Lecture 16-18: APIs & App Frontend

Advanced Practical Data Science, MLOps

AC215

Pavlos Protopapas
Institute for Applied Computational Science, Harvard

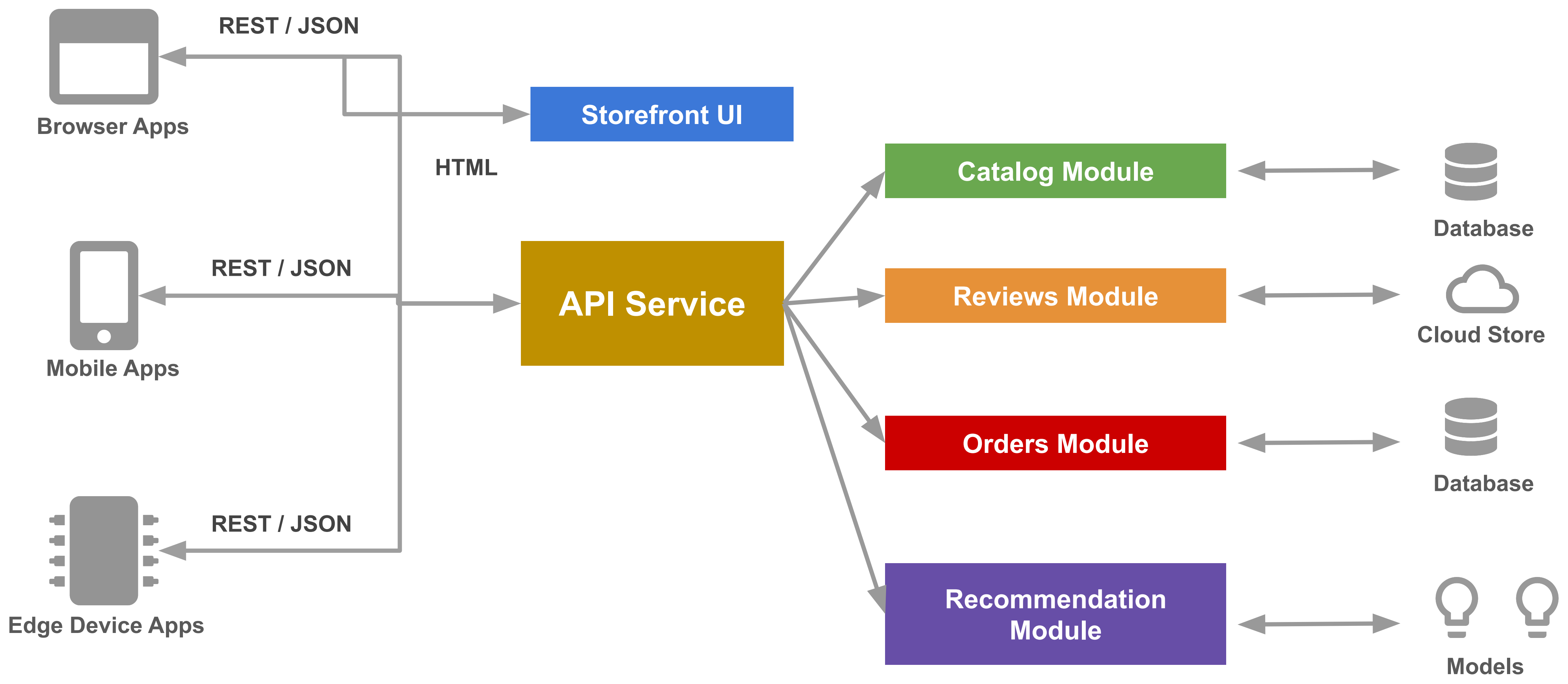
Outline

1. Recap
2. APIs
3. App Frontend (Simple)
4. Model Serving
5. Frontend Frameworks

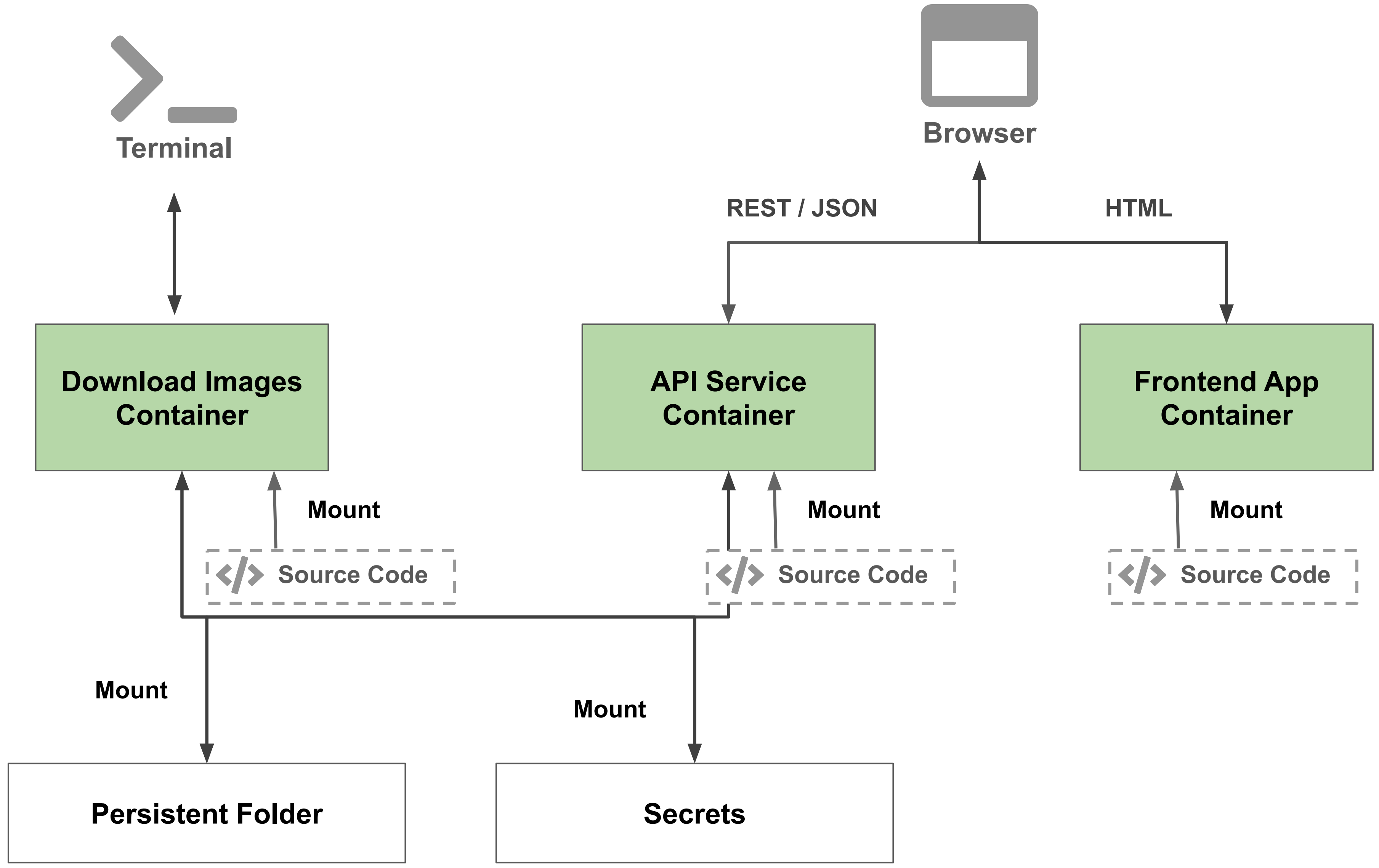
Outline

1. **Recap**
2. APIs
3. App Frontend (Simple)
4. Model Serving
5. Frontend Frameworks

