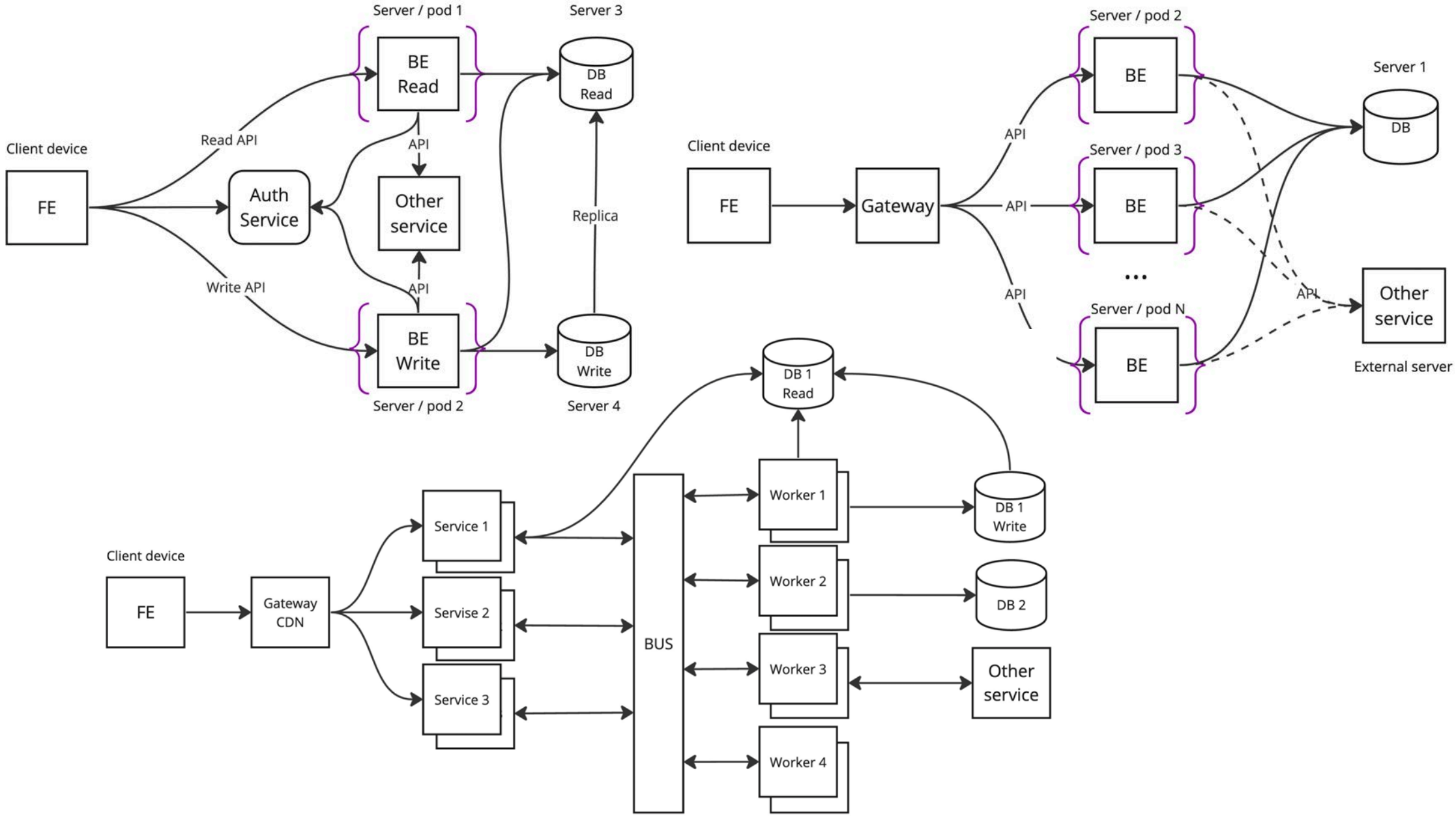
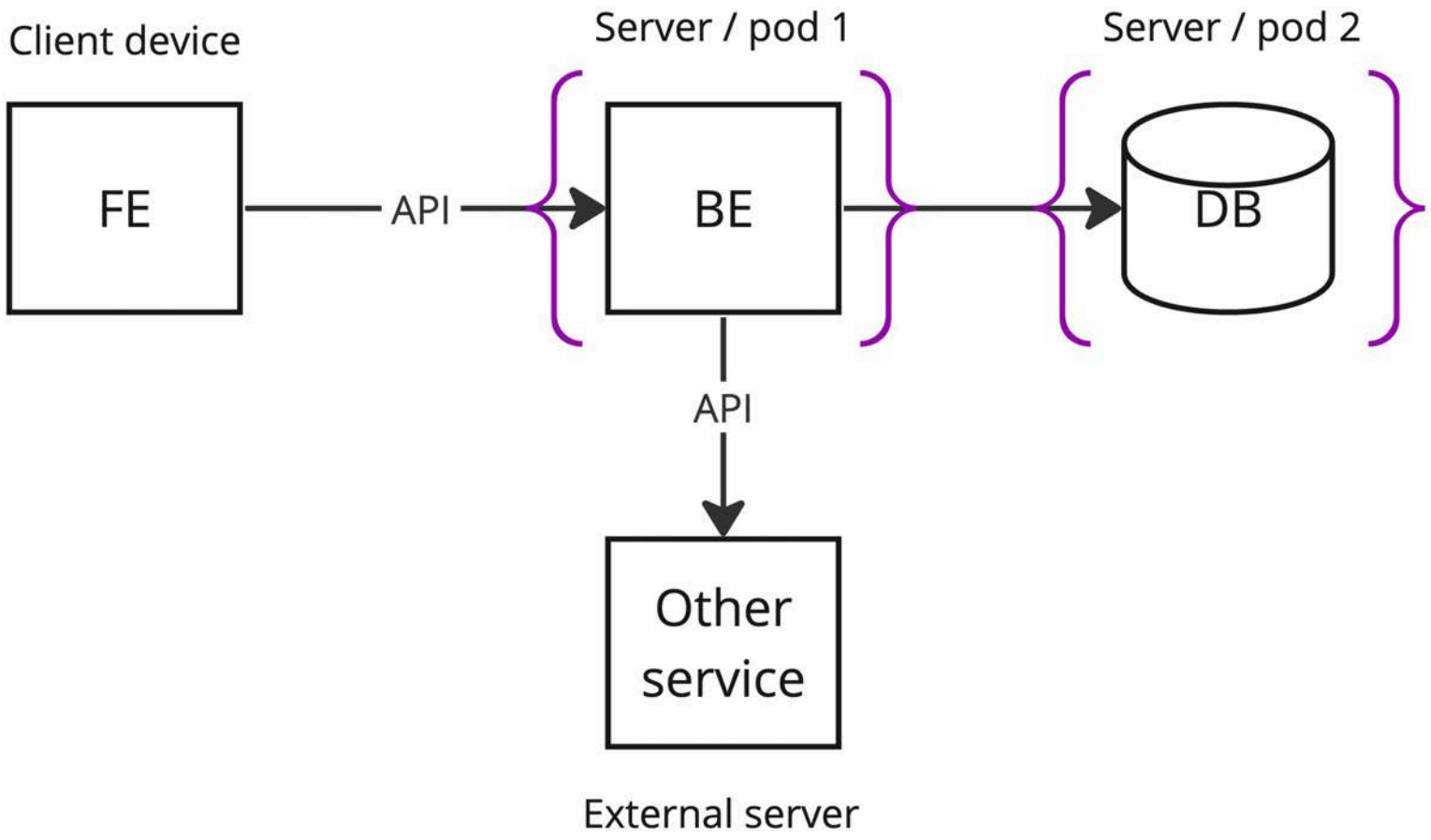
How to improve application and code quality without overengineering

