legacy implementation via the service API
high coupling of the service API to the legacy implementation

Agenda

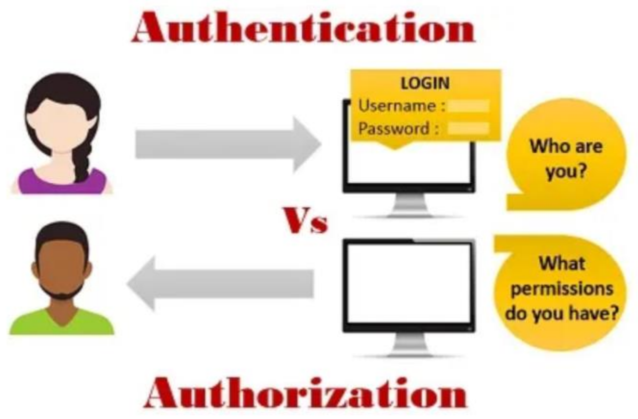
Design an API REST

- Authentication
- Synchronous vs Asynchronous APIs
- Health reports
- Documentation & versions

From the books to the practical and real use cases.

Authentication vs Authorization

- **Authentication** is the process of validating your credentials (such as user username and password) to verify your identity and whether you are the person you claim to be, or not.
- **Authorization** is the process to determine whether the authenticated user has access to a particular resource.

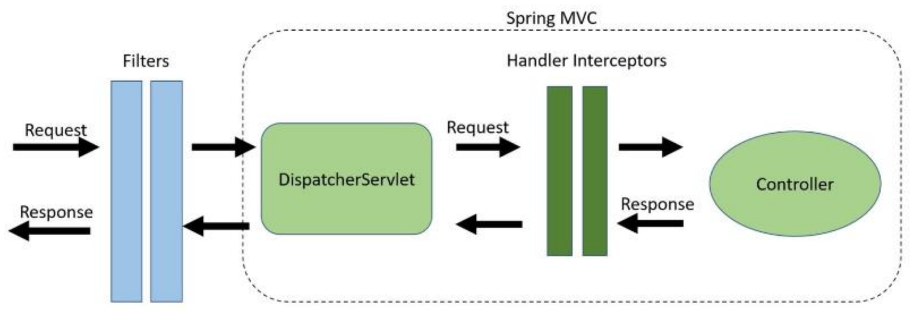


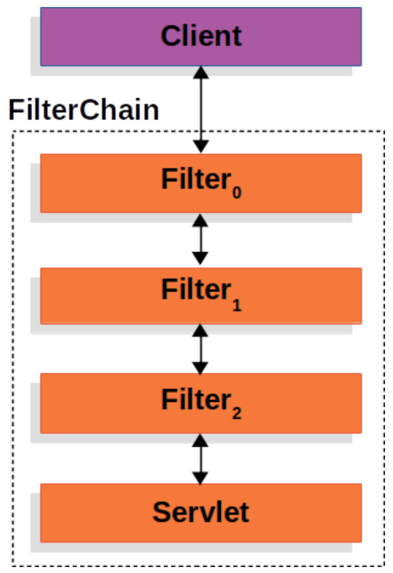
API Authentication methods

- **Basic Authentication:** Basic authentication is a simple, HTTP-based authentication scheme that allows clients to authenticate with a server by sending a username and password in plain text as part of the HTTP request.
- **API Key Authentication**: API key-based authentication involves sending an API key along with a request. An API key is a unique identifier that is issued by the API provider to authorized users or applications.
- **JWT-Based Authentication**: JWTs (JSON Web Tokens) are a compact and URL-safe means of representing claims to be transferred between parties. JWTs consist of three parts separated by dots: a header, a payload, and a signature. The header specifies the algorithm used to sign the token, the payload contains the claims, and the signature is used to verify the integrity of the token.
- **OAuth 2.0**: OAuth 2.0 is an authorization framework that allows users or applications to access resources from an API without giving the API access to their credentials, such as a username and password.
- **OIDC**: OpenID Connect (OIDC) is an authentication protocol that extends the OAuth 2.0 framework by providing an identity layer on top of it. OIDC enables users to authenticate with a web application using an identity provider, such as Google or Facebook.
- .. Several other authentication methods available!

Servlet Filter

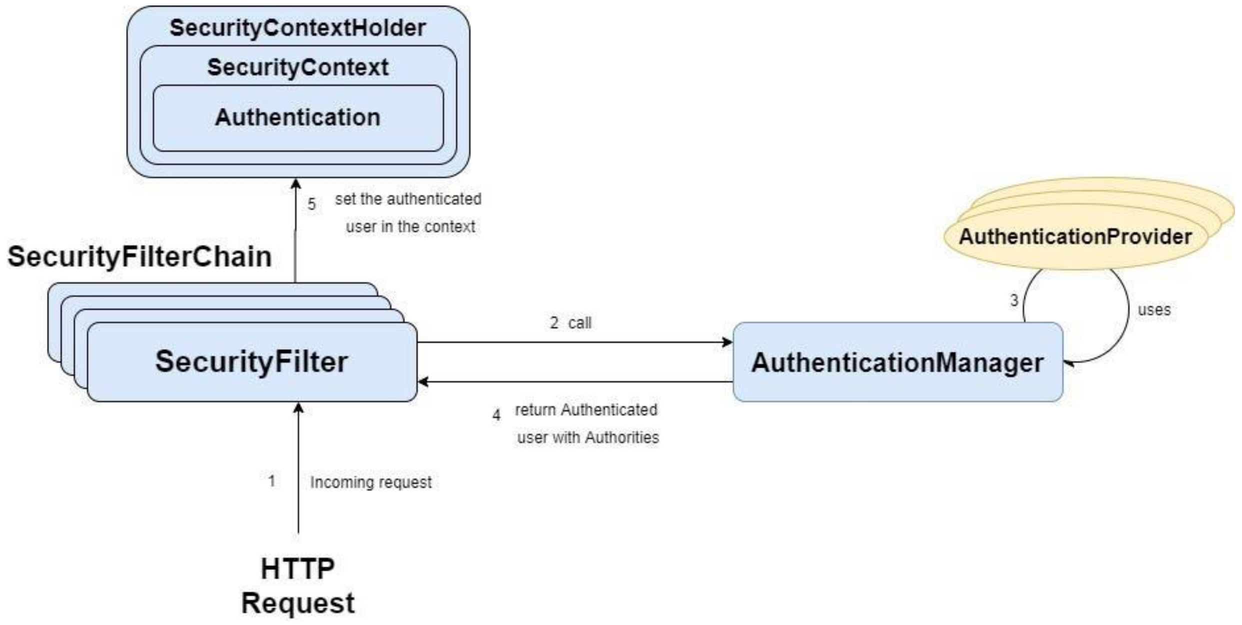
- Servlet Filter is used to **intercept the client request** and do some **pre-processing**. It can also intercept the response and do post-processing before sending to the client in web application. Common tasks:
  - Logging request parameters to log files; **authentication and authorization** of request for resources; formatting of request body or header before sending it to servlet; compressing the response data sent to the client; alter response by adding some cookies, header information etc.

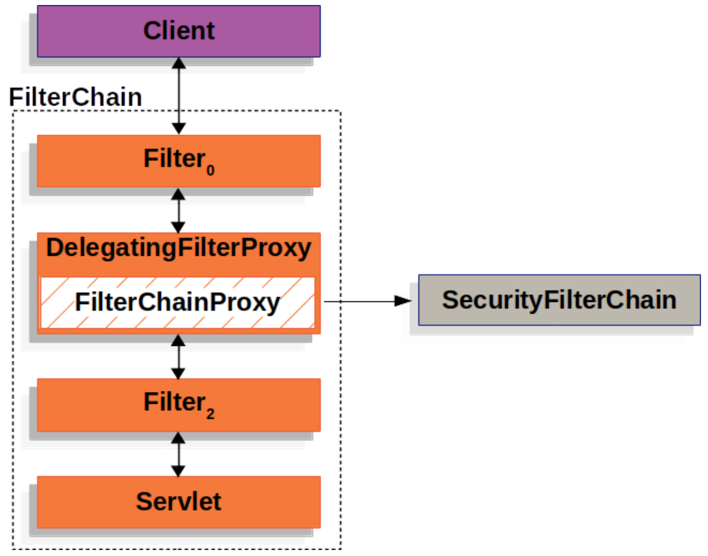




Security in Sprint Boot

- **SecurityFilterChain** : Spring Security maintains a filter chain internally where each of the filters is invoked in a specific order.





Authentication: using Filtering

If the authentication service lives in a different microservice, you can use for instance filtering for authentication.

```java
public class APIKeyFilter extends OncePerRequestFilter {
....
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
        //Perform call to other endpoint..

        HttpHeaders headers = new HttpHeaders();
        headers.set("x-api-key", request.getHeader("x-api-key"));
        headers.set("x-api-secret", request.getHeader("x-api-secret"));
        headers.setContentType(MediaType.APPLICATION_JSON);
        String requestBody = "{}";
        HttpEntity<String> requestInvoke =
                new HttpEntity<String>(requestBody, headers);
        ResponseEntity<String> responseInvoke
                = restTemplate.postForEntity(authEndPoint, requestInvoke, String.class);
        if (!responseInvoke.getStatusCode().equals(HttpStatus.OK)) {
            response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Invalid API key");
            return;
        }
        // Build user
        ....
        // Return user
        authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
        SecurityContextHolder.getContext().setAuthentication(authentication);
        filterChain.doFilter(request, response);
    }
```

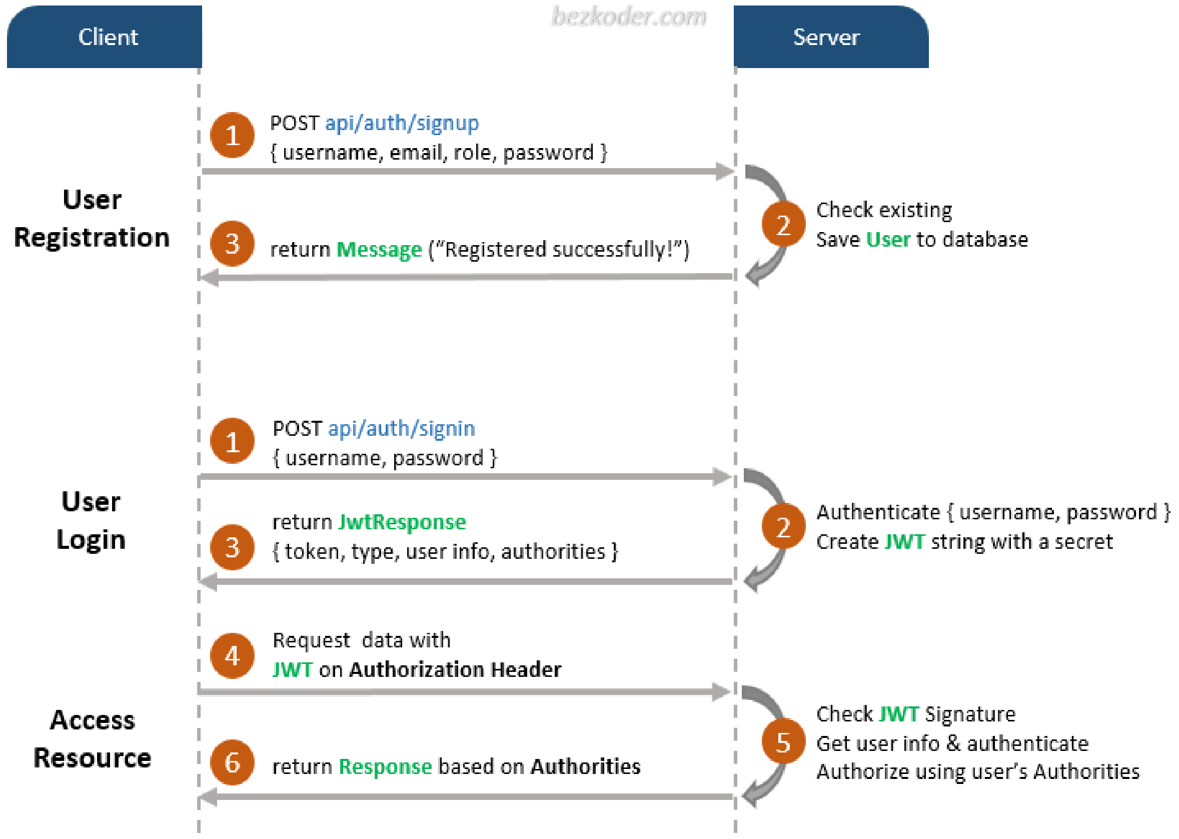
Call another authentication service

Practical Example

Authentication with JWT - Flow

Flow:

1. User makes a request to the service, seeking to create an account.
2. A user submits a request to the service to authenticate their account.
3. An authenticated user sends a request to access resources.