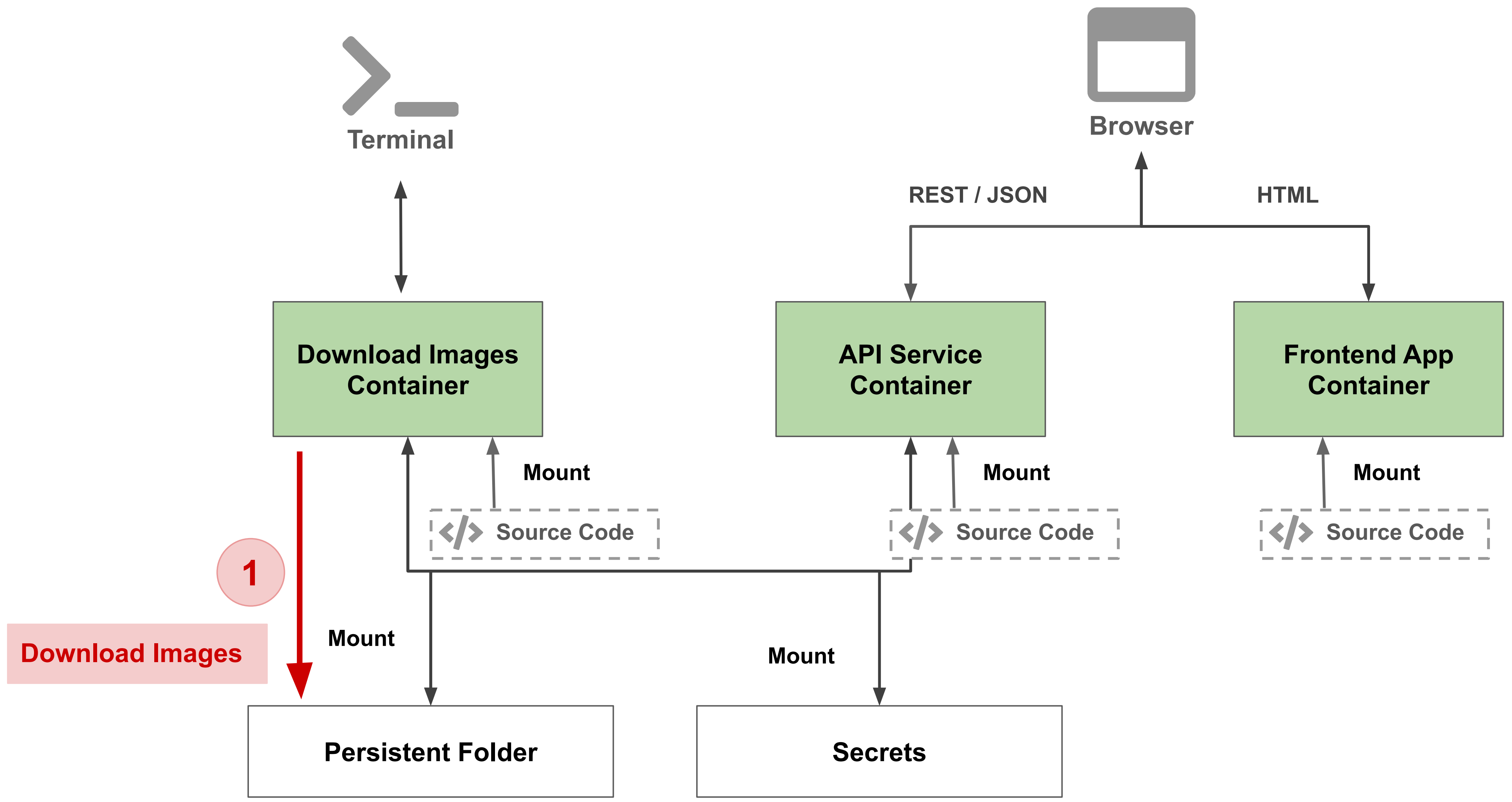
Microservice Architecture



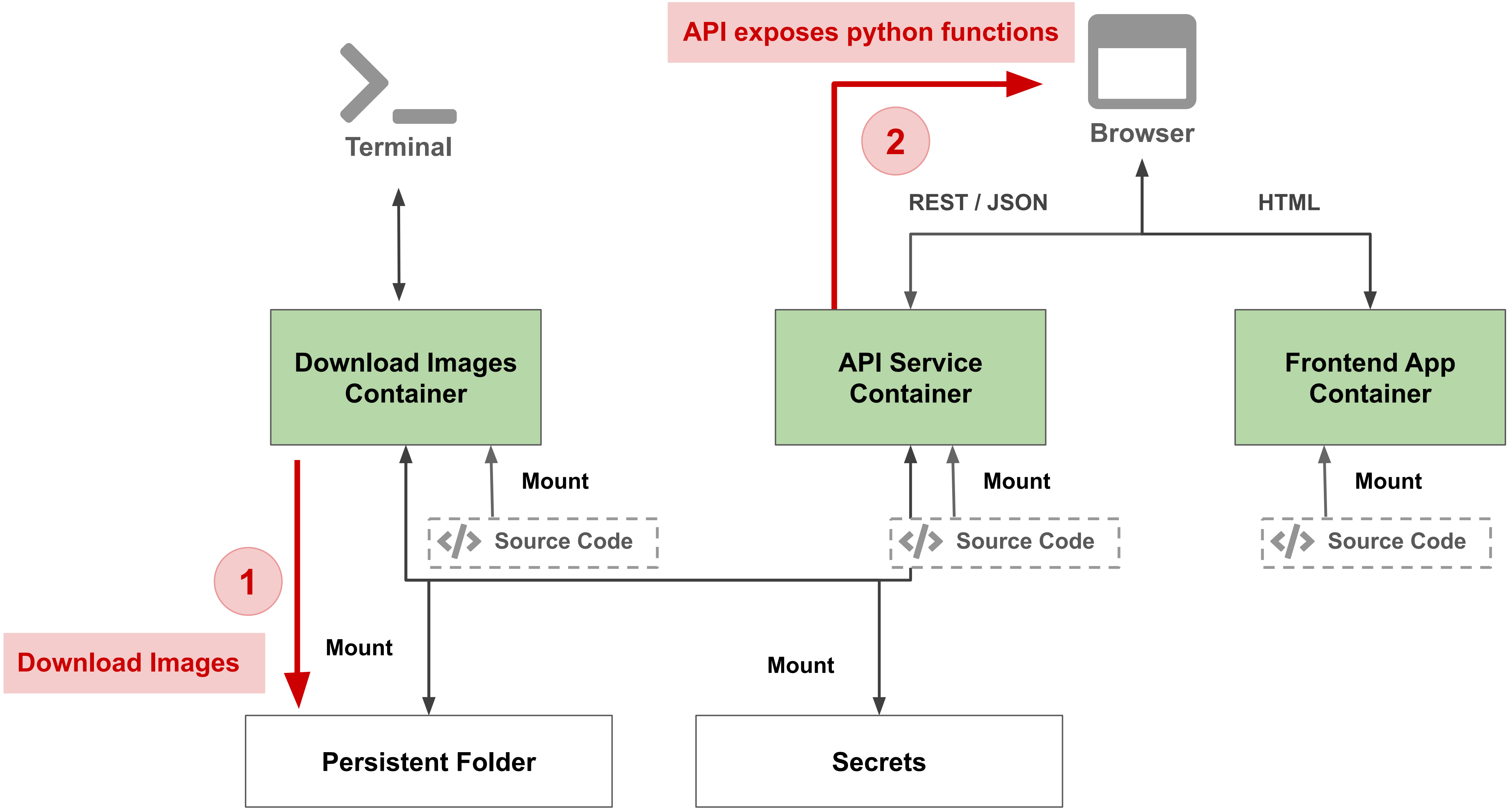
What we built so far



What we built so far



What we built so far



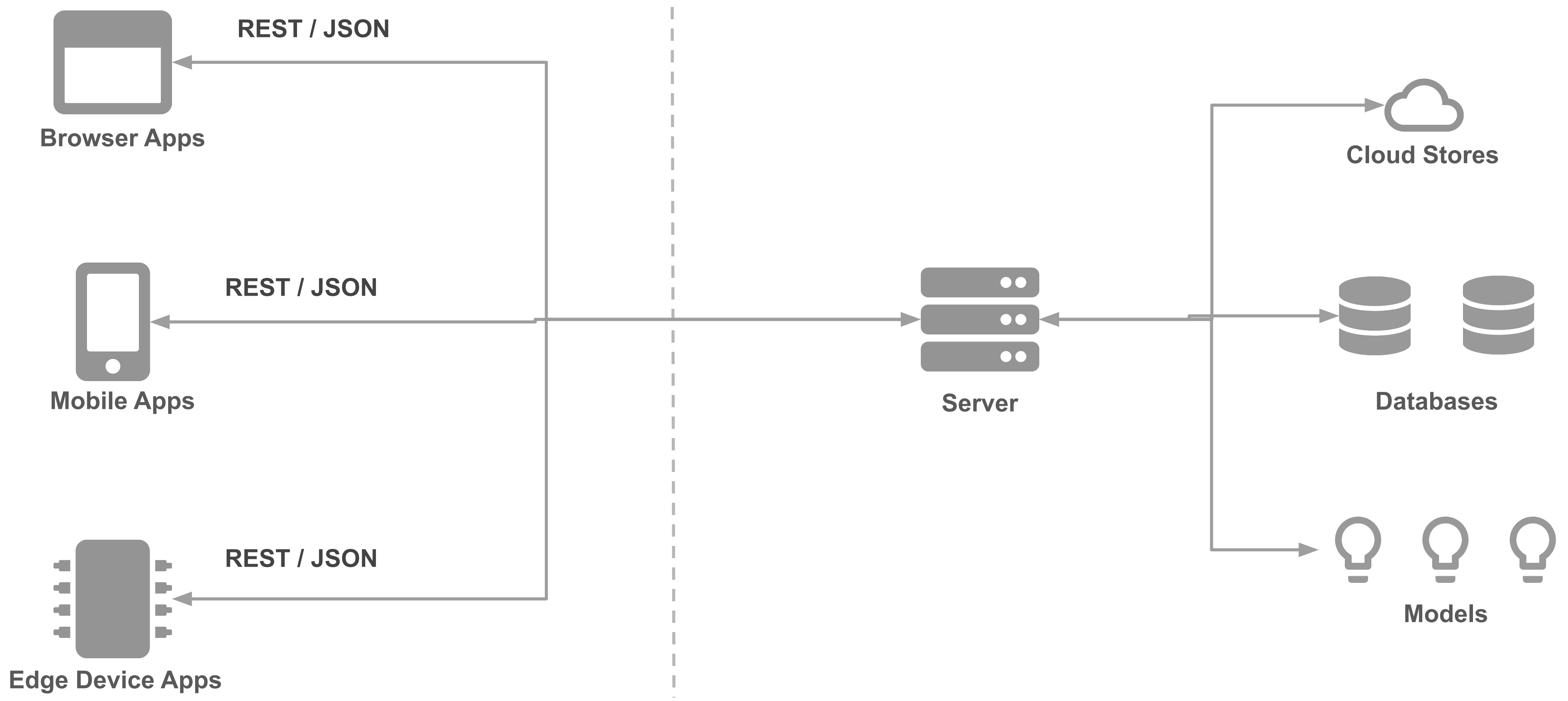
Outline

What is an API

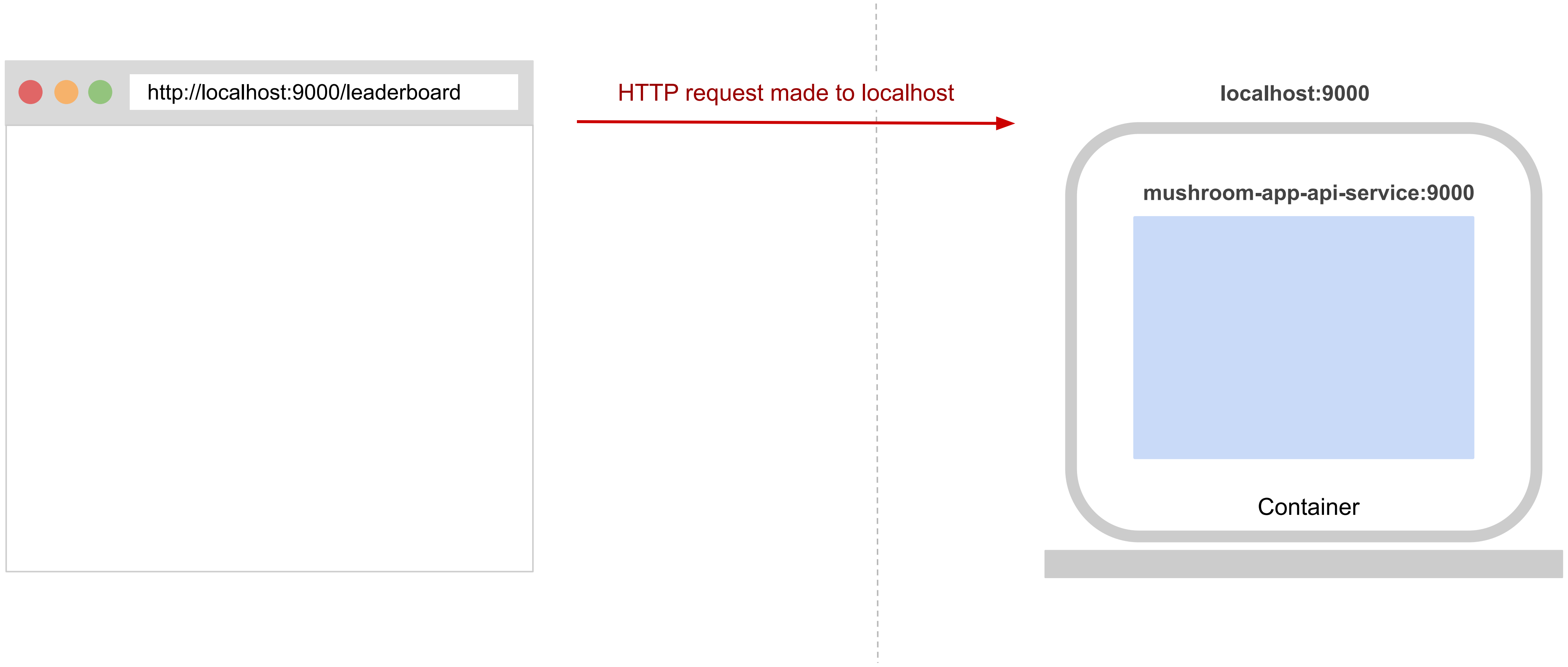
- API is **Application Programming Interface**
- **Web API** is one that can be access using HTTP/S
- A **REST API** is a Web API that follows the HTTP method constraints - get, post, put, delete
- We will use **FastAPI** a Python framework to build REST APIs

APIs

We will be using the term **API** to refer to REST API, which will be used to connect to various components



How does an API work

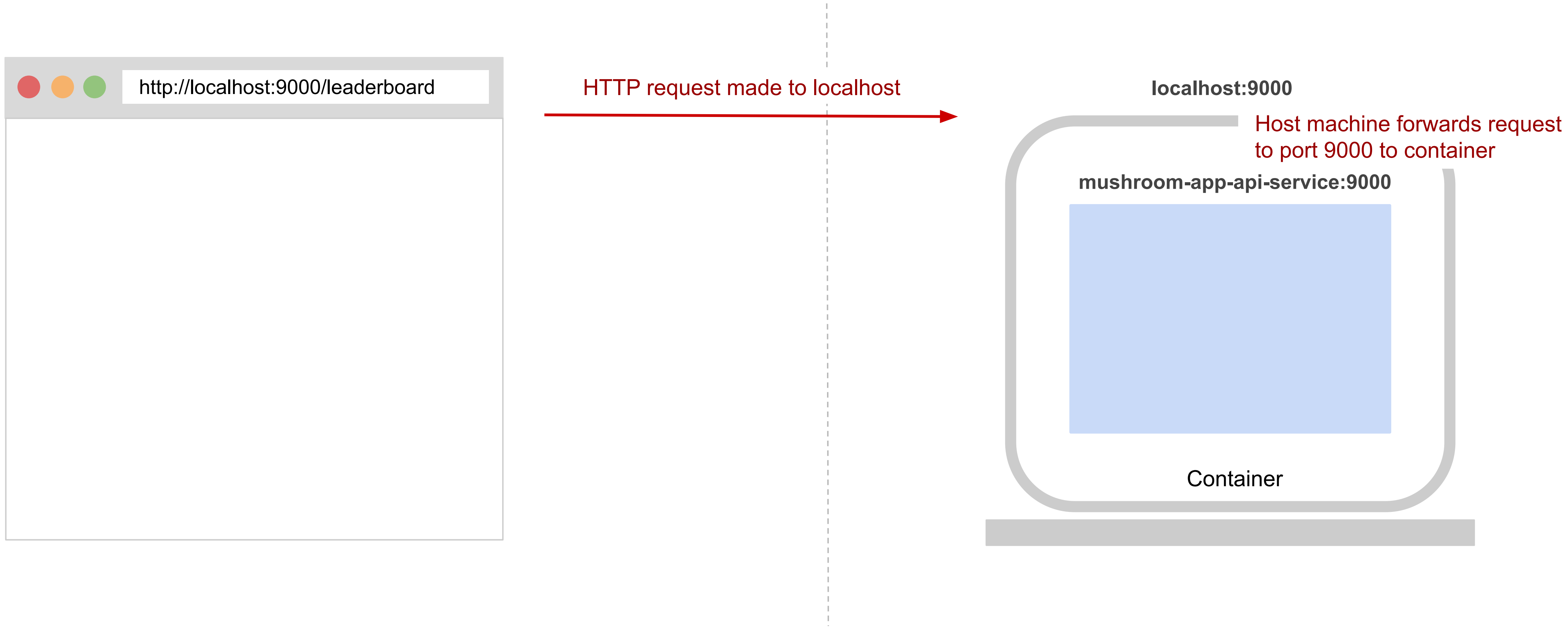


http://localhost:9000/leaderboard
HTTP request made to localhost
localhost:9000
mushroom-app-api-service:9000
Container

