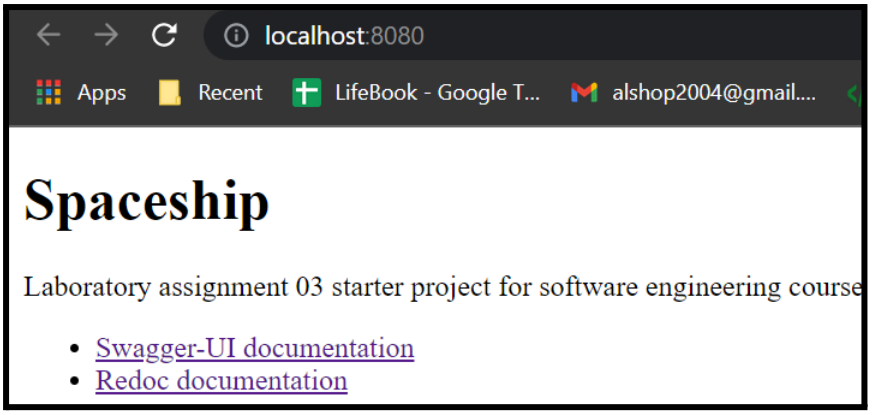
Python

Step 1(Create Dokerfile)

```
FROM python:slim-bullseye

WORKDIR /app
COPY . .
RUN pip install -r requirements/backend.in

CMD ["uvicorn", "spaceship.main:app", "--host=0.0.0.0", "--port=8080"]
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker build -t python-image .
[+] Building 45.3s (10/10) FINISHED
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker images
REPOSITORY      TAG       IMAGE ID       CREATED              SIZE
python-image    latest    f49e3f12c49f   About a minute ago   193MB
```



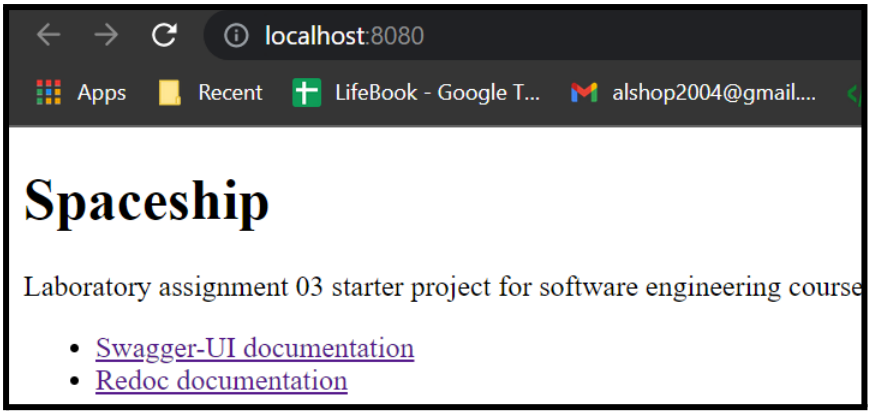
Build Quality: Not optimized
Basic image: medium size
Assembly time: 45.3 s
The size: 193 MB
Build: docker build -t python-image .
Run: docker run -dp 8080:8080 python-image
Link: LINK

Step 2(Make small changes)

```
    @app.get('/', include_in_schema=False, response_class=FileResponse)
    async def root() -> str:
        return 'build/index.html'

    return app
#Alex Shopiak was here
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker build -t python-image .
[+] Building 14.6s (10/10) FINISHED
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker images
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
python-image    latest    5951eb9b337f   2 minutes ago    193MB
```



Build Quality: Not optimized
Basic image: medium size
Assembly time: 14.6 s
The size: 193 MB
Build: docker build -t python-image .
Run: docker run -dp 8080:8080 python-image
Link: LINK