Client
bezkoder.com
Server
User Registration
1 POST api/auth/signup { username, email, role, password }
2 Check existing Save User to database
3 return Message ("Registered successfully!")
User Login
1 POST api/auth/signin { username, password }
2 Authenticate { username, password } Create JWT string with a secret
3 return JwtResponse { token, type, user info, authorities }
Access Resource
4 Request data with JWT on Authorization Header
5 Check JWT Signature Get user info & authenticate Authorize using user's Authorities
6 return Response based on Authorities

Example here: https://github.com/bastiao/springboot3-springsecurity6-jwt
Forked from buingoctruong

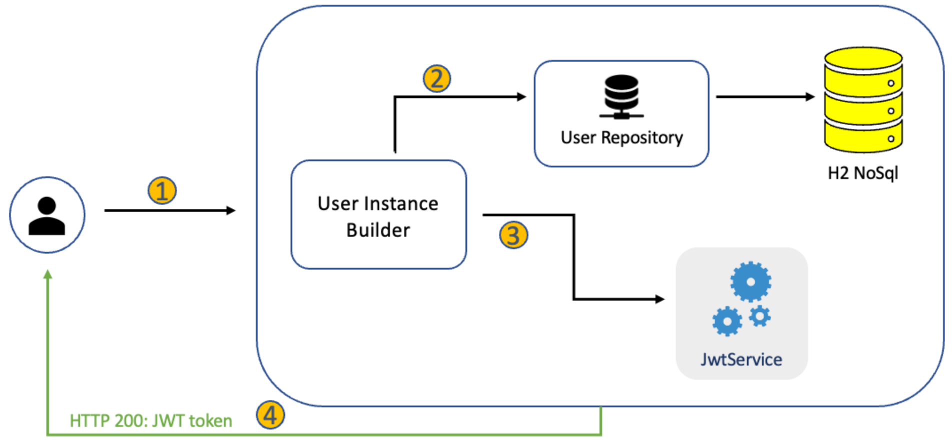
Practical Example

Authentication with JWT – Sign-up

1. The process starts when a user submits a request to the service. A user object is then generated from the request data, with the password being encoded using the PasswordEncoder.
2. The user object is stored in the database using the UserRepository, which leverages Spring Data JPA.
3. The JwtService is invoked to generate a JWT for the User object.
4. The JWT is encapsulated within a JSON response and subsequently returned to the user.

```
$ curl http://192.168.0.95:9001/api/v1/auth/signup -
X POST -H "Content-Type: application/json" -d
'{"email":"admin@localhost","password":"admin",
"firstName":"Admin", "lastName": "Localhost"}'
```

```
{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbkBsb2NhbGh
vc3QiLCJpYXQiOjE3MDEwOTM0OTcsImV4cCI6MTcwMTA5NDkz
N30.7HkDglcU1SbNyeZw6u9g-4wjzHvixZO-emvFOJY5zD8"}
```



Practical Example

Authentication with JWT - Sign-in & access

1. The process begins when a user sends a sign-in request to the Service. An Authentication object called UsernamePasswordAuthenticationToken is then generated, using the provided username and password.
2. The AuthenticationManager is responsible for authenticating the Authentication object, handling all necessary tasks. If the username or password is incorrect, an exception is thrown, and a response with HTTP Status 403 is returned to the user.
3. After successful authentication, an attempt is made to retrieve the user from the database. If the user does not exist in the database, a response with HTTP Status 403 is sent to the user. However, since we have already passed step 2 (authentication), this step is not crucial, as the user should already be in the database.
4. Once we have the user information, we call the JwtService to generate the JWT.
5. The JWT is then encapsulated in a JSON response and returned to the user.

Practical Example

Authentication with JWT – Resource access

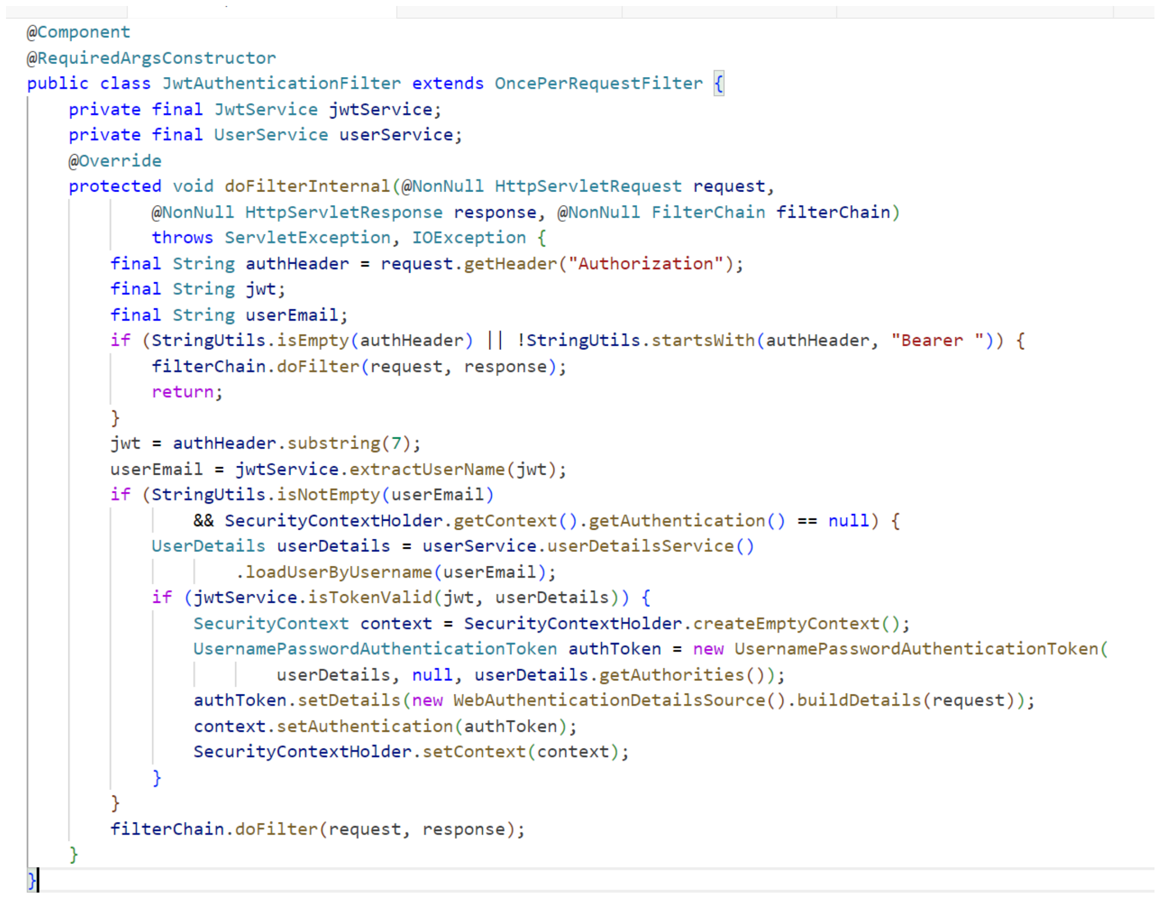
1. The process starts when the user sends a request to the Service. The request is first intercepted by JwtAuthenticationFilter, which is a custom filter integrated into the SecurityFilterChain.
2. As the API is secured, if the JWT is missing, a response with HTTP Status 403 is sent to the user.
3. When an existing JWT is received, JwtService is called to extract the userEmail from the JWT. If the userEmail cannot be extracted, a response with HTTP Status 403 is sent to the user.
4. If the userEmail can be extracted, it will be used to query the user's authentication and authorization information via UserDetailsService.
5. If the user's authentication and authorization information does not exist in the database, a response with HTTP Status 403 is sent to the user.
6. If the JWT is expired, a response with HTTP Status 403 is sent to the user.
7. Upon successful authentication, the user's details are encapsulated in a UsernamePasswordAuthenticationToken object and stored in the SecurityContextHolder.
8. The Spring Security Authorization process is automatically invoked.
9. The request is dispatched to the controller, and a successful JSON response is returned to the user.

Practical Example

Authentication with JWT - Sign-in & access

```
$ curl
http://192.168.0.95:900
1/api/v1/resource -X
GET -H "Authorization:
Bearer
eyJhbGciOiJIUzI1NiJ9.ey
JzdWIiOiJhZG1pbkBsb2Nhb
Ghvc3QiLCJpYXQiOjE3MDEw
OTM0OTcsImV4cCI6MTcwMTA
5NDkzN30.7HkDglcU1SbNye
Zw6u9g-4wjzHvixZO-
emvFOJY5zD8"
```

