PAPER • **OPEN ACCESS**

The Single Page Application architecture when developing secure Web services

To cite this article: D V Kornienko *et al* 2021 *J. Phys.: Conf. Ser.* **2091** 012065

View the article online for updates and enhancements.

The Single Page Application architecture when developing secure Web services

D V Kornienko, S V Mishina and M O Melnikov

Bunin Yelets State University, 28 Kommunarov street, Yelets 399770, Russia

Abstract. The article is devoted to the development of a prototype of a secure single page-application (SPA) web service for automating user information accounting. The relevance of this study is very high due to the use of web services. The article provides a classification of web applications and shows the features of the architecture of a single page application used in the development of the service. Based on a comparative analysis of the architectural styles of the API, the most appropriate style was selected. Considered and taken into account the key points of the development of a secure application programming interface (API), the requirements that must be met by RESTful API services. The work used popular authentication schemes (methods). A comparative characteristic of web frameworks of the Python programming language is given, on the basis of which a tool for implementing a web service is selected. Shows the main advantages of using Python when developing paged web services and the security tools included in the standard package of the Flask web services development framework. Shows how to securely prototype a Python RESTful SPA Web Service API using Flask. An example of using the Swagger tool to describe the specifications of the developed API is given. The process of setting up the application is considered in detail. The main recommendations for securing a web application, setting up a database and a web server are listed. The key points of ensuring the protection of the developed web application are considered. Conclusions are made regarding the choice of the architectural style of the application API, the most suitable tools and technologies for the software implementation of the service.

1. Introduction

In the context of the global digitalization of modern society, the trend of a massive transition of software solutions to online is becoming more and more noticeable [1]. Earlier developed software is being rebuilt for the format of interaction on the Web. Web solutions are preferred over mobile or desktop platforms when creating new services. In particular, the field of educational services is moving to the concept of a single information online space. This transition is motivated by a more convenient and accessible mechanism of interaction between the software and the target audience of the product. Any online resources do not impose strict hardware requirements on the user (storage memory, Random Access Memory, central processor unit power), do not tie the interaction to a specific type of platform (personal computer, smartphone, specialized device). They organize the work exclusively through the Internet browser. In addition, the share of online education and e-commerce platforms is growing, which also leads to increased attention and demand for websites and web applications [2].

However, the complexity of the logic of websites and information platforms is constantly increasing. Therefore, ordinary static websites or dynamic solutions based on content management systems (Wordpress, Joomla, 1C-Bitrix) can no longer satisfy the needs of customers when it comes to non-linear education platforms or e-commerce segment, and not trivial landing pages. Many questions arise both in functional terms and in relation to how efficiently the process of development, expansion, protection and maintenance of a software product proceeds. As a result, static websites have been almost completely replaced by full-featured web applications.

A web application is no longer a pre-built collection of HTML (HyperText Markup Language) pages, style files, scripts, and media files. Web applications are based on a client-server architecture based on the separation of a functional block and an external presentation. The business logic of the application is executed on a dedicated server (back-end), while the client interacts through the user interface in their Internet browser (front-end). This approach allows you to generate HTML markup content in real-time, with a minimum number of reloads on the viewed web page. Almost all e-commerce sites, social networks, instant messengers, online programs, forums and search engines are web applications [3].

E.V. Voevodina and Mikishanina E.A. in their article note the relevance of using web applications in the educational environment [4]. They describe a web application, which is a graphical editor that organizes the work with functions and their graphs. This service can simplify the solution of mathematical problems of high complexity when preparing students for exams.

Gladun A. is considering another way to use web applications in education. The work describes a web service that allows you to automatically control the knowledge gained by students in the framework of e-learning [5].

An equally urgent issue is the development of a secure service to automate the maintenance of information about users [6]. This article is devoted to the development of such a solution.

2. Materials and methods

The purpose of the research is to develop a web service-an indicator of user information. This article discusses the use of the service as an indicator of students ' professional competencies. The service application programming interface (further API) must conform to the principles of REST (Representational State Transfer) and meet the requirements of a RESTful architecture.

The following tasks precede the implementation of this application:

- analyze current technologies and approaches used in the development of web applications;
- consider the architectural styles of the API;
- study the limitations and requirements arising from the use of a RESTful architecture;
- analyze the key principles of ensuring the security of REST API web services;
- select the appropriate tools for software implementation of the software product.