Browser

Local computer / Server

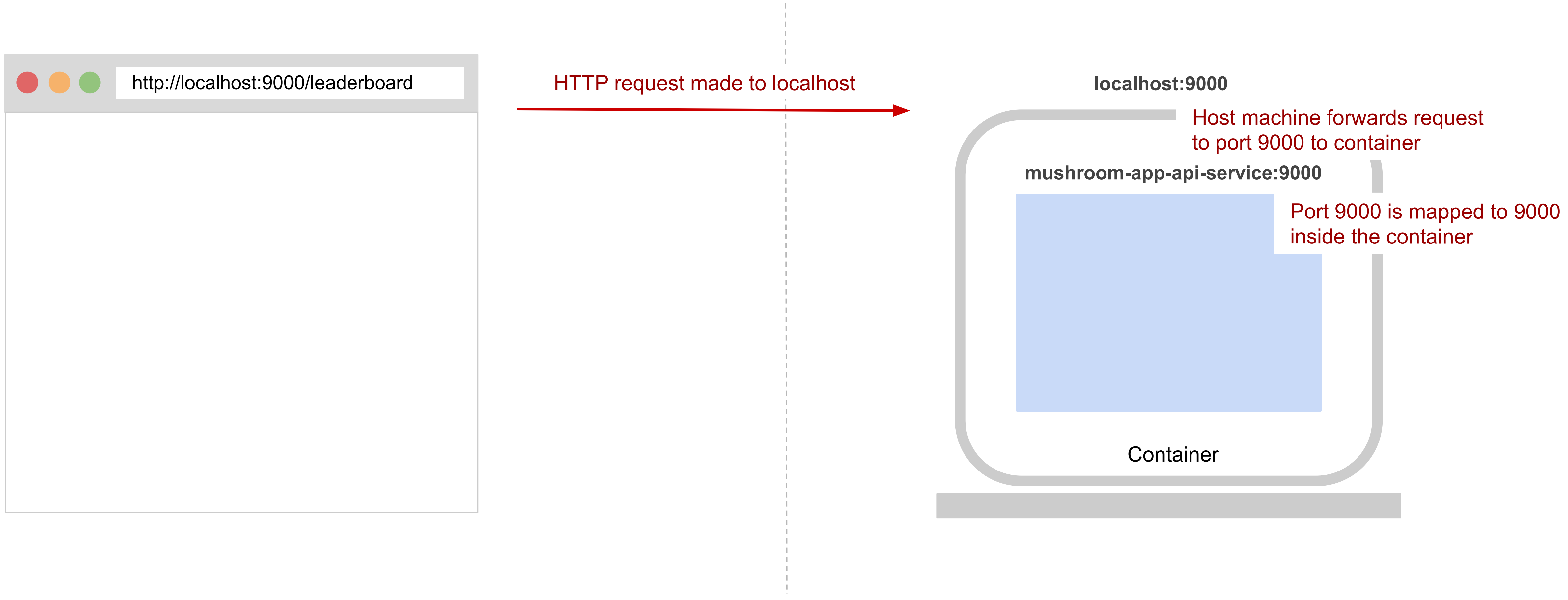
How does an API work


http://localhost:9000/leaderboard
HTTP request made to localhost
localhost:9000
Host machine forwards request to port 9000 to container
mushroom-app-api-service:9000
Container

Browser

Local computer / Server

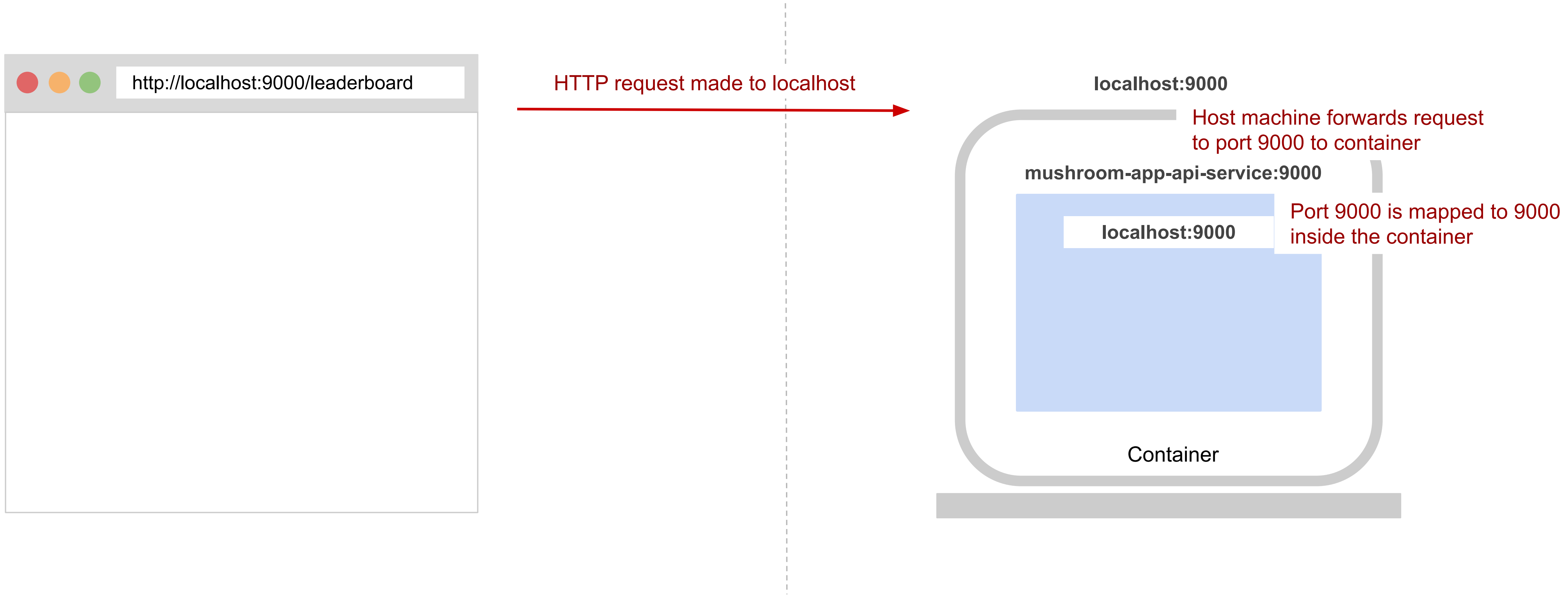
How does an API work


http://localhost:9000/leaderboard
HTTP request made to localhost
localhost:9000
Host machine forwards request to port 9000 to container
mushroom-app-api-service:9000
Port 9000 is mapped to 9000 inside the container
Container

Browser

Local computer / Server

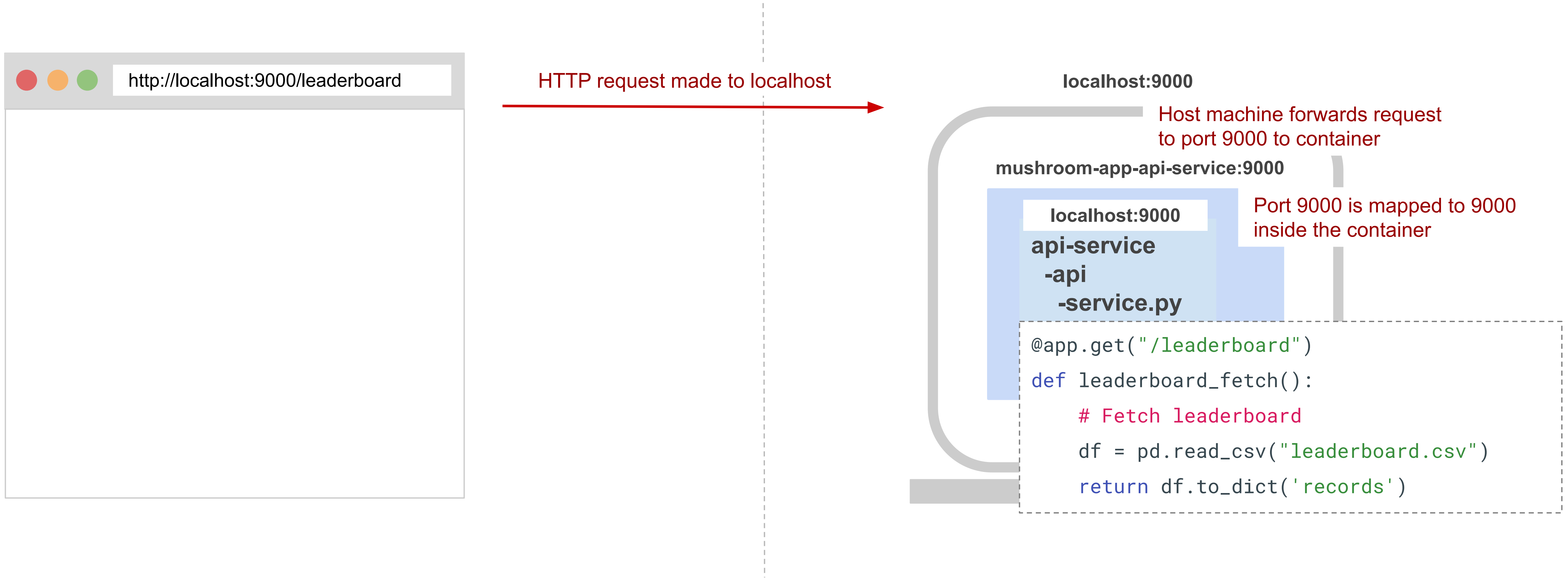
How does an API work


http://localhost:9000/leaderboard
HTTP request made to localhost
localhost:9000
Host machine forwards request to port 9000 to container
mushroom-app-api-service:9000
localhost:9000
Port 9000 is mapped to 9000 inside the container
Container

Browser

Local computer / Server

How does an API work

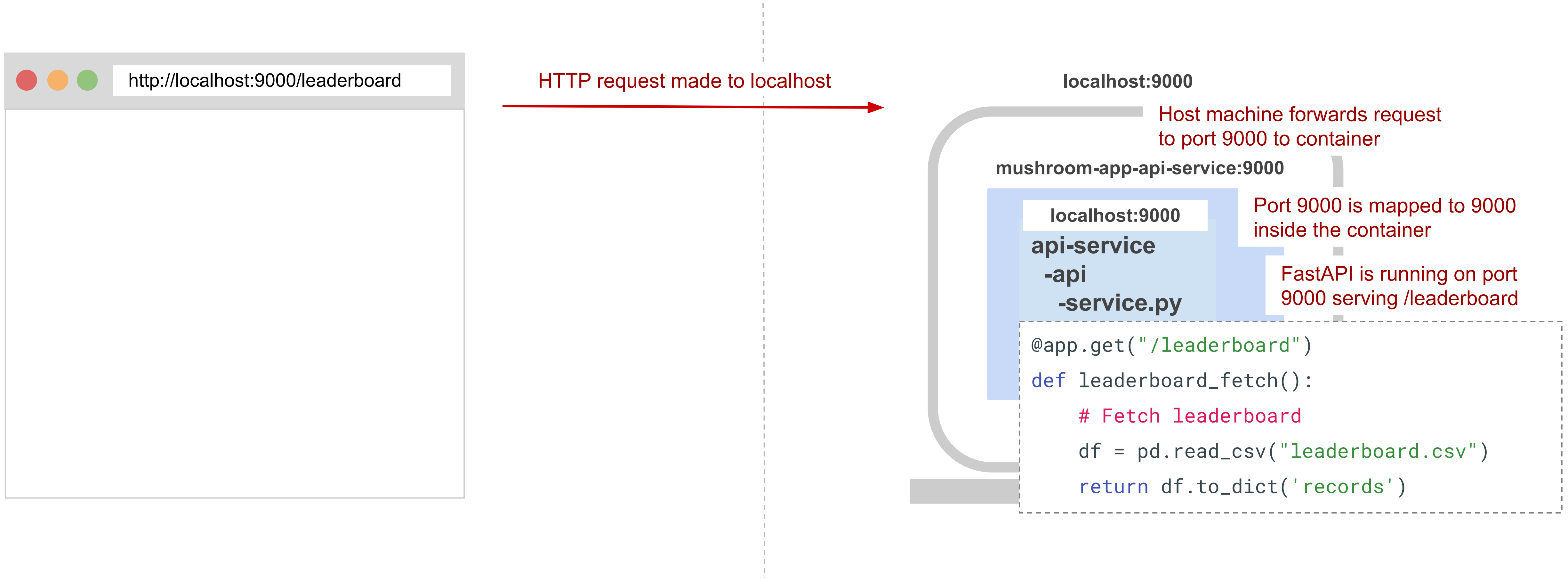

http://localhost:9000/leaderboard
HTTP request made to localhost
localhost:9000
Host machine forwards request to port 9000 to container
mushroom-app-api-service:9000
localhost:9000
Port 9000 is mapped to 9000 inside the container
api-service
-api
-service.py

```
@app.get("/leaderboard")
def leaderboard_fetch():
    # Fetch leaderboard
    df = pd.read_csv("leaderboard.csv")
    return df.to_dict('records')
```

