PYTHON BACKEND FEJLESZTÉS

7. forduló

CAMBRIDGE MOBILE TELEMATICS

A kategória támogatója: Cambridge Mobile Telematics (TrueMotion)

RENDELKEZÉSRE ÁLLÓ IDŐ:

25:00

Ismertető a feladathoz

Decorators, Pattern matching

Felhasznált idő: 01:48/25:00 **Elért pontszám:** 0/21

1. feladat 0/13 pont

Create a decorator which parses the payload from context-local state and passes the created CrashPrediction instance to the decorated function with name *model*.

Define your decorator the way that the decorated function does not lose its original name.

```
class ModelBase(ABC):
    @classmethod
    def create(cls, fields: Dict):
        return cls(**fields)
class CrashPrediction(ModelBase):
    def __init__(self, latitude: float, longitude: float, confidence: float):
        self.latitude = latitude
        self.longitude = longitude
        self.confidence = confidence
    def __str__(self):
        return f"latitude: {self.latitude}, longitude: {self.longitude}, confidence: {self.confidence}
class RequestContext:
    __model_context: ContextVar[Optional[Dict]] = ContextVar('model_context', default=None)
    @staticmethod
```

```
    def set_payload(model: Dict) -> None:
        RequestContext.__model_context.set(model)
    @staticmethod
    def get_payload() -> Optional[Dict]:
        return RequestContext.__model_context.get()
def inject_model(model_class: ModelBase):
    raise NotImplementedError
@inject_model(CrashPrediction)
def request_handler(param_1: str, model: CrashPrediction):
    print(f"param_1: {param_1}")
    print(f"Request model: {model}")
       # usage example
if __name__ == "__main__":
    RequestContext.set_payload({
        "longitude": 19.042935,
        "latitude": 47.498830,
        "confidence": 0.5
    })
    request_handler(param1="test_param")
```

Válasz

```
    def inject_model(model_class: ModelBase):
        def decorator_inject_model(f: Callable[..., Any]) -> Any:
            @wraps(f)
            def wrapper_inject_model(*args: Any, **kwargs: Any) -> Any:
                payload = RequestContext.get_payload()
                kwargs['model'] = CrashPrediction(**payload)
                return f(*args, **kwargs)
            return wrapper_inject_model
        return decorator_inject_model
```

(Answer A)

```
    def inject_model(model_class: ModelBase):
        def decorator_inject_model(f: Callable[..., Any]) -> Any:
            original_function_name = f.__name__
            def wrapper_inject_model(*args: Any, **kwargs: Any) -> Any:
                payload = RequestContext.get_payload()
                kwargs['model'] = CrashPrediction(**payload)
                f.__name__ = original_function_name
                return f(*args, **kwargs)
            return wrapper_inject_model
        return decorator_inject_model
```

(Answer B)

```
    def inject_model(model_class: ModelBase):
        def decorator_inject_model(f: Callable[..., Any]) -> Any:
            original_function_name = f.__name__
```

```
            def wrapper_inject_model(*args: Any, **kwargs: Any) -> Any:
                payload = RequestContext.get_payload()
                model = CrashPrediction(payload)
                f.__name__ = original_function_name
                return f(*args, model=model)
            return wrapper_inject_model
        return decorator_inject_model
```

(Answer C)

```
    def inject_model(model_class: ModelBase):
        def decorator_inject_model(f: Callable[..., Any]) -> Any:
            wraps(f)
            payload = RequestContext.get_payload()
            kwargs['model'] = CrashPrediction(**payload)
            return f(model=model)
        return decorator_inject_model
```

(Answer D)

Magyarázat

A: Passes the *param1* and *model* params to the function and the name of the function does not change

B: Function name changes

C: *param1* param is not passed to the function

D: Throws error

2. feladat 0/8 pont

Write pattern matcher for logs and ping messages. Handle the message only if **id** is present in the user, otherwise throw an **UnsupportedMessageType** exception.

This assignment requires knowledge about pattern matching, introduced in Python3.10 (beta).

Example messages:

```
{
    'message_type': 'PING',
    'value': 'PING',
    'user': {
        'id': '1687654'
    },
    'additional_info': {}
},
{
```

```
    'message_type': 'LOG',
    'message': 'Log message',
    'client': {
        'id': '1687654'
    },
    'additional_info': {}
}
```

Válasz

```
case {'message_type': MessageType.PING.value, 'value': data, 'user': user} | {'message_type': Mes
    return process_log_like(data, user)
case _:
    raise UnsupportedMessageType()
```

```
case {'message_type': MessageType.PING.value} | {'message_type': MessageType.LOG.value, 'message'
    if 'id' is not in user:
        raise UnsupportedMessageType()
    return process_log_like(data, user)
```

```
case guard(lambda user: 'id' in user) {'message_type': MessageType.PING.value, 'value': data, 'us
    return process_log_like(data, user)
```

```
case message:
    if message['type'] in ('PING', 'LOG') and 'id' in user:
        return process_log_like(data, user)
```

Magyarázat

Only the correct answer runs.

Világos