At the first stage, it is necessary to decide which category the developed web application will belong to.

Web applications can be classified into several types (by the type of main components): backend, frontend and SPA (single page application).

Backend applications (server-side applications) collect the main functionality and logic of the application, interact with the database (hereinafter referred to as the database). They are developed using a number of programming languages, the most popular of which are: Python, Ruby, PHP, C # (.NET), Node.js. However, production-ready solutions are almost impossible to realize without the use of special tools called web frameworks. Each of the programming languages listed above has its own set of frameworks for creating web applications:

- Python: Django, Flask, FastAPI, Tornado;
- PHP: Laravel, Simfony, Yii2;
- Ruby: Ruby on Rails, Sinatra;
- Node.js: Express.js, Meteor.js;
- C#: ASP .NET Core.

Frameworks take on some of the developer's responsibilities and help in ensuring application security, working with typical redirection, authentication, registration and other functionality. The application is deployed on a specially configured web server and independently generates the HTML representation of the pages (however, the browser will reload the page in most cases).

Frontend applications (client-side applications) run directly in the user's web browser. Developed using JavaScript, TypeScript and Vue.js, ReactJS, AngularJS frameworks. They are applicable in cases when the application does not need to work with the database and store client information for longer than one session. Fully responsible for the presentation of the HTML code.

SPA is a combination of solutions, dividing a web application into two (frontend and backend) and organizing interaction between them. The backend part deals with logic, processes requests, works with the database, and the frontend part forms an external view based on the data received from the backend and displays the result in the browser. The exchange of information between the client and server parts of the application, as a rule, occurs through a specially developed application programming interface. Almost all modern web applications are based on the SPA concept. Therefore, it is this concept that forms the basis of the competency accounting service.

An API is a set of methods and rules by which applications exchange messages and transfer data. As a rule, these are classes, functions, structures of one program, interacting with other programs. The API not only helps to organize interactions, but it also plays an important role in securing that interaction. Such an interface hides the specifics of the service implementation and works within the framework of what the developers provide. Using the application API, the operations that are available to the user are defined, restrictions are imposed, inputs and outputs are specified [7]. In addition, the API has a number of architectural styles that reflect certain aspects of how the interface works.

Applications of this type have complex logic and many components, which significantly affects the security of the system [8]. There are many factors to consider when organizing security. In addition to securing the client and server side, you should focus on developing a secure API. Intrusion or accidental client error can result service unavailability, data loss, or complete application failure. This can critically affect the user experience, and as a result, bring massive losses to the business. Therefore, ensuring security is a topical issue when developing any web service.

The implementation of the API is based on the use of protocols and software specifications that build syntactic and semantic rules for transmitted messages. Such specifications shape the architecture of the API.

There are many styles containing standardization schemas and data exchange rules. The most common architectural styles are RPC (gRPC), REST, SOAP, and GraphQL.

The architectural style, represented by the RPC (Remote Procedure Call) specification, allows a function to be called remotely in the context of an HTTP API. The client gets most of the capabilities when calling a particular procedure remotely, by means of personal formation of a special message for the server. The server deserializes the incoming message [9], runs the requested function and returns the result.

RPC is a new version of RPC designed to address the accumulated problems of the previous version of the specification. gRPC also adds a wide range of features like load balancing, message tracing, and more. In addition, the new standard has started using the concept of protobuffs and runs on top of HTTP / 2. Protobuf is a. proto file, which is a dictionary on the basis of which data is encoded and decoded in a special binary format. That is, it is a kind of new serialization format (alternative to JSON / XML) for exchanging information between client and server [10]. The innovation allows you to maintain high performance in an environment of heavy data exchange in real time. It is also possible to use SSL / TLS, OAuth 2.0, tokens for authentication, or write your own implementation of secure authentication.

RPC (gRPC) is most applicable in the development of high-performance microservices [11] for internal communication, as well as in the creation of APIs for remote control systems, for example, in the IoT (Internet Of Things) field.

GraphQL is another spec introduced by the developers at Facebook. GraphQL is focused on representing data in a graph format. Unlike the creators of a tabular view from relational databases, the developers of GraphQL started from the document structure borrowed from document-oriented databases like MongoDB. The implementation is based on a graph of entities (nodes) that refer to each other (edges).