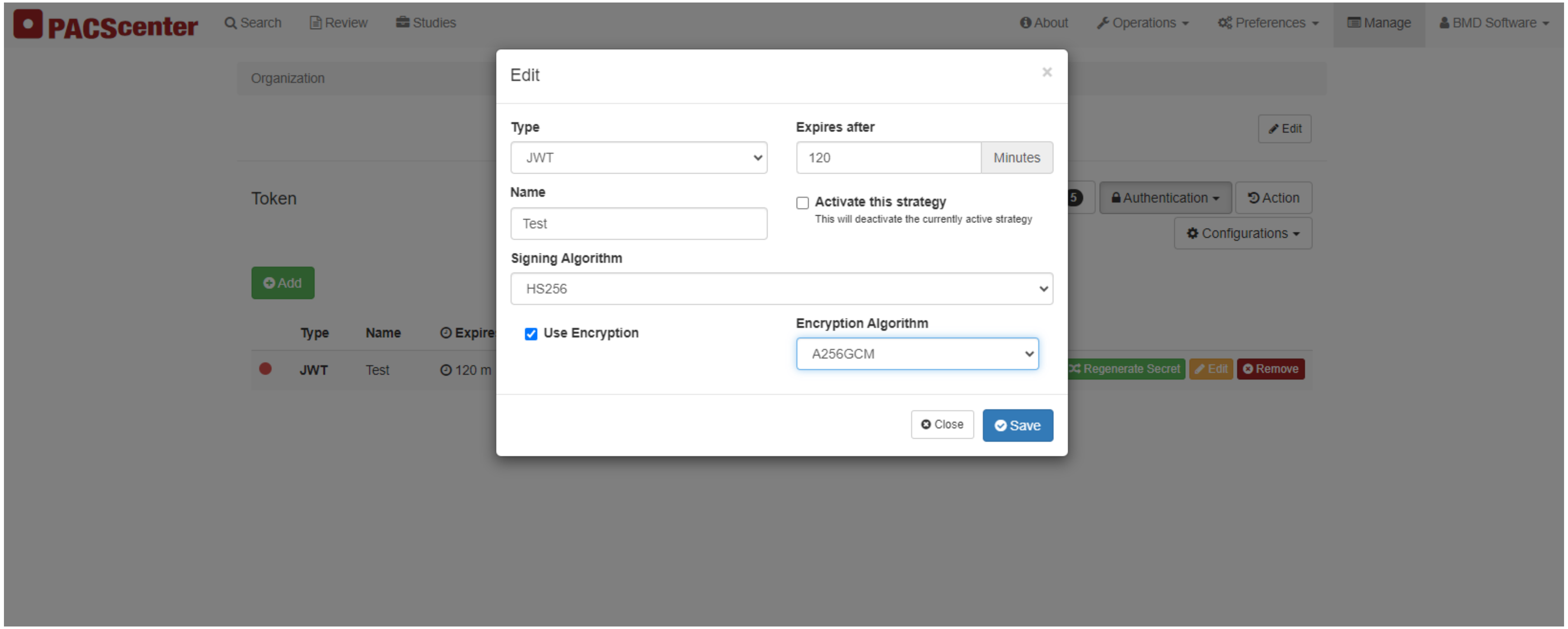
Response: Here is your resource



```java
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    private final JwtService jwtService;
    private final UserService userService;
    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response, @NonNull FilterChain filterChain)
            throws ServletException, IOException {
        final String authHeader = request.getHeader("Authorization");
        final String jwt;
        final String userEmail;
        if (StringUtils.isEmpty(authHeader) || !StringUtils.startsWith(authHeader, "Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }
        jwt = authHeader.substring(7);
        userEmail = jwtService.extractUserName(jwt);
        if (StringUtils.isNotEmpty(userEmail)
                && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = userService.userDetailsService()
                    .loadUserByUsername(userEmail);
            if (jwtService.isTokenValid(jwt, userDetails)) {
                SecurityContext context = SecurityContextHolder.createEmptyContext();
                UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());
                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                context.setAuthentication(authToken);
                SecurityContextHolder.setContext(context);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

Practical Example

Real use case: integrate with JWT



Edit

Type: JWT

Expires after: 120 Minutes

Name: Test

[ ] Activate this strategy
This will deactivate the currently active strategy

Signing Algorithm: HS256

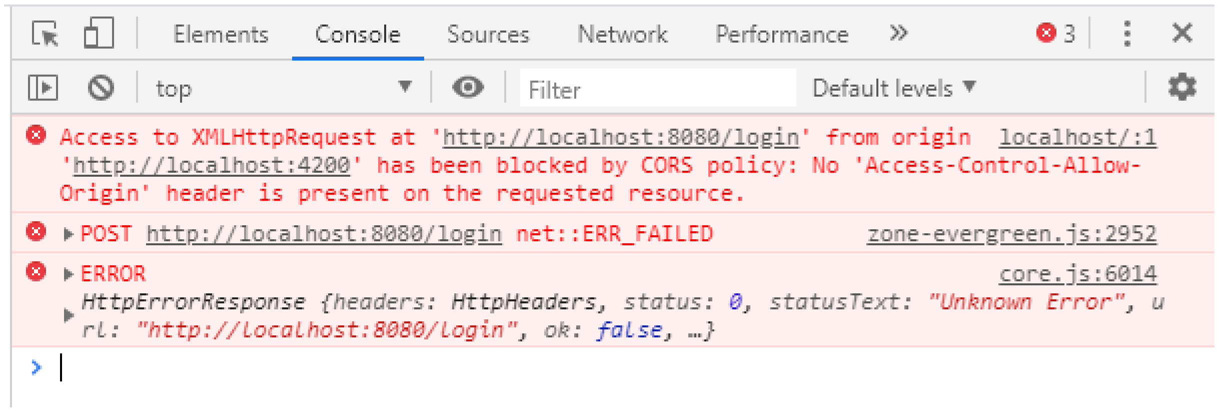
[✓] Use Encryption

Encryption Algorithm: A256GCM

Close | Save

	Type	Name	Expire
●	JWT	Test	120 m

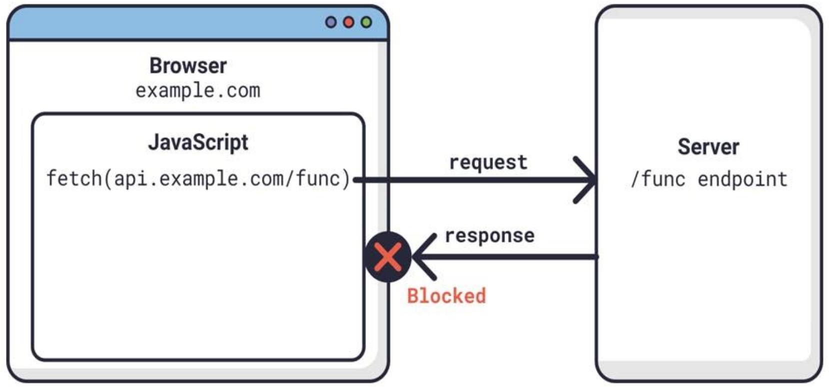
API: CORS (Cross-origin resource sharing) issue



```
Access to XMLHttpRequest at 'http://localhost:8080/login' from origin  localhost/:1
'http://localhost:4200' has been blocked by CORS policy: No 'Access-Control-Allow-
Origin' header is present on the requested resource.
▸POST http://localhost:8080/login net::ERR_FAILED          zone-evergreen.js:2952
▸ERROR                                                              core.js:6014
▸HttpErrorResponse {headers: HttpHeaders, status: 0, statusText: "Unknown Error", u
 rl: "http://localhost:8080/login", ok: false, …}
>
```

API: CORS (Cross-origin resource sharing)

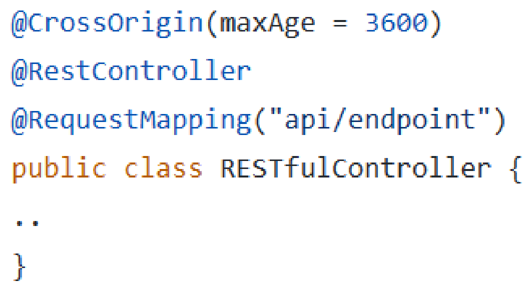
- Cross-origin resource sharing (CORS) is:
  - a mechanism for integrating applications
  - a a browser mechanism which enables controlled access to resources located outside of a given domain.



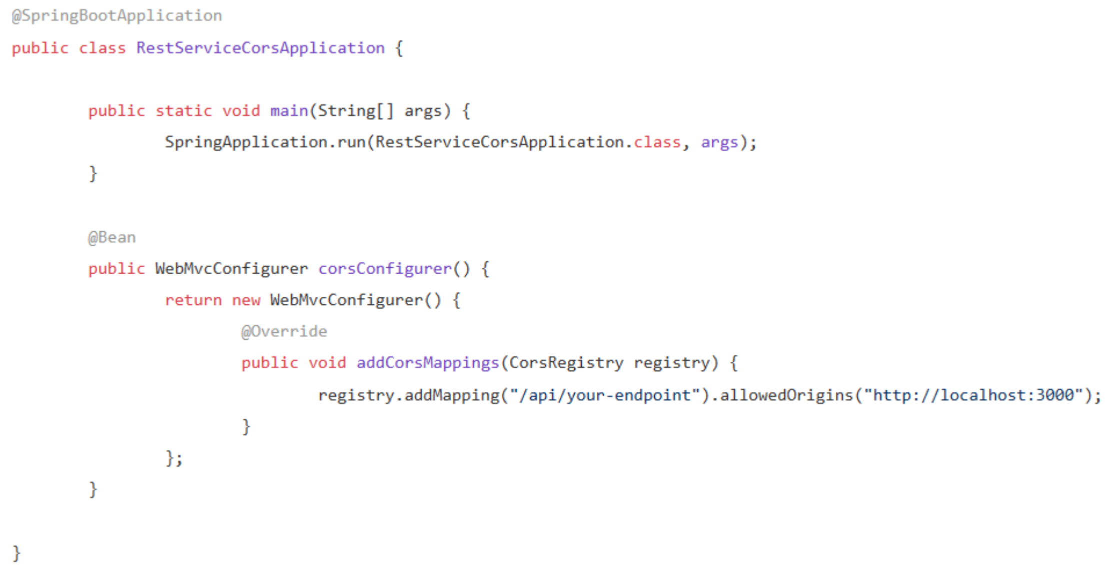
This can also arise in your lab work. For instance: http://localhost:3000 may raise also a CORS issue while accessing http://localhost:8080.

API: CORS (Cross-origin resource sharing)

- Global Cors Configurator: for entry point, by using *WebMVcConfigurer*
- @CrossOrigin on the Controller
  - Controller or Method Level



```
@CrossOrigin(maxAge = 3600)
@RestController
@RequestMapping("api/endpoint")
public class RESTfulController {
..
}
```



```
@SpringBootApplication
public class RestServiceCorsApplication {

	public static void main(String[] args) {
		SpringApplication.run(RestServiceCorsApplication.class, args);
	}

	@Bean
	public WebMvcConfigurer corsConfigurer() {
		return new WebMvcConfigurer() {
			@Override
			public void addCorsMappings(CorsRegistry registry) {
				registry.addMapping("/api/your-endpoint").allowedOrigins("http://localhost:3000");
			}
		};
	}