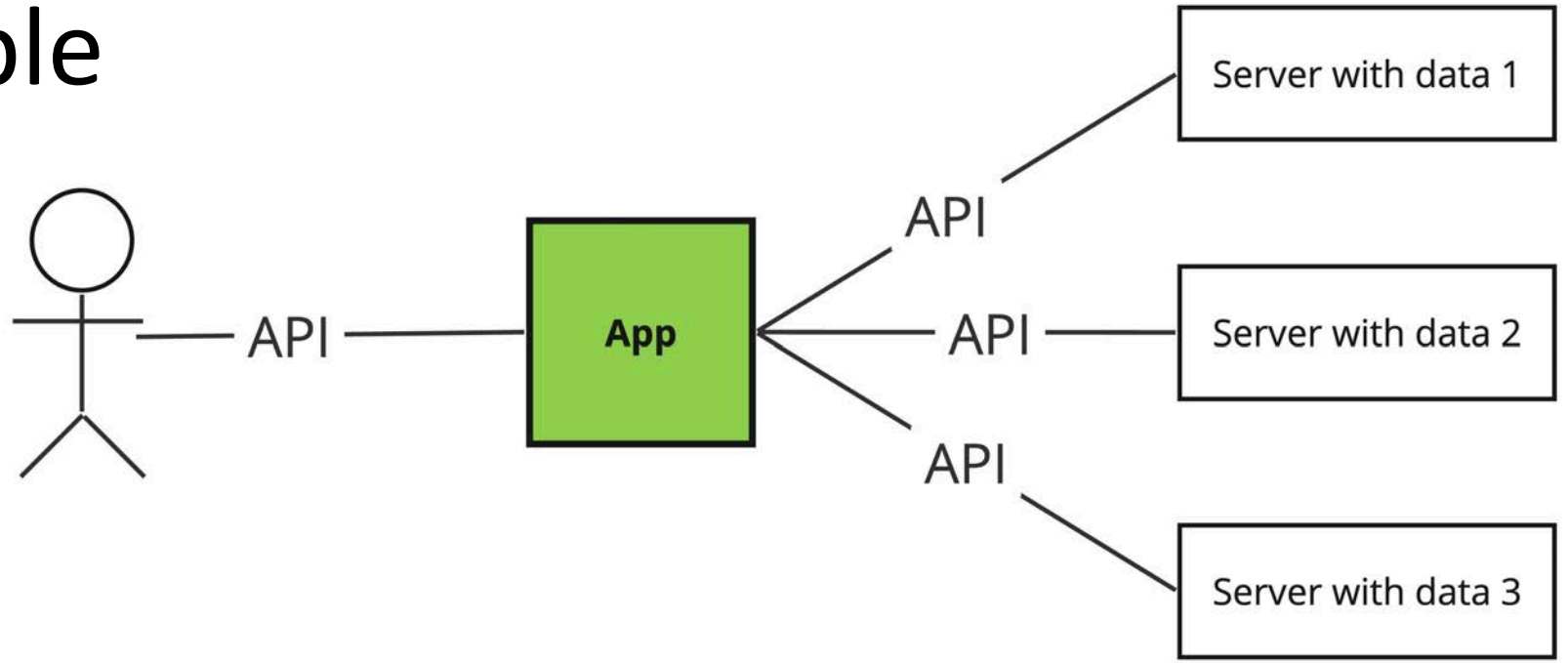
Aleksei Sharypov


Server / pod 1
Server 3
BE
Read
DB
Read
Client device
Read API
API
FE
Auth
Service
Other
service
Replica
Write API
API
BE
Write
DB
Write
Server / pod 2
Server 4
Server / pod 2
Server 1
BE
DB
Client device
API
Server / pod 3
FE
Gateway
API
BE
API
...
API
Server / pod N
Other
service
BE
External server
DB 1
Read
Client device
Worker 1
DB 1
Write
Service 1
FE
Gateway
CDN
Servise 2
Worker 2
DB 2
BUS
Service 3
Worker 3
Other
service
Worker 4

