

Lecture 14: APIs & Frontend

AC215

Pavlos Protopapas / Shivas Jayaram

SEAS/ Harvard



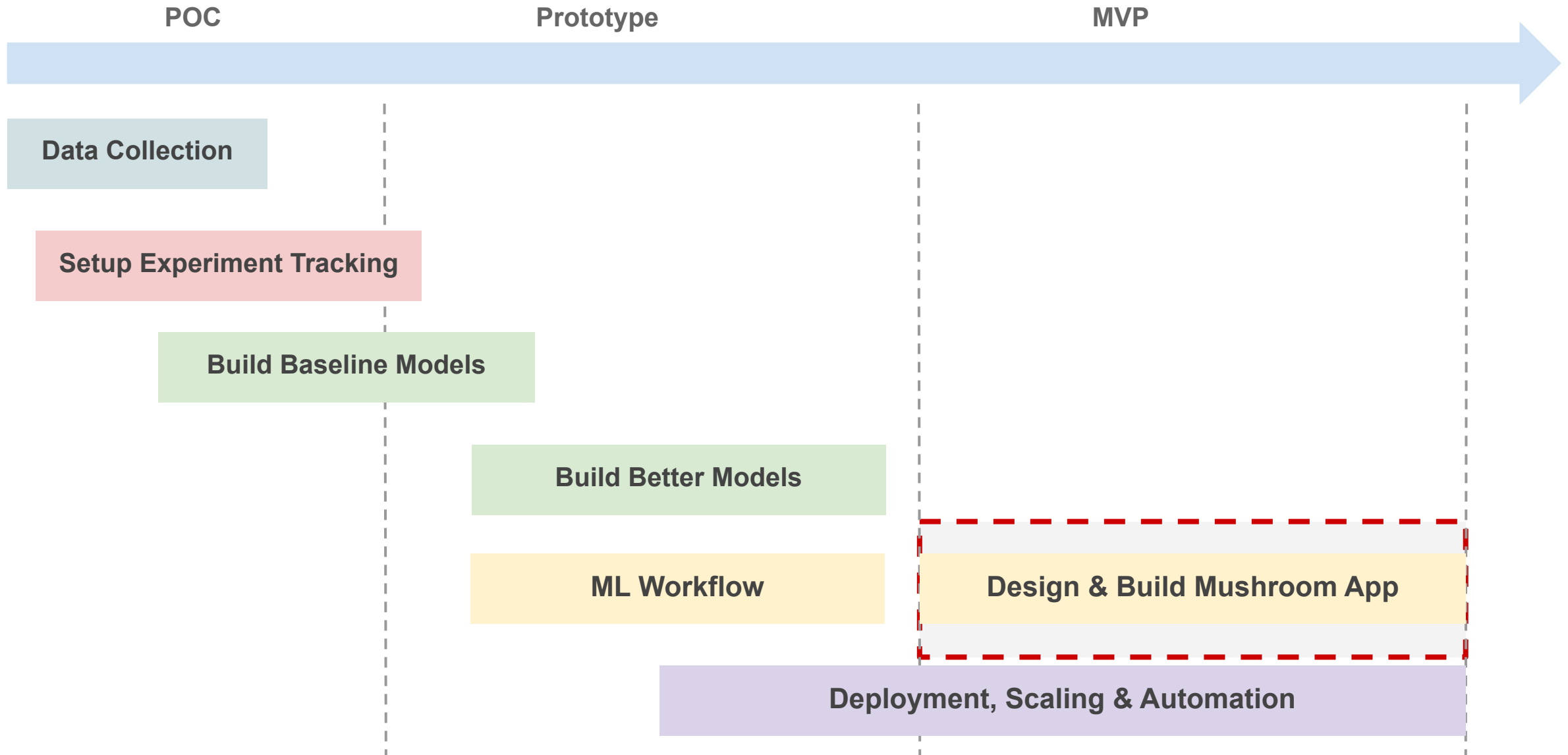
Outline

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

Outline

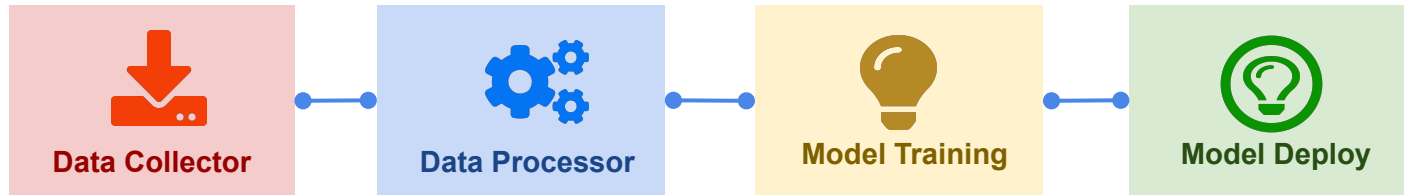
1. **Recap**
2. APIs
3. Frontend (Simple)
4. Model Serving
5. Frontend Frameworks

Recap: Mushroom App Status



Recap: Mushroom App Development

ML Pipeline



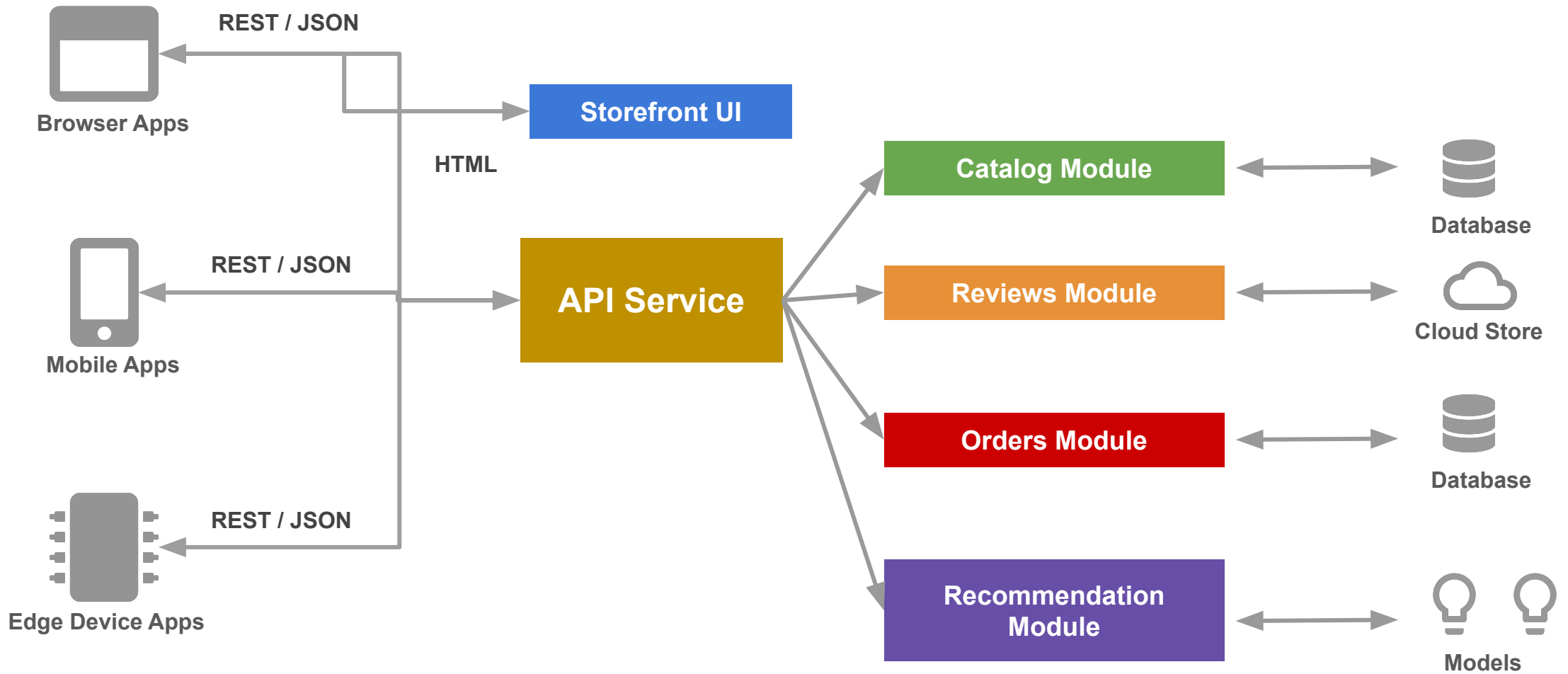
App Dev



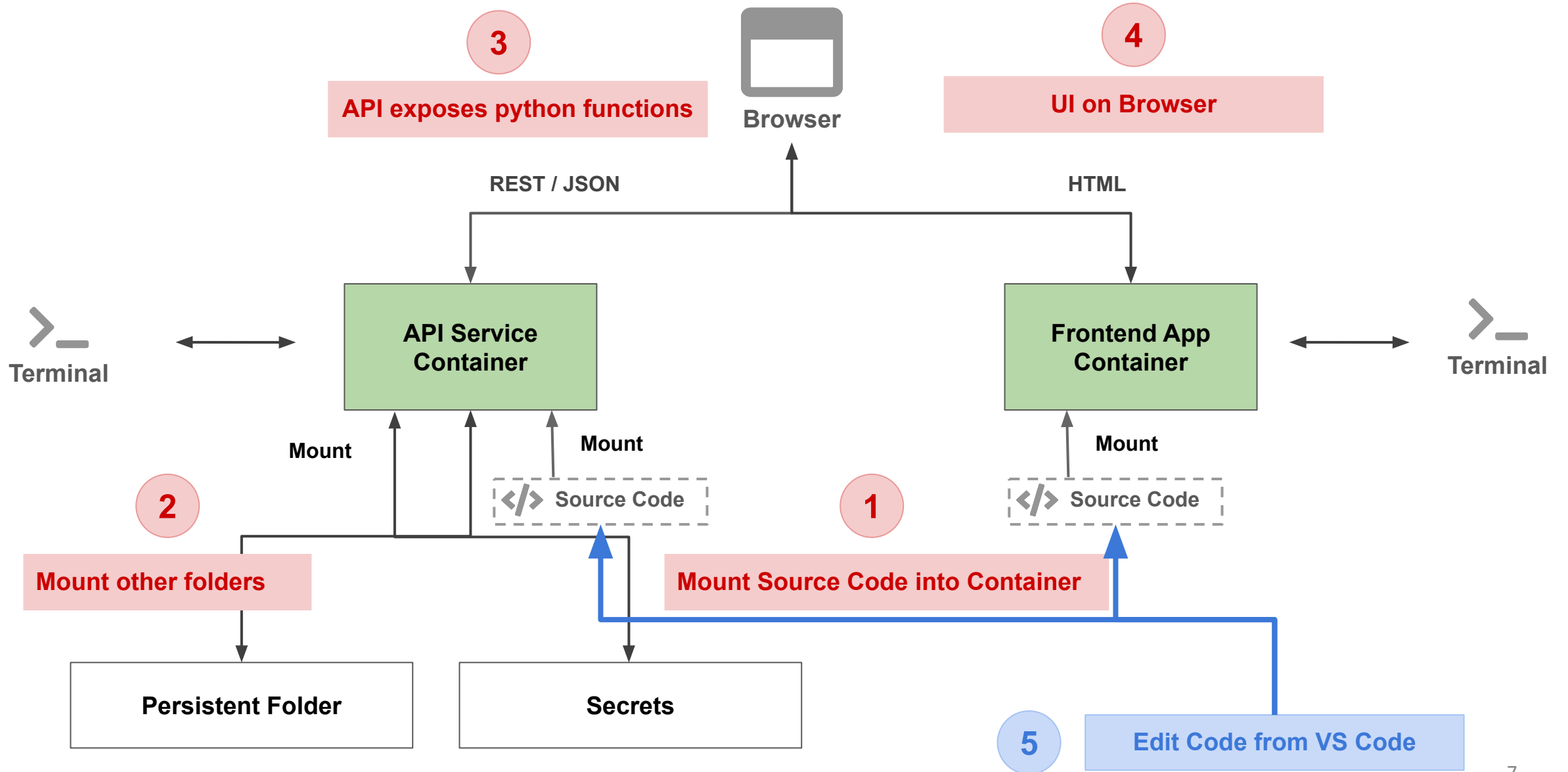
Google Cloud Platform



Recap: Microservice Architecture



What we built so far

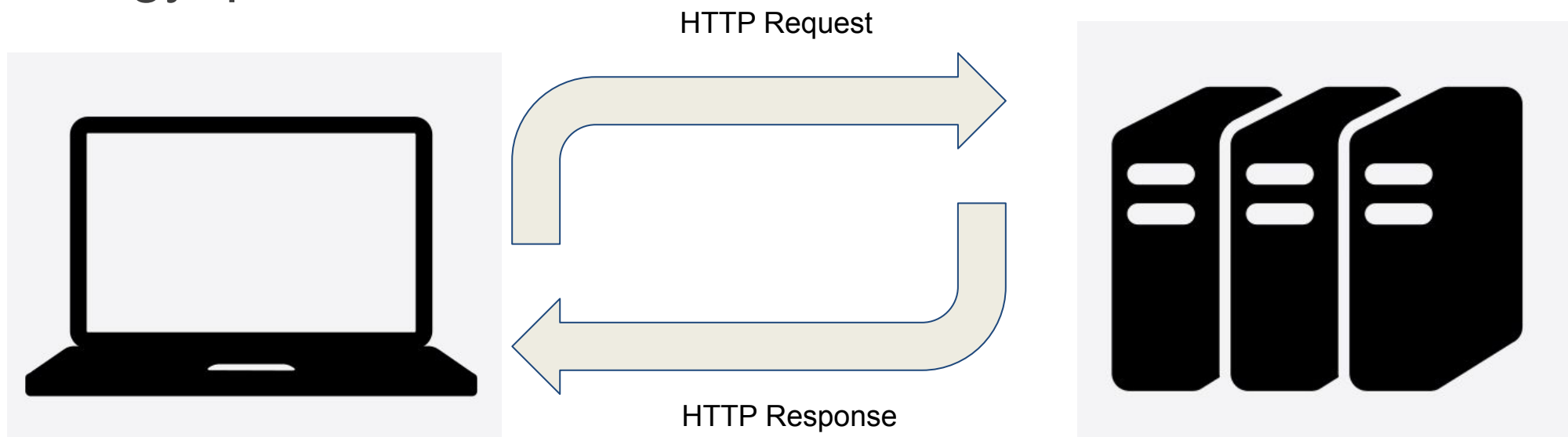


Outline

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

Review: What is HTTP?

