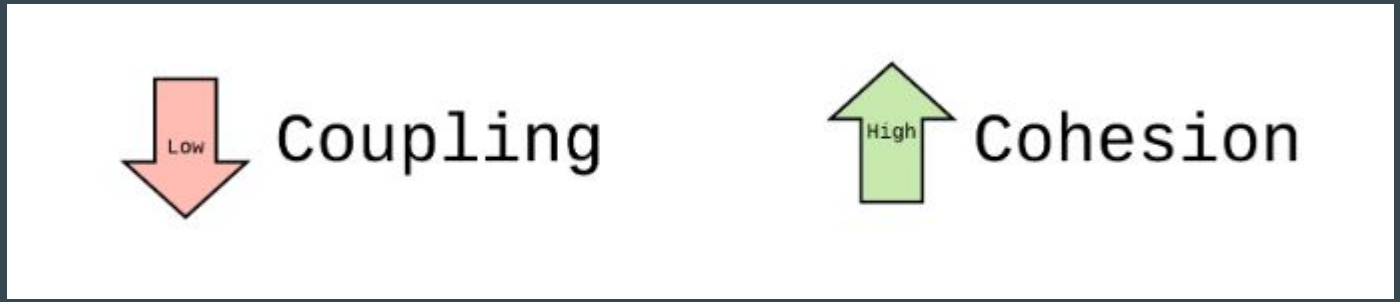
Dependency Injection

•••

Python

Goals

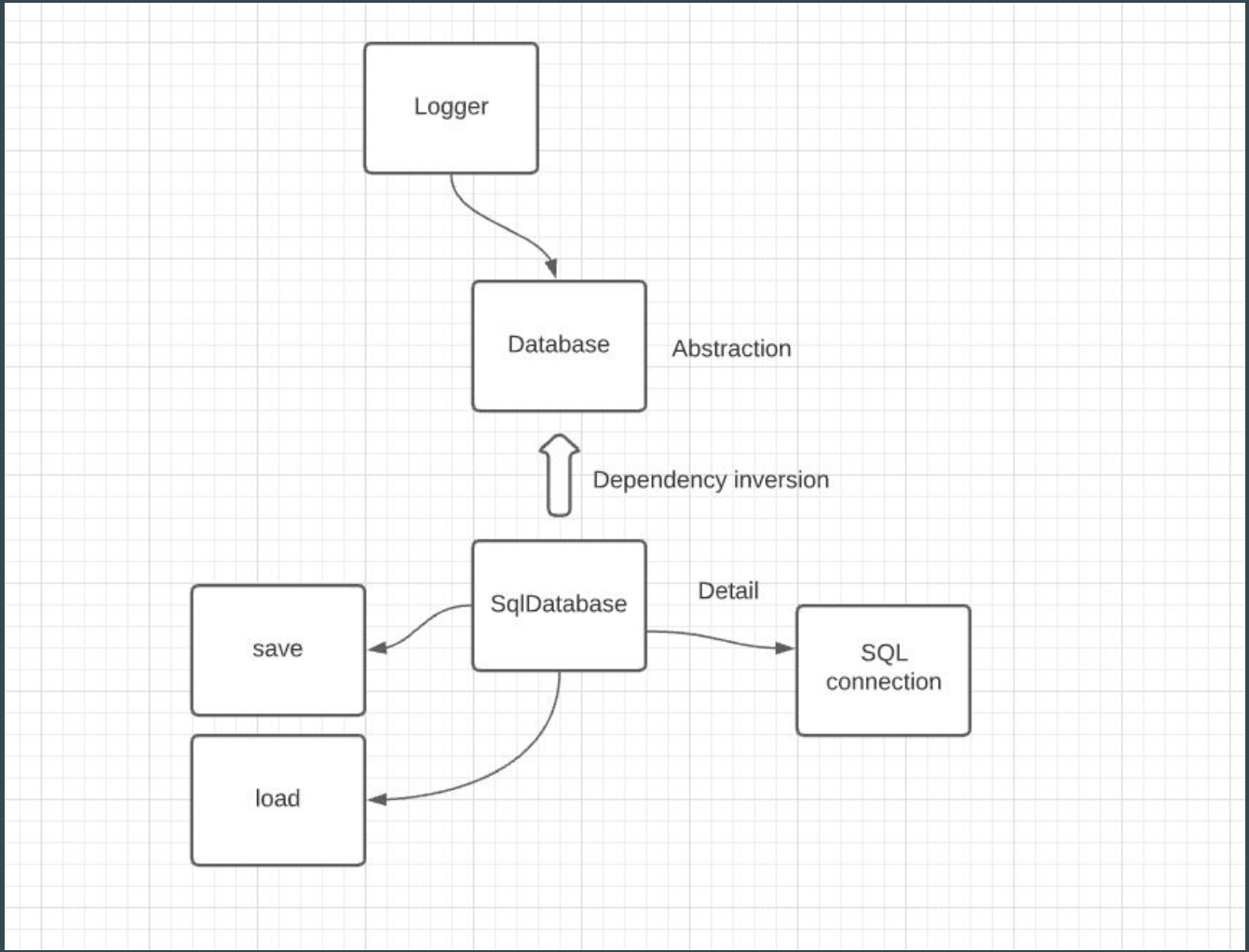
- We will learn what is coupling & cohesion
- Show how to use dependencies inversion and dependency injection pattern in python and it’s benefits
- Show how to use dependency injection mechanism using library
- Bonus