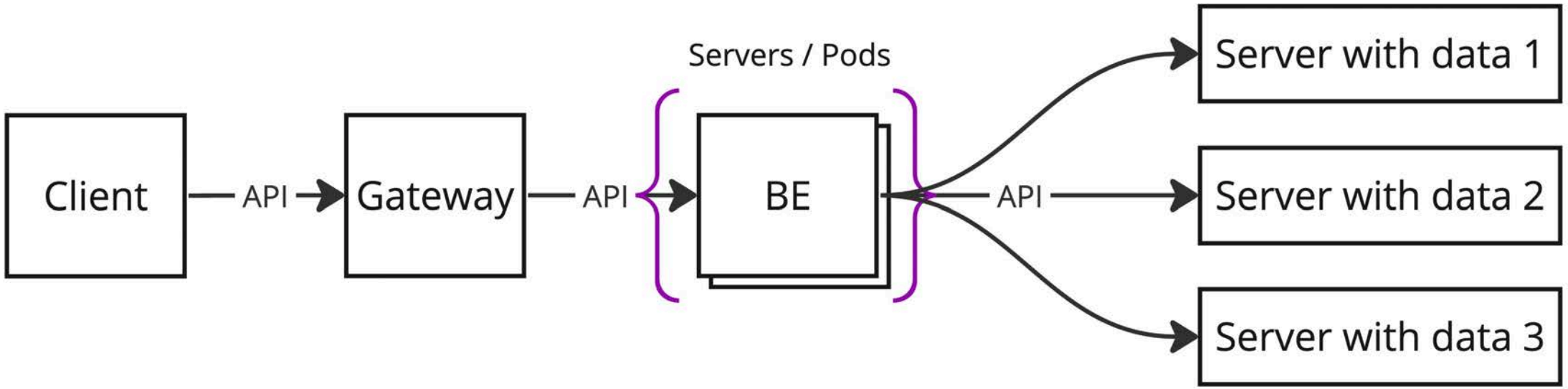
- FastAPI
- Django
- Flask



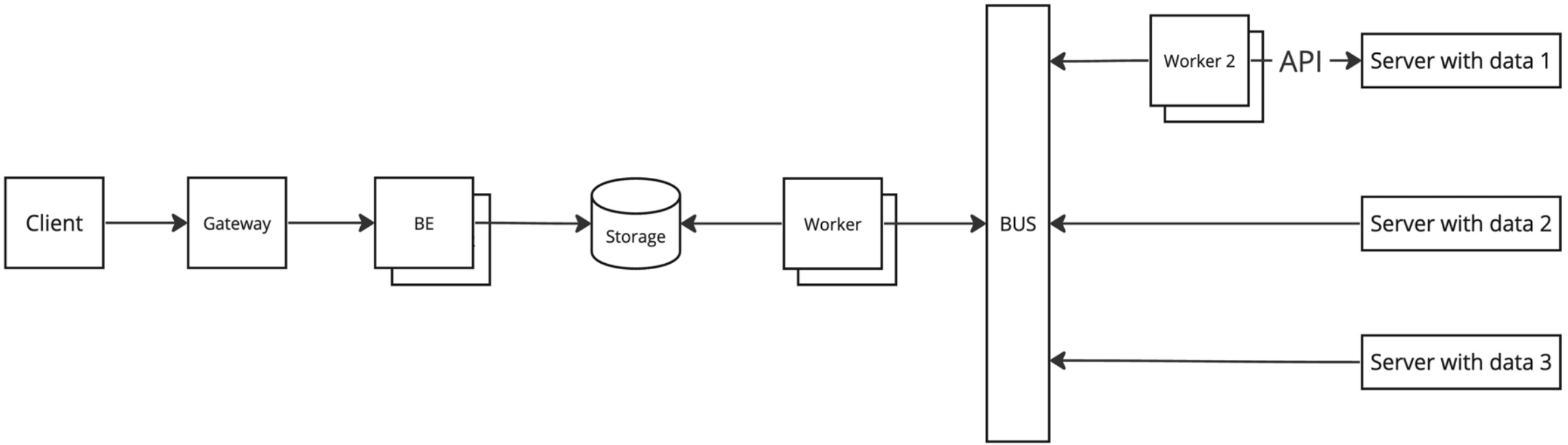
Example



- The statistics analysis system requires a separate API endpoint that will return statistics in a specific format (format provided).
- Existing service APIs must be used as data sources.
- The output data requires aggregation and processing (algorithm provided).



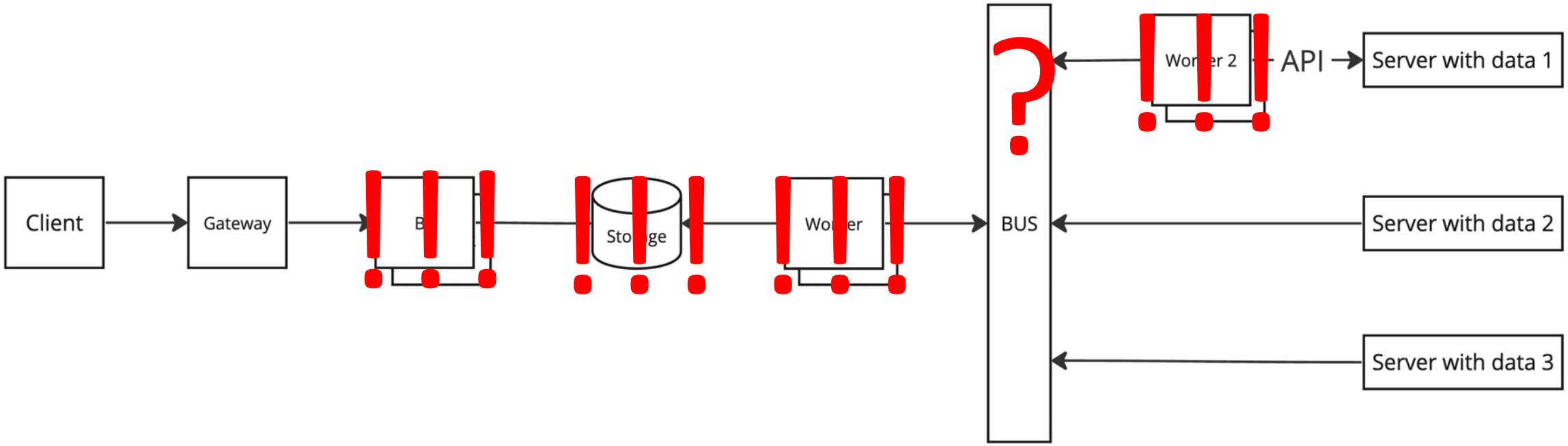
- A sufficient solution would be to implement a simple application using FastAPI or any other framework.
- It should:
  - asynchronously request sources via API in real-time,
  - process the received results,
  - return JSON in response to the query.