}
```

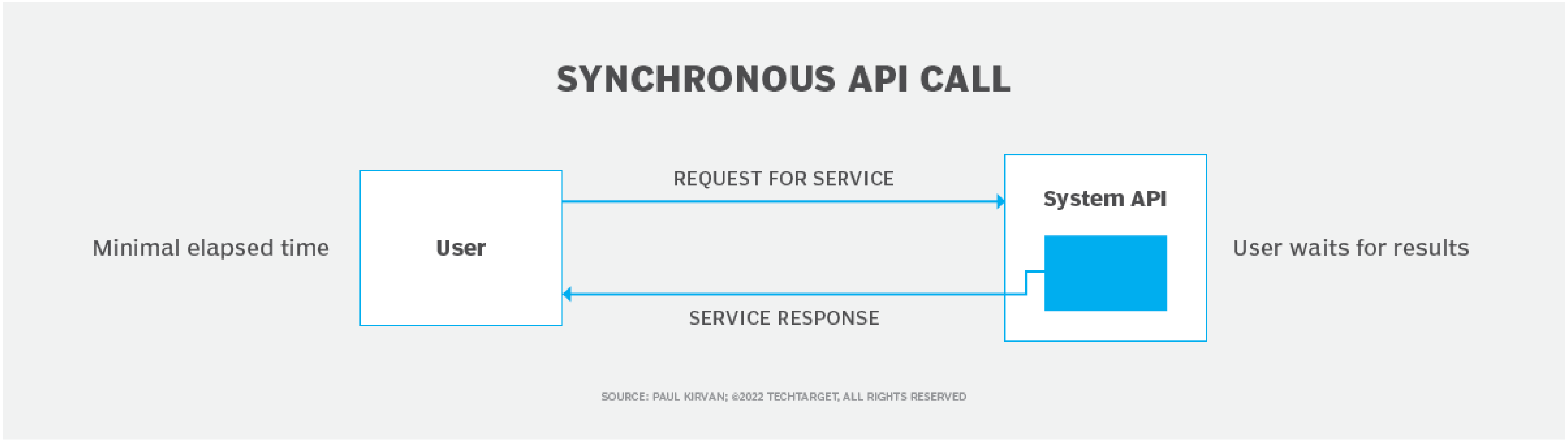
Agenda

Design an API REST

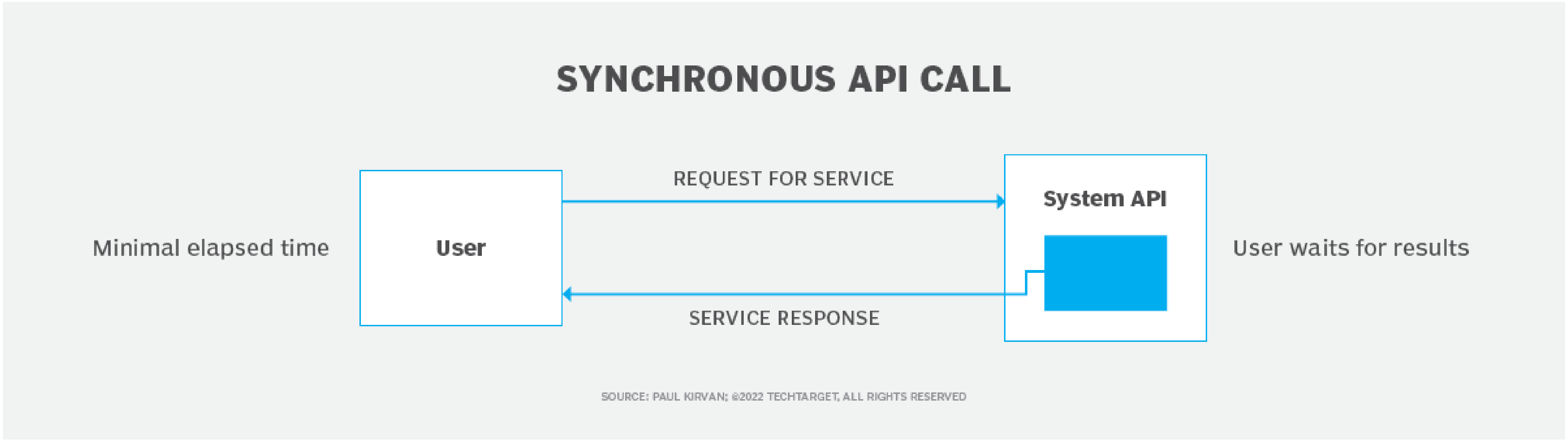
- Authentication
- Synchronous vs Asynchronous APIs
  - Sync API
  - Async API
  - WebSocket
- Health reports
- Documentation & versions

From the books to the practical and real use cases.

RESTful API: Synchronous

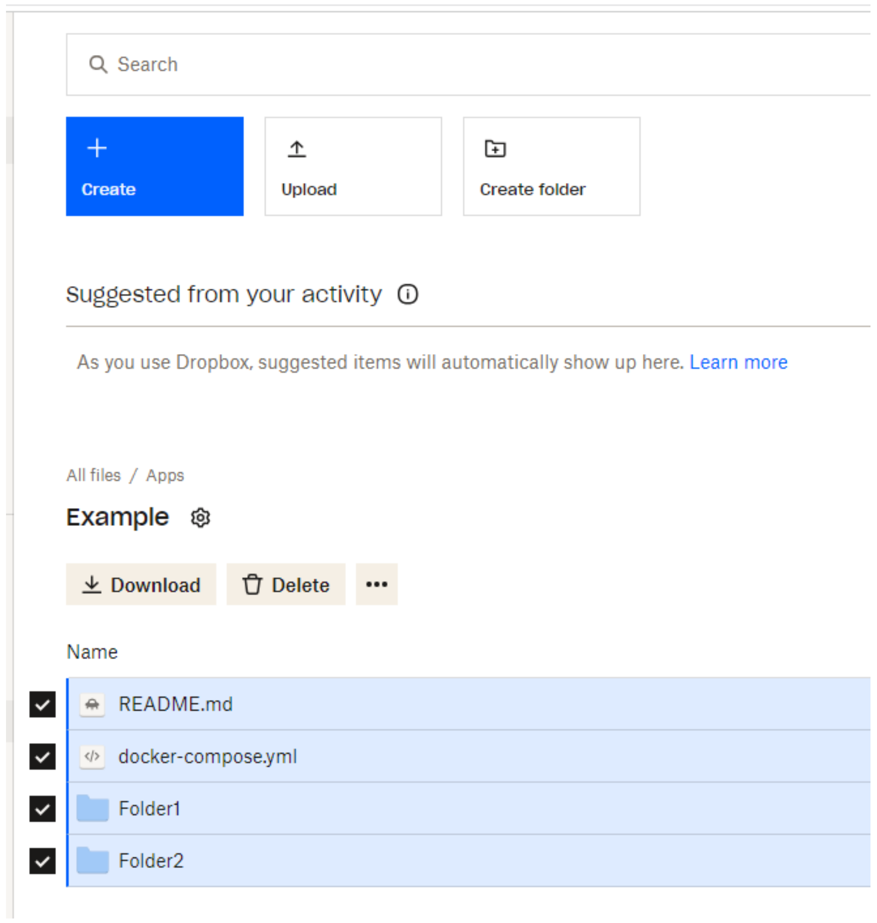


RESTful API: Synchronous



What happen if the service response takes longer? For instance 10 minutes?

Build a file system: download feature



Search

Create

Upload

Create folder

Suggested from your activity

As you use Dropbox, suggested items will automatically show up here. Learn more

All files / Apps

Example

Download

Delete

Name

- ✓ README.md
- ✓ docker-compose.yml
- ✓ Folder1
- ✓ Folder2

UC: As a user, I want the ability to download a list of files as a zip file.

Acceptance Criteria:

1. As a user, I should be able to select multiple files from a list.
2. Once I have selected the files, there should be an option to download them as a zip file.
3. The zip file should contain all the selected files in their original format.
4. If there are any errors during the download process, clear error messages should be displayed to inform me of the issue.

Practical Example

Build a file system: download feature

```java
@RestController
@RequestMapping("/api/files")
public class ExampleDownloadController {

    @PostMapping("/download")
    public ResponseEntity<byte[]> downloadFilesAsZip(@RequestBody List<String> fileNames) {
        try {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ZipOutputStream zipOut = new ZipOutputStream(baos);
            for (String fileName : fileNames) {
                ZipEntry zipEntry = new ZipEntry(fileName);
                zipOut.putNextEntry(zipEntry);
                // all zip magic here, can be also recursive, etc
            }
            zipOut.finish();
            zipOut.close();

            HttpHeaders headers = new HttpHeaders();
            headers.setContentType(MediaType.APPLICATION_OCTET_STREAM);
            headers.setContentDispositionFormData("attachment", "files.zip");
            return new ResponseEntity<>(baos.toByteArray(), headers, HttpStatus.OK);

        } catch (IOException e) {
            // Handle exceptions, log them, and return an error response...
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

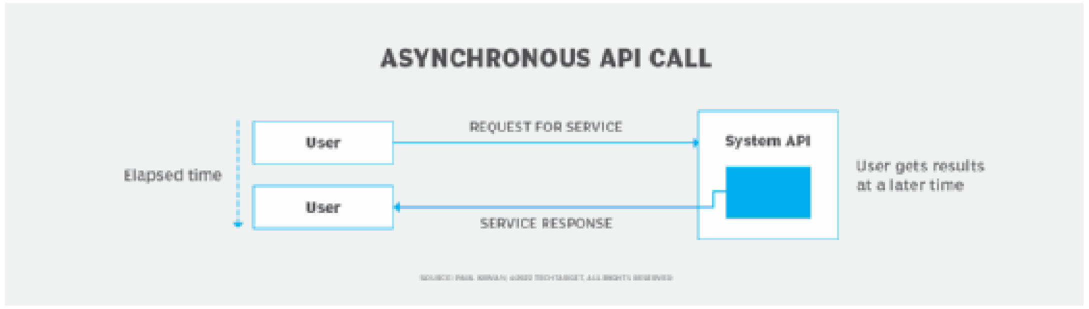
What happens if there is 1 million files in the directory?

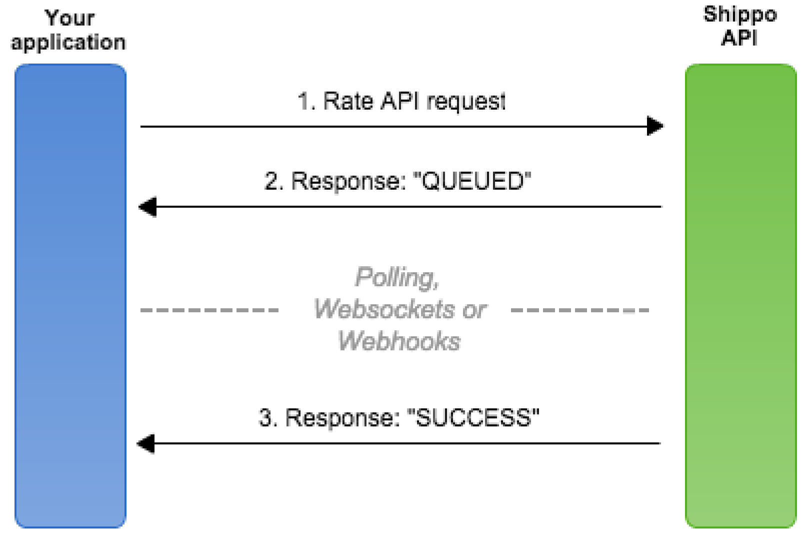
How long can it take?

Async API can be a solution!