Browser

Local computer / Server

How does an API work

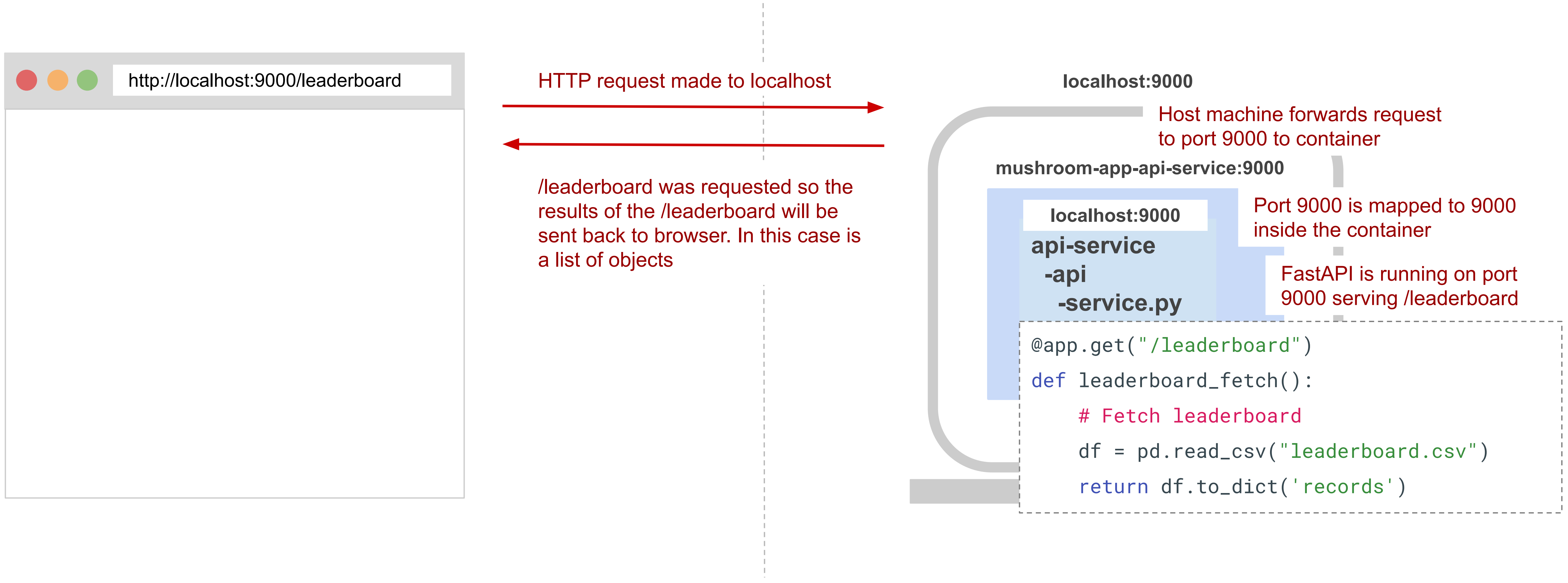

http://localhost:9000/leaderboard
HTTP request made to localhost
localhost:9000
Host machine forwards request to port 9000 to container
mushroom-app-api-service:9000
localhost:9000
Port 9000 is mapped to 9000 inside the container
api-service
-api
-service.py
FastAPI is running on port 9000 serving /leaderboard

```
@app.get("/leaderboard")
def leaderboard_fetch():
    # Fetch leaderboard
    df = pd.read_csv("leaderboard.csv")
    return df.to_dict('records')
```

Browser

Local computer / Server

How does an API work

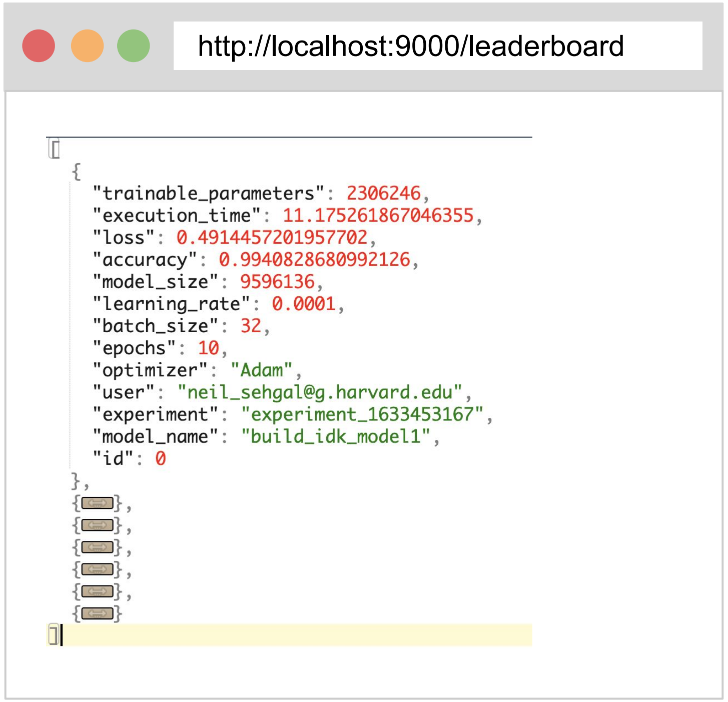

http://localhost:9000/leaderboard
HTTP request made to localhost
/leaderboard was requested so the results of the /leaderboard will be sent back to browser. In this case is a list of objects
localhost:9000
Host machine forwards request to port 9000 to container
mushroom-app-api-service:9000
localhost:9000
Port 9000 is mapped to 9000 inside the container
api-service
-api
-service.py
FastAPI is running on port 9000 serving /leaderboard

```python
@app.get("/leaderboard")
def leaderboard_fetch():
    # Fetch leaderboard
    df = pd.read_csv("leaderboard.csv")
    return df.to_dict('records')
```

Browser

Local computer / Server

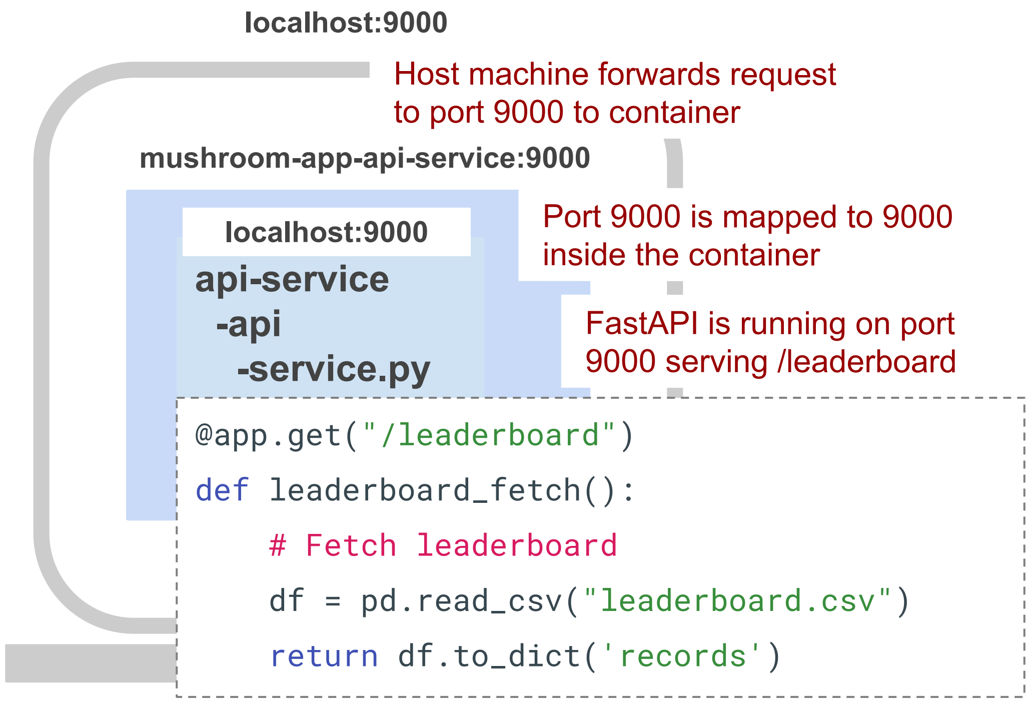
How does an API work



```
[
  {
    "trainable_parameters": 2306246,
    "execution_time": 11.175261867046355,
    "loss": 0.4914457201957702,
    "accuracy": 0.9940828680992126,
    "model_size": 9596136,
    "learning_rate": 0.0001,
    "batch_size": 32,
    "epochs": 10,
    "optimizer": "Adam",
    "user": "neil_sehgal@g.harvard.edu",
    "experiment": "experiment_1633453167",
    "model_name": "build_idk_model1",
    "id": 0
  },
  {↔},
  {↔},
  {↔},
  {↔},
  {↔},
  {↔}
]
```

HTTP request made to localhost

/leaderboard was requested so the results of the /leaderboard will be sent back to browser. In this case is a list of objects

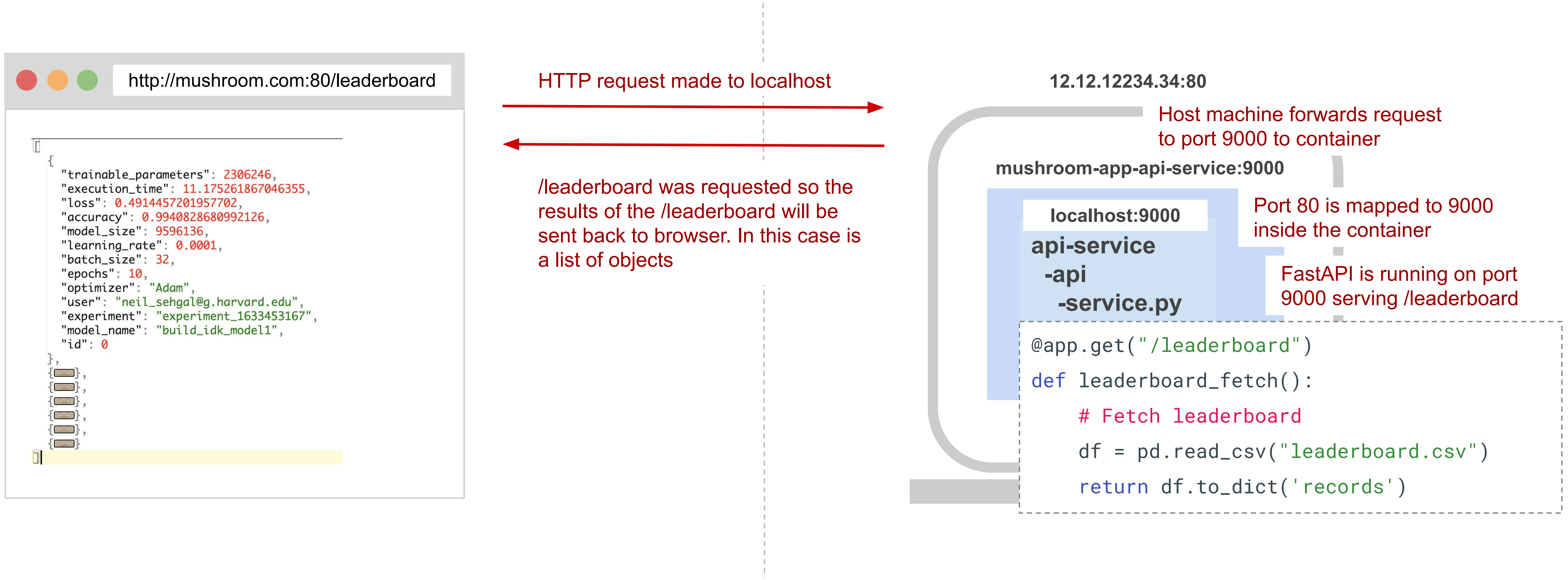

localhost:9000
Host machine forwards request to port 9000 to container
mushroom-app-api-service:9000
localhost:9000
Port 9000 is mapped to 9000 inside the container
api-service
-api
-service.py
FastAPI is running on port 9000 serving /leaderboard

```python
@app.get("/leaderboard")
def leaderboard_fetch():
    # Fetch leaderboard
    df = pd.read_csv("leaderboard.csv")
    return df.to_dict('records')
```

Browser

Local computer / Server

How does an API work (In Production)



http://mushroom.com:80/leaderboard

```
[
  {
    "trainable_parameters": 2306246,
    "execution_time": 11.175261867046355,
    "loss": 0.4914457201957702,
    "accuracy": 0.9940828680992126,
    "model_size": 9596136,
    "learning_rate": 0.0001,
    "batch_size": 32,
    "epochs": 10,
    "optimizer": "Adam",
    "user": "neil_sehgal@g.harvard.edu",
    "experiment": "experiment_1633453167",
    "model_name": "build_idk_model1",
    "id": 0
  },
  {…},
  {…},
  {…},
  {…},
  {…},
  {…}
]
```

HTTP request made to localhost

/leaderboard was requested so the results of the /leaderboard will be sent back to browser. In this case is a list of objects