- HyperText Transfer Protocol: method for **transporting information** where **client** (such as a web browser) makes **request** and web **server** issues a **response**
 - content can be anything from text to images to video
- HTTPS: encryption for **secure** communication over network
- Analogy: post office



Review: What is a port?

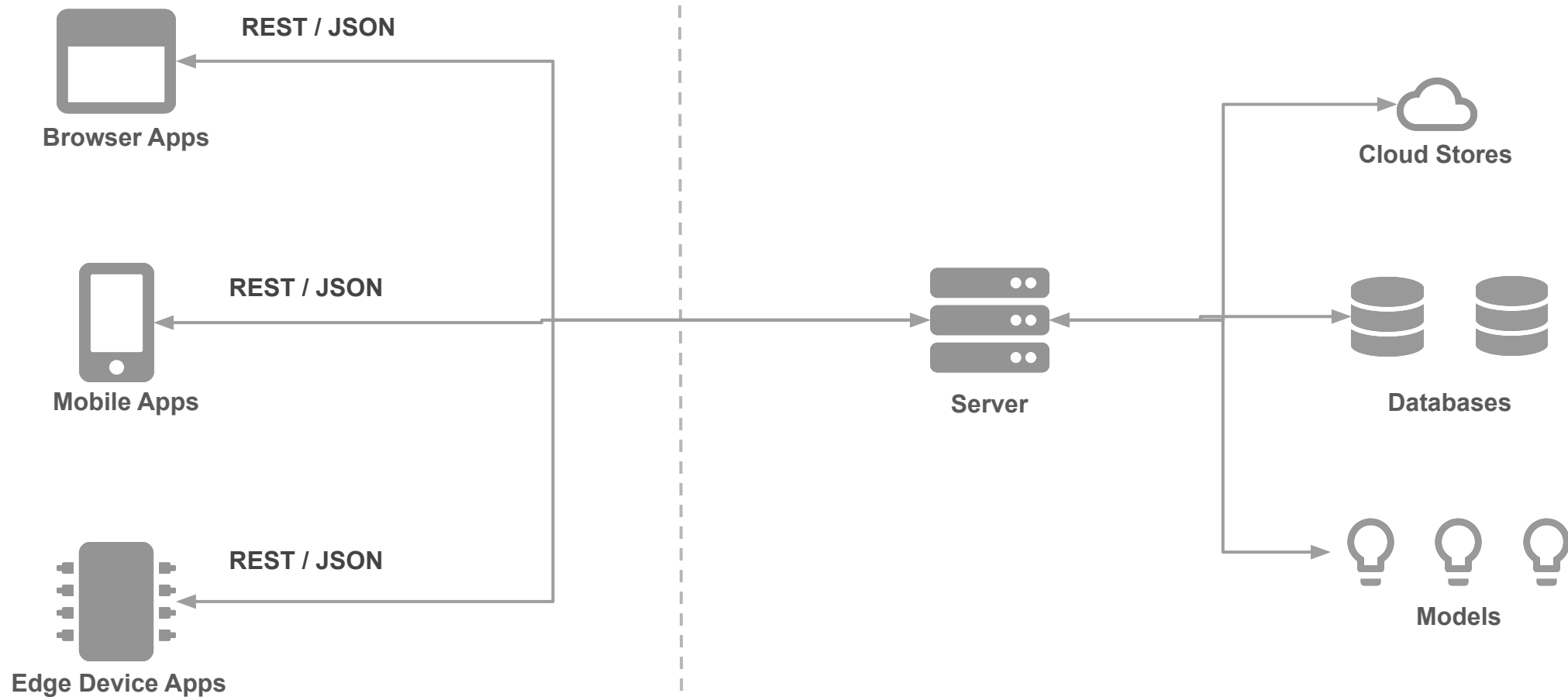
- **communication endpoint** where network connections start and end
- lets **computers differentiate** between different kinds of **data** (emails, webpages, etc.)
 - Port 22 = SSH
 - Port 25 = SMTP (email)
 - Port 80 = HTTP
 - Port 443 = HTTPS

What is an API

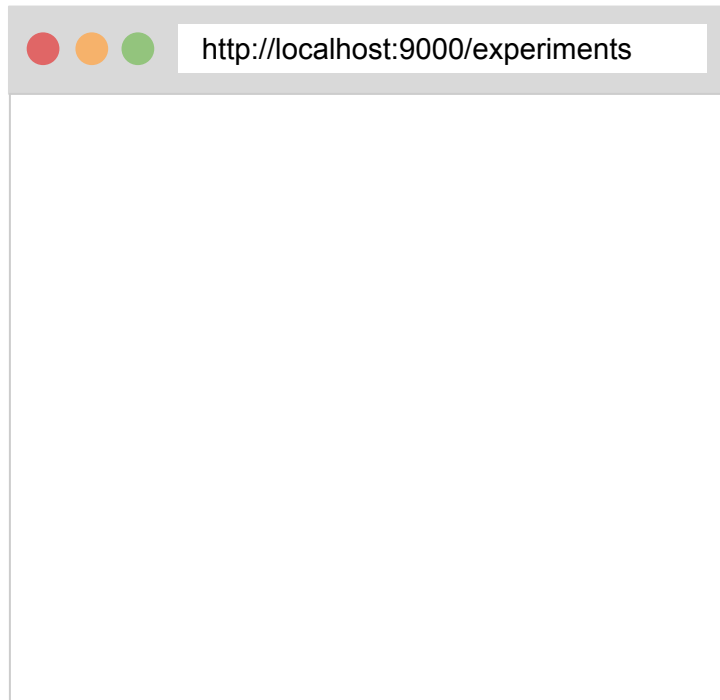
- API is **Application Programming Interface**
- **Web API** is an API that can be access using HTTP/S
- A **REST API** is a Web API that follows the HTTP method constraints - get, post, put, delete
- We will use **FastAPI** a Python framework to build REST APIs

APIs

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

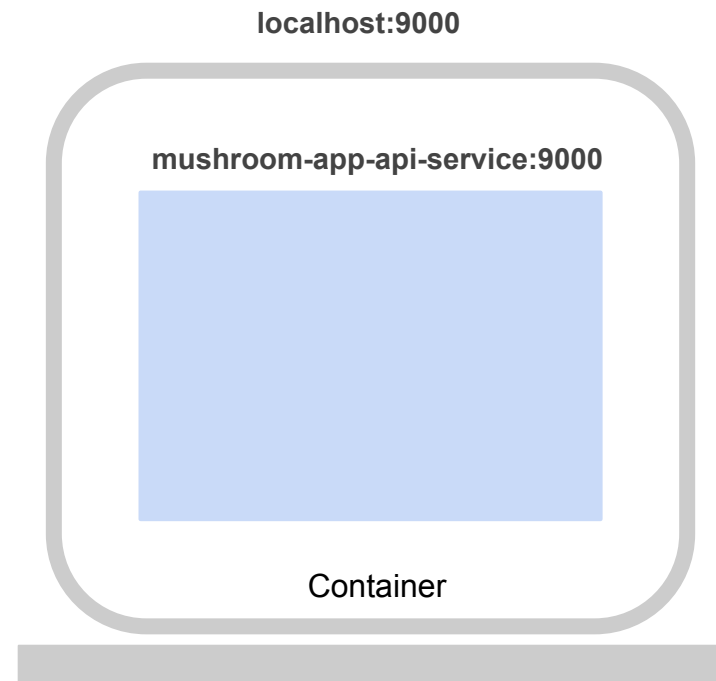


How does an API work



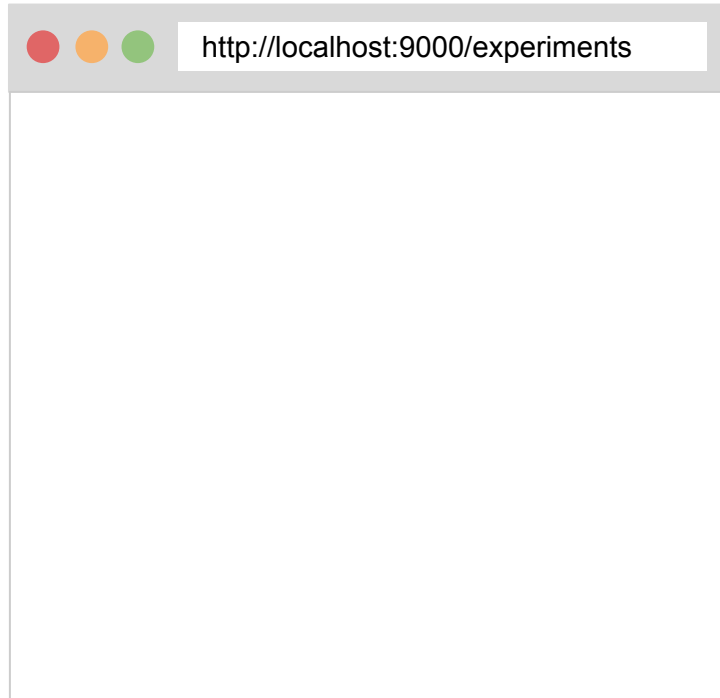
Browser

HTTP request made to localhost



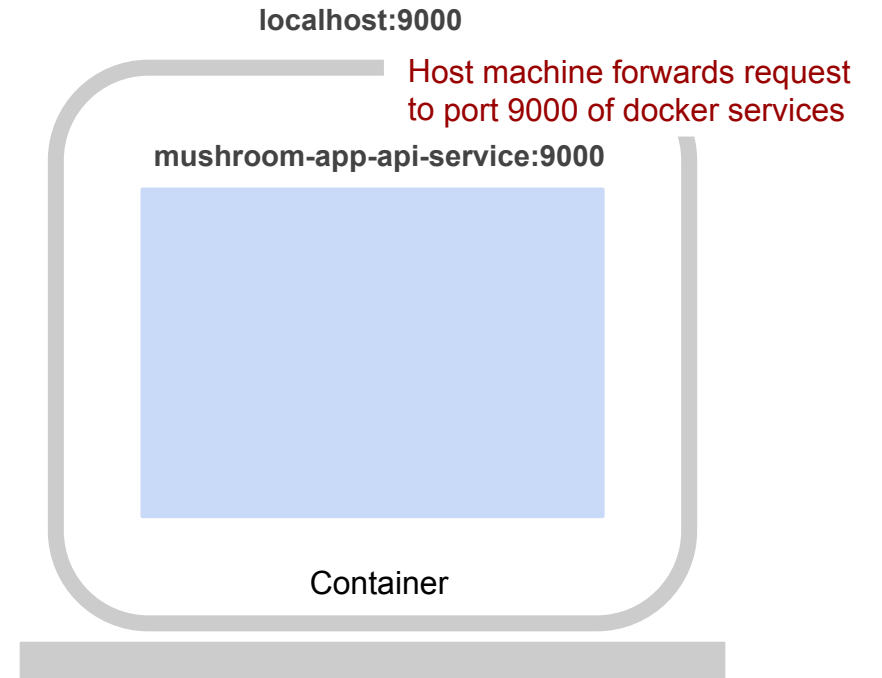
Local computer / Server

How does an API work



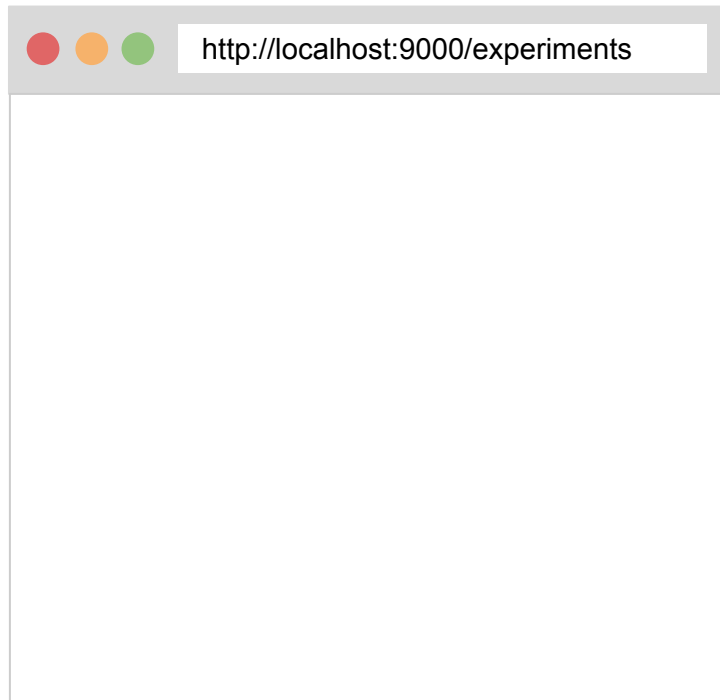
Browser

HTTP request made to localhost



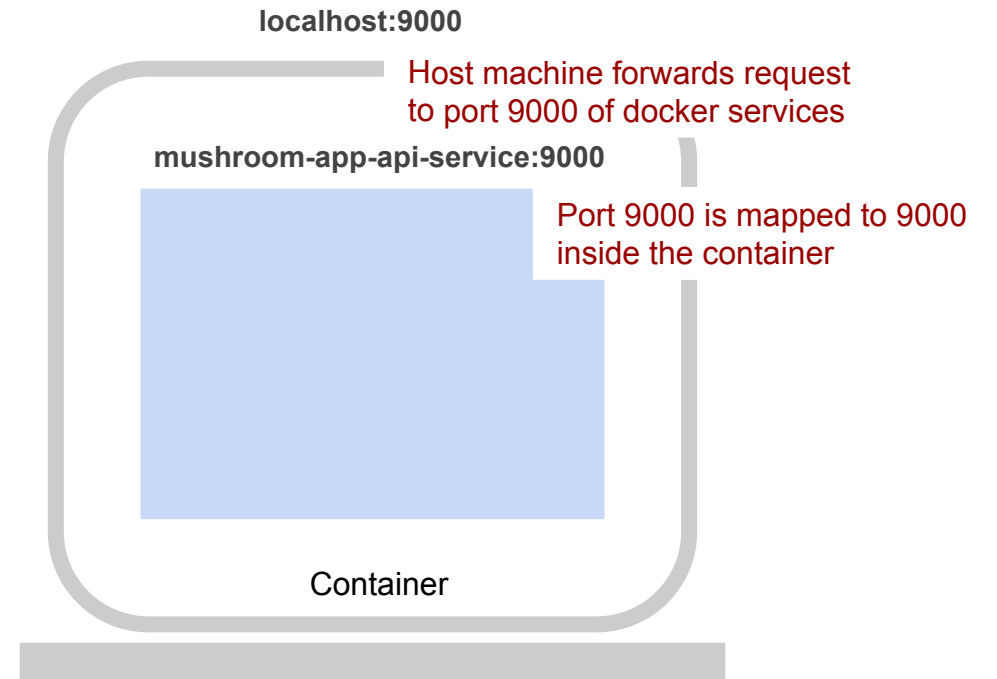
Local computer / Server

How does an API work



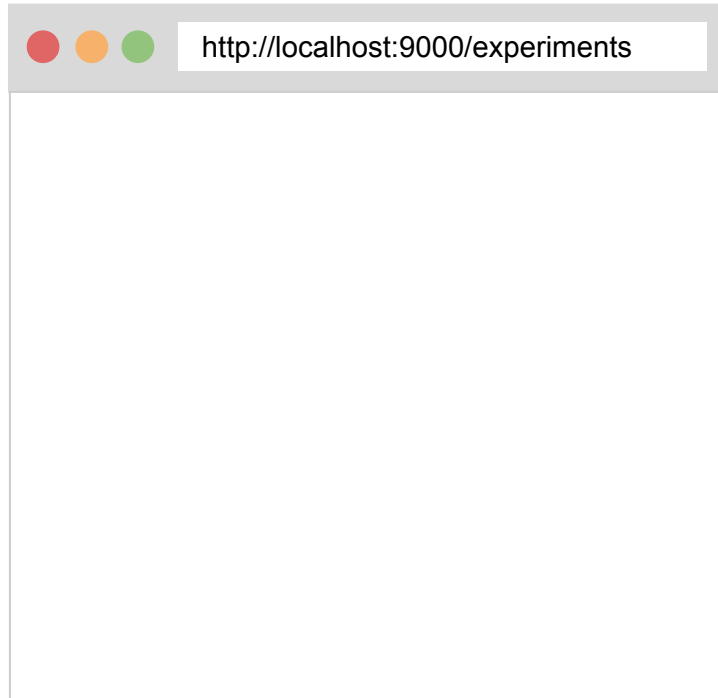
Browser

HTTP request made to localhost

A red arrow points from the browser window towards the server diagram, indicating the direction of the HTTP request.