- High coupling. If the coupling is high it’s like using a superglue or welding. No easy way to disassemble.
- High cohesion. High cohesion is like using the screws. Very easy to disassemble and assemble back or assemble a different way. It is an opposite to high coupling.

Live code example

- Firebase logger example
- Refactor logger using “Dependency inversion” principle
- Refactor logger using python “Dependency Injector library”


Logger
Database
Abstraction
Dependency inversion
SqlDatabase
Detail
save
load
SQL
connection

```python
import os


class ApiClient:

    def __init__(self, api_key: str, timeout: int):
        self.api_key = api_key  # <-- dependency is injected
        self.timeout = timeout  # <-- dependency is injected

class Service:

    def __init__(self, api_client: ApiClient):
        self.api_client = api_client  # <-- dependency is injected

def main(service: Service):  # <-- dependency is injected
    ...


if __name__ == '__main__':
    main(
        service=Service(
            api_client=ApiClient(
                api_key=os.getenv('API_KEY'),
                timeout=os.getenv('TIMEOUT'),
            ),
        ),
    )
```

```python
from dependency_injector import containers, providers
from dependency_injector.wiring import inject, Provide


class Container(containers.DeclarativeContainer):

    config = providers.Configuration()

    api_client = providers.Singleton(
        ApiClient,
        api_key=config.api_key,
        timeout=config.timeout.as_int(),
    )

    service = providers.Factory(
        Service,
        api_client=api_client,
    )

@inject
def main(service: Service = Provide[Container.service]):
    ...

if __name__ == '__main__':
    container = Container()
    container.config.api_key.from_env('API_KEY')
    container.config.timeout.from_env('TIMEOUT')
    container.wire(modules=[sys.modules[__name__]])

    main()  # <-- dependency is injected automatically

    with container.api_client.override(mock.Mock()):
        main()  # <-- overridden dependency is injected automatically

```

Bonus

How to use this library in django?

References

Python

- Dependency injector: https://python-dependency-injector.ets-labs.org/
- Import as antipattern: https://www.youtube.com/watch?v=qkGxy4c64Jg&ab_channel=PyConUK
- DI Quickly, PyBay: https://www.youtube.com/watch?v=_Ney7NA2u_M&ab_channel=SFPython

Java:

DI in spring

- https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring