ПРИЛОЖЕНИЕ А

Создание контроллера и DTO для него

```
#[Route(path: '/anecdote')]
class AnecdoteController extends AbstractController
{
    public function __construct(
        private readonly ValidatorService    $validator,
        private readonly SerializerInterface $serializer,
    ) { }
    #[Route(path: '', name: 'apiGetAnecdoteList', methods:
↪ Request::METHOD_GET)]
    public function getAnecdoteList(AnecdoteService
↪ $anecdoteService): JsonResponse
    {
        return $this->json(
            data: $anecdoteService->getAnecdoteList(),
            status: Response::HTTP_OK,
        );
    }
    #[Route(path: '', name: 'apiCreateAnecdote', methods:
↪ Request::METHOD_POST)]
    public function createUser(Request $request, AnecdoteService
↪ $anecdoteService): JsonResponse
    {
        $data =
↪ $this->serializer->deserialize($request->getContent(),
↪ AnecdoteBaseRequestDTO::class, 'json');
        $this->validator->validate(body: $data, groupsBody:
↪ ['register']);
        return $this->json(
            data: $anecdoteService->createAnecdote($data),
            status: Response::HTTP_CREATED,
        );
    }
```

```
26      #[Route(path: '/{id<\d+>}', name: 'apiEditAnecdote', methods:
    ↪   Request::METHOD_PATCH)]
27      public function editAnecdote(
28          Anecdote        $id,
29          Request         $request,
30          AnecdoteService $anecdoteService,
31      ): JsonResponse
32      {
33          $data =
    ↪   $this->serializer->deserialize($request->getContent(),
    ↪   AnecdoteBaseRequestDTO::class, 'json');
34          $this->validator->validate(body: $data, groupsBody:
    ↪   ['edit']);
35
36          return $this->json(
37              data: $anecdoteService->editAnecdote($id, $data),
38              status: Response::HTTP_CREATED,
39          );
40      }
41      #[Route(path: '/{id<\d+>}', name: 'apiDeleteAnecdote',
    ↪   methods: Request::METHOD_DELETE)]
42      public function deleteUser(Anecdote $id, AnecdoteService
    ↪   $anecdoteService): JsonResponse
43      {
44          $anecdoteService->deleteAnecdote($id);
45
46          return $this->json(
47              data: [],
48              status: Response::HTTP_NO_CONTENT,
49          );
50      }
51  }
```

```
1  class AnecdoteBaseRequestDTO
2  {
```

```
3       public function __construct(
4           #[Assert\NotNull(groups: ['register'])]
5           #[Assert\Type(type: 'string', groups: ['register',
    ↪  'edit'])]
6           #[Assert\Length(max: 127, groups: ['register', 'edit'])]
7           public ?string $title = null,
8
9           #[Assert\NotNull(groups: ['register'])]
10          #[Assert\Type(type: 'string', groups: ['register',
    ↪  'edit'])]
11          public ?string $text = null,
12
13          #[Assert\NotNull(groups: ['register'])]
14          #[Assert\Type(type: 'string', groups: ['register',
    ↪  'edit'])]
15          public ?bool $category = null,
16      ) { }
17  }
```

```
1   class AnecdoteBaseResponseDTO
2   {
3       public int $id;
4       public string $title;
5       public string $text;
6       public string $category;
7
8       public function __construct(Anecdote $anecdote)
9       {
10          $this->id = $anecdote->getId();
11          $this->title = $anecdote->getTitle();
12          $this->text = $anecdote->getText();
13          $this->category = $anecdote->getCategory();
14      }
15  }
```

