ДОДАТОК А

Фрагменти коду blockchainauth

Клас AuthRequest

```
class AuthRequest(AuthMessage):
""" Interface for creating signed auth request tokens, as
well as decoding
and verifying them.
"""
verify_methods = [
is_expiration_date_valid,
is_issuance_date_valid,
do_signatures_match_public_keys,
do_public_keys_match_issuer
]
def __init__(self, private_key, domain_name,
manifest_uri=None, redirect_uri=None, scopes=None,
expires_at=None, crypto_backend=default_backend()):
""" private_key should be provided in HEX, WIF or binary
format
domain_name should be a valid domain
manifest_uri should be a valid URI
redirect_uri should be a valid URI
scopes should be a list
expires_at should be a float number of seconds since
the epoch
"""
if not manifest_uri:
manifest_uri = domain_name + '/manifest.json'
```

```
if not redirect_uri:
redirect_uri = domain_name
if not scopes:
scopes = []
if not expires_at:
expires_at = time.time() + 3600 # next hour
66
self.private_key = private_key
self.domain_name = domain_name
self.manifest_uri = manifest_uri
self.redirect_uri = redirect_uri
self.scopes = scopes
self.expires_at = expires_at
self.tokenizer = Tokenizer(crypto_backend=crypto_backend)
def _payload(self):
now = time.time()
payload = {
'jti': str(uuid.uuid4()),
'iat': str(now),
'exp': str(now + self.expires_after),
'iss': None,
'public_keys': [],
'domain_name': self.domain_name,
'manifest_uri': self.manifest_uri,
'redirect_uri': self.redirect_uri,
'scopes': self.scopes
}
if self.private_key:
public_key =
BitcoinPrivateKey(self.private_key).public_key()
```

```
address = public_key.address()
payload['public_keys'] = [public_key.to_hex()]
payload['iss'] = make_did_from_address(address)
return payload
@classmethod
def fetch_app_manifest(cls, token):
# decode the token
try:
decoded_token = cls.decode(token)
except DecodeError:
return None
try:
return
requests.get(decoded_token['payload']['manifest_uri']).json()
except (requests.exceptions.RequestException,
ValueError):
# ValueError for non-json responses
return None
67
def redirect_url(self):
return 'blockstack:' + self.token()
Клас AuthResponse
class AuthResponse(AuthMessage):
""" Interface for creating signed auth response tokens, as
well as decoding
and verifying them.
"""
verify_methods = [
is_expiration_date_valid,
is_issuance_date_valid,
```

```
do_signatures_match_public_keys,
do_public_keys_match_issuer,
do_public_keys_match_username
]
def __init__(self, private_key, profile=None, username=None,
expires_at=None,
crypto_backend=default_backend()):
""" private_key should be provided in HEX, WIF or binary
format
profile should be a dictionary
username should be a string
expires_at should be a float number of seconds since
the epoch
"""
if not private_key:
raise ValueError('Private key is missing')
if not profile:
profile = {}
if not expires_at:
expires_at = time.time() + 30 * 24 * 3600 # next month
self.private_key = private_key
self.public_key =
BitcoinPrivateKey(self.private_key).public_key()
self.address = self.public_key.address()
self.profile = profile
self.username = username
self.expires_at = expires_at
68
self.tokenizer = Tokenizer(crypto_backend=crypto_backend)
def _payload(self):
```

```
now = time.time()
return {
'jti': str(uuid.uuid4()),
'iat': str(now),
'exp': str(now + self.expires_after),
'iss': make_did_from_address(self.address),
'public_keys': [self.public_key.to_hex()],
'profile': self.profile,
'username': self.username
}
```

ВІДОМІСТЬ МАГІСТЕРСЬКОЇ АТЕСТАЦІЙНОЇ РОБОТИ

№ док.	*Позначення*	*Найменування*	*Дод. від*
1		Текстові документи	
2	ГЮІК.ХХХХХХ.166Стз.09ПЗ	Пояснювальна записка	81 ст.
3		Графічні документи	
4		Презентаційний матеріал	15 А4.
5		Інші матеріали	
		Рецензія	1 с.
		Відгук керівника	1 с
		Електронна версія ПЗ	1 шт.

Ізм	*Аркуш*	*№ докум.*	*Підпис*	*Дата*	ГЮІК.ХХХХХХ.166Стз.09ВД		
Розроб.		Слиш О.В.			Децентралізована система ідентифікації Відомість атестаційної роботи	*Літ.*	*Лист* 1 / *Листо* 1
Перевір.		Мелникова О.А.				Т	
Н.контр.		Шевченко Л.П.				ХНУРЕ Кафедра БІТ	
Затверд.		Халімов Г.З.					