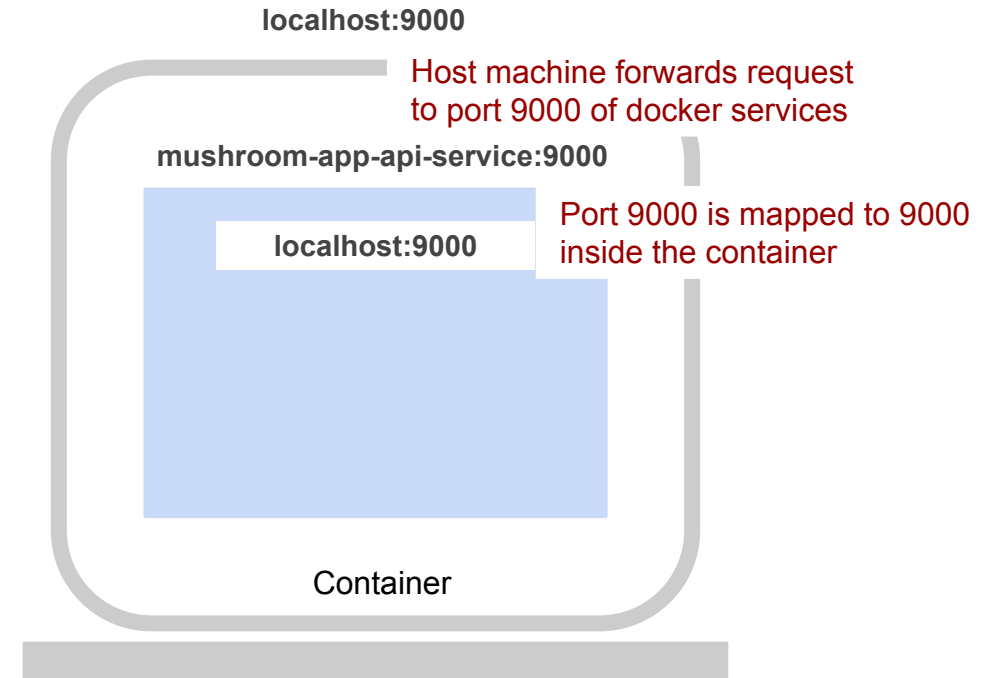
Local computer / Server

How does an API work



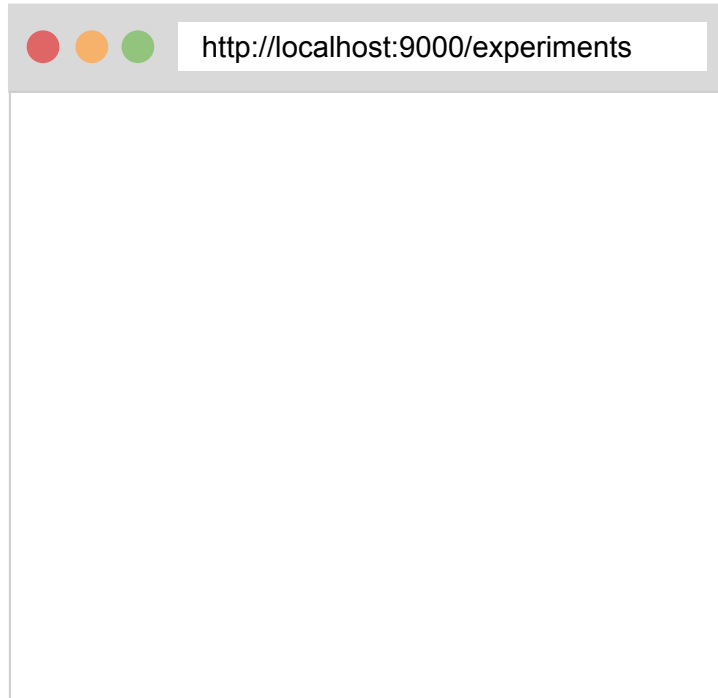
Browser

HTTP request made to localhost



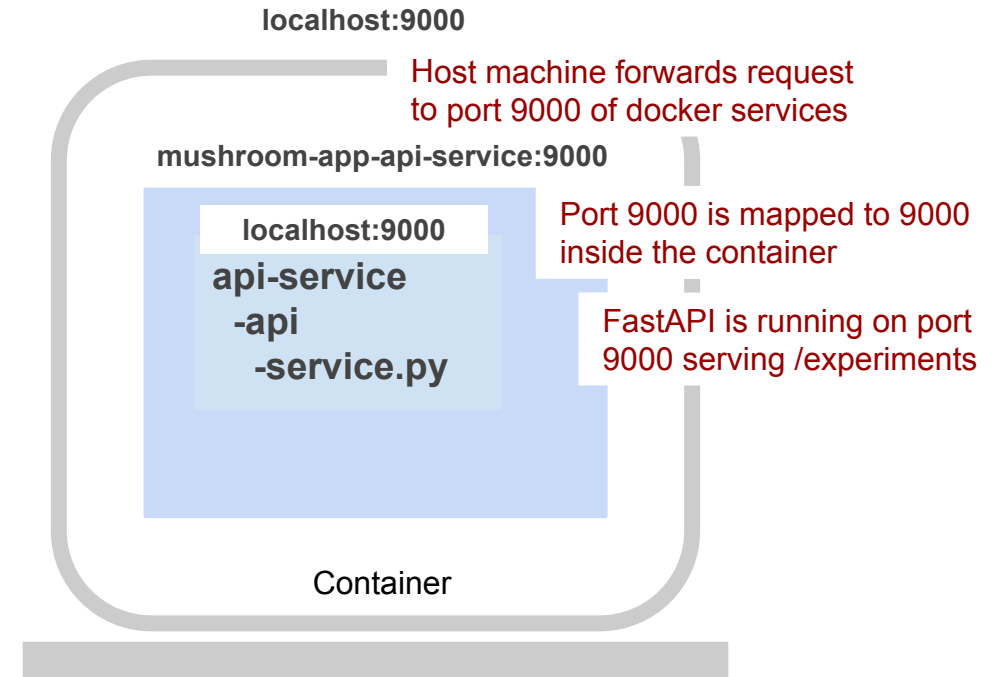
Local computer / Server

How does an API work



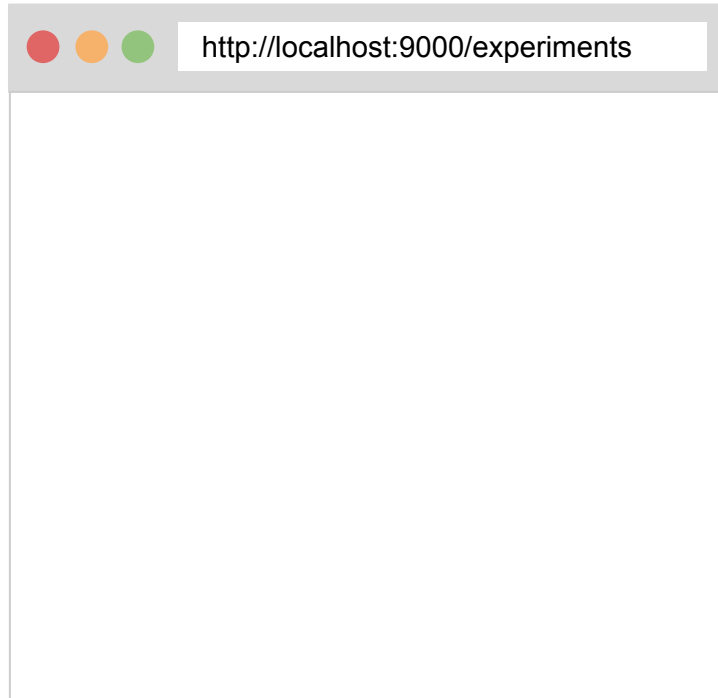
Browser

HTTP request made to localhost



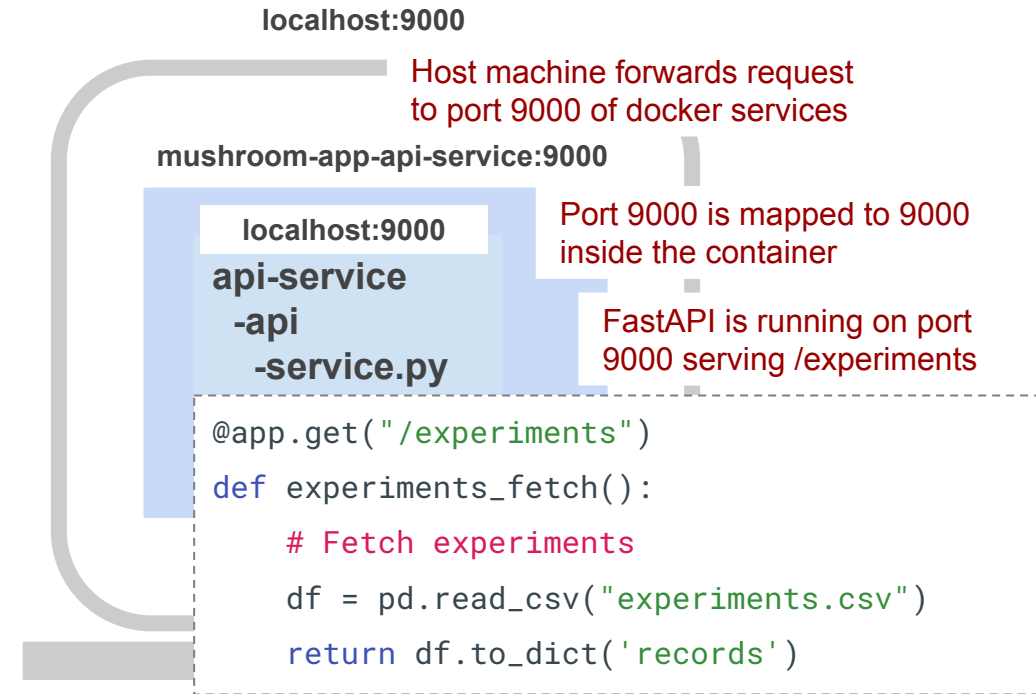
Local computer / Server

How does an API work



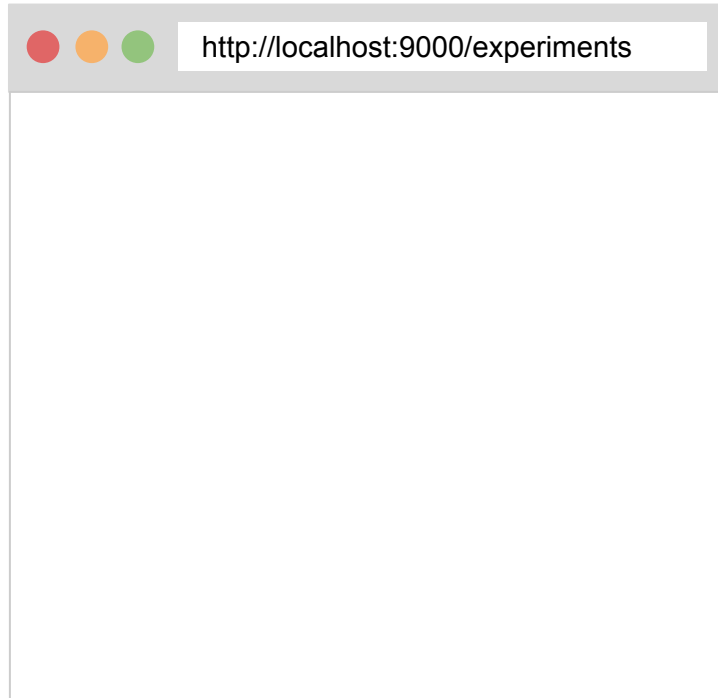
Browser

HTTP request made to localhost



Local computer / Server

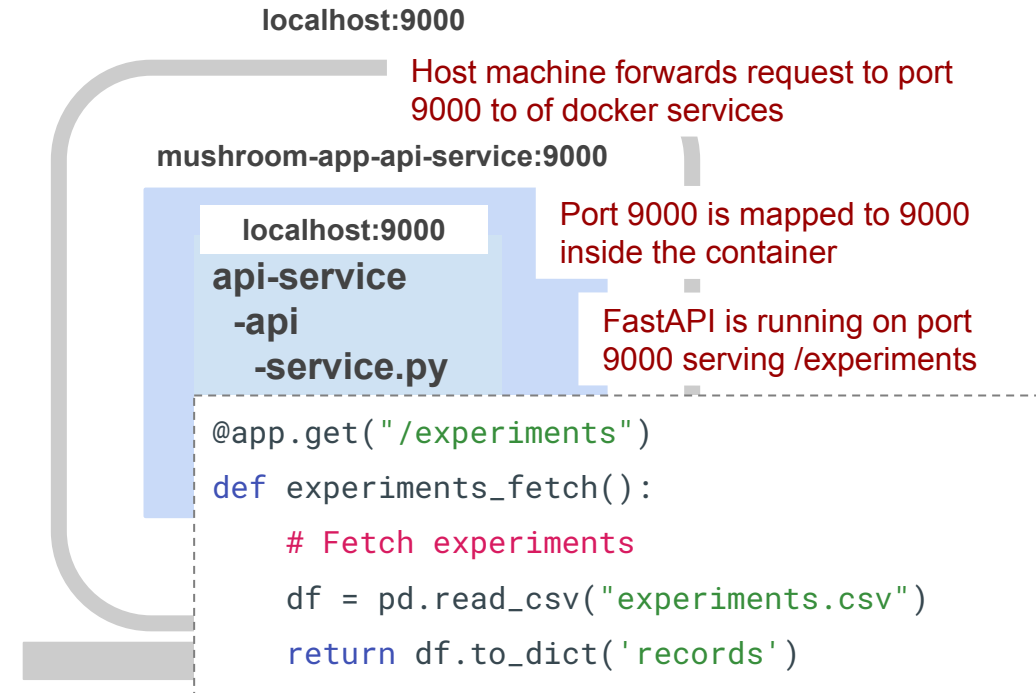
How does an API work



Browser

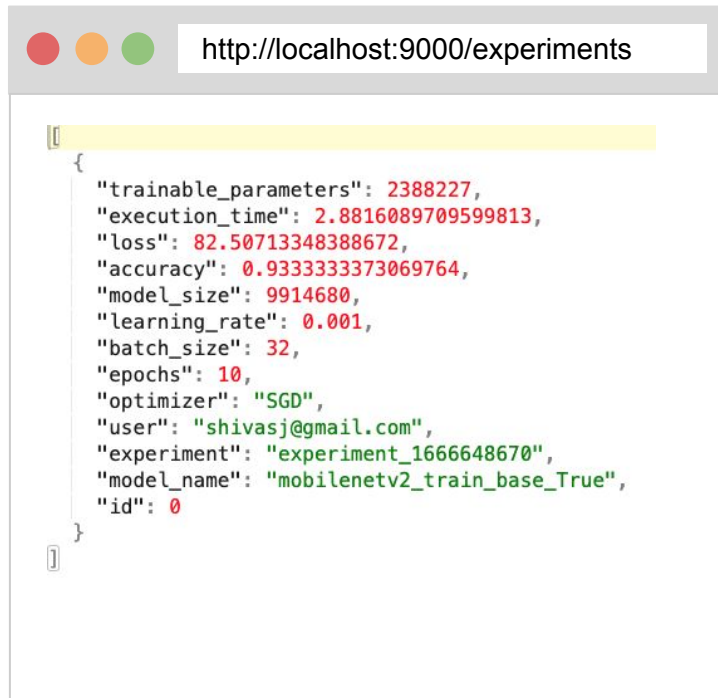
HTTP request made to localhost

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



Local computer / Server

How does an API work

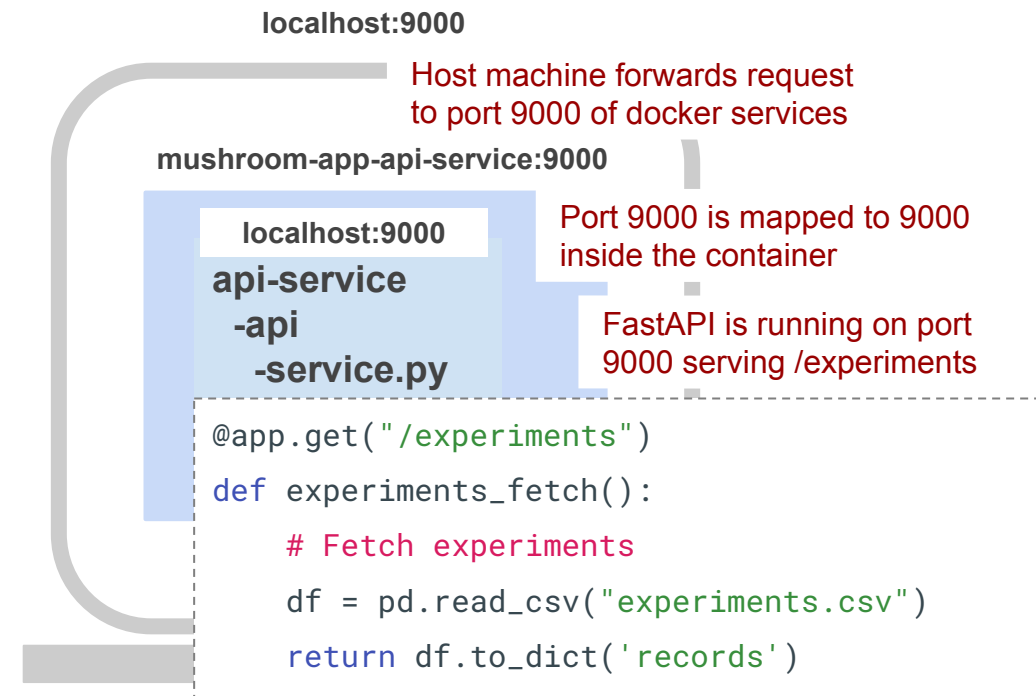


```
{
  "trainable_parameters": 2388227,
  "execution_time": 2.8816089709599813,
  "loss": 82.50713348388672,
  "accuracy": 0.9333333373069764,
  "model_size": 9914680,
  "learning_rate": 0.001,
  "batch_size": 32,
  "epochs": 10,
  "optimizer": "SGD",
  "user": "shivasj@gmail.com",
  "experiment": "experiment_1666648670",
  "model_name": "mobilenetv2_train_base_True",
  "id": 0
}
```

Browser

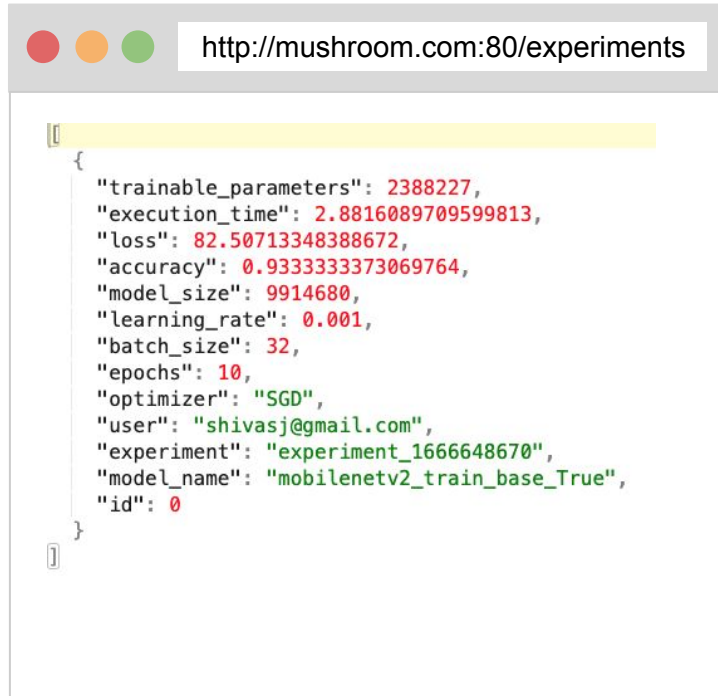
HTTP request made to localhost

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



Local computer / Server

How does an API work (In Production)