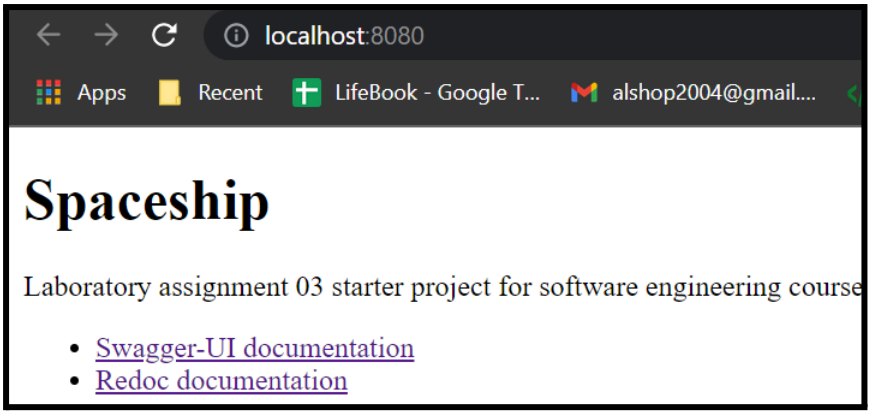
Step 3(Optimizing the Dockerfile)

```
FROM python:slim-bullseye

WORKDIR /app
COPY requirements/backend.in .
RUN pip install -r backend.in

COPY . .
CMD ["uvicorn", "spaceship.main:app", "--host=0.0.0.0", "--port=8080"]
```

```
    async def root() -> str:
        return 'build/index.html'

    return app
#Alex Shopiak was here
#Alex Shopiak was here twice
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker build -t python-image .
[+] Building 10.1s (10/10) FINISHED
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker images
REPOSITORY     TAG       IMAGE ID       CREATED              SIZE
python-image   latest    50ed317c4dcb   About a minute ago   193MB
```



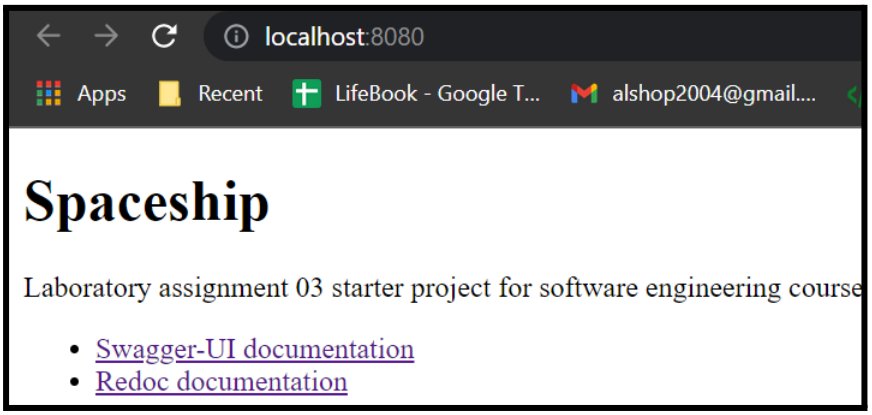
Build quality: optimized
Basic image: medium size
Build time: 10.1 s
The size: 193 MB
Build: docker build -t python-image .
Run: docker run -dp 8080:8080 python-image
Link: LINK

Step 4(Replace the basic image with a lighter one)

```
FROM python:alpine

WORKDIR /app
COPY requirements/backend.in .
RUN pip install -r backend.in

COPY . .
CMD ["uvicorn", "spaceship.main:app", "--host=0.0.0.0", "--port=8080"]
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker build -t python-image .
[+] Building 23.5s (11/11) FINISHED
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker images
REPOSITORY     TAG       IMAGE ID       CREATED         SIZE
python-image   latest    8b555320c633   2 minutes ago   123MB
```



Build quality: optimized
Basic image: lightweight
Assembly time: 23.5 s
The size: 123 MB
Build: docker build -t python-image .
Run: docker run -dp 8080:8080 python-image

Link: LINK

Step 5(Compare 2 types of images)