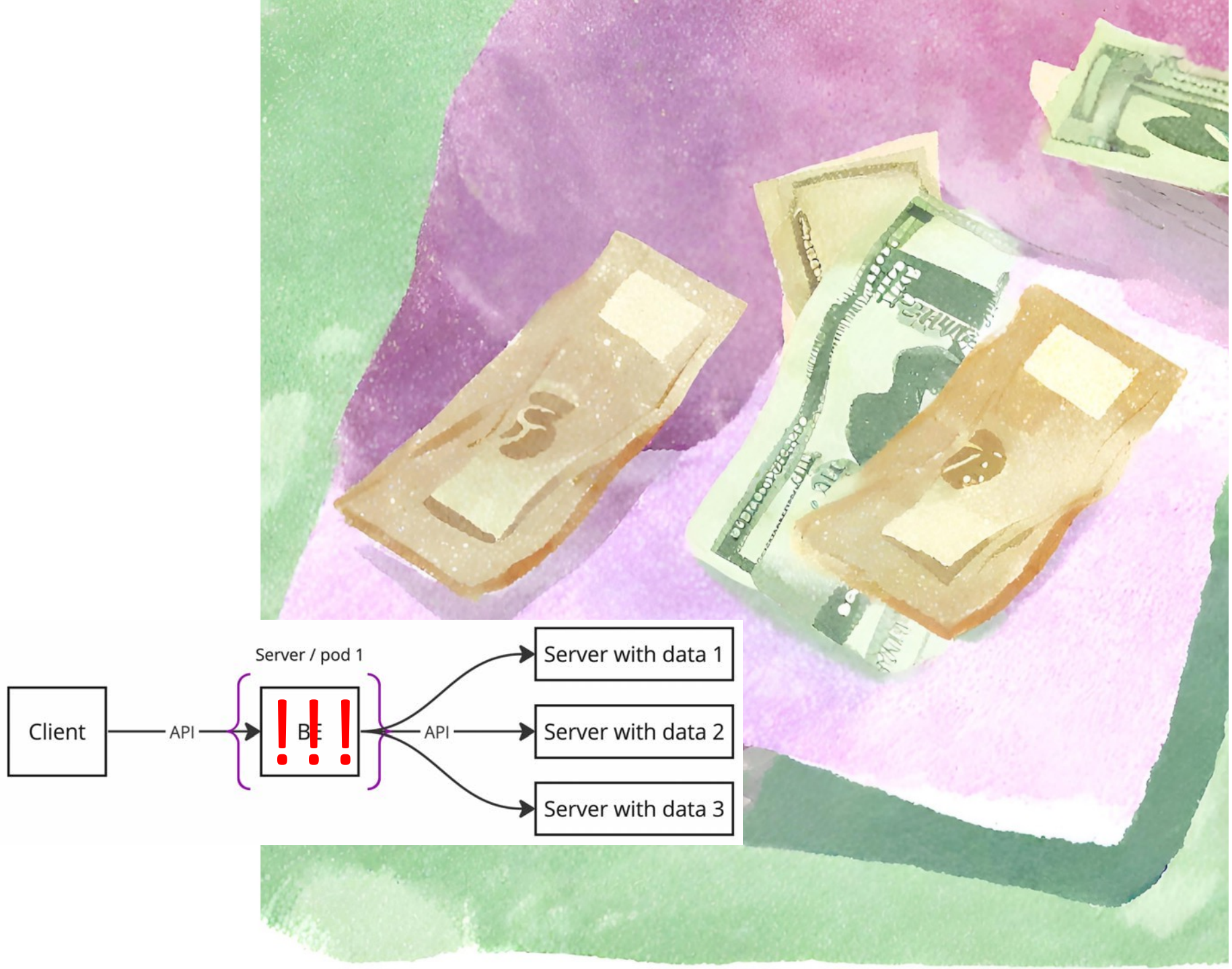


- The backend, through message queues, will receive data from sources via additional workers.
- It will store them in a local repository and serve the results from there.

~~Quality~~

~~Tests~~


Client
Gateway
BUS
Server with data 1
API
Server with data 2
Server with data 3


Server / pod 1
Client
API
BE
!!!
API
Server with data 1
Server with data 2
Server with data 3

1. An overengineered solution that exceeded the actual requirements.
2. Increased development costs and time.
3. Compromised code quality due to rushed development and postponed testing.
4. Difficulty in accommodating changes and modifications in the future, leading to higher maintenance costs.