12.12.12234.34:80

Host machine forwards request to port 9000 to container

mushroom-app-api-service:9000

localhost:9000

api-service
-api
-service.py

Port 80 is mapped to 9000 inside the container

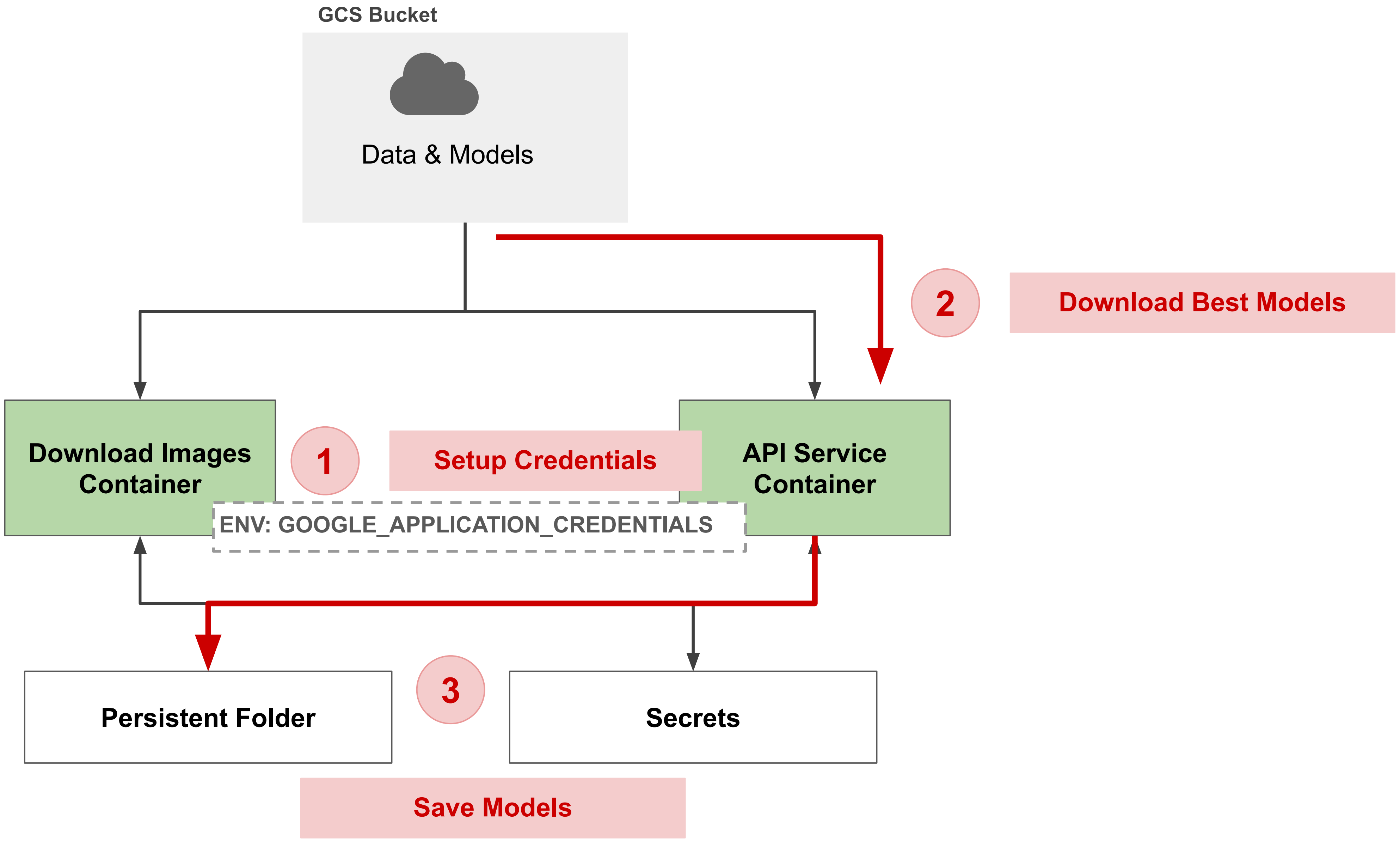
FastAPI is running on port 9000 serving /leaderboard

```python
@app.get("/leaderboard")
def leaderboard_fetch():
    # Fetch leaderboard
    df = pd.read_csv("leaderboard.csv")
    return df.to_dict('records')
```

Browser

GCP Server

Tutorial: Setup GCP Credentials/ Download Best Models

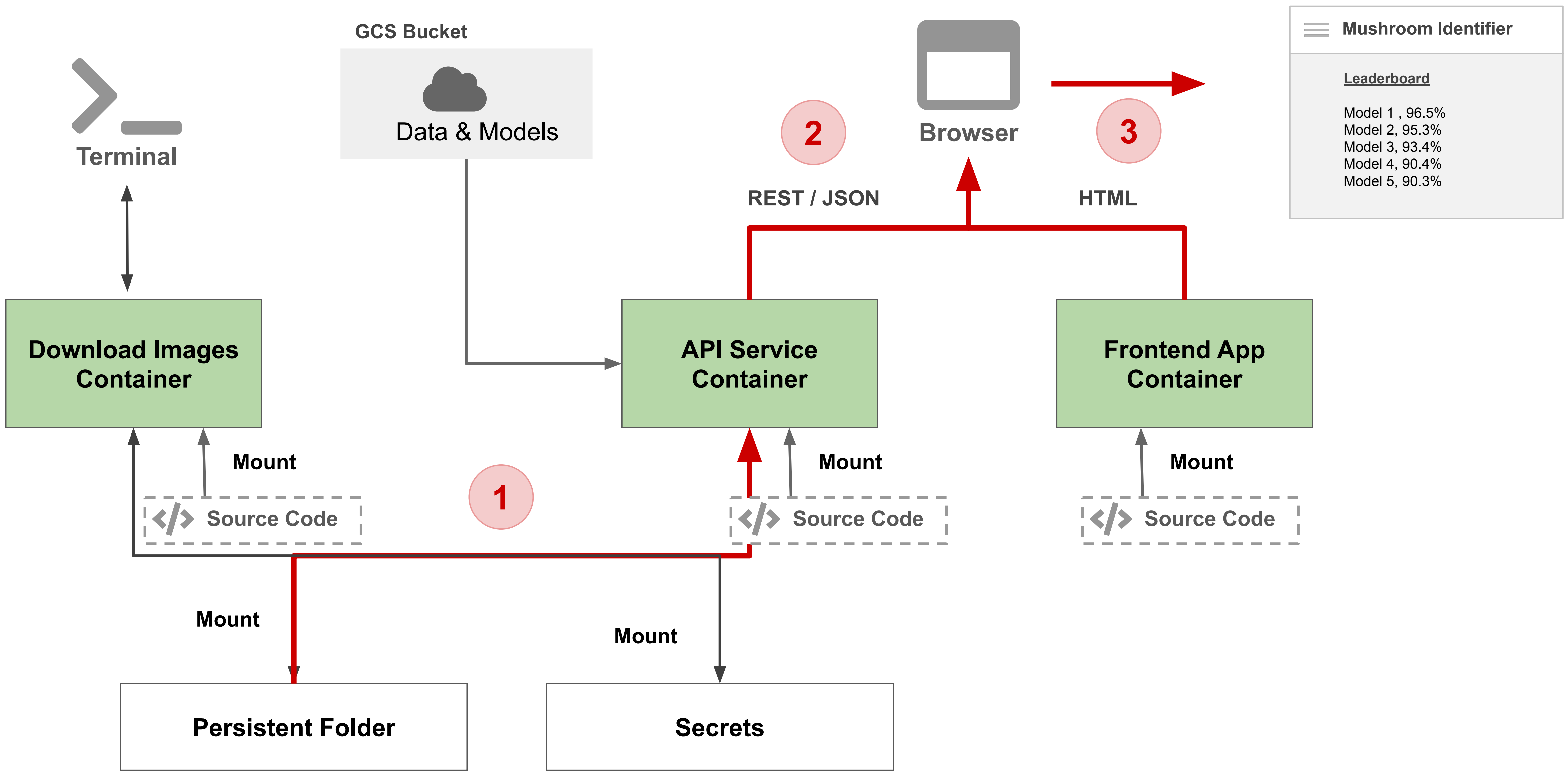


Tutorial: Setup GCP Credentials/ Download Best Models

Mushroom App - Setup GCP Credentials

Mushroom App - Download Best Models

Tutorial: APIs & Frontend App



Tutorial: APIs & Frontend App

Mushroom App - APIs & Frontend App

Outline

App Frontend

HTML

- Is Hyper Text Markup Language (Remember Markdowns)
- Browsers use HTML to display web pages

CSS

- Cascading style sheets
- Used to format & style web pages

Javascript

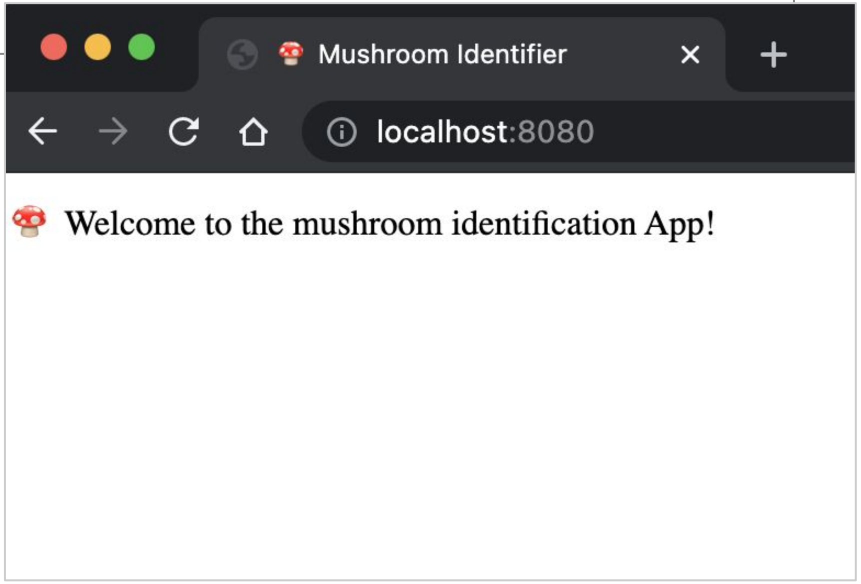
- Programming language understood by browser

App Frontend

```
<!DOCTYPE html>
<html>
<head>
    <title>🍄 Mushroom Identifier</title>
</head>
<body>
    🍄 Welcome to the mushroom identification App!
</body>
</html>
```

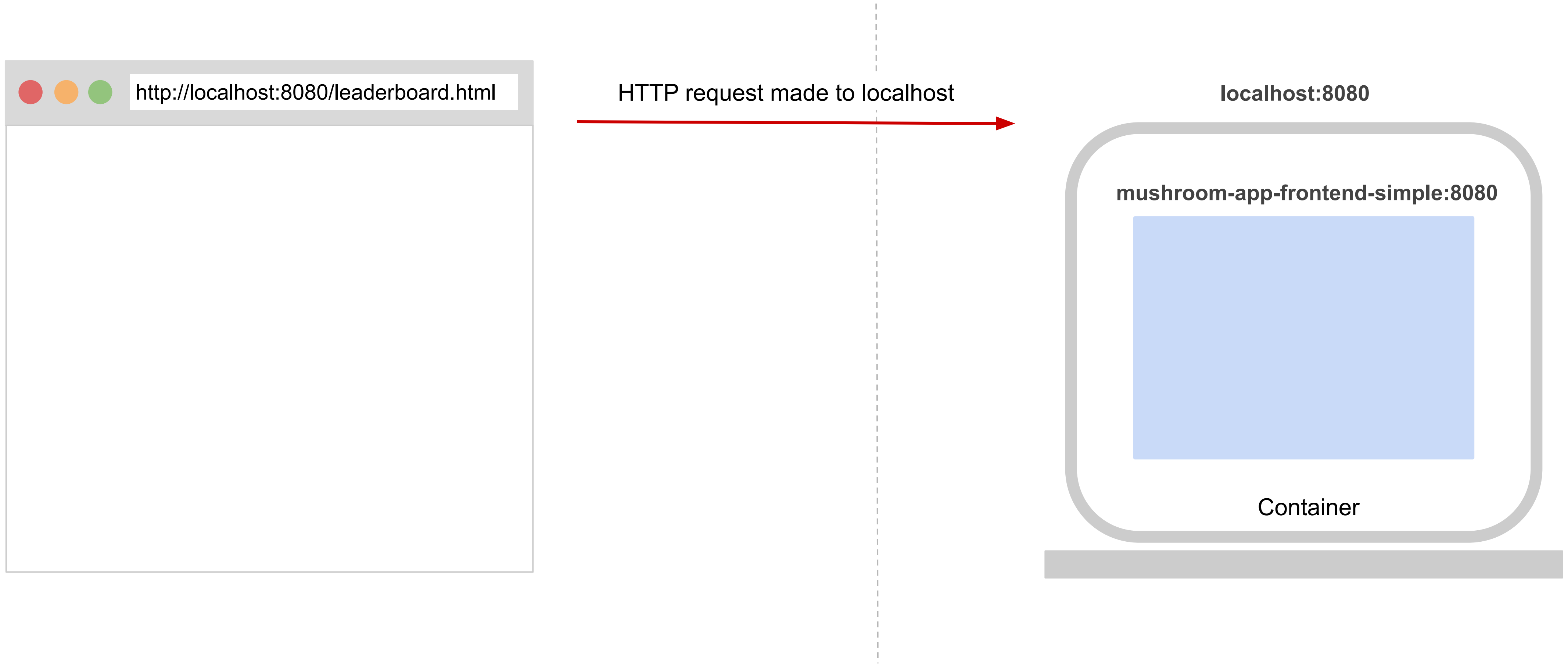
Browser Title

Web page details



🍄 Welcome to the mushroom identification App!

