Développez des applications web performantes avec Litestar

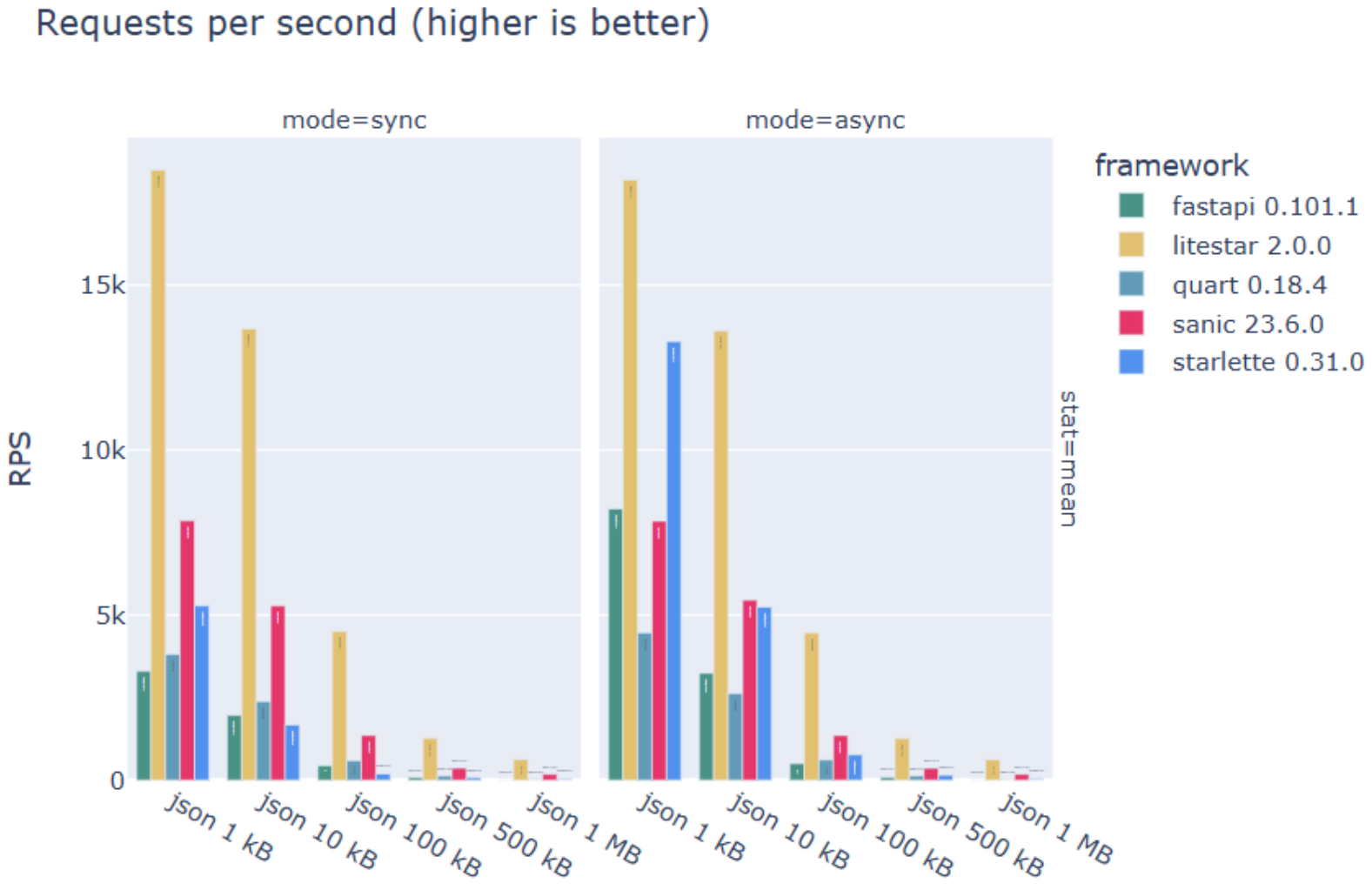
Irina Mitiaïeva, Cyril Monmouton

Qu'est-ce que Litestar ?