ПРИЛОЖЕНИЕ Б

Создание авторизации

```
class SecurityService
{
    private const string SUBJECT = 'Код авторизации';

    private const string ACCESS_TOKEN_LIFETIME = '+10 minutes';
    private const string REFRESH_TOKEN_LIFETIME = '+90 days';

    public function __construct(
        #[Autowire(service: YandexMailerService::class)]
        private MailerServiceInterface   $mailer,
        protected EntityManagerInterface $entityManager,
    ) { }
    public function sendCode(LoginDTO $DTO): void
    {
        $code = new Code();
        if (!$this->entityManager->getRepository(User::class)->
        findOneByEmail($DTO->email) instanceof User) {
            throw new ApiException(
                message: "Пользователь по указанному email не
    ↪ найден",
                status: Response::HTTP_NOT_FOUND,
            );
        }
        $code
            ->setEmail($DTO->email);

        $this->entityManager->persist($code);
        $this->entityManager->flush();
        $this->mailer->send(self::SUBJECT, $code->getCode(),
    ↪ (array)$DTO->email);
    }

```

```
31      public function verifyCode(LoginDTO $DTO): array
32      {
33          $code = $this->entityManager->getRepository(Code::class)->
34          findOneBy([
35              'code' => $DTO->code,
36              'email' => $DTO->email,
37              'status' => CodeStatus::ACTIVE->value,
38          ]);
39
40          if (!$code instanceof Code) {
41              throw new ApiException(
42                  'Неверный код авторизации',
43                  status: Response::HTTP_UNAUTHORIZED,
44              );
45          }
46          if ($code->getExpiredAt() < new \DateTime()) {
47              $code
48                  ->setStatus(CodeStatus::EXPIRED->value);
49              $this->entityManager->flush();
50              throw new ApiException(
51                  'Код авторизации истек',
52                  status: Response::HTTP_UNAUTHORIZED,
53              );
54          }
55          $owner =
    ↪ $this->entityManager->getRepository(User::class)->
56          findOneBy([
57              'email' => $DTO->email,
58          ]);
59
60          $device = (new Device())
61              ->setOwner($owner)
62              ->setTokenExpiresAt((new
    ↪ \DateTime())->modify(self::ACCESS_TOKEN_LIFETIME))
```

```
63              ->setRefreshTokenExpiresAt((new
    ↪ \DateTime())->modify(self::REFRESH_TOKEN_LIFETIME));
64
65          $code
66              ->setStatus(CodeStatus::INACTIVE->value);
67
68          $this->entityManager->persist($device);
69          $this->entityManager->flush();
70
71          return [
72              'token' => $device->getToken(),
73              'refreshToken' => $device->getRefreshToken(),
74          ];
75      }
76      public function logout(string $apikey): void
77      {
78          $device =
    ↪ $this->entityManager->getRepository(Device::class)->
79          findOneBy([
80              'apikey' => $apikey,
81          ]);
82          $device->setStatus(DeviceStatus::INACTIVE->value);
83          $this->entityManager->flush();
84      }
85      public function refresh(?string $refreshToken): array {
86          $device =
    ↪ $this->entityManager->getRepository(Device::class)->
87          findOneBy([
88              'refreshToken' => $refreshToken,
89              'status' => DeviceStatus::ACTIVE->value,
90          ]);
91
92          if (!$device instanceof Device) {
93              throw new ApiException(
```

```
94                  message: 'Некорректный refresh токен',
95                  status: Response::HTTP_UNAUTHORIZED,
96              );
97          }
98
99          if ($device->getRefreshTokenExpiresAt() < new \DateTime())
    ↪ {
100             $device->setStatus(DeviceStatus::EXPIRED->value);
101             $this->entityManager->flush();
102
103             throw new ApiException(
104                 message: 'Refresh токен истёк',
105                 status: Response::HTTP_UNAUTHORIZED,
106             );
107         }
108
109         $device
110             ->setToken($this->generateToken())
111             ->setTokenExpiresAt((new
    ↪ \DateTime())->modify(self::ACCESS_TOKEN_LIFETIME))
112             ->setRefreshToken($this->generateToken());
113
114         $this->entityManager->flush();
115
116         return [
117             'token' => $device->getToken(),
118             'refreshToken' => $device->getRefreshToken(),
119         ];
120     }
121     public function generateToken(): string {
122         return md5(random_int(100000, 999999) . microtime());
123     }
124 }
```