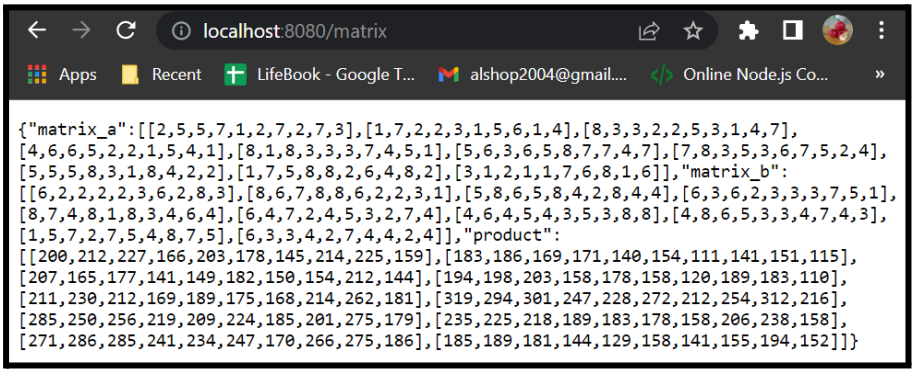
```
fastapi
pydantic
starlette
uvicorn[standard]
numpy
```

```
FROM python:alpine

WORKDIR /app
COPY requirements/backend.in .
RUN apk add g++
RUN pip install -r backend.in

COPY . .
CMD ["uvicorn", "spaceship.main:app", "--host=0.0.0.0", "--port=8080"]
```

```
@app.get('/matrix')
def matrix() -> dict:
    matrix_a = np.random.randint(1,9,(10,10))
    matrix_b = np.random.randint(1,9,(10,10))
    product = np.dot(matrix_a,matrix_b)
    res = {
        "matrix_a":matrix_a.tolist(),
        "matrix_b":matrix_b.tolist(),
        "product":product.tolist()
    }
    return res
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker build -t python-image .
[+] Building 403.1s (11/11) FINISHED
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker images
REPOSITORY     TAG       IMAGE ID       CREATED          SIZE
python-image   latest    09ddb7f891d9   2 minutes ago    488MB
```



Build quality: optimized
Basic image: lightweight
Build time: 403.1s
The size: 488 MB
Build: docker build -t python-image .
Run: docker run -dp 8080:8080 python-image

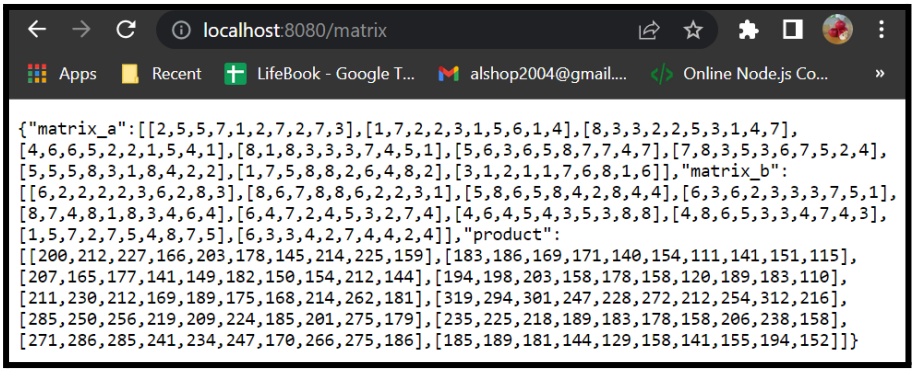
Notes: alpine version is not compatible with python wheels, so g++ must be installed before installing numpy

```
fastapi
pydantic
starlette
uvicorn[standard]
numpy
```

```
FROM python:slim-bullseye

WORKDIR /app
COPY requirements/backend.in .
RUN pip install -r backend.in

COPY . .
CMD ["uvicorn", "spaceship.main:app", "--host=0.0.0.0", "--port=8080"]
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker build -t python-image .
[+] Building 17.1s (11/11) FINISHED
```

```
PS C:\Users\Os\Desktop\method-lab3\python> docker images
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
python-image    latest    e05d03140401   4 minutes ago    285MB
```



Build quality: optimized
Basic image: medium size
Assembly time: 17.1 s
The size: 285 MB
Build: docker build -t python-image .
Run: docker run -dp 8080:8080 python-image
Notes: the slim-bullseye version can work with python wheels, so we install numpy right away
Link: LINK

Python. Conclusion

- Docker keeps the base image in the cache, so the 2nd, 3rd, 4th... build of the image is faster than the 1st: 45.3s -> 14.6s
  By optimizing the Dockerfile I was able to reduce the build time from 14.6s -> 10.1s with the base image in cache. Makes sense.
- It is not always beneficial to use a lightweight base image, because it will not be able to work with all dependencies. For example, when adding new dependencies, the following results are obtained:
  alpine - 403.1s 488MB

slim-bullseye - 17.1s 285MB

Although the base images were already in the cache. That is, we spent so many resources just to load the g++ compiler. Inefficient.