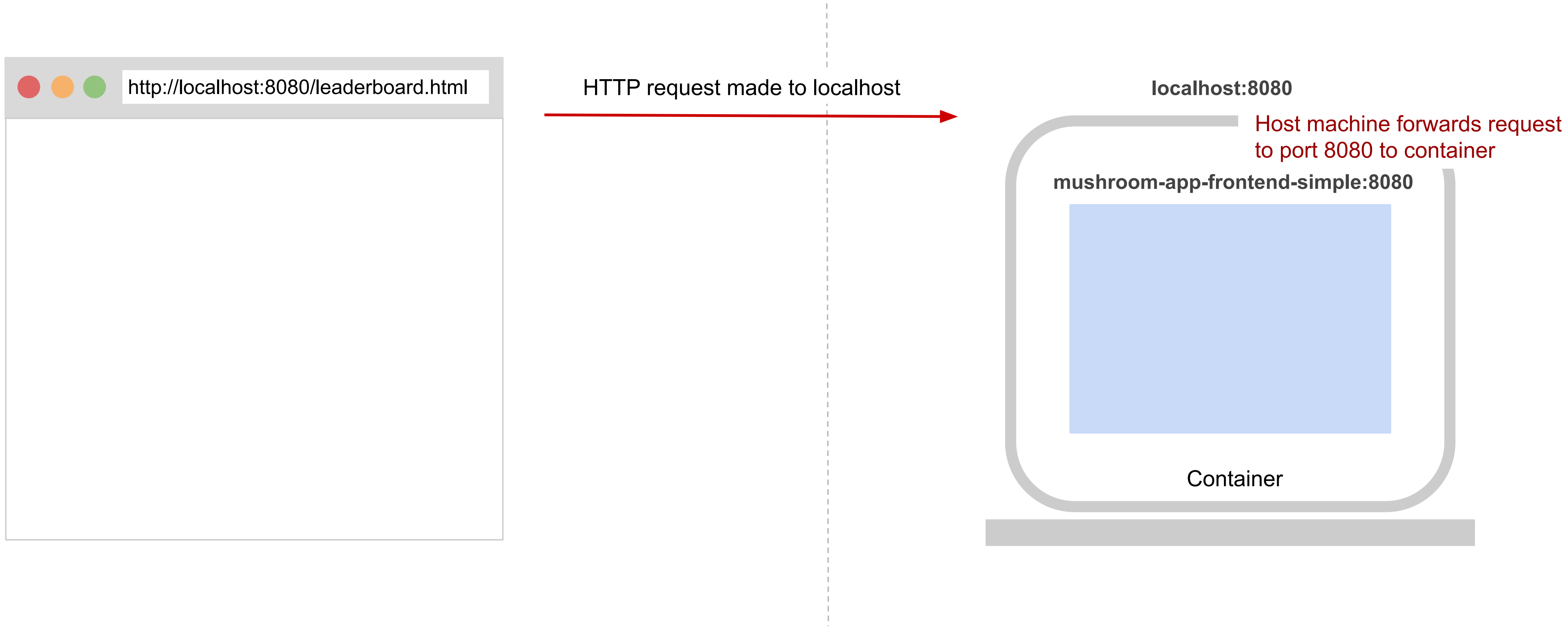
How does the App work


http://localhost:8080/leaderboard.html
HTTP request made to localhost
localhost:8080
mushroom-app-frontend-simple:8080
Container

Browser

Local computer / Server

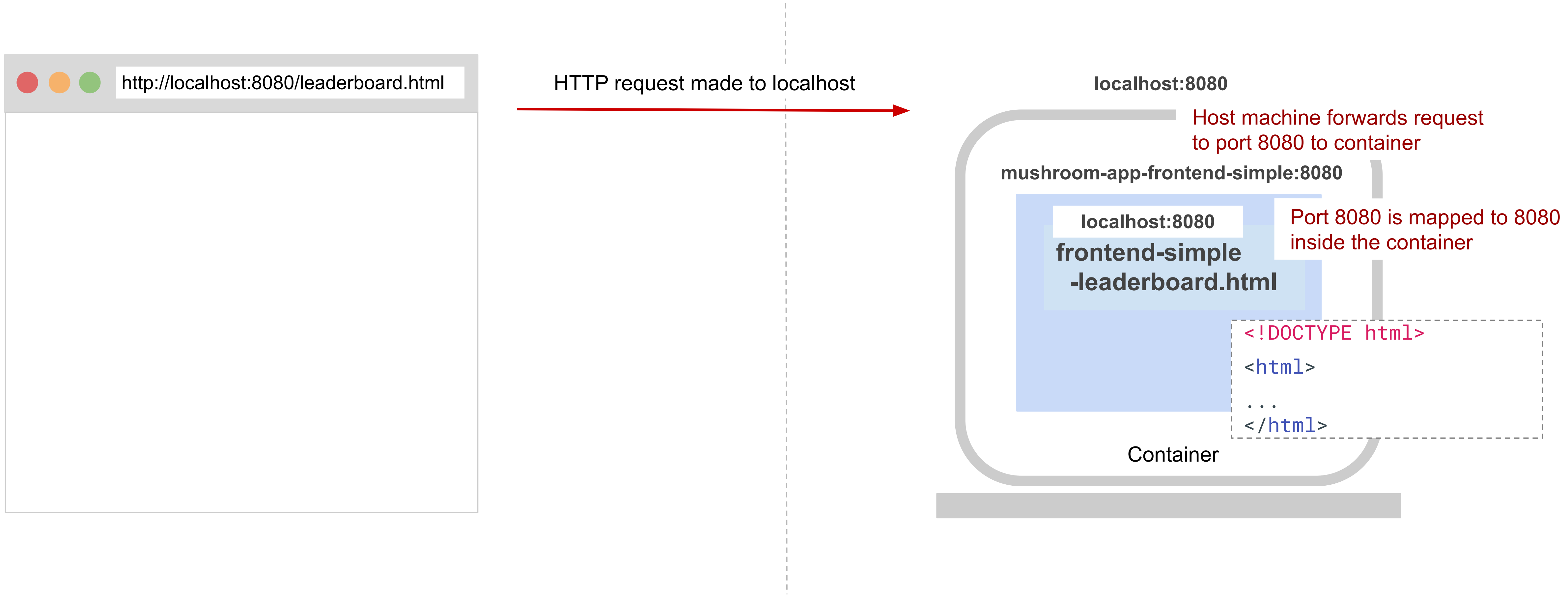
How does the App work


http://localhost:8080/leaderboard.html
HTTP request made to localhost
localhost:8080
Host machine forwards request to port 8080 to container
mushroom-app-frontend-simple:8080
Container

Browser

Local computer / Server

How does the App work

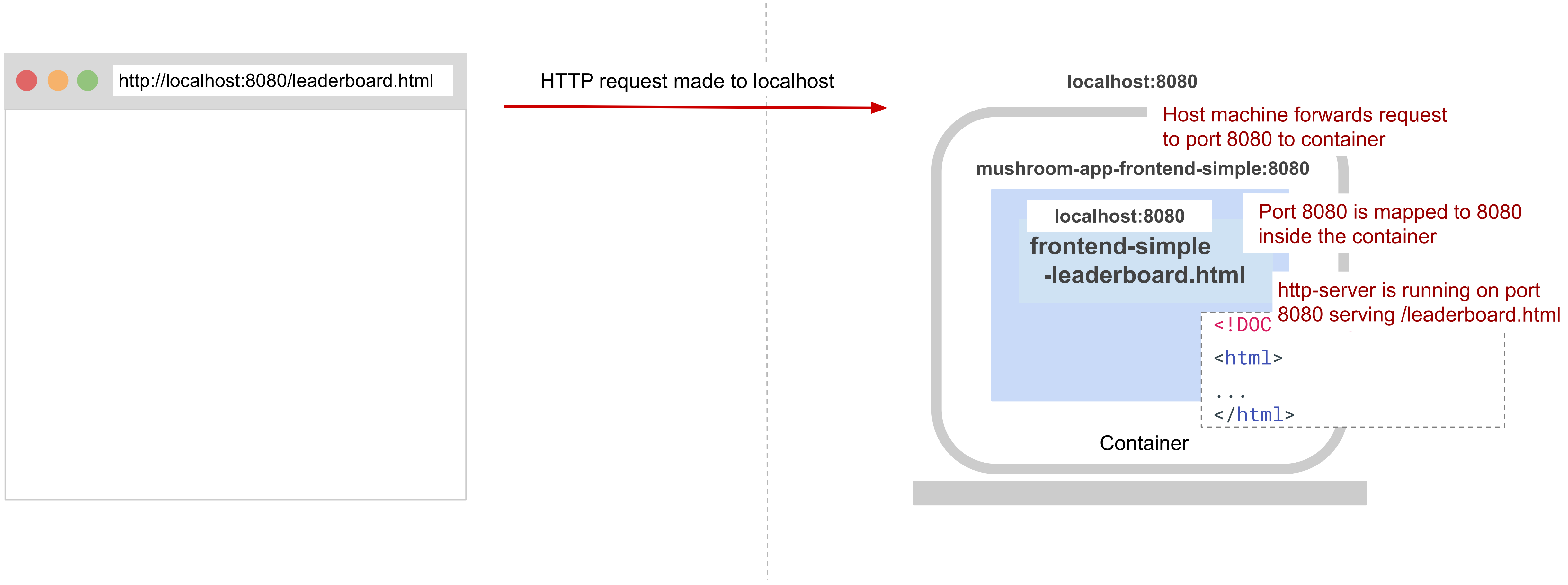

http://localhost:8080/leaderboard.html
HTTP request made to localhost
localhost:8080
Host machine forwards request to port 8080 to container
mushroom-app-frontend-simple:8080
localhost:8080
frontend-simple -leaderboard.html
Port 8080 is mapped to 8080 inside the container
Container

```
<!DOCTYPE html>
<html>
...
</html>
```

Browser

Local computer / Server

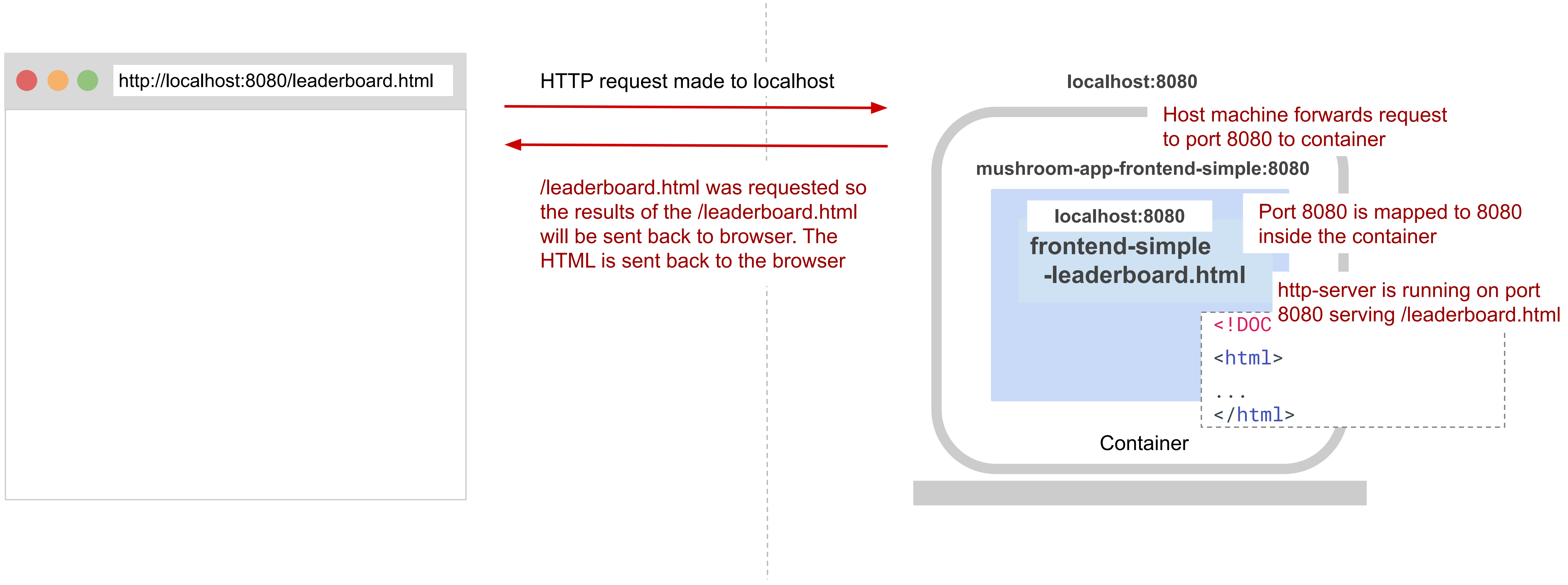
How does the App work


http://localhost:8080/leaderboard.html
HTTP request made to localhost
localhost:8080
Host machine forwards request to port 8080 to container
mushroom-app-frontend-simple:8080
localhost:8080
frontend-simple -leaderboard.html
Port 8080 is mapped to 8080 inside the container
http-server is running on port 8080 serving /leaderboard.html
<!DOC
<html>
...
</html>
Container