- → Framework web Python asynchrone et moderne
- → Rapide, typé, facile d'utilisation
- → v2 de Starlite, inspiré de Starlette
- → 7.6k étoiles sur GitHub
- → Dernière version: **2.18.0**
- → Date de sortie: 2021

https://litestar.dev/

Pourquoi Litestar ?



https://github.com/litestar-org/api-performance-tests

C'est parti!

```
pip install 'litestar[standard]'
```

Bonjour, ~~monde~~ Litestar!

app.py

```
from litestar import Litestar, get

@get("/")
async def index() -> str:
    return "Bonjour, Litestar!"

app = Litestar([index])
```

Organisation des routes en contrôleur

```python
from litestar import Litestar, Controller, get, post, put, patch, delete


class MyController(Controller):
    path = "/controller"

    @get()
    def get_handler(self) -> None: ...

    @post()
    def post_handler(self) -> None: ...

    @put()
    def put_handler(self) -> None: ...

    @patch()
    def patch_handler(self) -> None: ...

    @delete()
    def delete_handler(self) -> None: ...

app = Litestar(route_handlers=[MyController])
```

Gestion des paramètres

- → Paramètres « path »
- → Paramètres « query »
- → litestar.params.Parameter()
- → Entêtes et cookies

Paramètres « path »

```python
@get(path="/path-date/{your_date:str}")
async def path_date(votre_date: date) -> int:
    return (date.today() - votre_date).days

@get(path="/path-int/{version:int}")
async def path_int(
        version: Annotated[
            int,
            Parameter(
                ge=3,
                le=13,
                description="Validation et documentation pour les paramètres de chemins",
            )
        ]
) -> int:
    return version
```

Paramètres « query »

```python
@get(path="/query-str")
async def query_str(nom: str) -> str:
    return f« Salut ici, {nom}"

@get("/query-coercion")
async def query_coercion(nombre: int, nombre_flottant: float, strings: list[str])
-> dict[str, Any]:
    return {
        "int": nombre,
        "float": nombre_flottant,
        "list": strings,
    }
```