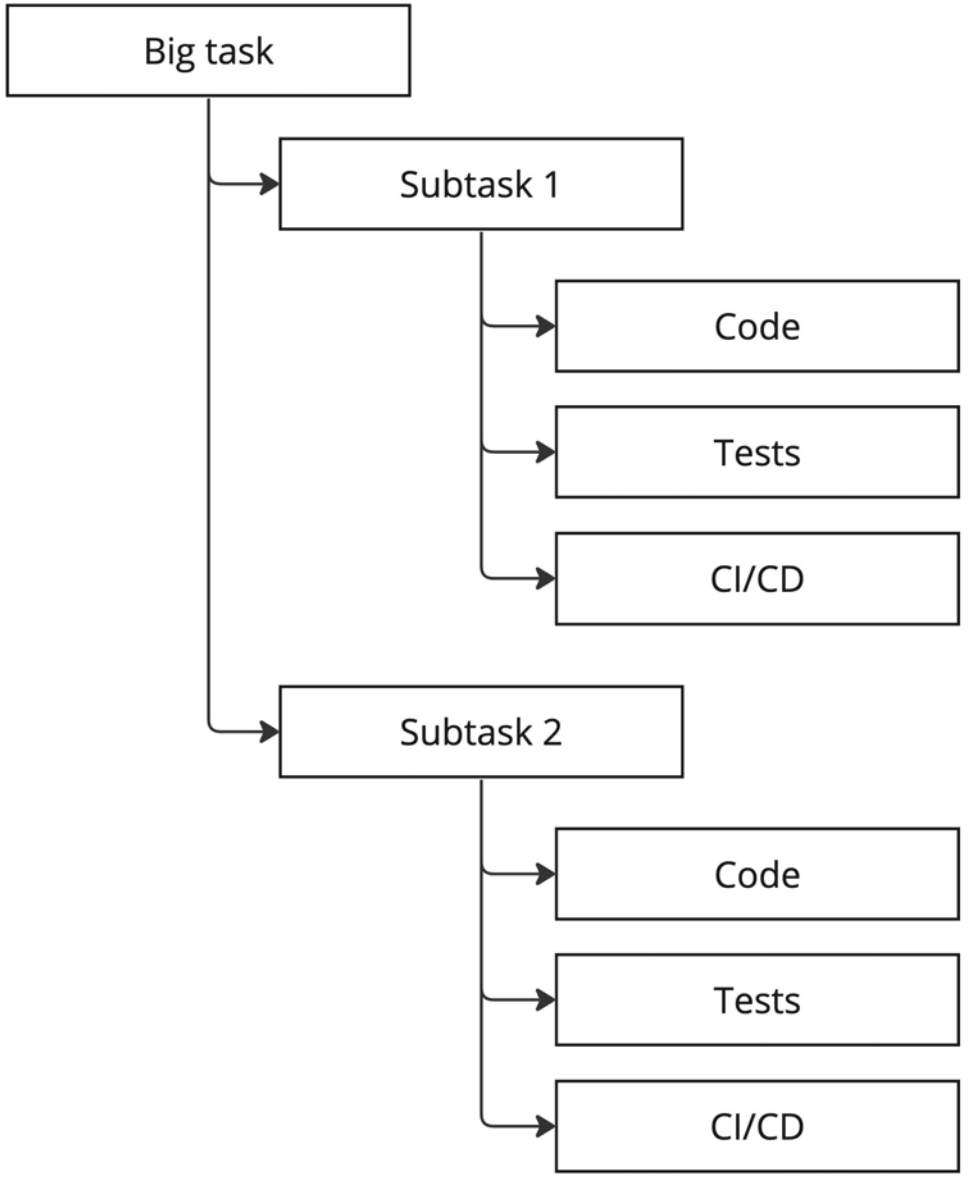
How to improve

1. Experienced developer.
2. Task refinement process.
3. Architecture.
4. Iterative process.
5. Refactoring and optimization.
6. **Tests.**

Architectural design

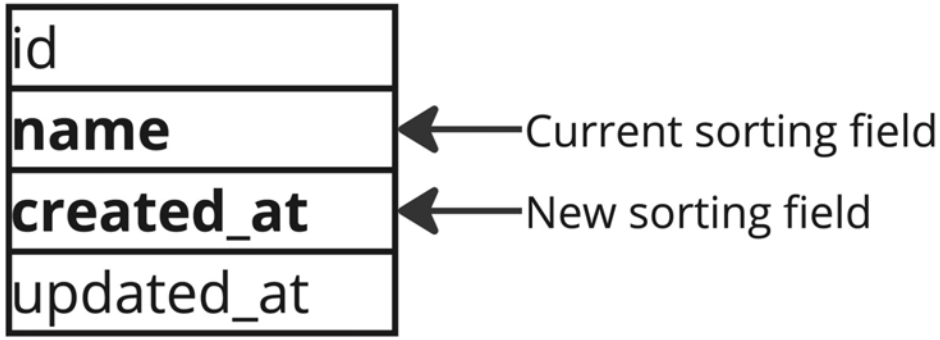
- Load capacity.
- Processing time.
- Correctness.
- Validation.
- Error handling.
- The possibility of using.

- Break down into subtasks.
- Work through each subtask:
  - Migrations.
  - Code revisions.
  - Tests.
  - Build and deployment to production.
  - Acceptance criteria.
- Specify dependencies.
- Establish coding and release order.



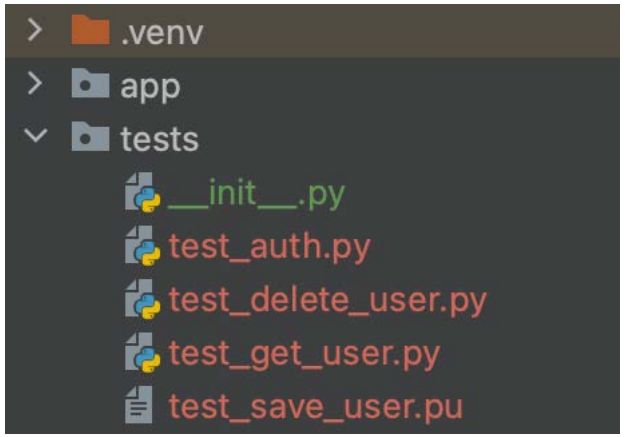
New sorting field

- Scenario 1
  - Create an index on the new field;
  - Deploy to production;
  - Modify the code;
  - Deploy to production;
  - Remove the old index if not in use;
  - Deploy to production.
- Scenario 2
  - Create an index on the new field;
  - Modify the code;
  - Remove the old index if not in use;
  - Deploy to production.