Browser

Local computer / Server

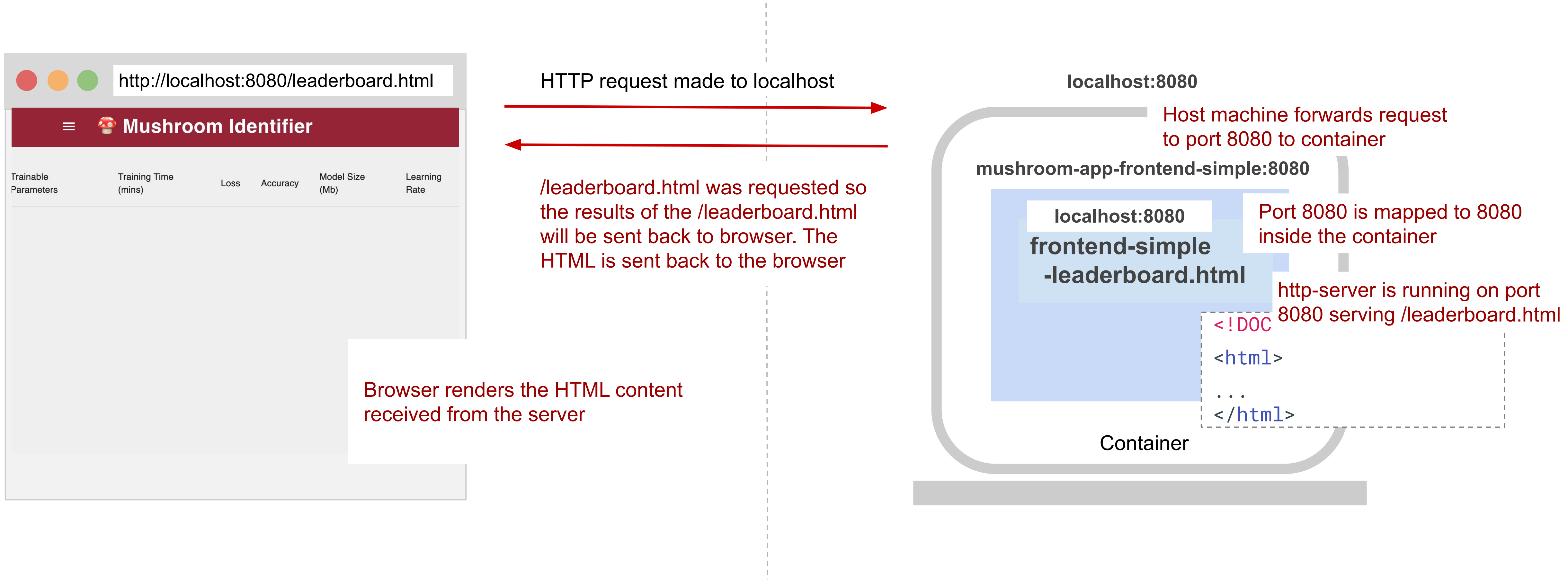
How does the App work


http://localhost:8080/leaderboard.html
HTTP request made to localhost
/leaderboard.html was requested so the results of the /leaderboard.html will be sent back to browser. The HTML is sent back to the browser
localhost:8080
Host machine forwards request to port 8080 to container
mushroom-app-frontend-simple:8080
localhost:8080
frontend-simple -leaderboard.html
Port 8080 is mapped to 8080 inside the container
http-server is running on port 8080 serving /leaderboard.html
<!DOC
<html>
...
</html>
Container

Browser

Local computer / Server

How does the App work

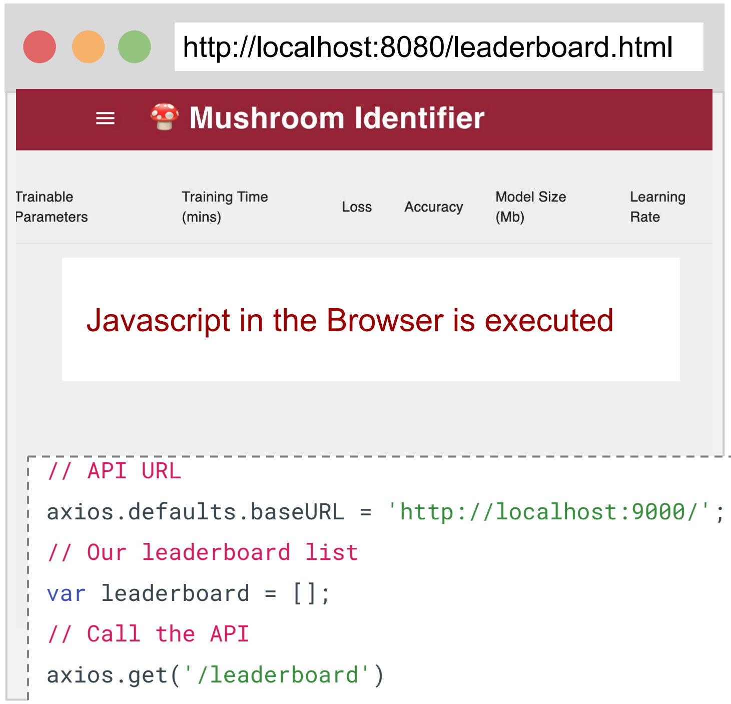


http://localhost:8080/leaderboard.html
Mushroom Identifier
Trainable Parameters
Training Time (mins)
Loss
Accuracy
Model Size (Mb)
Learning Rate
HTTP request made to localhost
/leaderboard.html was requested so the results of the /leaderboard.html will be sent back to browser. The HTML is sent back to the browser
Browser renders the HTML content received from the server
localhost:8080
Host machine forwards request to port 8080 to container
mushroom-app-frontend-simple:8080
localhost:8080
frontend-simple
-leaderboard.html
Port 8080 is mapped to 8080 inside the container
http-server is running on port 8080 serving /leaderboard.html
<!DOC
<html>
...
</html>
Container

Browser

Local computer / Server

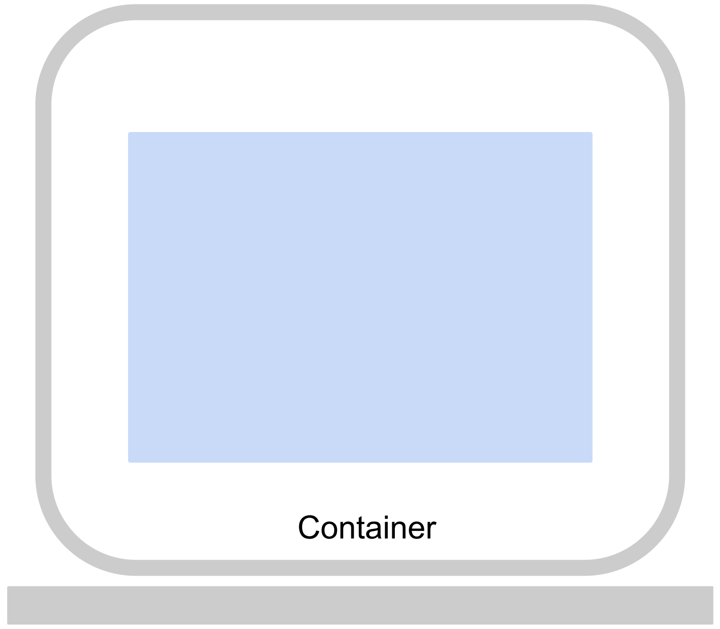
How does the App work



Javascript in the Browser is executed

```
// API URL
axios.defaults.baseURL = 'http://localhost:9000/';
// Our leaderboard list
var leaderboard = [];
// Call the API
axios.get('/leaderboard')
    .then((response) => {
        leaderboard = response.data;
        // Build the table
        buildLeaderboardTable(leaderboard);
    });
```

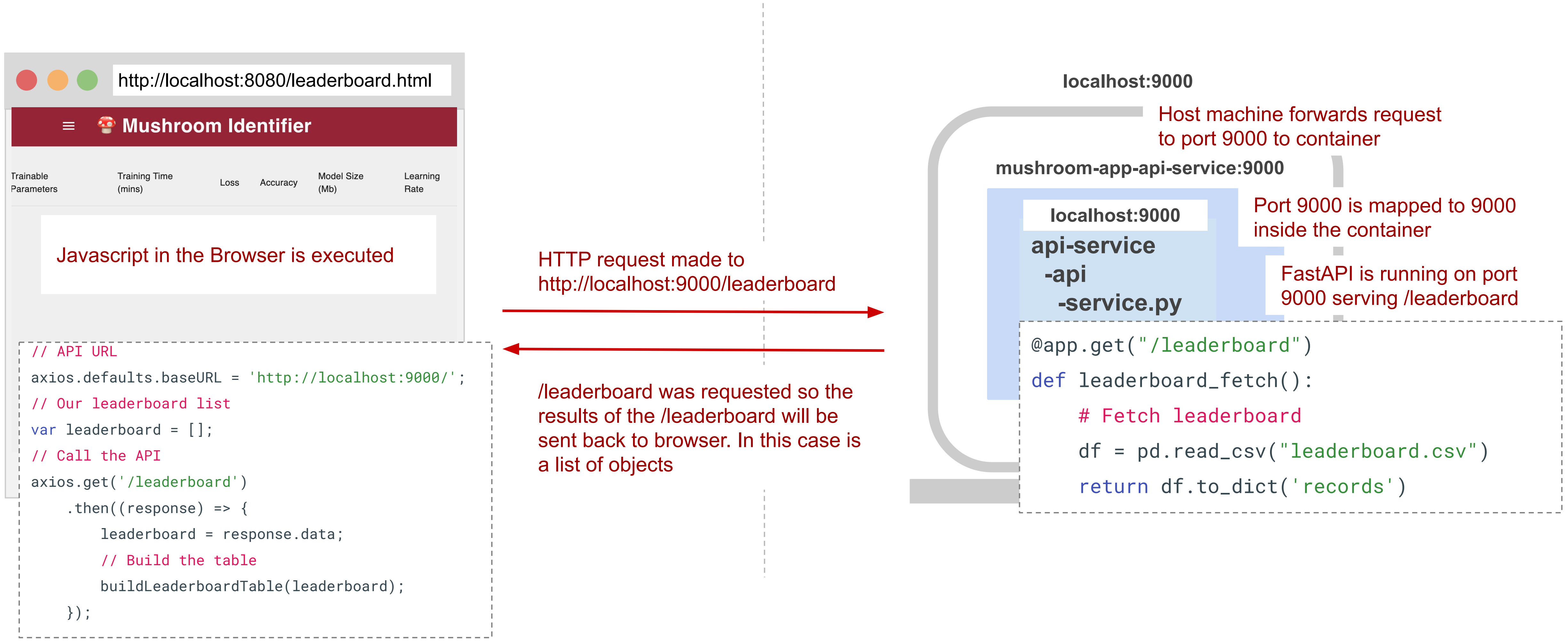
HTTP request made to http://localhost:9000/leaderboard



Browser

Local computer / Server

How does the App work



Javascript in the Browser is executed

```
// API URL
axios.defaults.baseURL = 'http://localhost:9000/';
// Our leaderboard list
var leaderboard = [];
// Call the API
axios.get('/leaderboard')
    .then((response) => {
        leaderboard = response.data;
        // Build the table
        buildLeaderboardTable(leaderboard);
    });
```

HTTP request made to http://localhost:9000/leaderboard

/leaderboard was requested so the results of the /leaderboard will be sent back to browser. In this case is a list of objects

Host machine forwards request to port 9000 to container

Port 9000 is mapped to 9000 inside the container

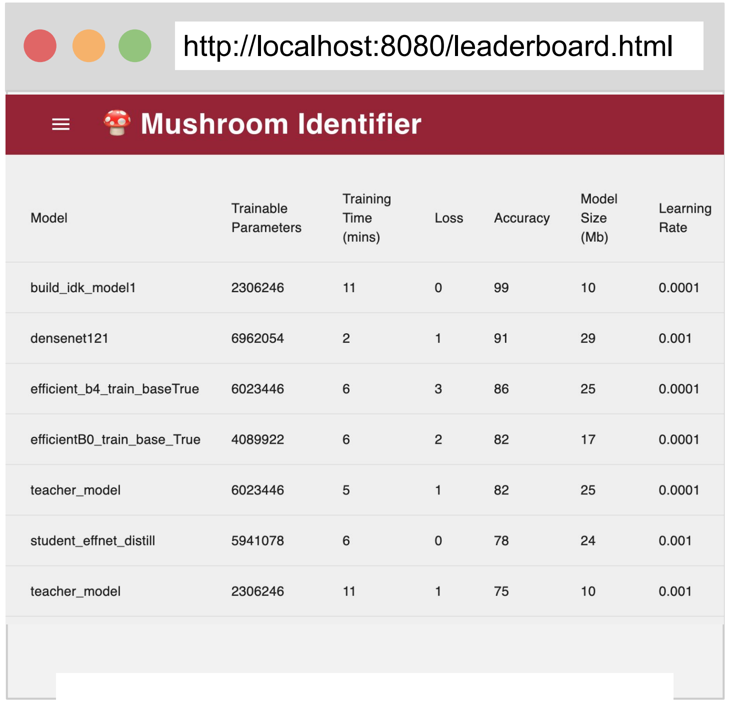
FastAPI is running on port 9000 serving /leaderboard

```
@app.get("/leaderboard")
def leaderboard_fetch():
    # Fetch leaderboard
    df = pd.read_csv("leaderboard.csv")
    return df.to_dict('records')
```