First of all, a schema is built that describes (using the special syntax of the SDL language) the exact queries and data types in the expected responses. A client with a drawn up scheme, even before sending a request, can make sure that the server has an answer to this request. On the backend, the operation is interpreted according to the scheme and sends a response in JSON format, with the requested data. As a result, we have a technology that, based on the request scheme, can receive data from multiple endpoints and issue a single response in a specific form determined by the requestor.

In addition, GraphQL supports a subscription mechanism that allows you to receive information in real time from a web server.

GraphQL is a great choice when you need to encapsulate the complex logic of the systems on which the API is developed, which is important when designing bulky projects and microservices. In addition, GraphQL provides the ability to work more efficiently with data on mobile devices (mobile API) by optimizing the payload of a single message.

SOAP or Simple Object Access Protocol is a protocol specification for building web communication. It is a W3C industry standard and uses the XML (eXtensible Markup Language) format as the underlying interoperability format. SOAP implements a stateful messaging mechanism that is used in transactions involving multiple parties or in complex secure transactions. Works with HTTP (HyperText Transfer Protocol), SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol) protocols. SOAP integrates with WS-Security protocols to guarantee the integrity and confidentiality of processed transactions with additional encryption [12]. By using XML, SOAP is the most verbose API style. Despite this, the XML interchange format has many implications that prevent SOAP from being a universal architectural style.

Thus, SOAP is most often used to provide reliable access within a company or directly with a client. Strong security capabilities make SOAP the preferred choice in the corporate and financial sectors.

REST (View State Transfer) is a style of API architecture for creating web services and services. It was presented in 2000 in a doctoral dissertation [13] by one of the creators of the HTTP protocol, Roy Fielding. REST is a simple interface for transferring information that does not use third-party software layers. That is when sending data, there is no conversion stage, the information is delivered in its original form, which has a beneficial effect on the client load, but adds a load on the network part.

Working with data is organized in JSON or XML formats. A web service that meets all the requirements and conditions of the REST architecture is called a RESTful web service.

RESTful architectural requirements (Fielding Constraints):

- Do not contain state (stateless): the state for processing a request can only be contained in the request itself, so the server does not store any session information.
- Caching.
- Common Interface: Allows for a consistent, application-independent interaction with the web server. It imposes a number of sub-restrictions: manipulating resources through their representations, identifying resources, "self-sufficiency" of messages, working with hypermedia.
- Focus on client-server architecture.
- Ability to deliver executable code to the client side.
- Independence from the number of levels / layers of the application.

In REST, all communication is based on the use of HTTP methods:

- GET - request to receive data;
- POST - sending / adding data;
- PUT - data editing;
- PATCH - partial data update;
- DELETE - deleting data.

The advantages and disadvantages of each of the schemes are presented in table 1.

Table 1. Advantages and disadvantages of the most popular schemes.

Schemas	Benefits	Disadvantages
RPC (gRPC)	simple interaction mechanism the ability to add functions with subsequent conversion to the end point high performance the ability to interact with security tools	low level of abstraction, which leads to difficulty in reuse difficulties in scaling problems with introspection of queries reduced security due to the inability to completely hide the details of the API implementation
GraphQL	to retrieve data of any complexity, one request is enough availability of a typed request data schema works effectively with related data detailed error log flexible customization of requests (format definition, portioning, single version)	Possible low performance if there are too many nested fields in one query difficulties with caching high threshold of entry, it is necessary to deal with SDL
SOAP	the ability to work with various transport protocols implementation of built-in error handling (Retry messages) security enhancements. detailed message format (lots of metadata)	support for a single XML format the need for a large service bandwidth due to the heavy weight of XML requires deep knowledge of protocols and data transfer rules
REST	openness of interaction simple implementation data caching at the HTTP level work with several formats of data presentation; stability due to the high level of abstraction	lack of a unified standardized structure high load on the network excessive or insufficient information, which may lead to the need to send an additional request

REST is most often used as a management API for CRUD (Create, Read, Update, Delete) to implement interaction with resources in lightweight scalable services. A resource is usually a data model object or a database table.

Of all the specifications, REST implements the highest level of abstraction and is best suited for developing simpler CRUD APIs. Maintains a balance of reliability and ease of use. The client-server bias and caching propensity (important for listing competencies, sorting and filtering) are also advantages for services like the one discussed in this article. The large, in comparison with other specifications, the load on the network is equalized by the specifics of the problem being solved (one student with a low degree of probability will have thousands of entries and add them to the platform in batches of several thousand, and the development of a separate mobile application is not initially planned). Therefore, it is REST that forms the basis of the architectural style of the project prototype being developed.

Building secure RESTful APIs also imposes certain standard requirements:

- Using the HTTPS protocol: encryption is required to ensure the integrity of the transmitted data.
- Rate-limits: It is necessary to monitor the load on the API. Dropping requests in case of overload or connecting additional systems for balancing.
- Authentication: User / application / device identification.
- Audit log: Recording actions by creating an entry in the logs log.
- Control of access rights (authorization): Determination of access rights for working with resources.
- Access to the business logic of the application.

When working with the REST architecture, it is customary to distinguish two levels of security, since the REST API is an interface for network interaction with a web application:

- the first level - getting access to the API;
- second level - getting access directly to the application.

Protection of the API level implies proper organization of authentication, authorization, registration and separation of access rights. The developer must ensure that only authorized clients can use the API and only have access to authorized operations.

Application layer security includes vulnerability checking of service endpoints - the URLs responsible for interacting with the interface.

The OWASP (Open Web Application Security Project) community has identified (at various levels) about ten API development vulnerabilities. The most dangerous are:

- Broken Object Level Authorization: No separation and enforcement of resource access control.
- Broken User Authentication: Vulnerabilities related to user authentication.
- Lack of Resources & Rate Limiting: Mismatch of checks and limits.
- Broken Function Level Authorization: Lack of access control.
- Mass Assignment: Vulnerabilities related to deserialization of received objects.
- Security Misconfiguration: Errors in working with application security settings.
- Insufficient Logging & Monitoring: Absence or incorrect vision of the logs and system monitoring logs.

First of all, it is necessary to pay attention to the problems associated with access control, authentication and authorization [14]. By project, the REST API is stateless, so access is controlled through local endpoints.

There are several common methods (authentication schemes): Basic Authentication, API Key, JSON Web Tokens, OAuth 2.0, Token-Base Authentication, Cookie-Based Authentication, and OpenID.

Basic Authentication (BA) is the simplest HTTP-based authentication scheme. The client or application forms an HTTP request, the header of which contains the "Authorization" field. A string of the form: Basic <username: password> (encoded in Base64) is passed as the value of this field. To ensure security and preserve the confidentiality of data, in combination with Basic Authentication, it is always necessary to use the HTTPS / TLS protocols [15], since the user ID and password enter the Network as plain text encoded in Base64 (which is easy to decode).

Cookie-Based Authentication is a method based on checking the content of cookies, inside which all the necessary information about the session is stored. The user initiates a login request. In case of a successful login, the server sends a response, in the header of which the Set-Cookies field is included, containing the name of the cookie-field, value, cookie expiration, and so on. The next time the user needs to access the API, he will pass in the request header the value of the saved Cookie-field JSESSIONID with the key "Cookie": Cookie: JSESSIONID = 0112. Cookie-based authentication is possible only if the server has a mechanism for storing and checking user sessions. As with Basic Authentication, use encrypted Internet protocols for security. In addition, Cookie-Based Authentication has a number of vulnerabilities, which can be secured by passing a special CSRF security token in the request header: X-Csrf-Token: token_value.

API Key – a key in the form of a character string that the user passes along with the request to the server. The server will confirm the identity of the client if its key is contained in the application database. The key itself is issued by the application when registering a user. This scheme is used to protect against unauthorized access and allows you to impose a limit on API calls. API key can be passed: as a request parameter, as a request header with a field, as a cookie value.

The API Key should only be accessible to the owner user and the server. Therefore, APIs using API Key as their primary authentication scheme should only use secure HTTPS / SSL protocols.

The Token-Base Authentication or Bearer Authentication scheme is based on the use of a "token signed by the server". This token is then sent to the server inside the Authorization request header. However, unlike Basic Authentication, the "Basic" keyword is replaced by "Bearer": Authorization: Bearer <token>. After receiving the token, the server validates it (does a user exist with such a token, the validity period has not expired, etc.). Token-Base Authentication can be used independently, when the server itself generates tokens for new users, or be part of a more complex authentication mechanism like OAuth 2.0 or OpenID Connect. To prevent the interception of the token by attackers, you must use the secure HTTPS protocol. The difference between tokens and API Key is that the key can only provide access to API methods without the ability to get the user's personal data.

JSON Web Tokens (JWT) is an authentication mechanism based on the use of a special type of token - JWT token. It is a JSON data structure. Such a token contains a header with general information, a body with a payload (user-id, group, data) and a cryptographic signature. This scheme is one of the most secure mechanisms for transferring data between two parties, therefore it is considered the preferred method for controlling REST API access. The user for working with the API, when sending a request, adds to it the personal JWT previously issued by the server.

OAuth 2.0 is a more complex protocol responsible for user authorization. Allows a web application to acquire rights to use client resources on another service. This makes it possible to provide a third-party application with limited access to the controlled resources of the user without directly transferring the login and password to this application. OAuth can be used on any platform, as the protocol relies on the underlying web technology stack, namely HTTP requests, responses, URL redirects [16], etc.

The scheme of the web service using OAuth 2.0 looks like this:

- The application is being authorized. As a result of successful passage, an access token is created (a key consisting of a set of characters). Presentation of this token indicates that the application has access to protected resources. As in classic authentication schemes, the access token can be presented via an HTTPS request with the appropriate header or as one of the GET request parameters.
- The application requests and receives protected resources.

This approach is the most complex authorization option, but only when it is used is the service able to uniquely identify the application requesting authorization. Otherwise, authorization is done entirely on the client side. The schema is often used when developing REST APIs that interact with third-party services.

OpenID is a standardized method for a decentralized authentication scheme. A distinctive feature is the ability to create a single account for authentication on several services at once (creating a unique digital identifier) through the services of an intermediary service. In terms of the mechanism of operation, this method is similar to OAuth 2.0, but OpenID is intended solely for user authentication. The new version of OpenID Connection was transformed into an authentication add-on over OAuth 2.0, received a reliable encryption mechanism and digital signatures.

It was decided to use JWT as the main access control scheme for the developed application, since the service does not use third-party resources in its work, and such tokens are easy to use, have a convenient data description format and do not create unnecessary network load. The use of the HTTPS protocol in conjunction with a cryptographic signature provides an optimal level of protection for the service.

When securing a web service, input control deserves special attention. You need to make sure that all the data that the application will subsequently operate on is valid and conforms to the API specification.

The developer community has formed a number of recommendations when validating input data:

- conduct data validation both on the client side and on the server side;
- you should not use your own validation mechanisms, you should use the built-in functions for checking a programming language or framework;
- it is always necessary to check the content type, size and length of the request;
- do not create manual queries to the database on the backend, use only parameterized queries;
- use allow lists on the server;
- keep logs of errors, attempts to enter data, etc.

An equally important task when developing a REST API is choosing the right tools for the backend. The Python programming language has long established itself in the web development market. A lot of high-quality modules, packages, frameworks and utilities have been created for "python", which greatly simplifies the development process. Over the years, the industry has developed a rich code base and a huge international community of developers. An important factor is the relatively low threshold of entry, the simple and concise syntax of the language. This and much more has affected the fact that the development of application prototypes, proof-of-concept and minimum viable products (MVP) is faster and most efficiently done in Python.

There are many powerful and versatile frameworks for developing services and web applications in the Python ecosystem. The most popular framework for API development is Flask.

Flask is a WSGI (Web Server Gateway Interface) micro-framework that provides minimal basic functionality with the ability to add modules as needed and at the discretion of the developer. Due to the modular approach, the programmer can assemble only those tools that will definitely be used in the development of the project [17]. The flexibility of the architecture and the presence of a large number of ready-made extensions make it possible to use Flask both in simple projects and in the development of complex platforms. In the developer community, it is often used to create RESTful web service APIs. All of this became a determining factor in choosing Flask as the basis for the development of an application prototype.