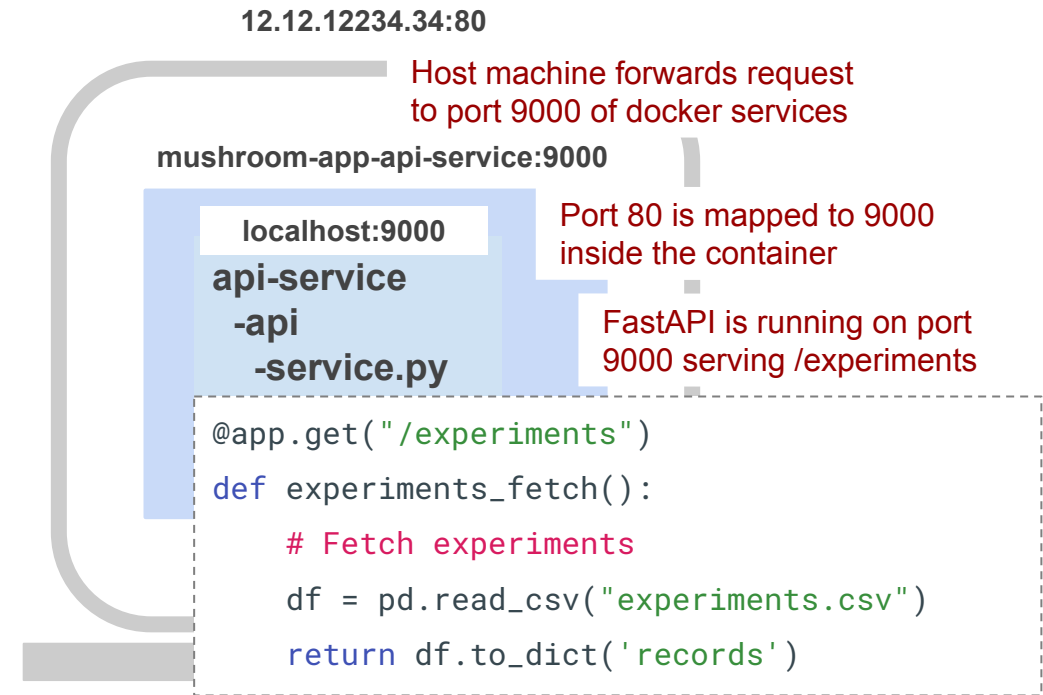


```
{
  "trainable_parameters": 2388227,
  "execution_time": 2.8816089709599813,
  "loss": 82.50713348388672,
  "accuracy": 0.9333333373069764,
  "model_size": 9914680,
  "learning_rate": 0.001,
  "batch_size": 32,
  "epochs": 10,
  "optimizer": "SGD",
  "user": "shivasj@gmail.com",
  "experiment": "experiment_1666648670",
  "model_name": "mobilenetv2_train_base_True",
  "id": 0
}
```

Browser

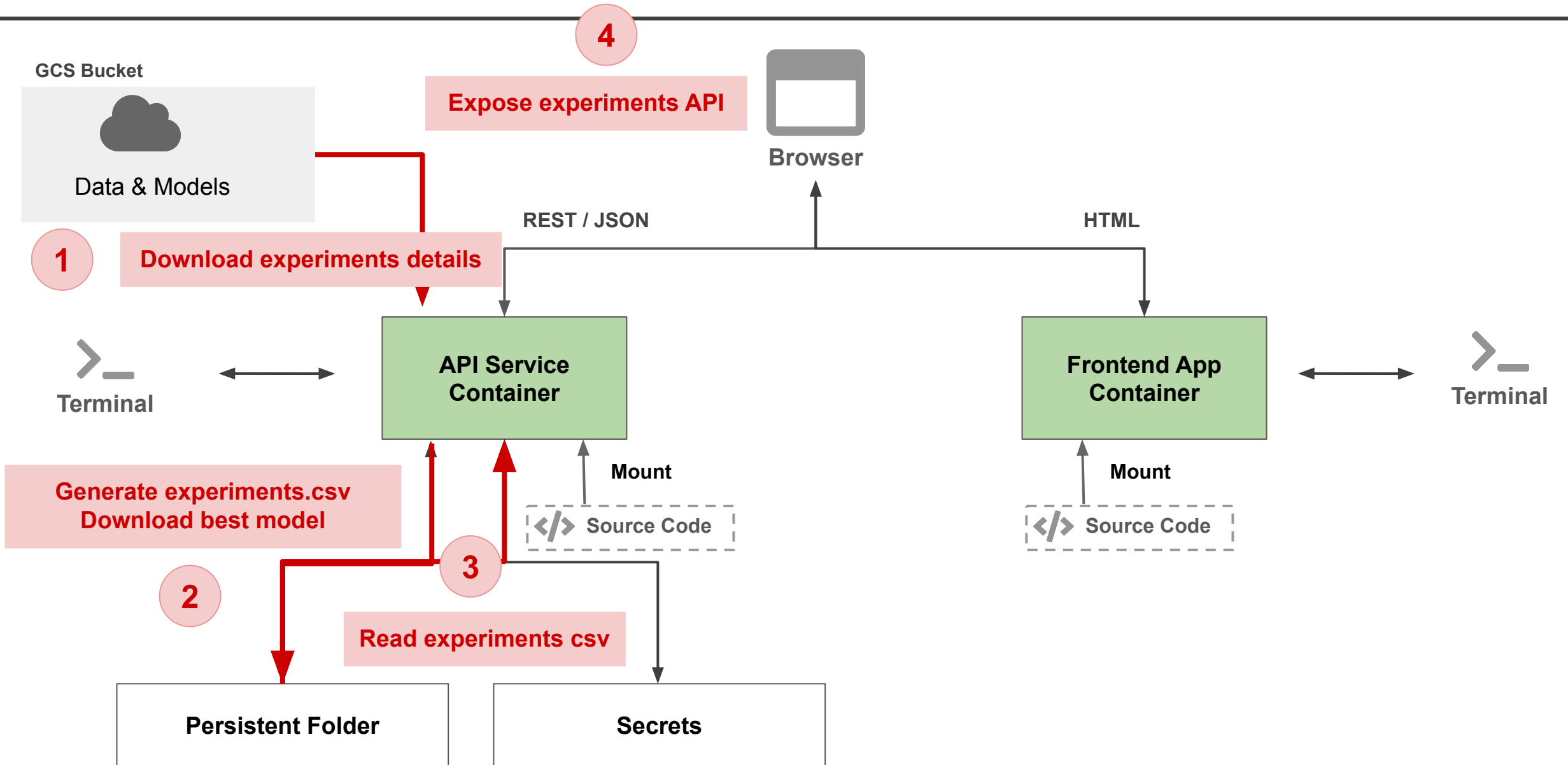
HTTP request made to localhost

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



GCP Server

Tutorial: APIs



Tutorial: APIs

Steps to create Mushroom App **APIs**:

- Download experiments from GCS bucket.
- Save best model in persistent store.
- Read experiments.csv using pandas.
- Expose data using an API.
- For detailed instructions, please refer to the following link
 - Mushroom App APIs. (<https://github.com/dlops-io/mushroom-app-v2#backend-apis>)

Outline

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

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

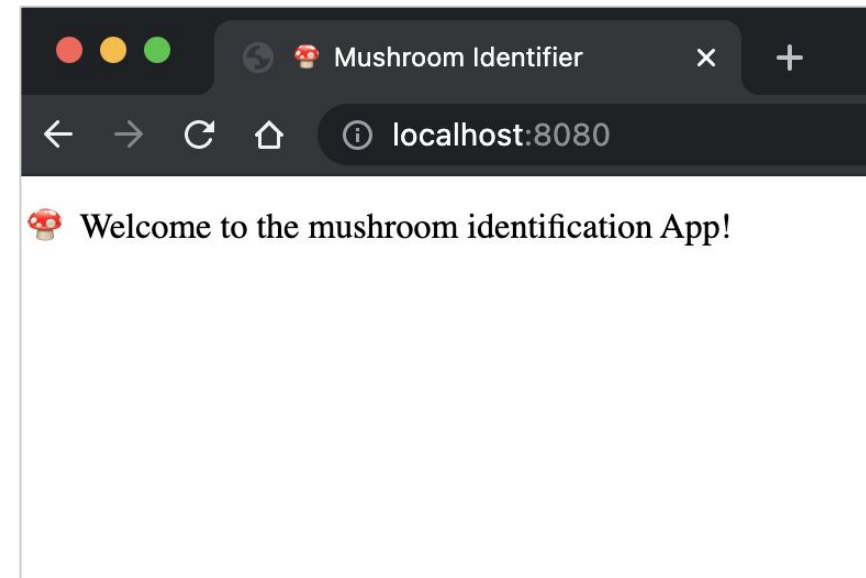
```
<!DOCTYPE html>
<html>
<head>
  <title>🍄 Mushroom Identifier</title>
  <style>body{background-color: #efefef;}</style>
</head>
<body>
  🍄 Welcome to the mushroom identification App!
</body>
<script>
  var input_file =
document.getElementById("input_file");
</script>
</html>
```

Browser Title

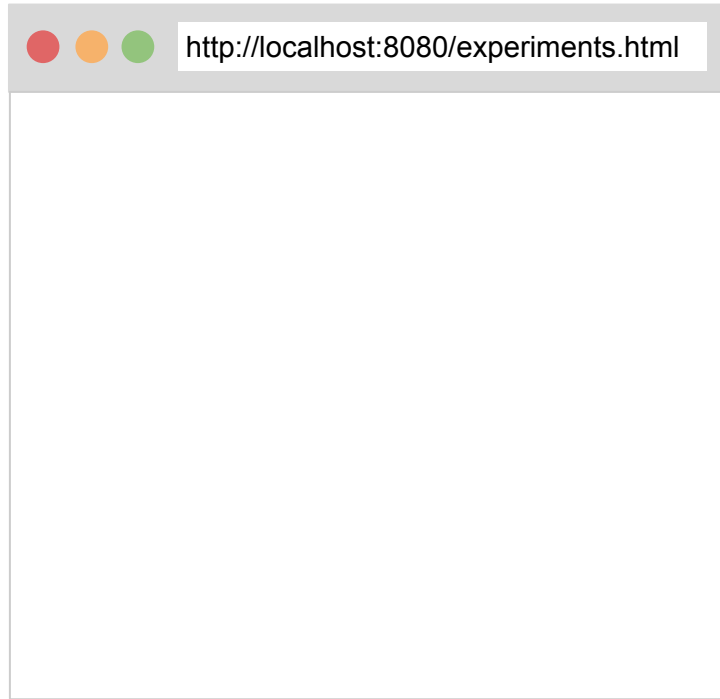
Page Style

Web page details

Web page scripts (Javascript)

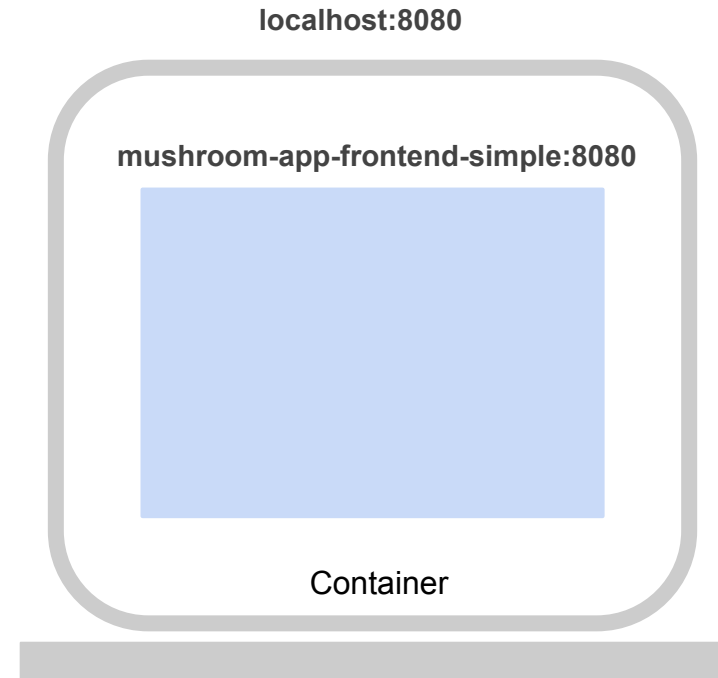


How does the App work



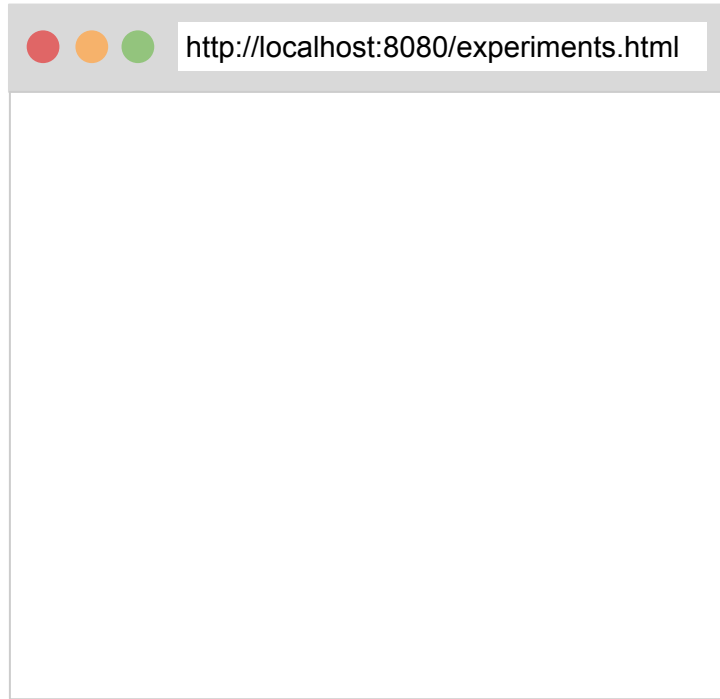
Browser

HTTP request made to localhost



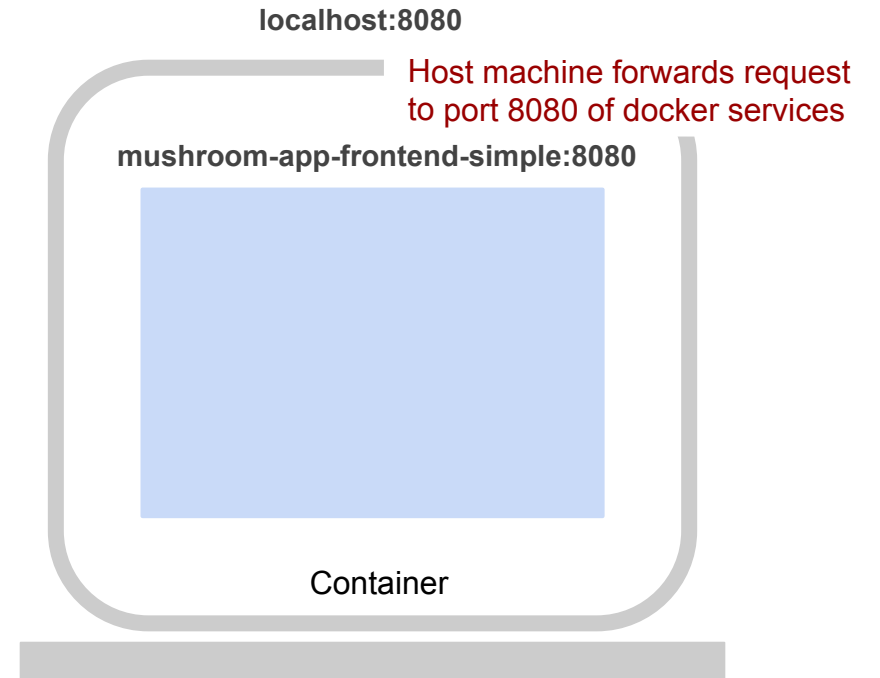
Local computer / Server

How does the App work



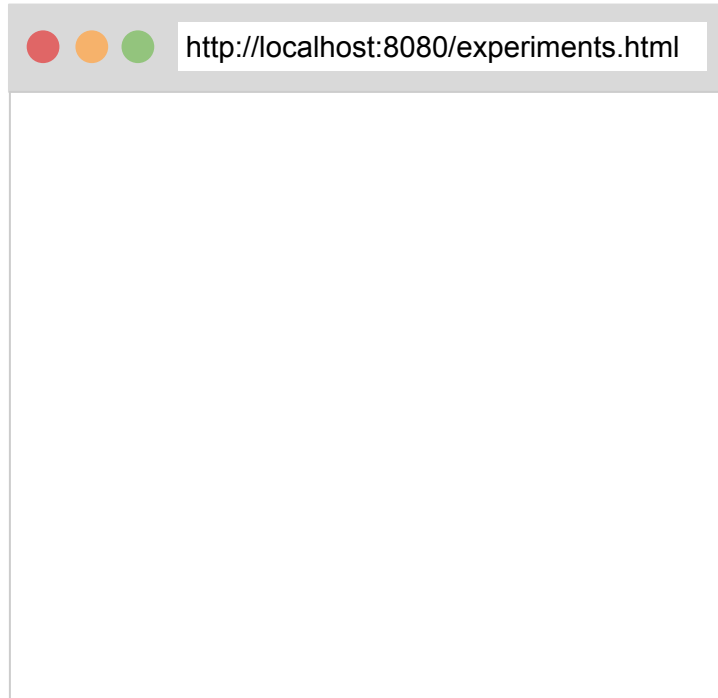
Browser

HTTP request made to localhost



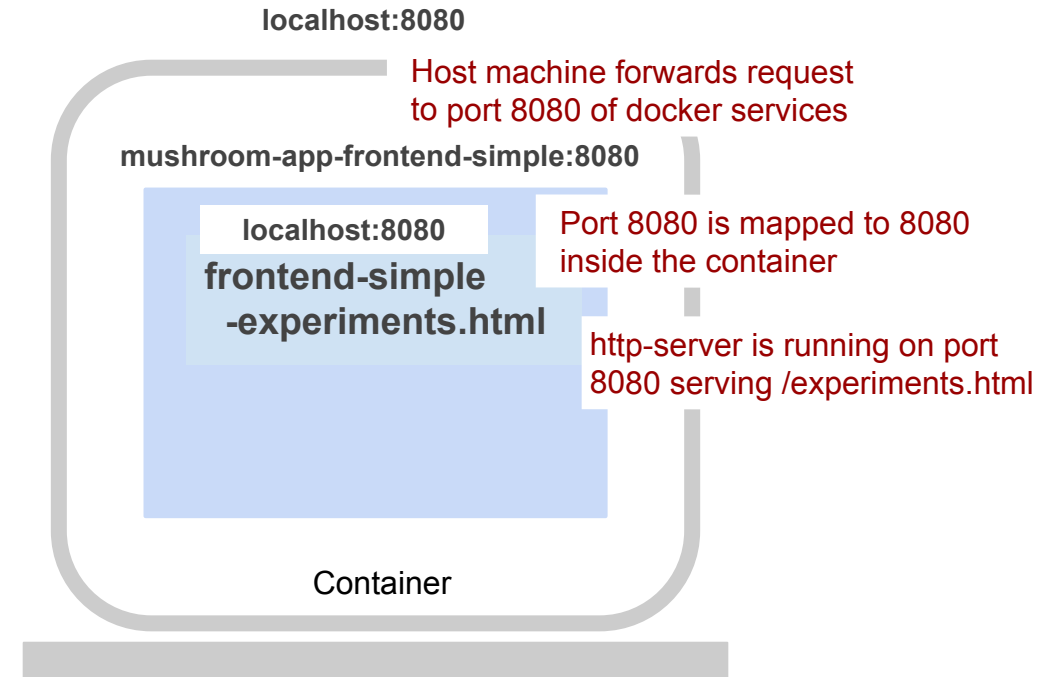
Local computer / Server

How does the App work



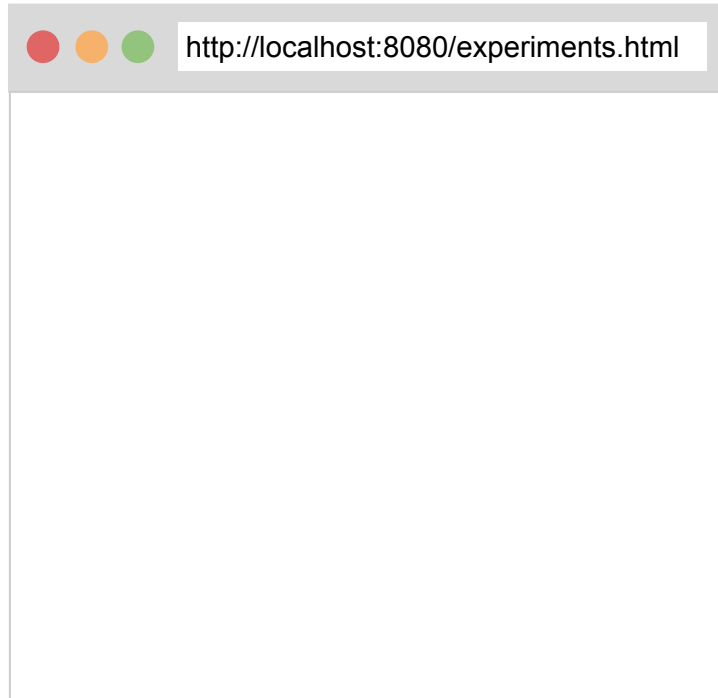
Browser

HTTP request made to localhost



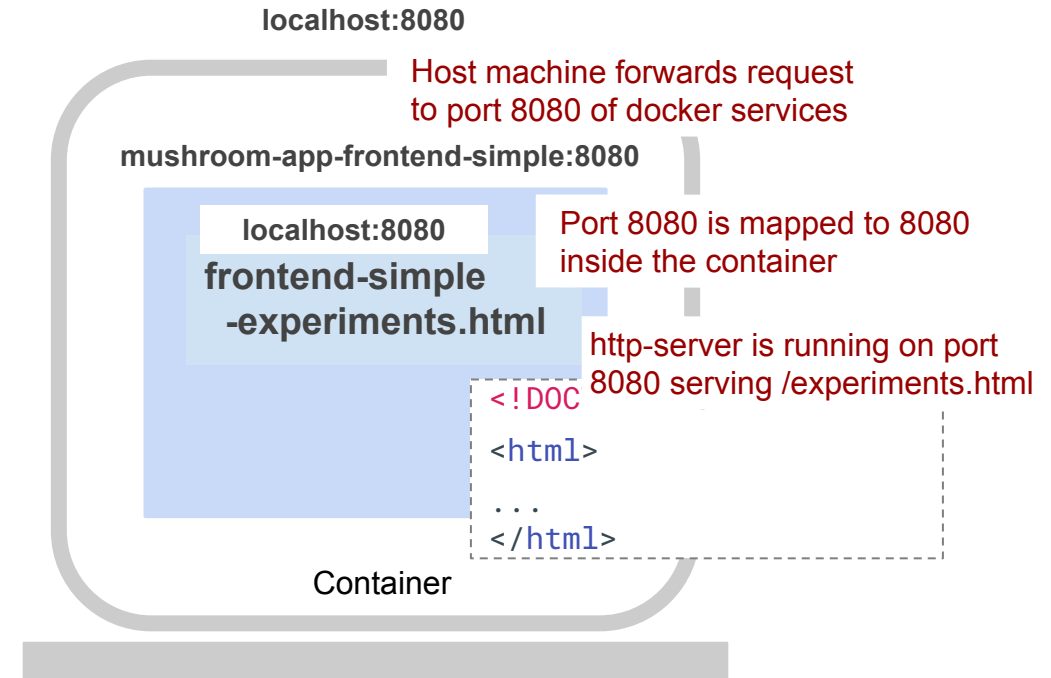
Local computer / Server

How does the App work



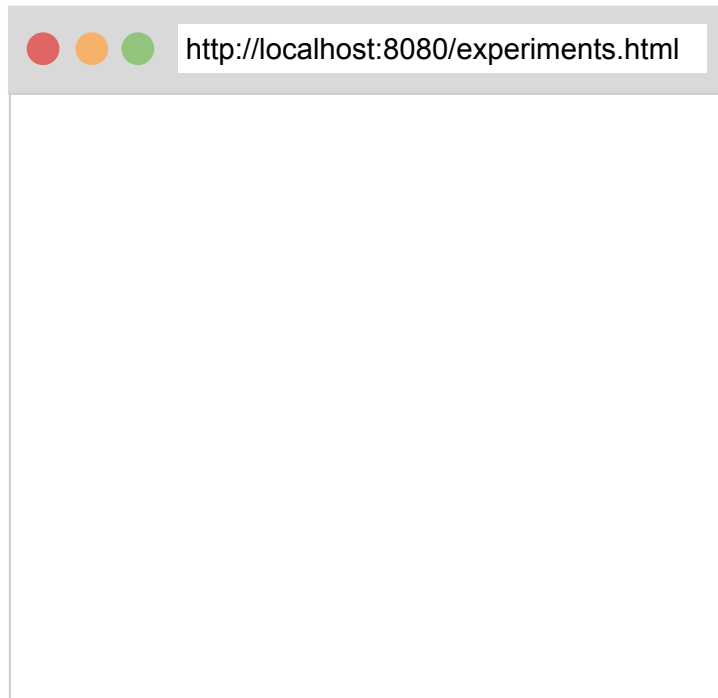
Browser

HTTP request made to localhost



Local computer / Server

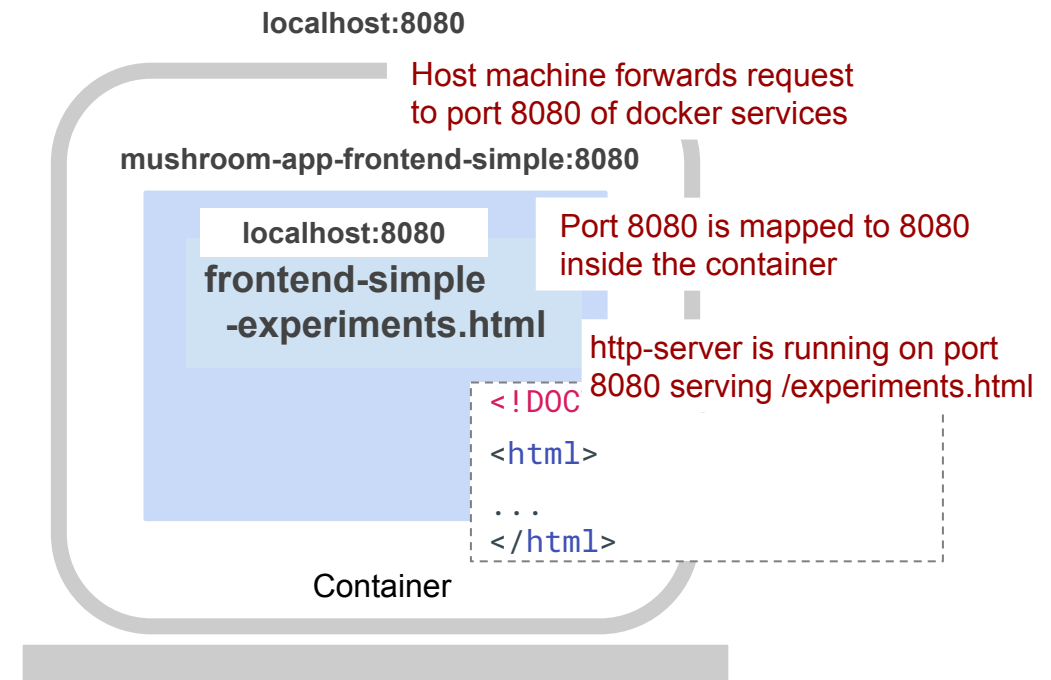
How does the App work



Browser

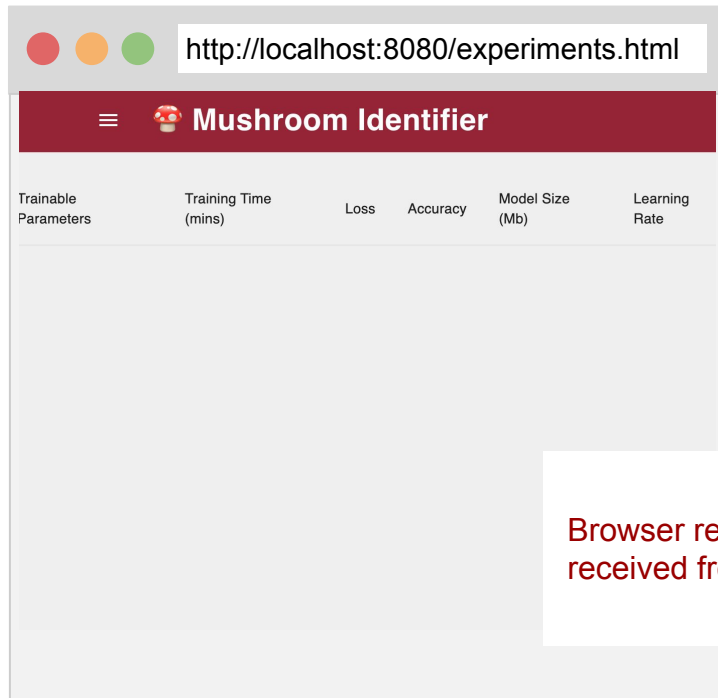
HTTP request made to localhost

`/experiments.html` was requested so the content of the `/experiments.html` will be sent back to browser. The HTML is sent back to the browser



Local computer / Server

How does the App work

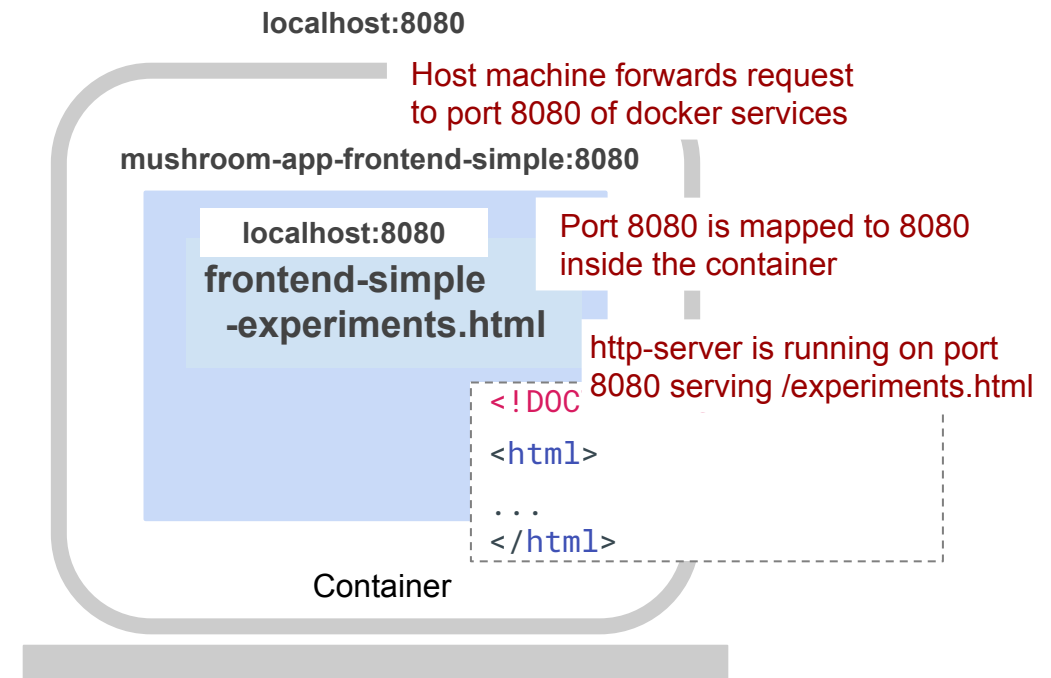


Browser

HTTP request made to localhost

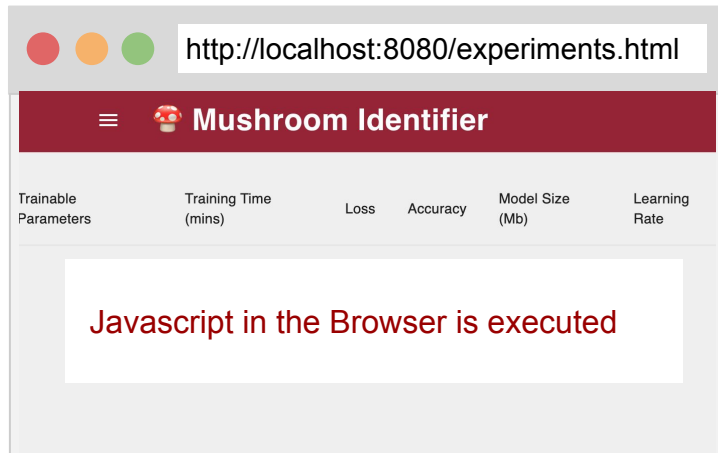
`/experiments.html` was requested so the content of the `/experiments.html` will be sent back to browser. The HTML is sent back to the browser

Browser renders the HTML content received from the server



Local computer / Server

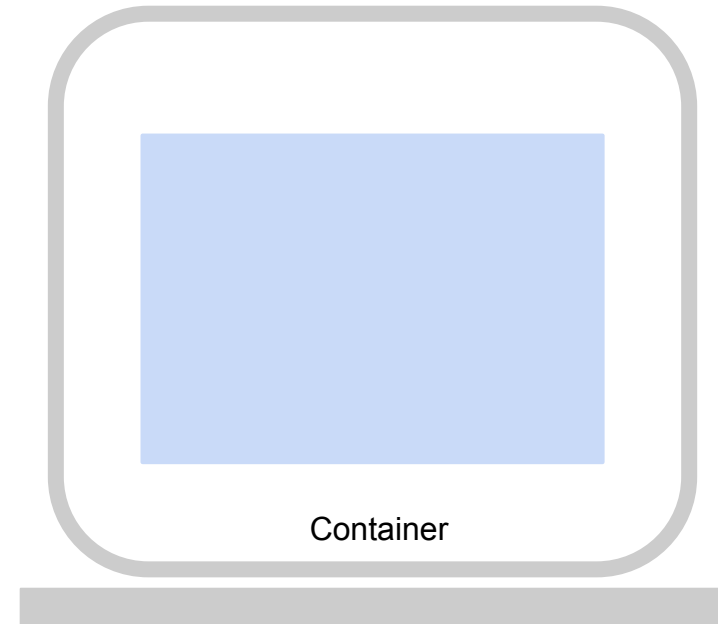
How does the App work



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

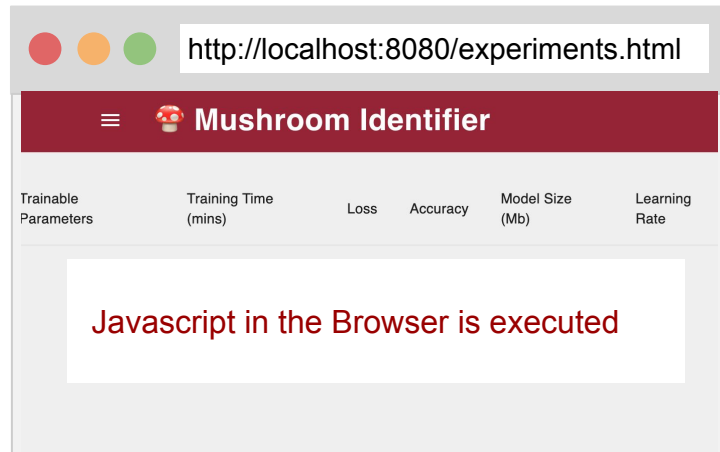
Browser

HTTP request made to
<http://localhost:9000/experiments>



Local computer / Server

How does the App work

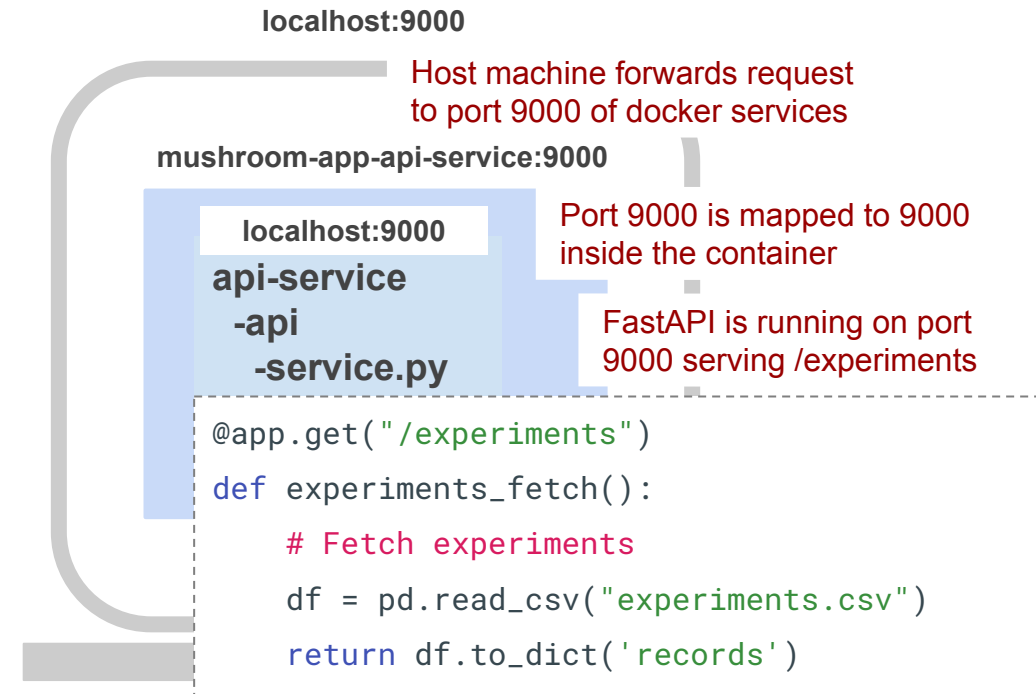


```
// API URL
axios.defaults.baseURL = 'http://localhost:9000/';
// Our experiments list
var experiments = [];
// Call the API
axios.get('/experiments')
  .then((response) => {
    experiments = response.data;
    // Build the table
    buildExperimentsTable(experiments);
  });
```