+31987 −157

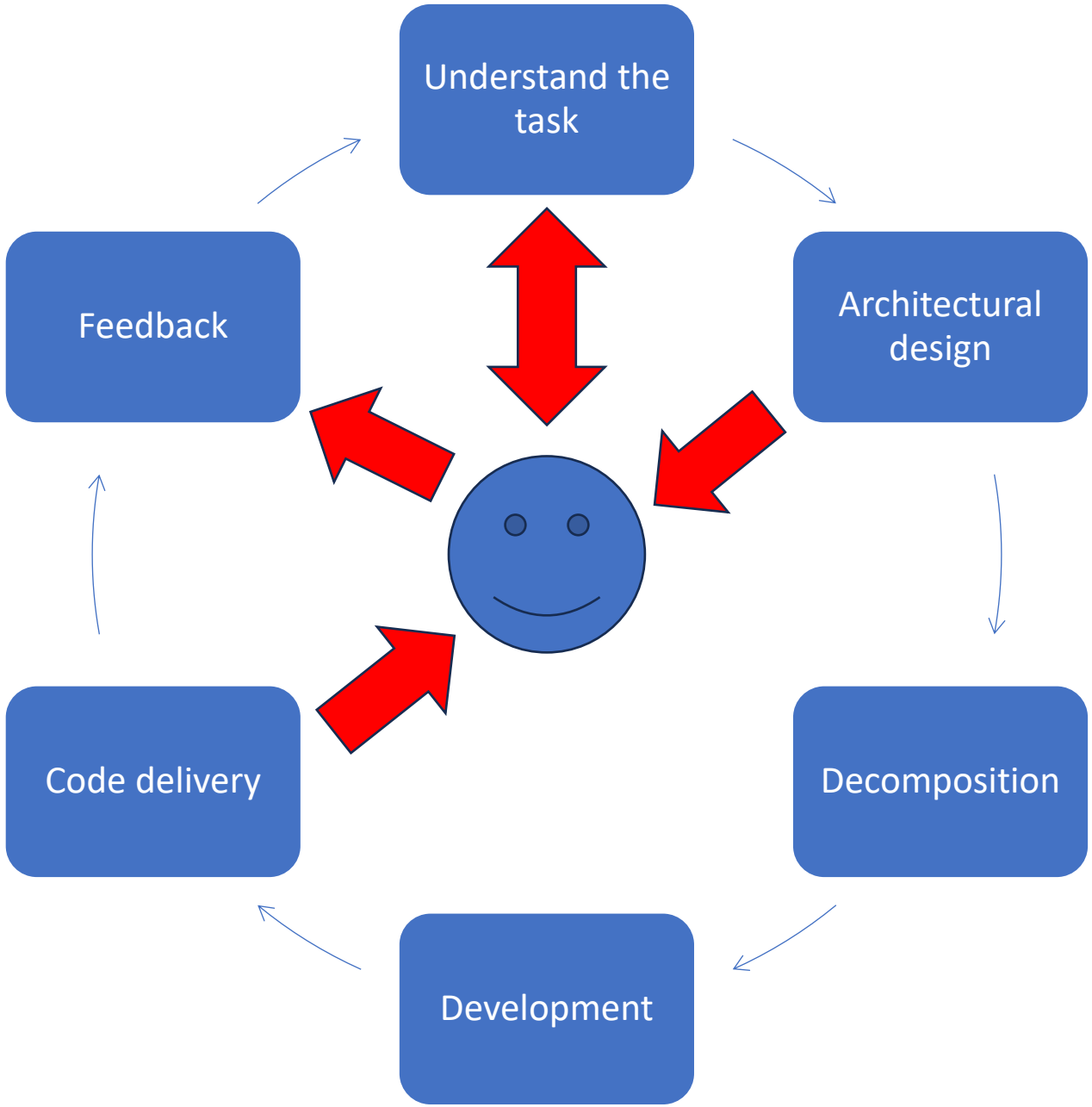

+105 −44




```python
from datetime import datetime
from fastapi.testclient import TestClient
from freezegun import freeze_time
from .main import app

client = TestClient(app)


Aleksei Sharypov
def test_my_info_success():
    initial_datetime = 1691377675
    with freeze_time(datetime.utcfromtimestamp(initial_datetime)):
        response = client.get("/my-info")
    assert response.status_code == 200
    result = {
        "user": {
            "id": 1,
            "firstname": "John",
            "lastname": "Smith",
            "phone": "+995000000000",
        },
        "timestamp": initial_datetime,
    }
    assert response.json() == result
    assert response.headers.get("X-Signature") == "479bb02f0f5f1249760573846de2dbc1"
```


Understand the task
Architectural design
Decomposition
Development
Code delivery
Feedback

“Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs. In essence when you refactor you are improving the design of the code after it has been written.”