For developing secure APIs, Flask provides a number of loadable modules [18]:

- Flask-Login and Flask_simplelogin for managing user sessions. Due to these extensions, registration, authentication and authorization mechanisms are integrated.
- Flask-Admin - an extension that adds an analogue of the administrator's personal account to manage the service and monitor systems.
- Flask-WTF– module for creating secure forms (validation, csrf tokens, etc.).
- Flask_jwt - extension for working with JWT authentication scheme.
- Flask_restful / Flask_restplus - Sets of helper objects and functions that simplify REST API development.
- Flask-Security - a package for managing user roles and groups.

The disadvantage of many developed services is that the description of the API itself is rigidly tied to the implementation in a specific language or its framework [19]. The open source tool Swagger is used to solve this problem and create a flexible unified API.

Swagger is a language and technology stack agnostic framework for describing the specification and documentation of RESTful APIs. It allows you to configure REST API without access to the source code of the application, through a special file with the YML (Yandex Market XML) or JSON extension [20]. A documentation package is also automatically generated based on this file.

3. Results

The server on which the developed web application will be launched is also a potentially vulnerable point for attackers. To reduce the risk of hacking, you need to disable root logins of the linux server by editing the / etc / ssh / sshd_config. In the PermitRootLogin option, change the value from yes to no.
The next step is installing and configuring the firewall. It is necessary to close access to all ports except 80 (HTTP), 443 (HTTPS) and 22 if we work with the server via SSH: *sudo ufw allow ssh hhtp 443 / tcp –force enable.*

In addition, you should replace the standard Flask application server with a more advanced WSGI server, gunicorn. It is a robust production server that many developers take as a standard when developing Python web applications. Configure it so that the service listens for requests from private port 8000: *gunicorn -b localhost: 8000 -w 4 restapiservice: app*. However, application web traffic can only enter ports 80 and 443 open on the firewall. At the same time, to ensure the security of the application, all traffic arriving on HTTP port 80 must be redirected to encrypted port 443 (HTTPS). To work with the HTTPS protocol, you need to obtain an SSL security certificate (for example, Let's Encrypt): *sudo Apt install python-certbot-nginx / sudo certbot –nginx -d <application_domain> -d www. <application_domain>*

An Nginx proxy will be used to redirect traffic. Listening port settings, HTTPS connection and traffic redirection occur through the setting of the configuration file / etc / nginx / sites-enabled / default [21].

The default SQLite database is also not an acceptable solution. The application will use the reliable and efficient PostgreSQL DBMS (Database Management System), which is the de facto standard on the Python + Django / Flask technological stack.

However, no DBMS can efficiently service the large number of independent processes that access it. Too many requests (real users or those created by an intruder) will lead to the failure of the database. To solve this problem, we will use the PgBouncer connection puller. Puller proxies all incoming requests in small chunks, creating specialized queues.
Initially, when developing a web application, you need to pay attention to the form of storing confidential application data: keys, addresses, database passwords, etc. The simplest and most reliable option is to use environment variables. The programmer creates a bat / bash script that, when launched on the server, will write the necessary data to the variables of the current environment. You can then use the os python module to read the required information and configure your Flask application. Due to this, secret data does not appear openly in the code.

When creating a user model, it is important to pay attention to the work with the "Password" field. Passwords should not be explicitly stored in the database. For security reasons, it is not the passwords themselves that are used to record and compare passwords, but their hashes. To do this, inside the set_password and check_password methods of the User model, we will use the functions of the security package - generate_password_hash and check_password_hash.

Next, let's look at the method responsible for adding a new competency. In this case, the user must be authorized. The function will be called every time the API receives a POST request like this:
curl -X POST -H "Content-Type: application / json" \ -d '{"title": "Achievement Ttitle", "type": "Achievement Type", "date": "Achievement Date", "proof" : "Achievement Proof URL"} '\ https: // site / api / achievement /

Getting the contents of the fields of the model should be done through a call to the get method, and in the absence of the corresponding field, the value of the variable is set to None. In addition, in order to ensure security, the work with the database is carried out through the objects of the built-in ORM (Object-Relational Mapping), and not by direct SQL (structured query language) queries.