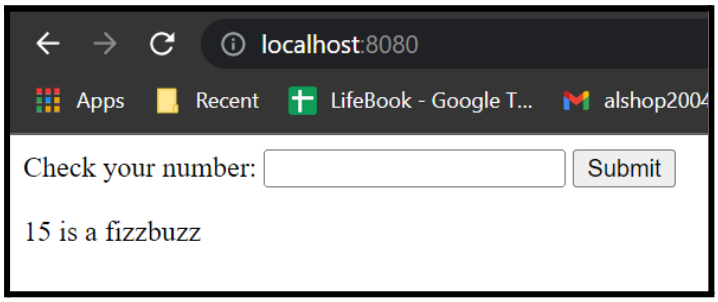
GOLANG

Step 1(Create the Dokerfile)

```
FROM golang:1.19
WORKDIR /app

COPY go.mod go.sum ./
RUN go mod download

COPY . .
RUN go build -o build/fizzbuzz

EXPOSE 8080
CMD ["./build/fizzbuzz","serve"]
```

```
[+] Building 174.3s (12/12) FINISHED
```

```
PS C:\Users\Os\Desktop\method-lab3\golang> docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
go-image     latest    8c2cbd6be77e   17 minutes ago   1.01GB
```



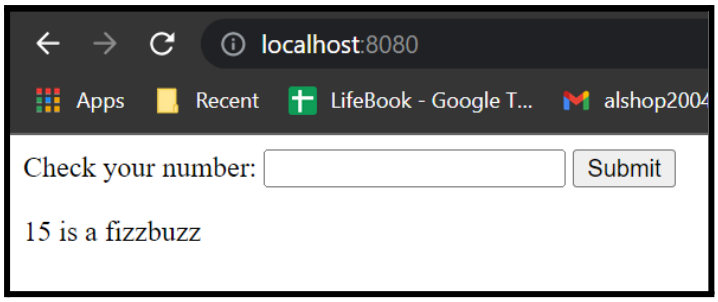
Build time: 174.3c
The size: 1.01GB
Build: docker build -t go-image .
Run: docker run -dp 8080:8080 go-image
Notes: in fact, we only need the fizzbuzz.exe file to run, because that's what we built it for + the html page
Link: LINK

Step 2(Make a multi-stage building)

```
FROM golang:1.19 AS builder
WORKDIR /app

COPY go.mod go.sum ./
RUN go mod download

COPY . .
RUN CGO_ENABLED=0 go build -o build/fizzbuzz

FROM scratch
COPY --from=builder /app/build/fizzbuzz ./
COPY --from=builder /app/templates/index.html ./templates/

EXPOSE 8080
CMD ["./fizzbuzz","serve"]
```

```
PS C:\Users\Os\Desktop\method-lab3\golang> docker build -t go-image .
[+] Building 6.0s (14/14) FINISHED
```

```
PS C:\Users\Os\Desktop\method-lab3\golang> docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
go-image     latest    6706de8eaeac   4 minutes ago    9.58MB
```



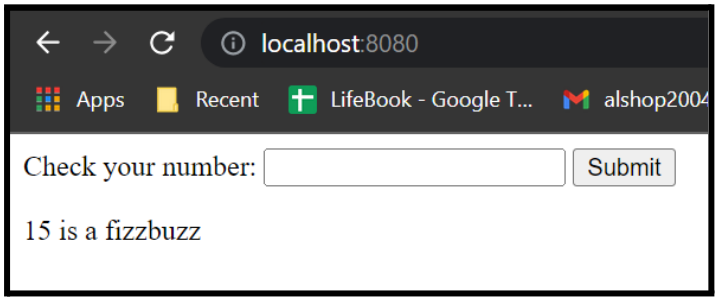
Build time: 6s
The size: 9.58MB
Build: docker build -t go-image .
Run: docker run -dp 8080:8080 go-image
Notes: Our new image consists of an executable and an html page. Yes files are enough since we have a static binary. Yes, this image is 100 times lighter than the previous one, it is quite an interesting idea how to get rid of "unprofitable" parts of the project, namely, not to transfer to the final image those files that will not participate in the program's work in the future.
Link: LINK

