KEA

(Kubernetes Empowerer to API)

Requirements and analysis model

Product description

The product is a platform for deploying, managing, and scaling machine learning models in production. It offers a secure, flexible environment for automating ML tasks like model versioning, routing, and monitoring. With Kubernetes integration and containerization support, it's designed for developers, ML engineers, and enterprises needing scalable, resilient ML infrastructure.

Team K8C: Tsurkan Daniel; Dandamaev Gadji; Tsaturyan Konstantin; Smolkin Mikhail

Project repo: https://github.com/fanglores/Advanced-Software-Design

This report: https://github.com/fanglores/Advanced-Software-Design
/blob/master/Practice%20Tasks/Final_Task/K8C_FinalTask1_(Task7).pdf

Roles

ML Engineer

Description: This role joins professionals involved in the development, deployment, and monitoring of ML models. They want to simplify the deployment process, automate API documentation, and ensure efficient request validation and caching, ultimately enhancing their workflow and model performance.

API Consumer

Description: This role includes all users interacting with APIs to integrate ML models into their applications. They want to access reliable and well-documented APIs, enabling seamless integration of ML models into their business applications and ensuring optimal performance and usability.

Personas

ML Engineer (Maria, 32 years old)

Goals:

- Deploy and version ML models in Kubernetes.
- Automatic API documentation and request validation.
- Flexibility for different ML frameworks.

Pain points:

- Manual API documentation.
- Difficulties in monitoring model performance.

Backend Developer (Alexander, 28 years old)

Goals:

- Use automatic OpenAPI schema generation.
- Easily add API endpoints with request validation and security.

Pain points:

- Manual API documentation.
- Challenges with integrating authorization and managing access control.

Personas

API Consumer (Sergey, 30 years old)

Goals:

- Get documentation for quick access to ML models.
- Work with reliable and validated APIs.

Pain points:

- Incomplete or outdated documentation.
- API instability and delays.

Corporate Client (Yandex, Sber)

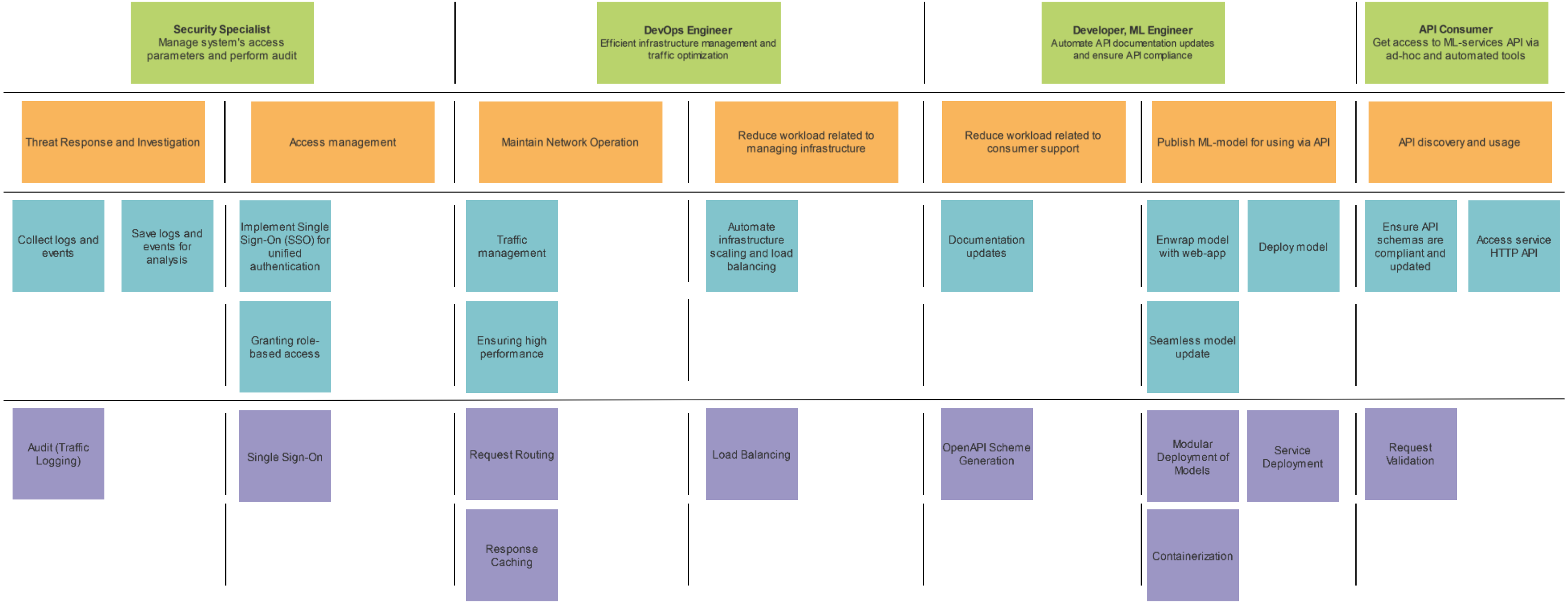
Goals:

- Scalable and secure deployment of ML models.
- Integration of the API gateway into existing infrastructure.