The description of the API request specification itself, the setting of the handler and security restrictions are moved to the swagger.yml file. In this file, you need to specify the authentication scheme that the application will use, as well as a special function that contains the implementation of this scheme in Python:

components:

```
    securitySchemes:
    bearerAuth:
    type: https
    scheme: bearer
   bearerFormat: JWT
securityDefinitions:
   bearerAuth:
   type:bearerAuth
   x-basicInfoFunc: "auth.jwt_auth"
```

In the case of a developed web service, the schema handler (jwt_auth function) is located in a separate auth.py file.

In the swagger.yml file, each request to the API is represented by a separate block, which contains all the service options in a declarative style. In the specification block, you must specify: path, HTTP request type, response and security.

A description of the specification for adding competencies:

```
paths:
/achievement:
post:
      operationId: " achievement.create"
      tags:
 responses:
        201:
          description: "Successfully created achievement in list"
        406:
          description: "achievement already exists"
      security:
        - bearerAuth: []
```

Now, when trying to access the POST /achievement API method, the system will ask the user for authorization.

The rest of the API methods are created in the same way.

After completing all the procedures for securing the server side, you must also take into account a number of considerations related to securing the client side of the application.

A web application developed and configured in this way closes most of the vulnerabilities of the frontend / backend side and allows the client to safely use all the functionality of the service.

4. Discussion

As it turned out during testing, the developed web service, with a rapid increase in the number of incoming requests (on the order of several hundred), may experience a number of performance problems. This is due to the technology stack used, the choice of which was justified by the priority in the simple and fast development of an application prototype. Using the synchronous Flask framework like a bottleneck for system speed and responsiveness.

Flask, as a web service development tool, is often criticized by the Python developer community.
In 2018, Sebastian Ramirez developed a Python web framework called FastAPI. According to Uber developers Pierre Molino, Yaroslav Dudin and Sai Sumant Miryal, FastAPI should be the best framework for developing REST API services and a more performant analogue of Flask [22]. Ramirez's framework is completely asynchronous and easily integrates with OpenAPI-schema, making it easier to work with Swagger and ReDoc. Despite the relatively short term on the market, FastAPI has established itself as a quality tool. Many large companies have started using it to develop their APIs. So in the project discussion thread on GitHub, Microsoft Lead Engineer Khan Kabir posted the following comment: "I have been using FastAPI very often in recent days. I definitely plan to use it for all ML services of my team at Microsoft." [23].

Also, as an alternative to Flask, many developers, in particular Adam Hopkins, recommend considering another relatively "young" framework - Sanic [24]. It was also conceived as a multithreaded, asynchronous counterpart to Flask. It is a micro-framework with a modular structure and a number of features such as basic routing of static files, special extensions for creating CRUD, mechanisms of non-blocking operations, and others.

In addition, Rhea Mutafis criticizes the use of Python because of its relatively poor performance, dictated by the interpretability of the language [25]. As an alternative, Google engineers offer their own development - the Golang programming language. Its main advantages are strong typing, the presence of a compiler, and goroutines are an efficient analogue of threads. In addition, Go provides the ability to work with static typing when developing web applications, which makes it much easier to test and maintain the product in the future.

The transition to an alternative technology stack, for example, the FastAPI framework, will help to further improve the performance of the service, as well as expand the potential for scaling the system.

5. Conclusions

Thus, SPA APIs have become a staple in the development of modern web and mobile applications. They allow you to organize interactions between applications, services, and software platforms. Due to their versatility and ease of use, REST architectural style interfaces account for about 80% of all public and proprietary APIs.

In the course of the research, a prototype of a protected SPA RESTful API web service was developed, with the help of which users will be able to keep track of personal information related to various areas of their professional, educational or personal activities. The Python programming language and its Flask web microframework significantly increased the efficiency, quality and speed of MVP development, and the Swagger framework made it possible to achieve independence of the developed REST API from the technology stack.

The project has a high potential for expanding and scaling by adding new functionality, replacing the used database with a more flexible No-SQL document-oriented version (for example, MongoDB), implementing advanced caching based on the in-memory Redis database, authorization through social networks. In addition, it is possible to switch to a more efficient asynchronous backend framework, refine and optimize the UX / UI of the client side of the application.

The prospect of further research on this issue is to find ways to restructure the application by moving to a modular architecture. This will allow you to flexibly customize the project for the needs of customers, including organizations that are not tied to working with clients from any one field of activity.