Practical Example

RESTful API: Asynchronous





https://docs.goshippo.com/docs/api_concepts/asynchronusapicall/

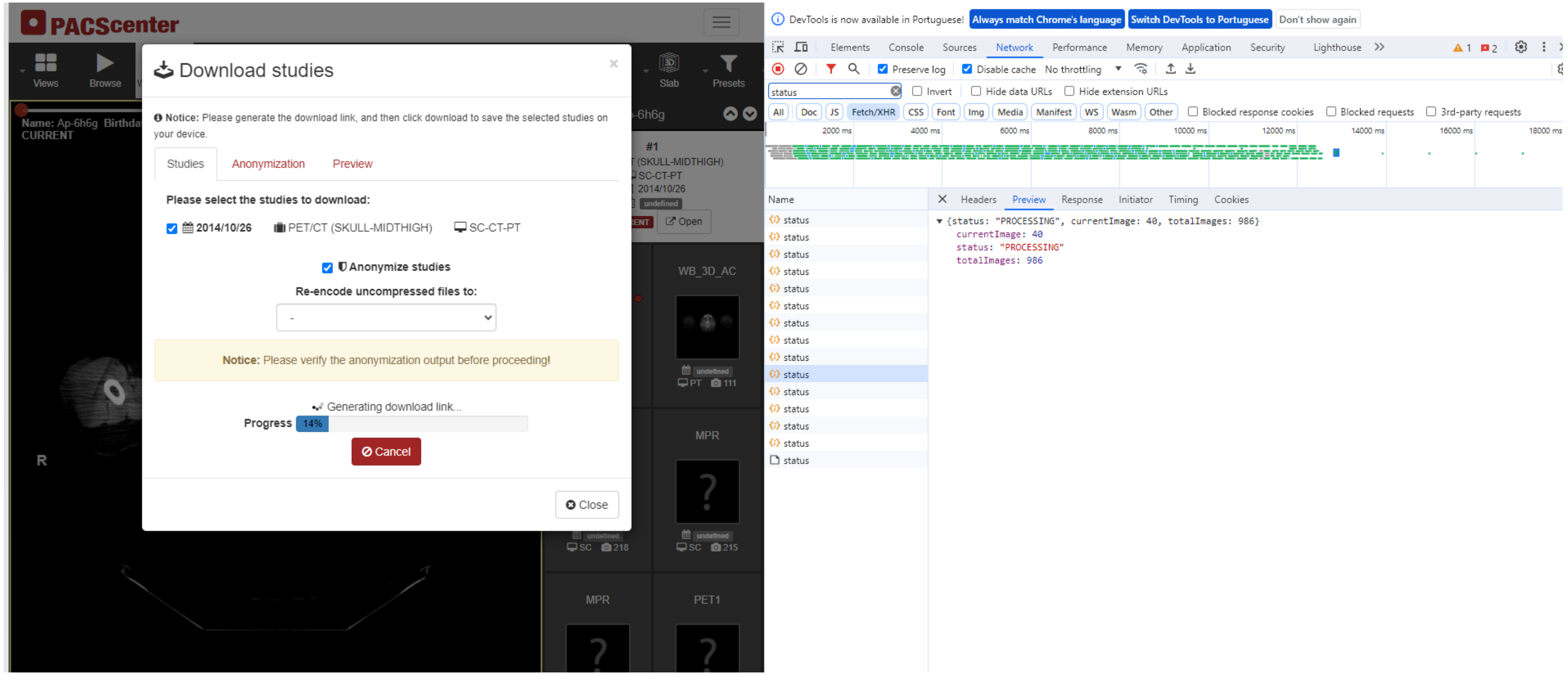
Build a file system: download feature (async)

```java
@RestController
@RequestMapping("/api/files")
public class FileController {
    private final AsyncTaskExecutor asyncTaskExecutor;
    private final DownloadTaskManager downloadTaskManager;

    @Autowired
    public FileController(ThreadPoolTaskExecutor asyncTaskExecutor, DownloadTaskManager downloadTaskManager) {
        this.asyncTaskExecutor = asyncTaskExecutor;
        this.downloadTaskManager = downloadTaskManager;
    }

    @PostMapping("/download")
    public ResponseEntity<DownloadTaskStatus> initiateFileDownload(@RequestBody List<String> fileNames) {
        String taskId = UUID.randomUUID().toString();
        CompletableFuture.runAsync(() -> downloadFilesAsZip(fileNames, taskId), asyncTaskExecutor);
        return ResponseEntity.ok(new DownloadTaskStatus(taskId, DownloadTaskStatus.Status.IN_PROGRESS));
    }

    @GetMapping("/download/status/{taskId}")
    public ResponseEntity<DownloadTaskStatus> getDownloadStatus(@PathVariable String taskId) {
        DownloadTaskStatus.Status status = downloadTaskManager.getTaskStatus(taskId);
        if (status == null) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(new DownloadTaskStatus(taskId, status));
    }
```

```java
@Async
public void downloadFilesAsZip(List<String> fileNames, String taskId) {
    try {
        downloadTaskManager.setTaskStatus(taskId, DownloadTaskStatus.Status.IN_PROGRESS);

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        // ... (Same as previous example)

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_OCTET_STREAM);
        headers.setContentDispositionFormData("attachment", "files.zip");

        downloadTaskManager.setTaskStatus(taskId,
                                          DownloadTaskStatus.Status.COMPLETED);
        downloadTaskManager.setTaskResult(taskId,
                                          new ResponseEntity<>(baos.toByteArray(),
                                                               headers,
                                                               HttpStatus.OK));

    } catch (IOException e) {
        // Handle exceptions, log them, and update task status accordingly
        downloadTaskManager.setTaskStatus(taskId, DownloadTaskStatus.Status.FAILED);
    }
}
```

Practical Example

Real Use Case



Download studies

Notice: Please generate the download link, and then click download to save the selected studies on your device.

Studies | Anonymization | Preview

Please select the studies to download:

☑ **2014/10/26** PET/CT (SKULL-MIDTHIGH) SC-CT-PT

☑ **Anonymize studies**

Re-encode uncompressed files to: -

Notice: Please verify the anonymization output before proceeding!

Generating download link...

Progress 14%

Cancel

Close

DevTools is now available in Portuguese! Always match Chrome's language | Switch DevTools to Portuguese | Don't show again

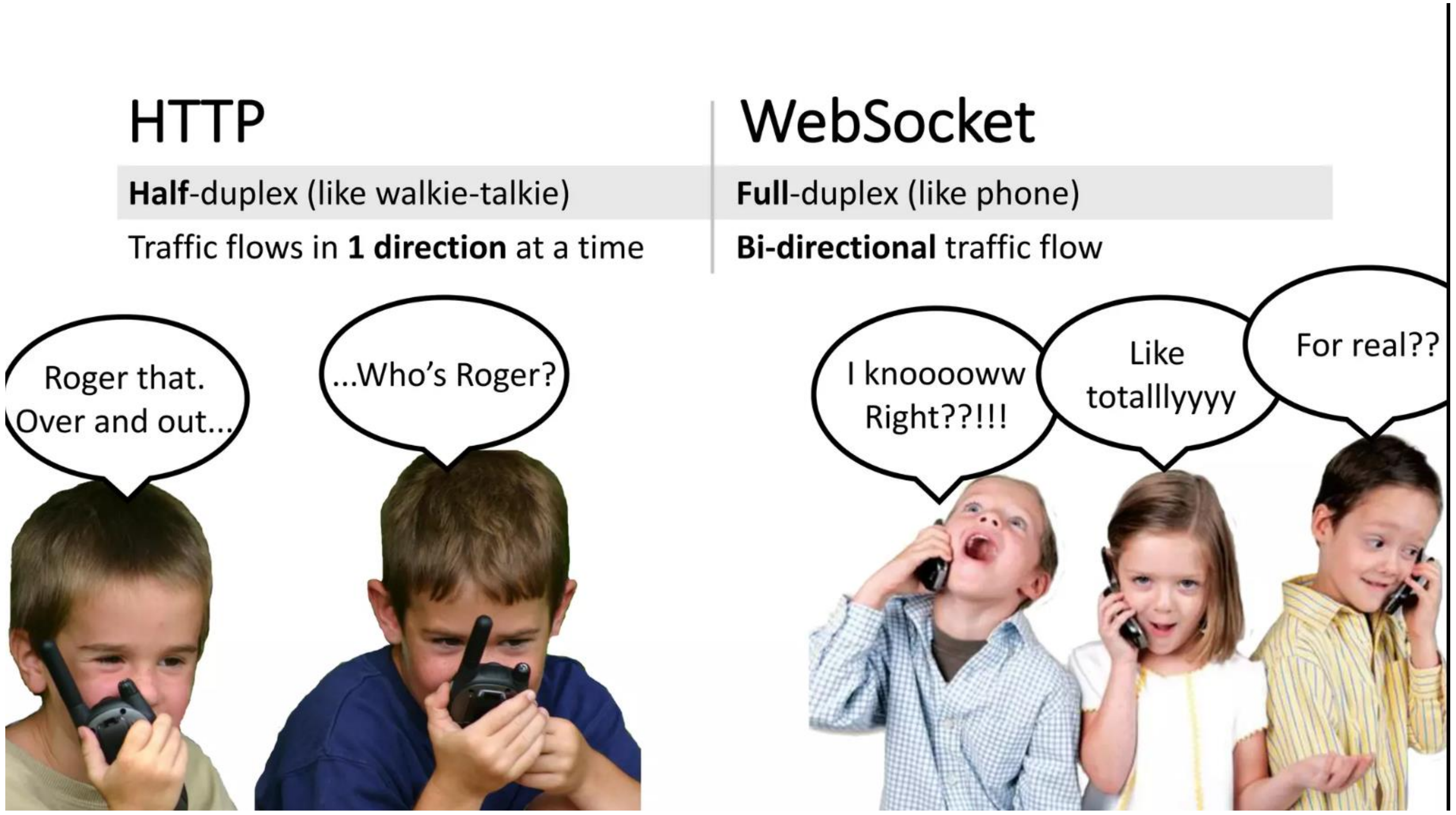
Filter: status

```
▼ {status: "PROCESSING", currentImage: 40, totalImages: 986}
    currentImage: 40
    status: "PROCESSING"
    totalImages: 986
```