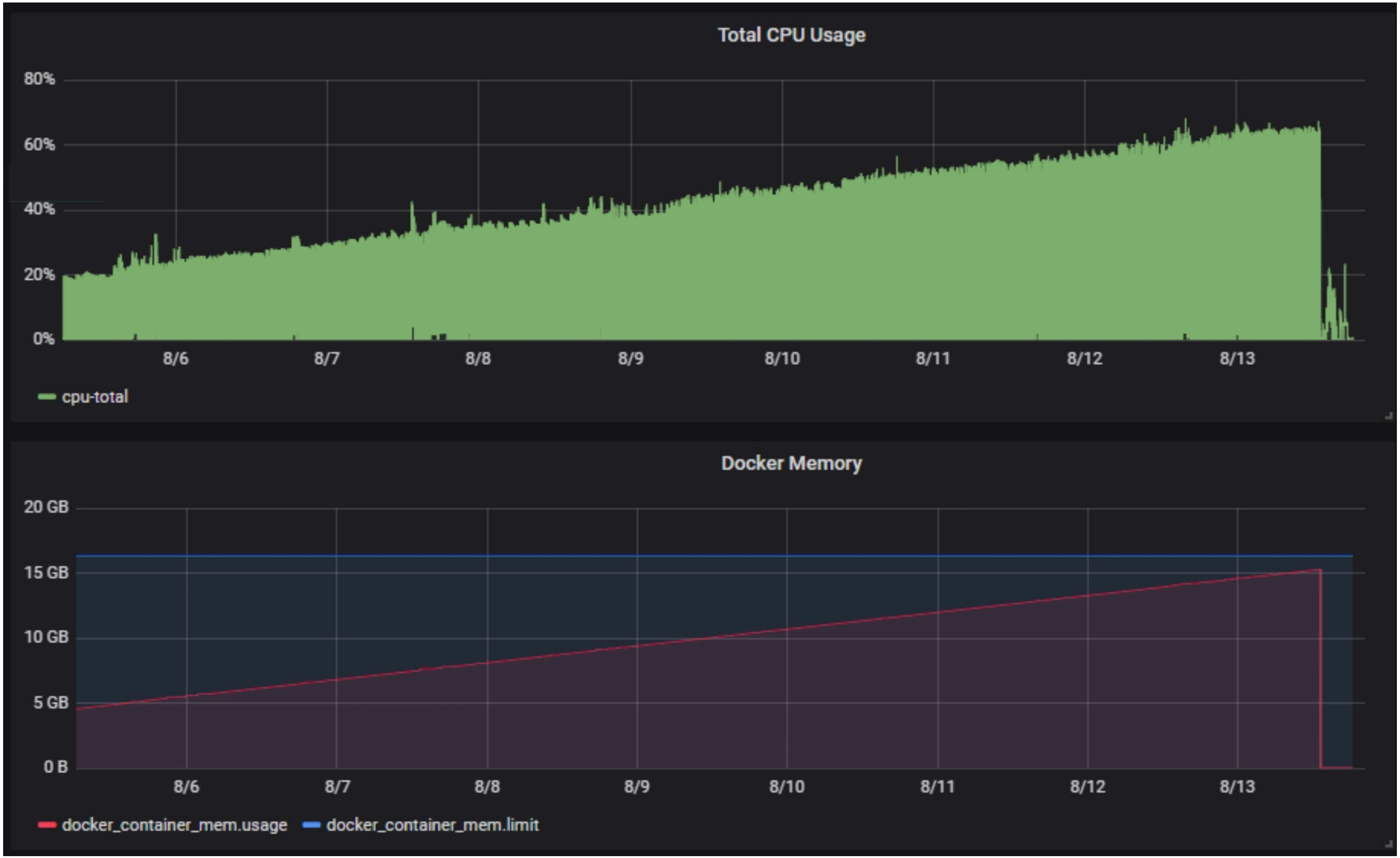
“Refactoring: Improving the Design of Existing Code” by Martin Fowler, with Kent Beck

Dirty spots for refactoring

- Duplicated code.
- Long methods.
- Large classes.
- Long parameter lists.
- Redundant temporary variables.
- Data classes.
- Ungrouped data.
- Debugging information.
- Others…

Optimization, not refactoring

- Optimization of queries to the database or other storage.
- Increasing speed.
- Reducing memory consumption.
- Fixing leaks.


Total CPU Usage
80%
60%
40%
20%
0%
8/6
8/7
8/8
8/9
8/10
8/11
8/12
8/13
cpu-total
Docker Memory
20 GB
15 GB
10 GB
5 GB
0 B
8/6
8/7
8/8
8/9
8/10
8/11
8/12
8/13
docker_container_mem.usage
docker_container_mem.limit