References

[1] Kornienko D V 2020 Organization of a system of digital education practices in the municipal sphere of general education *J. Phys.: Conf. Ser.* **1691(1)** 012108
[2] Mishina S V 2020 Strategies for students' lean competencies formation in the educational process of the university *J. Phys.: Conf. Ser.* **1691(1)** 012213
[3] Shcherbatykh S V and Mishina S V 2019 Development of professionally significant qualities of future economists by means of the hidden curriculum | Desarrollo de cualidades profesionalmente significativas de futuros economistas mediante el currículum oculto *Utopia y Praxis Latinoamericanat* **24** pp 387–95
[4] Volodina E V and Mikishanina E A 2020 Using the developed web-application for solving problems of increased complexity in mathematics *J. Phys.: Conf. Ser.* **1691** 012198
[5] Gladun A, Rogushina J, Garcı´a-Sanchez F, Martínez-Béjar R and TomásFernández-Breis J 2009 An application of intelligent techniques and semantic web technologies in e-learning environments *Exp. Sys. App.* **36**
[6] Kornienko D V, Shcherbatykh S V, Mishina S V and Popov S E 2020 The effectiveness of the pedagogical conditions for organizing the educational process using distance educational technologies at the university *J. Phys.: Conf. Ser.* **1691(1)** 012090
[7] Shevat A, Jin B and Sahni S 2018 *Designing Web APIs: Building APIs That Developers Love* (O'Reilly Media, City of Newton)
[8] Jiao J, Yang Y, Feng B, Wu S, Li Y and Zhang Q 2017 Distributed rateless codes with unequal error protection property for space information networks *Entropy* **19(1)** 38

[9] Abbaspour E, Fani B and Heydarian-Forushani E 2019 A bi-level multi agent-based protection scheme for distribution networks with distributed generation *Int. J. Electrical Power and Energy Sys.* **112** pp 209–20
[10] Cong W, Zheng Y, Zhang Z, Kang Q and Wang X 2017 Distributed Storage and Management Method for Topology Information of Smart Distribution Network *Dianli Xitong Zidonghua/Automation of Electric Power Sys.* **41(13)** pp 111–18
[11] Bonér J 2016 *Reactive Microservices Architecture: Design Principles for Distributed Systems* (O'Reilly Media, City of Newton)
[12] Song X, Zhang Y, Zhang S, Song S, Ma J and Zhang W 2018 Active distribution network protection mode based on coordination of distributed and centralized protection *Proc. 2017 China Int. Electrical and Energy Conf.* pp 180–83
[13] Fielding R 2000 *Architectural Styles and the Design of Network-based Software Architectures* (University of California, California)
[14] Tong B B, Zou G B and Shi M L 2013 A distributed protection and control scheme for distribution network with DG *Advanced Materials Research* pp 732–33 628–33
[15] Zhong S, Liu C, Yang Z and Yan D 2009 Privacy protection model for distributed service system in converged network *Int. Conf. E-Business and Inf. Sys. Security* (Institute of Electrical and Electronics Engineers, Piscataway)
[16] Maximov R V, Ivanov I I and Sharifullin S R 2017 Network topology masking in distributed information systems *CEUR Work. Proc.* **2081** pp 83–7
[17] Greenberg M 2016 *Developing web applications using Flask in Python* (DMK Press, Moscow)
[18] Dwyer G 2016 *Flask By Example* (Pack Publishing Ltd, UK)
[19] Percival G 2018 *Test Driven Development* (DMK Press, Moscow)
[20] Swagger https://community.sanicframework.org/t/how-do-i-find-out-which-companies-use-sanic/724
[21] Melnikov M O 2021 Setting up a Debian server for developing Python and Django web applications *Modern Science* **2(2)** https://www.elibrary.ru/item.asp?id=44789469
[22] Uber Engineering https://eng.uber.com/ludwig-v0-2/
[23] GitHub https://github.com/tiangolo/fastapi/pull/26
[24] Sanicframework https://swagger.io/
[25] Moutafis R 2020 Why Python is not the programming language of the future *Towards Datascience* https://towardsdatascience.com/why-python-is-not-the-programming-language-of-the-future-30ddc5339b66