WebSockets

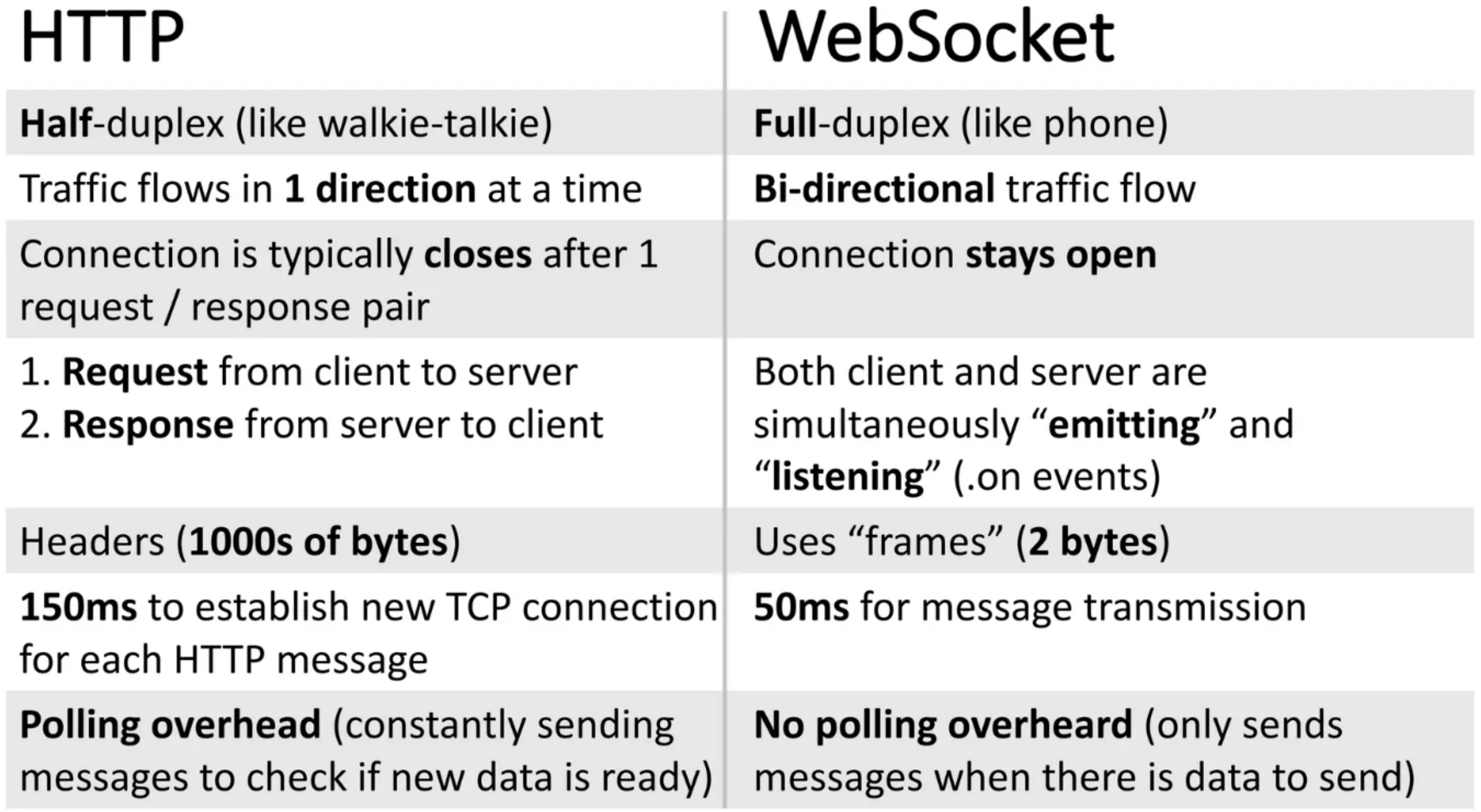
- **WebSocket** is a communication protocol that provides **full-duplex** communication channels over a single, long-lived connection. In the context of web development, WebSocket allows for real-time communication between a client and a server.
- Persistent 2-way connection between browser and server
- Common use cases:
  - Chat
  - Notifications
  - Online Game
  - Financial Trading
  - Live Maps
  - Collaborations
  - ..

HTTP RESTful API vs WebSocket APIs



HTTP	WebSocket
Half-duplex (like walkie-talkie)	**Full**-duplex (like phone)
Traffic flows in **1 direction** at a time	**Bi-directional** traffic flow

HTTP RESTful API vs WebSocket APIs



HTTP	WebSocket
Half-duplex (like walkie-talkie)	**Full**-duplex (like phone)
Traffic flows in **1 direction** at a time	**Bi-directional** traffic flow
Connection is typically **closes** after 1 request / response pair	Connection **stays open**
1. **Request** from client to server 2. **Response** from server to client	Both client and server are simultaneously "**emitting**" and "**listening**" (.on events)
Headers (**1000s of bytes**)	Uses "frames" (**2 bytes**)
150ms to establish new TCP connection for each HTTP message	**50ms** for message transmission
Polling overhead (constantly sending messages to check if new data is ready)	**No polling overheard** (only sends messages when there is data to send)

From Julien LaPointe

WebSocket in Spring Boot

1 Add Dependency

```xml
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
```

2 Configure WebSocket Broker

```java
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

  @Override
  public void configureMessageBroker(MessageBrokerRegistry config) {
    config.enableSimpleBroker("/topic");
    config.setApplicationDestinationPrefixes("/app");
  }

  @Override
  public void registerStompEndpoints(StompEndpointRegistry registry) {
    registry.addEndpoint("/gs-guide-websocket");
  }

}
```

3 Write a Controller (*/hello* forward to topic)

```java
@Controller
public class GreetingController {


  @MessageMapping("/hello")
  @SendTo("/topic/greetings")
  public Greeting greeting(HelloMessage message) throws Exception {
    Thread.sleep(1000); // simulated delay
    return new Greeting("Hello, " + HtmlUtils.htmlEscape(message.getName()) + "!");
  }

}
```

4 Write frontend / web client for WebSocket

```javascript
const stompClient = new StompJs.Client({
    brokerURL: 'ws://localhost:8080/gs-guide-websocket'
});

stompClient.onConnect = (frame) => {
    setConnected(true);
    console.log('Connected: ' + frame);
    stompClient.subscribe('/topic/greetings', (greeting) => {
        showGreeting(JSON.parse(greeting.body).content);
    });
};
```

Practical Example

https://www.baeldung.com/spring-boot-scheduled-websocket

Agenda

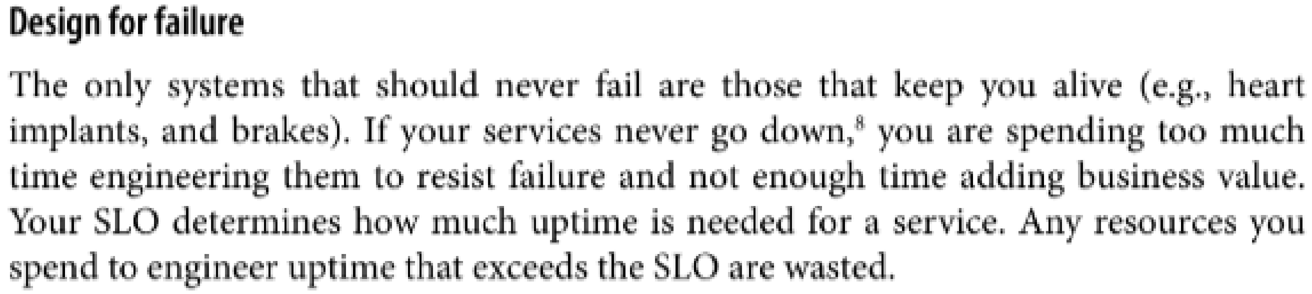
Design an API REST

- Authentication
- Synchronous vs Asynchronous APIs
- **Health reports**
- Documentation & versions

From the books to the practical and real use cases.

Health reporting

- Health reporting informs us of application life cycle state.
- It is different for telemetry data, which informs about application business objectives.



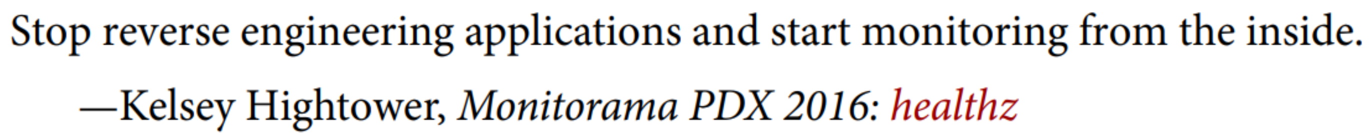
Design for failure

The only systems that should never fail are those that keep you alive (e.g., heart implants, and brakes). If your services never go down,[8] you are spending too much time engineering them to resist failure and not enough time adding business value. Your SLO determines how much uptime is needed for a service. Any resources you spend to engineer uptime that exceeds the SLO are wasted.

"Cloud Native Infrastructure", Justin Garrison & Kris Nova

How this can be implemented?

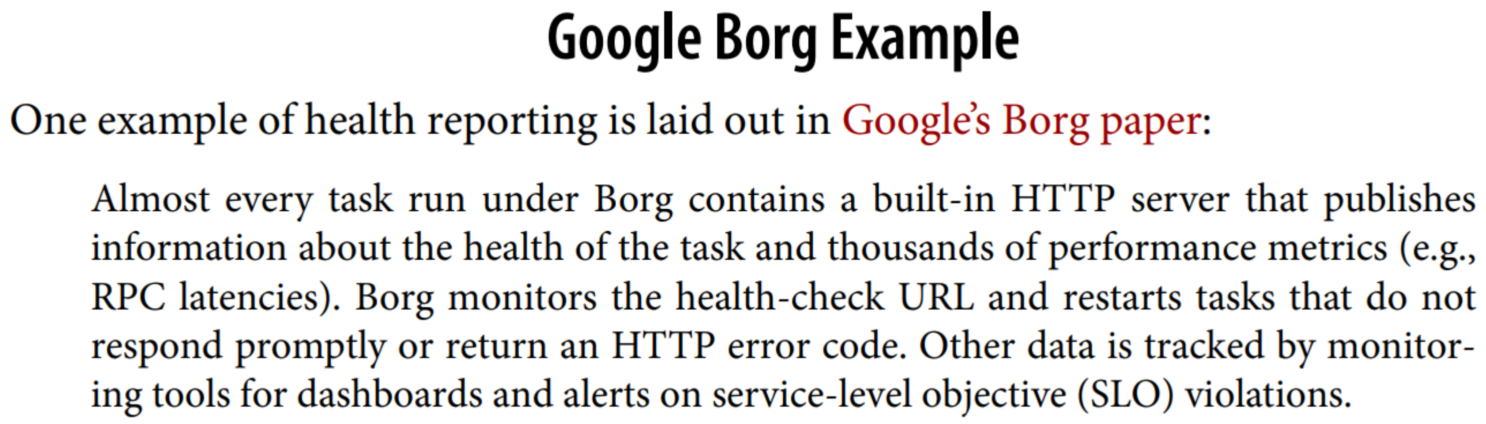
Health reporting



Stop reverse engineering applications and start monitoring from the inside.

—Kelsey Hightower, *Monitorama PDX 2016: healthz*

- Long road to achieve high availability
- Everything can start by an API



Google Borg Example

One example of health reporting is laid out in Google's Borg paper:

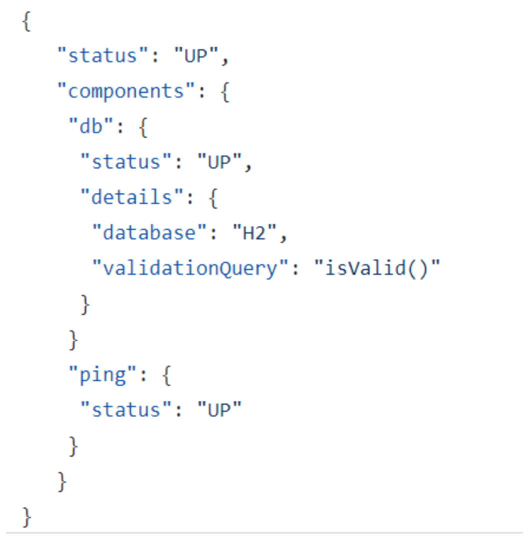
Almost every task run under Borg contains a built-in HTTP server that publishes information about the health of the task and thousands of performance metrics (e.g., RPC latencies). Borg monitors the health-check URL and restarts tasks that do not respond promptly or return an HTTP error code. Other data is tracked by monitoring tools for dashboards and alerts on service-level objective (SLO) violations.