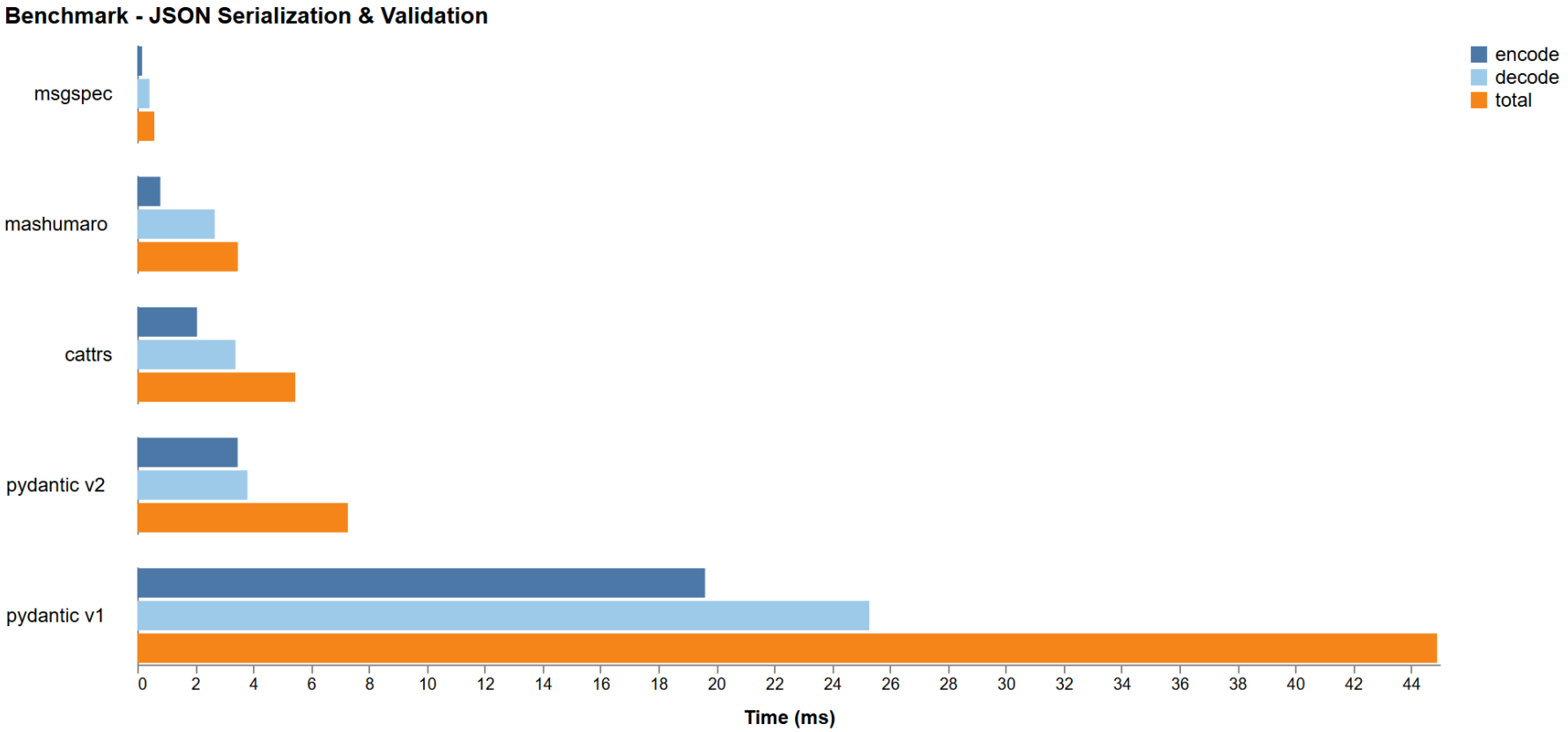
Entêtes et cookies

```
from litestar.exceptions import PermissionDeniedException

TOKEN_VALIDE = "super-secret-secret"
VALEUR_COOKIE_VALIDE = "cookie-secret"
BD_UTILISATEURS = {1: { "id": 1, « nom": "John Doe" }}

@get(path="/header-cookies/{id_utilisateur:int}/")
async def get_user(
        id_utilisateur: int,
        cle_api: Annotated[str, Parameter(header="X-API-KEY")],
        cookie: Annotated[str, Parameter(cookie="my-cookie-param")],
) -> dict[str, str]:
    if cle_api != TOKEN_VALIDE or cookie != VALEUR_COOKIE_VALIDE :
        raise PermissionDeniedException
    return BD_UTILISATEURS[id_utilisateur]
```

Modèles de données



Modèles de données

```python
from enum import Enum
from uuid import UUID, uuid4
from msgspec import Struct, field
from litestar.dto.msgspec_dto import MsgspecDTO
from litestar.dto.config import DTOConfig


class RoleUtilisateur(Enum):
    admin = "admin"
    moderateur = "moderateur"
    editeur = "editeur"
    lecteur = " lecteur"


class Utilisateur(Struct):
    nom: str
    role: RoleUtilisateur
    id: UUID = field(default_factory=uuid4)


class UserCreateDTO(MsgspecDTO[Utilisateur]):
    config = DTOConfig(exclude={"id"})

class UserReadDTO(MsgspecDTO[Utilisateur]): ...
```

Contrôleur Utilisateur

path = "/users"
dto = UserCreateDTO
return_dto = UserReadDTO

À vous de jouer ;-)