Pain points:

- Challenges with integration and corporate standards.

Story map



Security Specialist Manage system's access parameters and perform audit			**DevOps Engineer** Efficient infrastructure management and traffic optimization		**Developer, ML Engineer** Automate API documentation updates and ensure API compliance			**API Consumer** Get access to ML-services API via ad-hoc and automated tools	
Threat Response and Investigation		Access management	Maintain Network Operation	Reduce workload related to managing infrastructure	Reduce workload related to consumer support	Publish ML-model for using via API		API discovery and usage	
Collect logs and events	Save logs and events for analysis	Implement Single Sign-On (SSO) for unified authentication	Traffic management	Automate infrastructure scaling and load balancing	Documentation updates	Enwrap model with web-app	Deploy model	Ensure API schemas are compliant and updated	Access service HTTP API
		Granting role-based access	Ensuring high performance			Seamless model update			
Audit (Traffic Logging)		Single Sign-On	Request Routing	Load Balancing	OpenAPI Scheme Generation	Modular Deployment of Models	Service Deployment	Request Validation	
			Response Caching			Containerization			

Use case diagram

(take from the previous task)

Show main and alternative scenarios on a diagram

Provide a link to structured textual use case scenarios in your project repo.

Interaction analysis

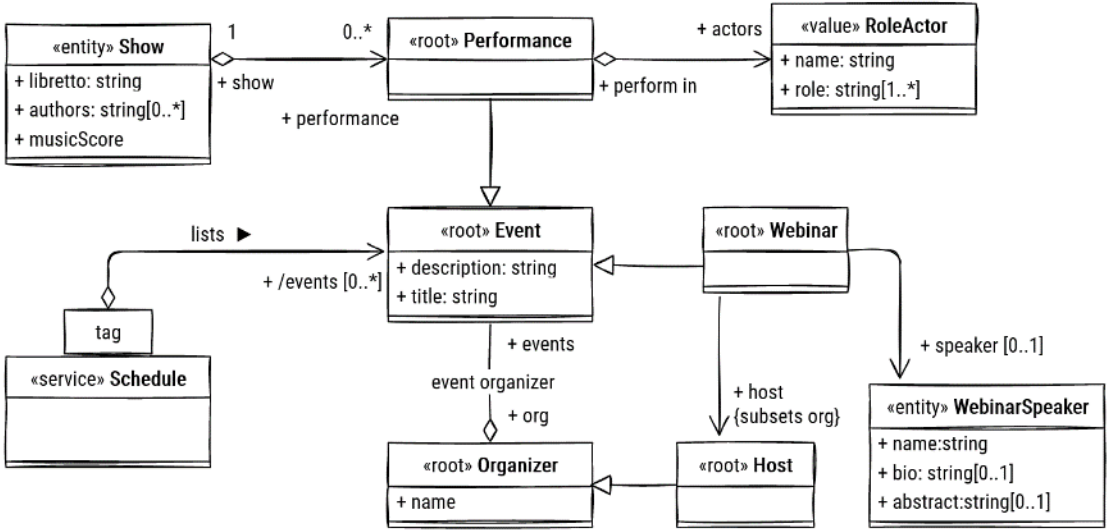
Show a cooperation on a diagram along with the use case

OR. Show as a table, columns: use case - cooperation name - used roles - candidate classes that play them

Final class diagram

The final class model should be consistent (well-formed) with interactions and use cases / stories.