Step 3(Use a different image)

```
FROM golang:1.19 AS builder
WORKDIR /app

COPY go.mod go.sum ./
RUN go mod download

COPY . .
RUN CGO_ENABLED=0 go build -o build/fizzbuzz

FROM gcr.io/distroless/static-debian11
COPY --from=builder /app/build/fizzbuzz ./
COPY --from=builder /app/templates/index.html ./templates/

EXPOSE 8080
CMD ["./fizzbuzz","serve"]
```

```
PS C:\Users\Os\Desktop\method-lab3\golang> docker build -t go-image .
[+] Building 11.4s (15/15) FINISHED
```

```
PS C:\Users\Os\Desktop\method-lab3\golang> docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
go-image     latest    e903c37bbac3   42 seconds ago       12MB
```



Build time: 11.4s
The size: 12MB
Build: docker build -t go-image .
Run: docker run -dp 8080:8080 go-image
Notes: The size of the image has increased slightly, since scratch is positioned as an "empty" image, and distroles as an image with the minimum necessary functionality.
Link: LINK

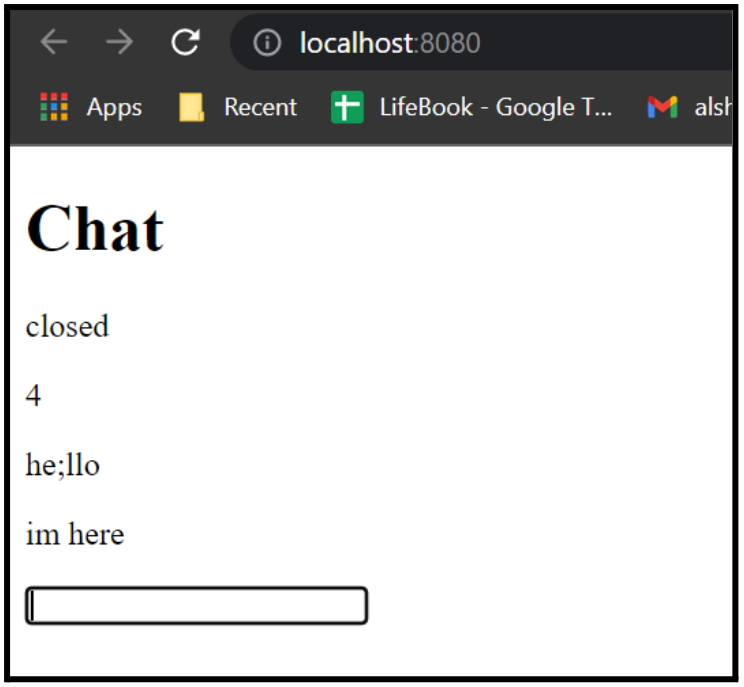
NODE.JS

Step 1(Create Dokerfile)

```
FROM node:19-alpine
WORKDIR /app

COPY package*.json ./
RUN npm i

COPY . .
EXPOSE 8080
CMD ["node", "server.js"]
```

```
PS C:\Users\Os\Desktop\method-lab3\nodejs> docker build -t node-image .
[+] Building 40.0s (11/11) FINISHED
```

```
PS C:\Users\Os\Desktop\method-lab3\nodejs> docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
node-image   latest    b0498c9046e0   52 seconds ago   179MB
```



Build time: 40.0s
The size: 179 MB
Build: docker build -t node-image .
Run: docker run -dp 8080:8080 node-image
Link: LINK

General report

- I learned how to use Docker, how to optimize the containerization process and the product (image) itself in various ways
- After conducting experiments, it turned out that docker keeps most of the data in the cache, which speeds up the rebuild of images (by 3 times in my case). First, you should add dependencies to the image, and then the main working code, this speeds up the process of building the image (by 1.4 times in my case). It is also possible to create a binary file in a harsh environment in an intermediate image, and transfer the binary itself and an "empty" scratch image to the final image to execute basic commands, this reduces the size of the image (by 100 times in my case). Sometimes it is not worth using lightweight basic images, so as not to load a bunch of additional applications for the correct operation of the project. When I used the standard base image instead of the lightweight one, the size of the final image was reduced by a factor of 1.7.
- While studying the theory, I encountered the problem of a large diversity of teams. In order to understand how containers work, image optimization and other functionality that is present in the official documentation, you need to make a little more effort than when learning git, for example.

- In general, this is an interesting idea. I liked that I could not clutter up my memory with all kinds of junk, but instead do it all "virtually" and keep my system clean.