Gestion des erreurs

```
from litestar import Litestar
from litestar.exceptions import ValidationException
from litestar.status_codes import HTTP_500_INTERNAL_SERVER_ERROR

import error_handler

app = Litestar(
   route_handlers=[],
   exception_handlers={
      # C'est à vous de créer ces deux fonctions:
      # ValueError: value_error_handler,
      # ValidationException: validation_exception_handler,
      HTTP_500_INTERNAL_SERVER_ERROR: error_handler.internal_server_error_handler,
   }
)


def internal_server_error_handler(request: Request, exc: Exception) -> Response:
   return Response(media_type=MediaType.TEXT,
                   content=f« Erreur serveur: {exc}",
                   status_code=500,
   )
```

https://docs.litestar.dev/latest/usage/exceptions.html

Journalisation

```
logging_config = LoggingConfig(
    root={"level": "INFO", "handlers": ["queue_listener"]},
    formatters={
        "standard": {"format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s"}
    },
    log_exceptions="always",
)
```

```
app = Litestar(
    ...
    logging_config=logging_config,
)
```

- Utilise des logs structurés
- Intégrer avec uvicorn / gunicorn
- Formateur personnalisés supportés

https://docs.litestar.dev/latest/usage/logging.html

Mise en cache

- Usage simple pour get/post/put/patch/delete
- Clé de mise en cache personnalisable
- Filtre de mise en cache personnalisable

```
@get("/cache-demo", cache=15, cache_key_builder=key_builder)
async def cached_during_15s_handler() -> str:
    await sleep(2)
    return "cached"

app = Litestar(
    [cached_during_15s_handler],
    response_cache_config=response_cache_config,
)
```

https://docs.litestar.dev/latest/usage/caching.html

Garde de sécurité

```python
from litestar.connection import ASGIConnection
from litestar.exceptions import NotAuthorizedException
from litestar.handlers import BaseRouteHandler


def api_key_guard(connection: ASGIConnection, _: BaseRouteHandler) -> None:
    """ @:param: connection: ASGIConnection est l'instance d'un objet 'Request'
    ou 'WebSocket' (ils héritent tout les deux de la classe ASGIConnection)
    """
    if "X-API-KEY" not in connection.headers:
        raise NotAuthorizedException
```

- Endpoint
- Contrôleur
- Routeur
- Application

https://docs.litestar.dev/latest/usage/security/guards.html

Injection des dépendences

```python
from litestar.di import Provide


app = Litestar(
    route_handlers=[di_demo],
    dependencies={"app_config": Provide(lambda: "My awesome app")},
)


@get("/di-demo")
async def di_demo(app_config: str) -> str:
    return f"I am {app_config}"
```

- Endpoint
- Contrôleur
- Routeur
- Application

https://docs.litestar.dev/latest/usage/dependency-injection.html

Observabilité

```
pip install 'litestar[opentelemetry]'
```

```
from litestar.contrib.opentelemetry import OpenTelemetryConfig, OpenTelemetryPlugin

app = Litestar(
    ...
    plugins=[OpenTelemetryPlugin(OpenTelemetryConfig())]
)
```

https://docs.litestar.dev/latest/usage/metrics/open-telemetry.html

Middlewares

```python
from litestar import Litestar
from litestar.types import ASGIApp, Scope, Receive, Send

def middleware_factory(app: ASGIApp) -> ASGIApp:
    async def mon_middleware(scope: Scope, receive: Receive, send: Send) -> None:
        print("Avant")
        await app(scope, receive, send)
        print("Après")

    return mon_middleware

@get("/test-middleware")
async def my_handler() -> int:
    print("Test middleware")
    return 42

app = Litestar(route_handlers=[my_handler], middleware=[middleware_factory])
```

- Endpoint
- Contrôleur
- Routeur
- Application

intégrés: CORS, CSRF, Compression, Limite, Journalisation, Session

https://docs.litestar.dev/2/usage/middleware/index.html

Merci!

https://github.com/irmiti/litestar-workshop