Health reporting

- You can have a /health or /status API.
- Response examples:

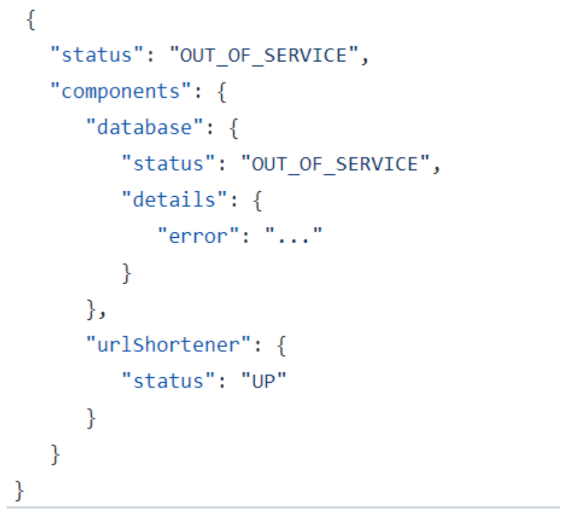
Basic URL health check:

Status: OK

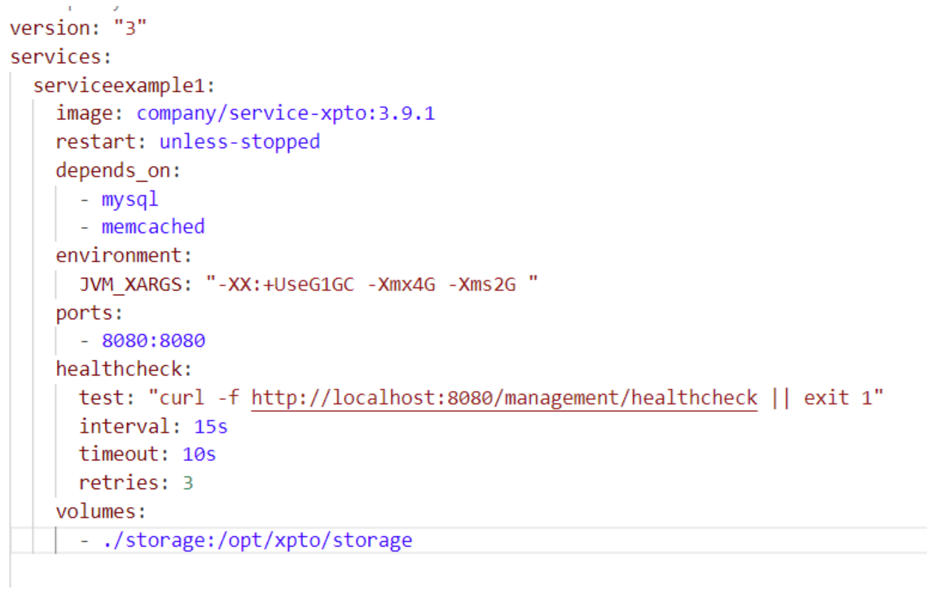


```
{
   "status": "UP",
   "components": {
    "db": {
     "status": "UP",
     "details": {
      "database": "H2",
      "validationQuery": "isValid()"
     }
    }
    "ping": {
     "status": "UP"
    }
   }
}
```

Status: **Error**



```
{
  "status": "OUT_OF_SERVICE",
  "components": {
     "database": {
        "status": "OUT_OF_SERVICE",
        "details": {
           "error": "..."
        }
     },
     "urlShortener": {
        "status": "UP"
     }
  }
}
```



```
version: "3"
services:
  serviceexample1:
    image: company/service-xpto:3.9.1
    restart: unless-stopped
    depends_on:
      - mysql
      - memcached
    environment:
      JVM_XARGS: "-XX:+UseG1GC -Xmx4G -Xms2G "
    ports:
      - 8080:8080
    healthcheck:
      test: "curl -f http://localhost:8080/management/healthcheck || exit 1"
      interval: 15s
      timeout: 10s
      retries: 3
    volumes:
      - ./storage:/opt/xpto/storage
```

Practical Example

Agenda

Design an API REST

- Authentication
- Synchronous vs Asynchronous APIs
- Health reports
- Documentation & versions

From the books to the practical and real use cases.

Documentation APIs – with OpenAPI

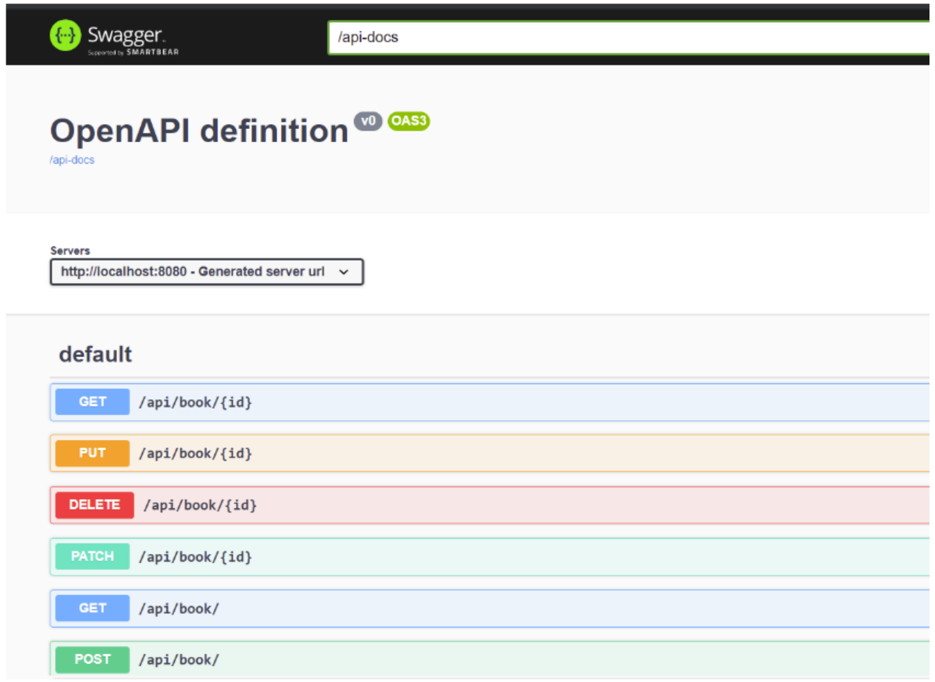
- The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to HTTP APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.
- A self-contained or composite resource which defines or describes an API or elements of an API. The OpenAPI document MUST contain at least one paths field, a components field or a webhooks field.
- YAML or JSON format

Documentation APIs

Spring Boot 3.x requires to use version 2 of springdoc-openapi:

```xml
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.2.0</version>
</dependency>
```