Browser

Local computer / Server

How does the App work



http://localhost:8080/leaderboard.html

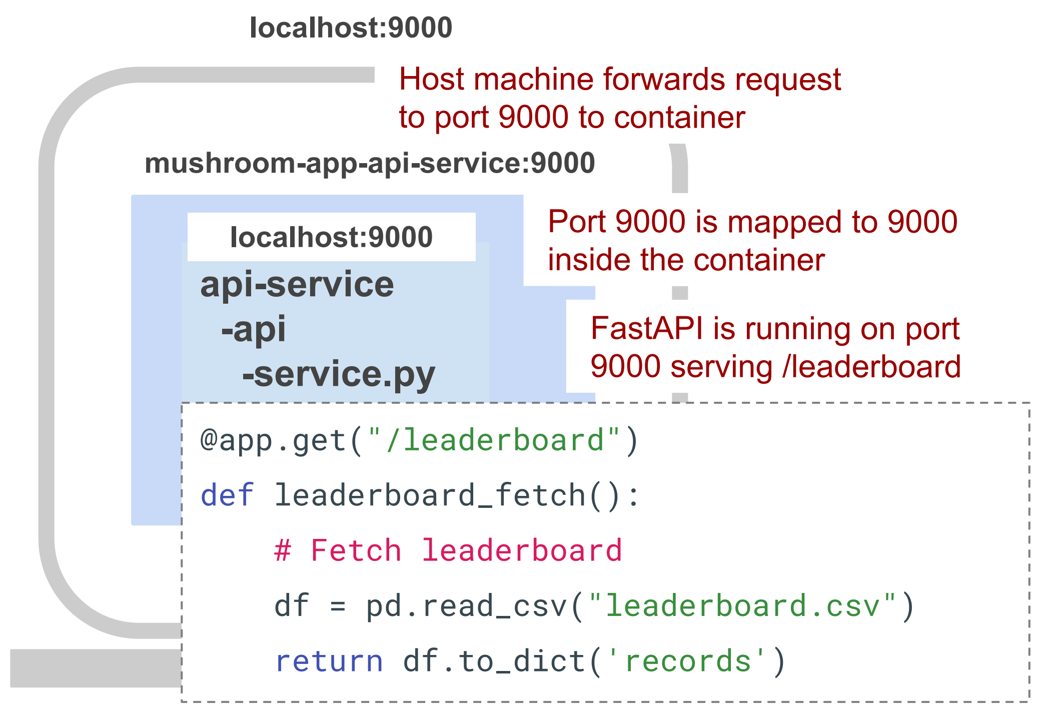
Mushroom Identifier

Model	Trainable Parameters	Training Time (mins)	Loss	Accuracy	Model Size (Mb)	Learning Rate
build_idk_model1	2306246	11	0	99	10	0.0001
densenet121	6962054	2	1	91	29	0.001
efficient_b4_train_baseTrue	6023446	6	3	86	25	0.0001
efficientB0_train_base_True	4089922	6	2	82	17	0.0001
teacher_model	6023446	5	1	82	25	0.0001
student_effnet_distill	5941078	6	0	78	24	0.001
teacher_model	2306246	11	1	75	10	0.001

Javascript displays the leaderboard data in the html page.

HTTP request made to http://localhost:9000/leaderboard

/leaderboard was requested so the results of the /leaderboard will be sent back to browser. In this case is a list of objects


localhost:9000
Host machine forwards request to port 9000 to container
mushroom-app-api-service:9000
Port 9000 is mapped to 9000 inside the container
localhost:9000
api-service -api -service.py
FastAPI is running on port 9000 serving /leaderboard

```
@app.get("/leaderboard")
def leaderboard_fetch():
    # Fetch leaderboard
    df = pd.read_csv("leaderboard.csv")
    return df.to_dict('records')
```

Browser

Local computer / Server

Outline

1. Recap
2. APIs
3. App Frontend (Simple)
4. **Model Serving**
5. Frontend Frameworks

Tutorial: Model Serving API

Mushroom App - Model Serving API

Outline

1. Recap
2. APIs
3. App Frontend (Simple)
4. Model Serving
5. **Frontend Frameworks**

Frontend

When we build our frontend we had a page for each component:

- index.html
- leaderboard.html
- predict.html

Frontend

When we build our frontend we had a page for each component:

- index.html
- leaderboard.html
- predict.html

Problems:

- Each of these had its own HTML, Javascript, CSS
- How do we share/reuse code across pages
- Each page is loaded separately in browser (Slow)

Frontend

Problems:

- Each of these had its own HTML, Javascript, CSS
- How do we share/reuse code across pages
- Each page is loaded separately in browser (Slow)

Solution:

- Create a single page app that manages HTML, Javascript, CSS as components
- Frontend App Frameworks to the rescue

Frontend Frameworks

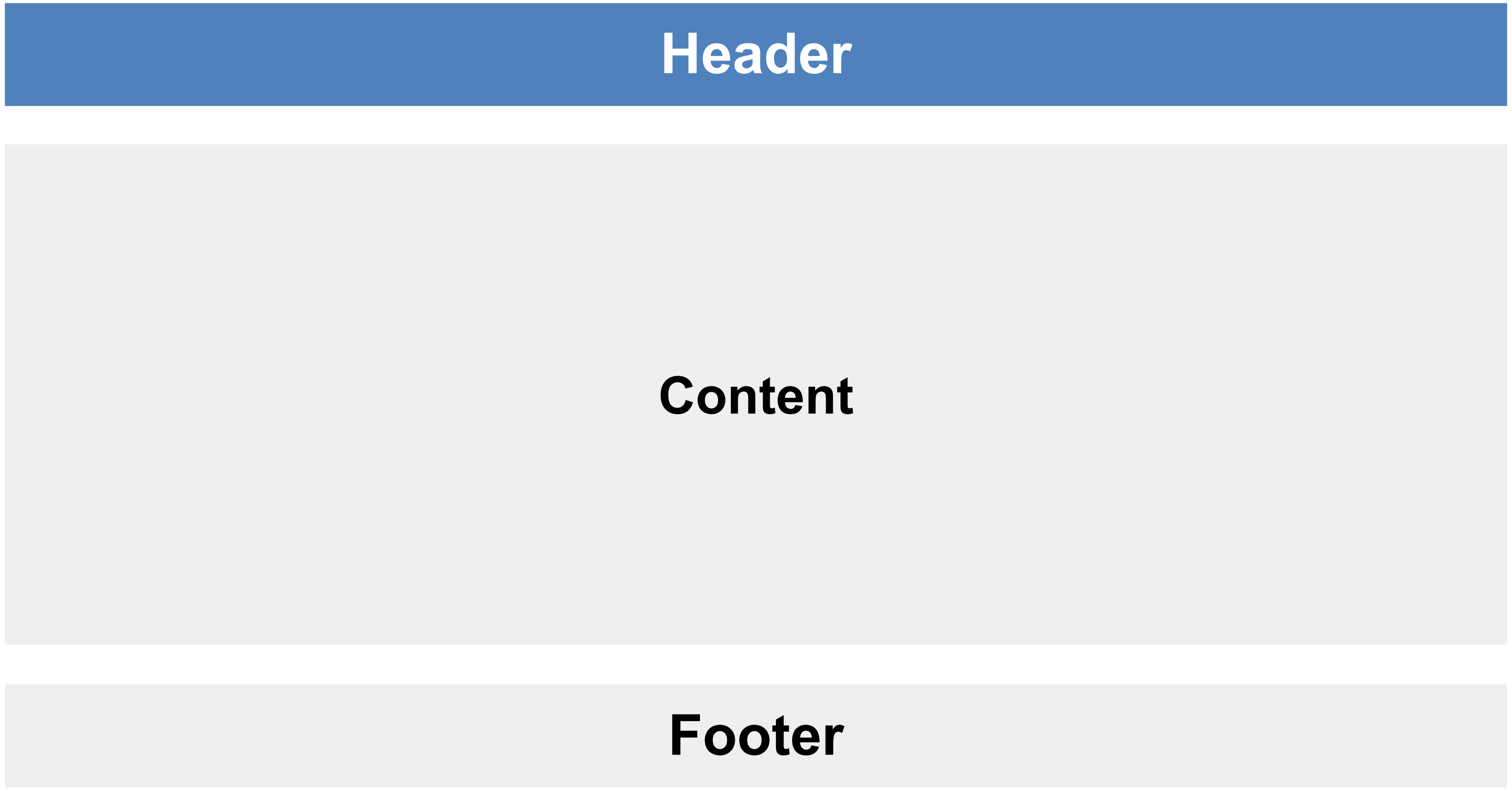
There major frontend app frameworks are:

- Angular (Google)
- React (Facebook)
- Vue

React

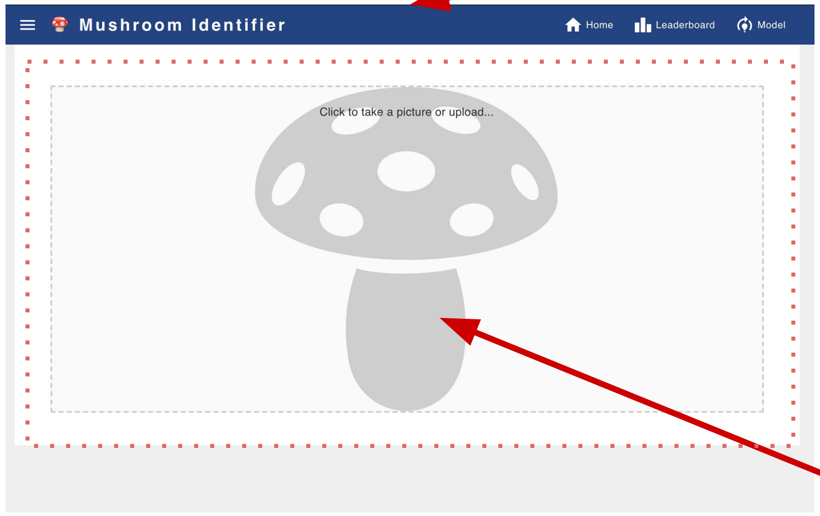
- Everything is a Component
- Uses JSX instead of Javascript
- JSX is an extension to JavaScript
- JSX is like a template language, but it comes with the full power of JavaScript

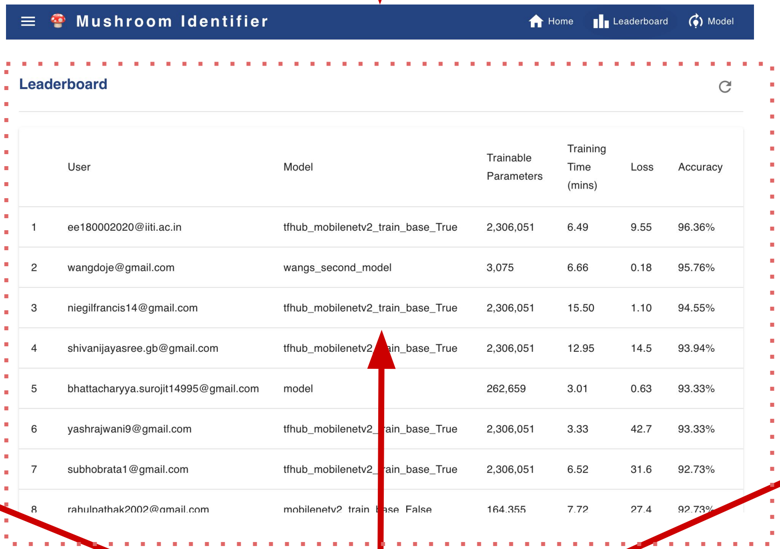
React App



React App

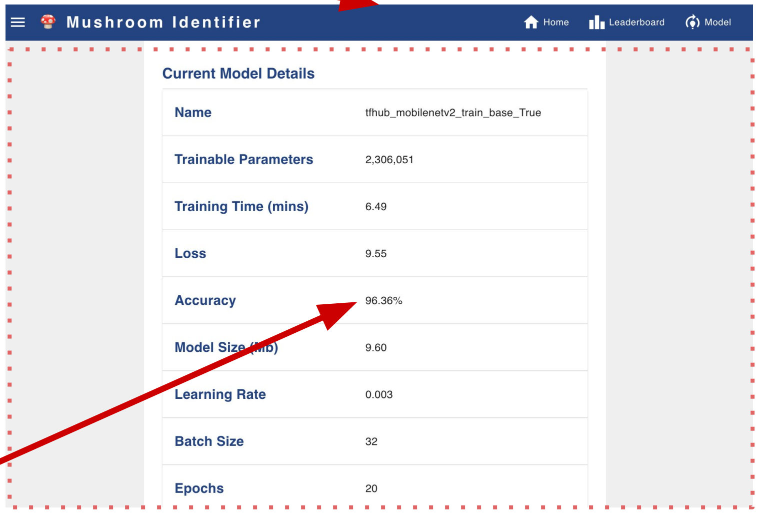
Header defined only once





Leaderboard

	User	Model	Trainable Parameters	Training Time (mins)	Loss	Accuracy
1	ee180002020@iiti.ac.in	tfhub_mobilenetv2_train_base_True	2,306,051	6.49	9.55	96.36%
2	wangdoje@gmail.com	wangs_second_model	3,075	6.66	0.18	95.76%
3	niegilfrancis14@gmail.com	tfhub_mobilenetv2_train_base_True	2,306,051	15.50	1.10	94.55%
4	shivanijayasree.gb@gmail.com	tfhub_mobilenetv2_train_base_True	2,306,051	12.95	14.5	93.94%
5	bhattacharyya.surojit14995@gmail.com	model	262,659	3.01	0.63	93.33%
6	yashrajwani9@gmail.com	tfhub_mobilenetv2_train_base_True	2,306,051	3.33	42.7	93.33%
7	subhobrata1@gmail.com	tfhub_mobilenetv2_train_base_True	2,306,051	6.52	31.6	92.73%
8	rahulpathak2002@gmail.com	mobilenetv2_train_base_False	164,355	7.72	27.4	92.73%



Current Model Details

Name	tfhub_mobilenetv2_train_base_True
Trainable Parameters	2,306,051
Training Time (mins)	6.49
Loss	9.55
Accuracy	96.36%
Model Size (Mb)	9.60
Learning Rate	0.003
Batch Size	32
Epochs	20

Content block switched for each page

THANK YOU