Check that DDD stereotypes are set



Detailed behaviour

Use an activity or state diagram to describe behavior of **one** of the dynamic classifiers in your model

Add the diagram here and explain how it works precisely in detail

Check that the overall model remains well-formed

Note: too small models may not be enough to demonstrate your qualification

Repository structure

Repository structure

```
.
├── product_img.jpg
├── README.md
├── General/
│   ├── Domain_description_en.md
│   ├── Task_description_en.md
│   ├── DFD0/
│   ├── StoryMap/
│   └── UseCases/
└── Practice Tasks/
    ├── Final_Task/
    ├── Task_1/
    ├── Task_2/
    ├── Task_3/
    ├── Task_4/
    ├── Task_5/
    ├── Task_6/
    └── Task_7/
```

Tools Used:

- Github
- Drawio
- Planttext

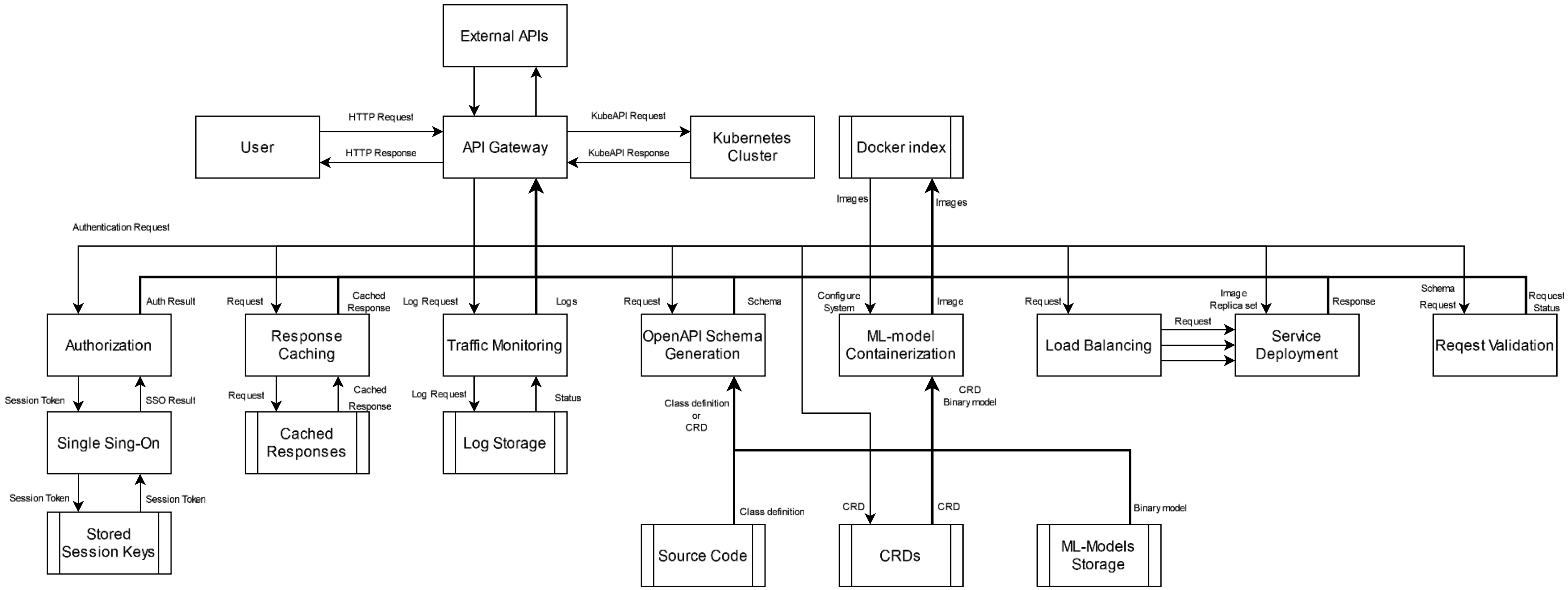
Team and roles

Roles, Personas, Story Map	Use cases, DDD	UML, DFD, Repository Management	Domain analysis, CRC Cards, UML
Tsurkan Daniel Tg: @crazy_deyzi	Dandamaev Gadji Tg: @dandamaev	Tsaturyan Konstantin Tg: @fanglores	Smolkin Mikhail Tg: @m0hnatik

Thanks for attention!

Now we are ready to answer your questions!

DFD (Level 0)


External APIs
User
HTTP Request
HTTP Response
API Gateway
KubeAPI Request
KubeAPI Response
Kubernetes Cluster
Docker index
Images
Images
Authentication Request
Auth Result
Request
Cached Response
Log Request
Logs
Request
Schema
Configure System
Image
Request
Image Replica set
Response
Schema Request
Request Status
Authorization
Response Caching
Traffic Monitoring
OpenAPI Schema Generation
ML-model Containerization
Load Balancing
Request
Service Deployment
Reqest Validation
Session Token
SSO Result
Request
Cached Response
Log Request
Status
Class definition or CRD
CRD Binary model
Single Sing-On
Cached Responses
Log Storage
Session Token
Session Token
Class definition
CRD
CRD
Binary model
Stored Session Keys
Source Code
CRDs
ML-Models Storage