Document code, design API and it will automatically generate API definition



OpenAPI definition v0 OAS3

/api-docs

Servers

http://localhost:8080 - Generated server url

default

GET /api/book/{id}

PUT /api/book/{id}

DELETE /api/book/{id}

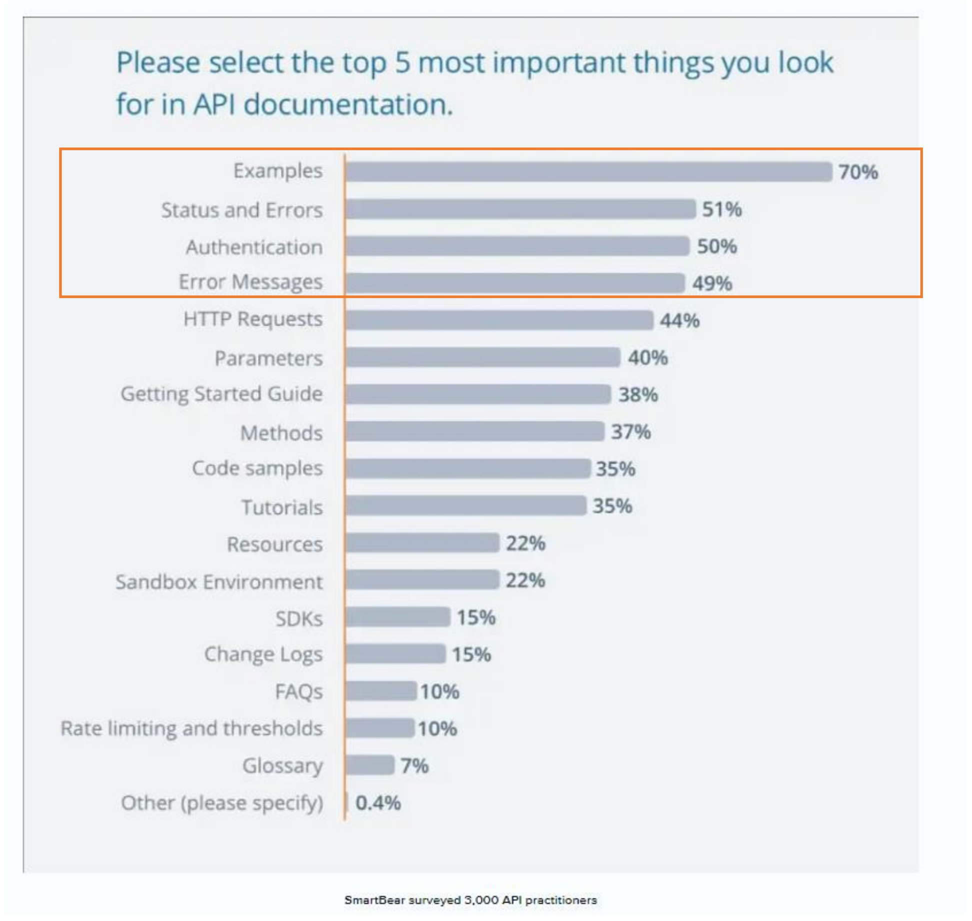
PATCH /api/book/{id}

GET /api/book/

POST /api/book/

https://www.baeldung.com/spring-rest-openapi-documentation

Documention of APIs

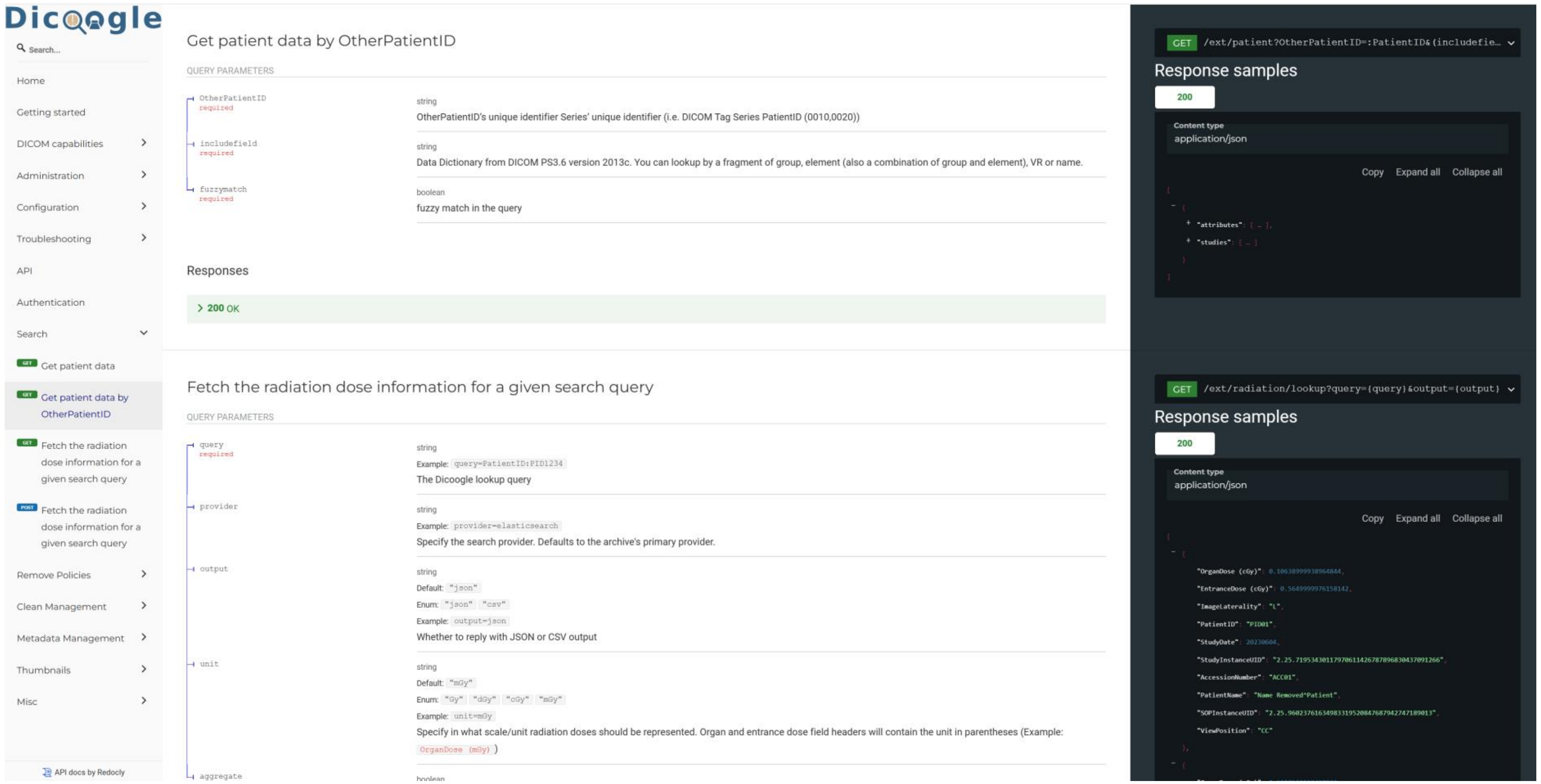


Version Software & API

- No real *standard* for set version in APIs, but there is good practices and strategies:
  - Version by: url path, query Parameter or header Version.
  - Communication of changes clearly
  - Backward compatibility *(if possible)*
  - Deprecate old versions gradually
  - Implement a Versioning Strategy That Suits Your Needs
- According to Semver, you:
  - *MAJOR version when you make incompatible API changes*
  - *MINOR version when you add functionality in a backward compatible manner*
  - *PATCH version when you make backward compatible bug fixes*

From www.semver.org

Real use case: document API



Get patient data by OtherPatientID

QUERY PARAMETERS

`OtherPatientID` required	string OtherPatientID's unique identifier Series' unique identifier (i.e. DICOM Tag Series PatientID (0010,0020))
`includefield` required	string Data Dictionary from DICOM PS3.6 version 2013c. You can lookup by a fragment of group, element (also a combination of group and element), VR or name.
`fuzzymatch` required	boolean fuzzy match in the query

Responses

> 200 OK

Fetch the radiation dose information for a given search query

QUERY PARAMETERS

`query` required	string Example: `query=PatientID:PID1234` The Dicoogle lookup query
`provider`	string Example: `provider=elasticsearch` Specify the search provider. Defaults to the archive's primary provider.
`output`	string Default: `"json"` Enum: `"json"` `"csv"` Example: `output=json` Whether to reply with JSON or CSV output
`unit`	string Default: `"mGy"` Enum: `"Gy"` `"dGy"` `"cGy"` `"mGy"` Example: `unit=mGy` Specify in what scale/unit radiation doses should be represented. Organ and entrance dose field headers will contain the unit in parentheses (Example: `OrganDose (mGy)`)
`aggregate`	boolean

GET `/ext/patient?OtherPatientID=:PatientID&(includefie…`

Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
[
 - {
    + "attributes": [ … ],
    + "studies": [ … ]
   }
]
```

GET `/ext/radiation/lookup?query={query}&output={output}`

Response samples

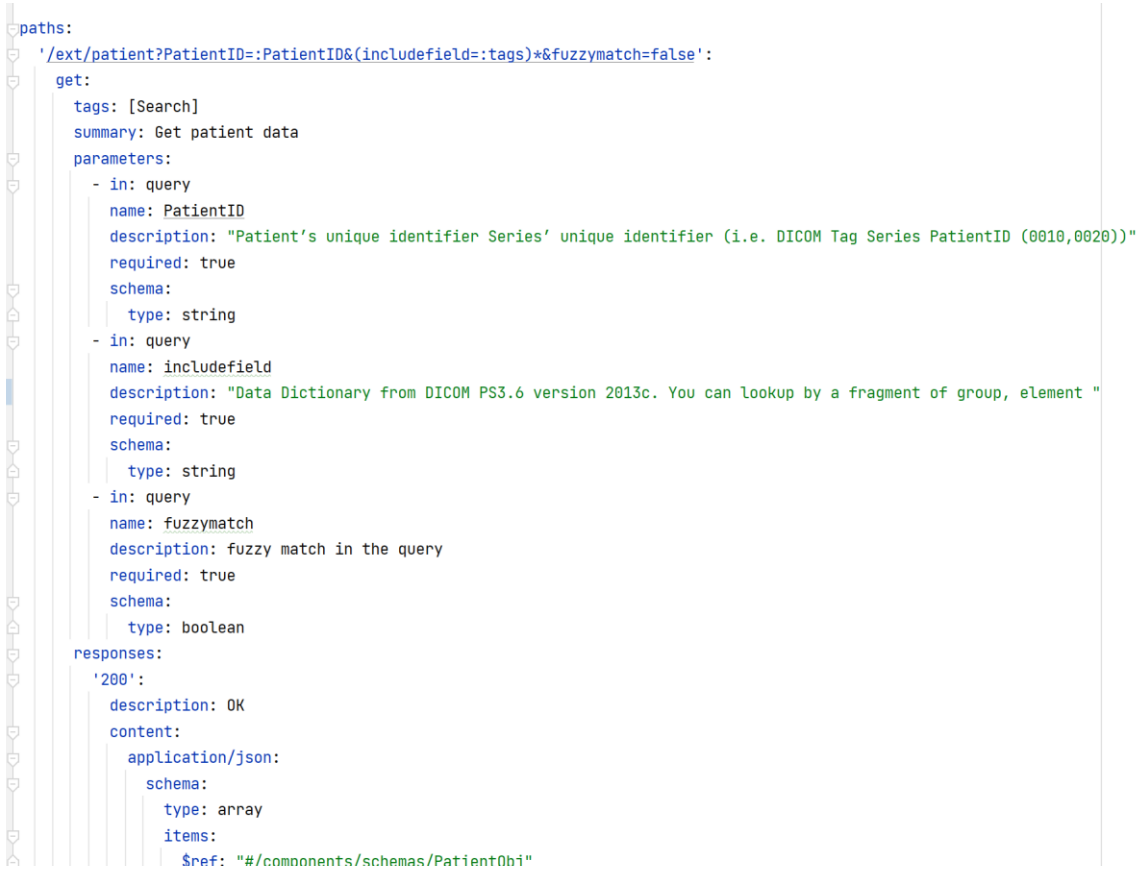
200

Content type
application/json

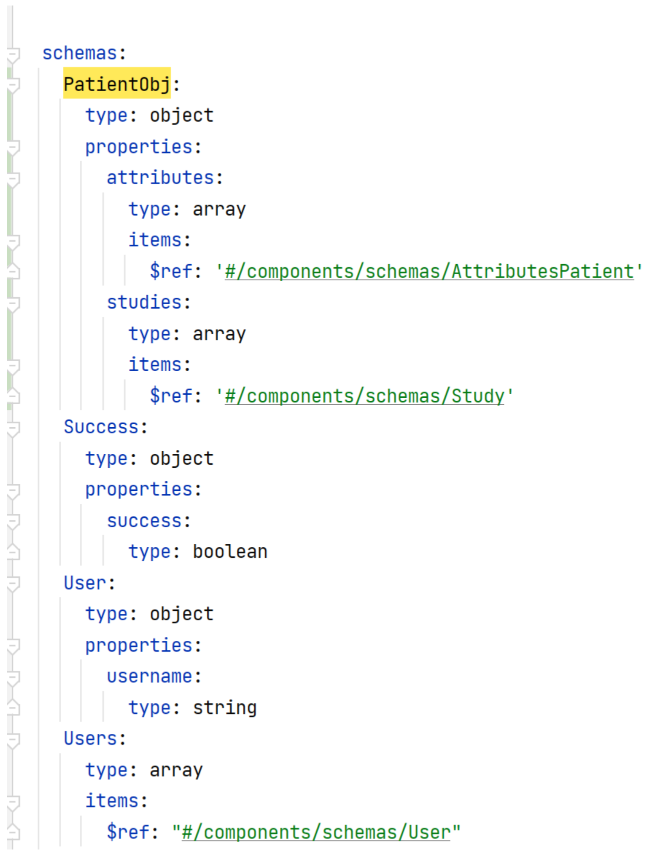
Copy Expand all Collapse all

```
[
 - {
      "OrganDose (cGy)": 0.10638999938964844,
      "EntranceDose (cGy)": 0.5649999976158142,
      "ImageLaterality": "L",
      "PatientID": "PID01",
      "StudyDate": 20230604,
      "StudyInstanceUID": "2.25.71953430117970611426787896830437091266",
      "AccessionNumber": "ACC01",
      "PatientName": "Name Removed^Patient",
      "SOPInstanceUID": "2.25.96023761634983319520847687942747189013",
      "ViewPosition": "CC"
   },
 - {
```

Real use case: document API



```yaml
paths:
  '/ext/patient?PatientID=:PatientID&(includefield=:tags)*&fuzzymatch=false':
    get:
      tags: [Search]
      summary: Get patient data
      parameters:
        - in: query
          name: PatientID
          description: "Patient’s unique identifier Series’ unique identifier (i.e. DICOM Tag Series PatientID (0010,0020))"
          required: true
          schema:
            type: string
        - in: query
          name: includefield
          description: "Data Dictionary from DICOM PS3.6 version 2013c. You can lookup by a fragment of group, element "
          required: true
          schema:
            type: string
        - in: query
          name: fuzzymatch
          description: fuzzy match in the query
          required: true
          schema:
            type: boolean
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: "#/components/schemas/PatientObj"
```



```yaml
schemas:
  PatientObj:
    type: object
    properties:
      attributes:
        type: array
        items:
          $ref: '#/components/schemas/AttributesPatient'
      studies:
        type: array
        items:
          $ref: '#/components/schemas/Study'
  Success:
    type: object
    properties:
      success:
        type: boolean
  User:
    type: object
    properties:
      username:
        type: string
  Users:
    type: array
    items:
      $ref: "#/components/schemas/User"
```

Biography & References

- “Cloud Native Infrastructure”, Justin Garrison & Kris Nova
- “Building Evolutionary Architectures”, Neal Ford, Rebecca Parsons & Patria Kua
- “Learn Microservices with Spring Boot”, Moises Macero
- https://www.baeldung.com/spring-boot-shared-secret-authentication
- https://docs.spring.io/spring-security/reference/servlet/authorization/authorize-http-requests.html
- https://github.com/buingoctruong/springboot3-springsecurity6-jwt/tree/master/src/main/java/com/truongbn/security
- https://frontegg.com/guides/api-authentication-api-authorization