Browser

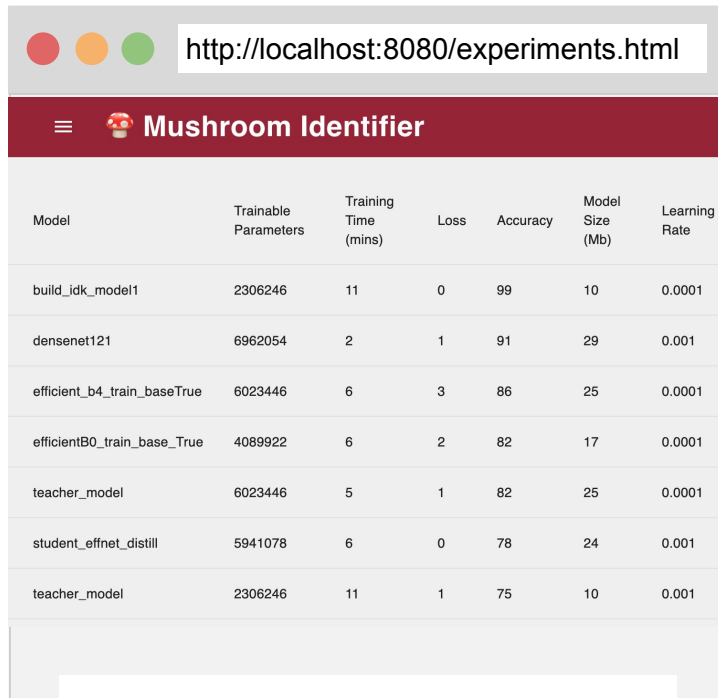
HTTP request made to `http://localhost:9000/experiments`

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



Local computer / Server

How does the App work



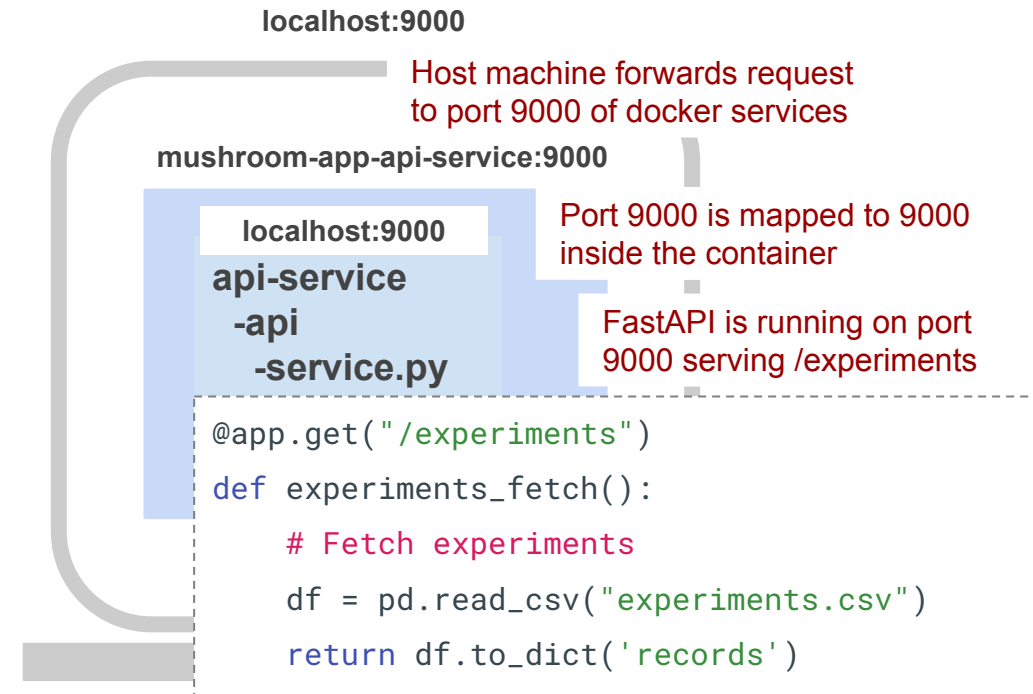
Model	Trainable Parameters	Training Time (mins)	Loss	Accuracy	Model Size (Mb)	Learning Rate
build_idk_model1	2306246	11	0	99	10	0.0001
densenet121	6962054	2	1	91	29	0.001
efficient_b4_train_baseTrue	6023446	6	3	86	25	0.0001
efficientB0_train_base_True	4089922	6	2	82	17	0.0001
teacher_model	6023446	5	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 experiments data in the html page.

Browser

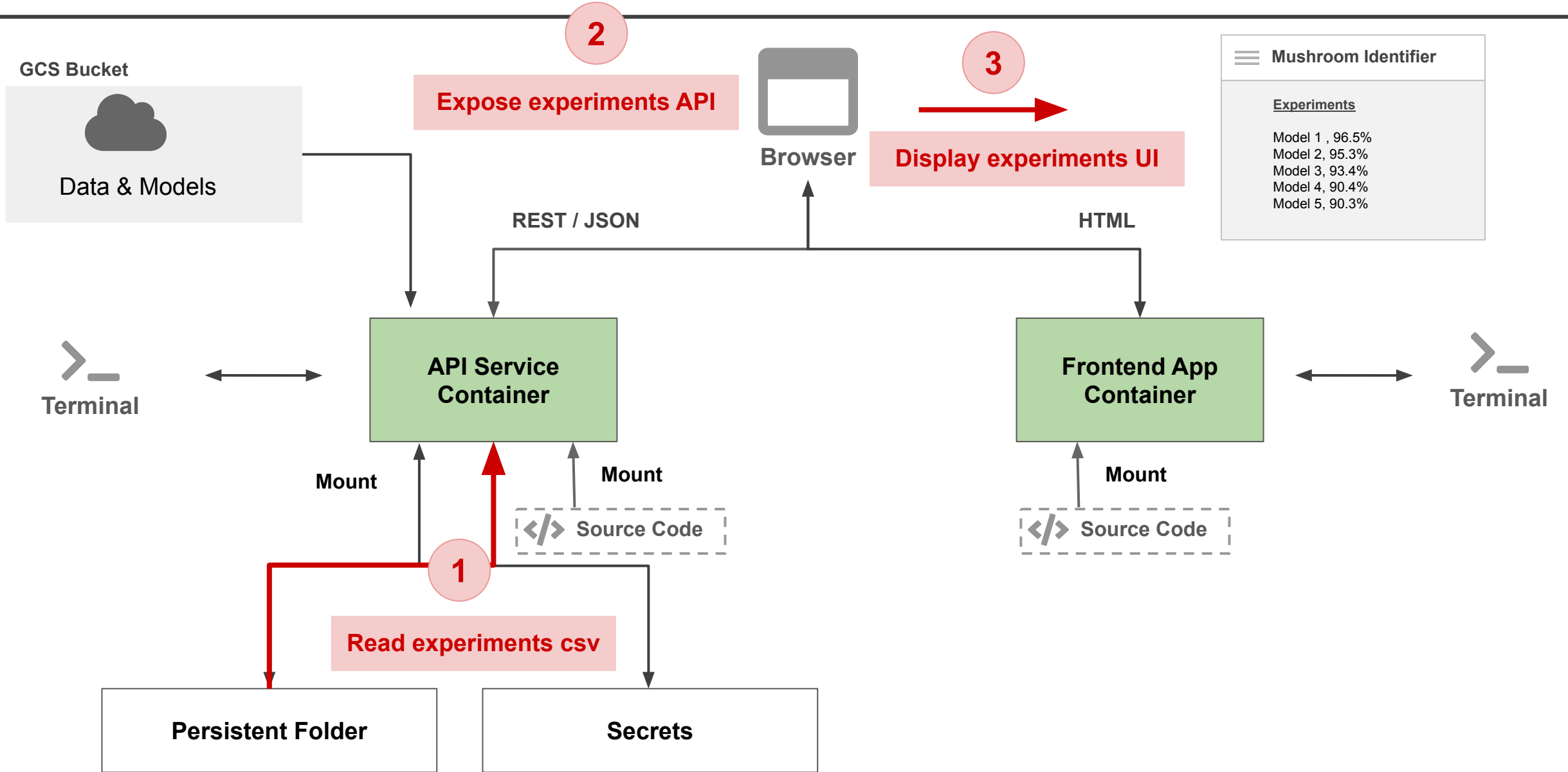
HTTP request made to
`http://localhost:9000/experiments`

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



Local computer / Server

Tutorial: Frontend Simple



Tutorial: Frontend Simple

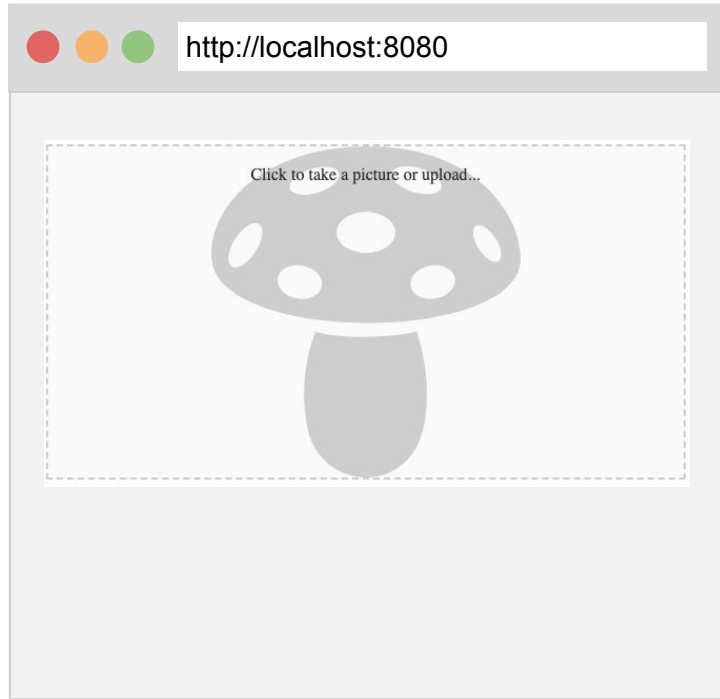
Steps to run Mushroom App **Frontend**:

- <https://github.com/dlops-io/mushroom-app-v2#frontend-app-simple>

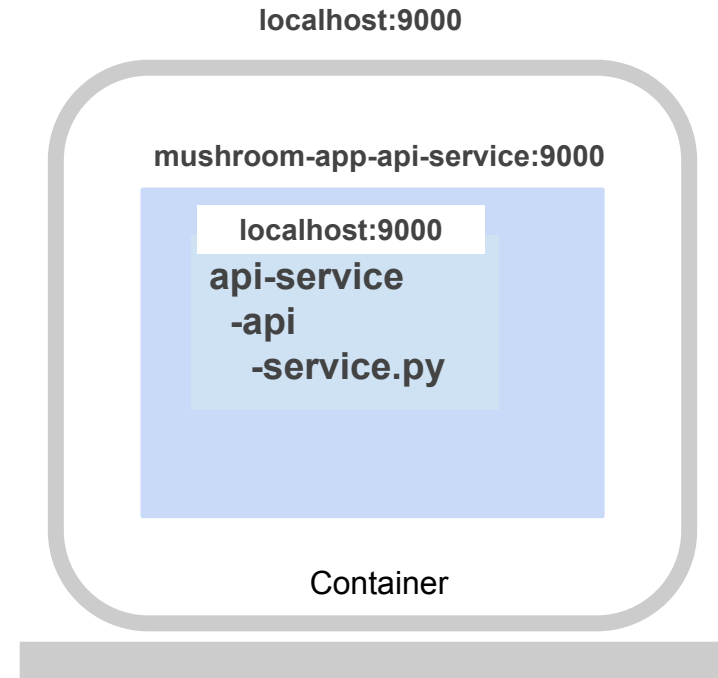
Outline

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

How does Model Serving work

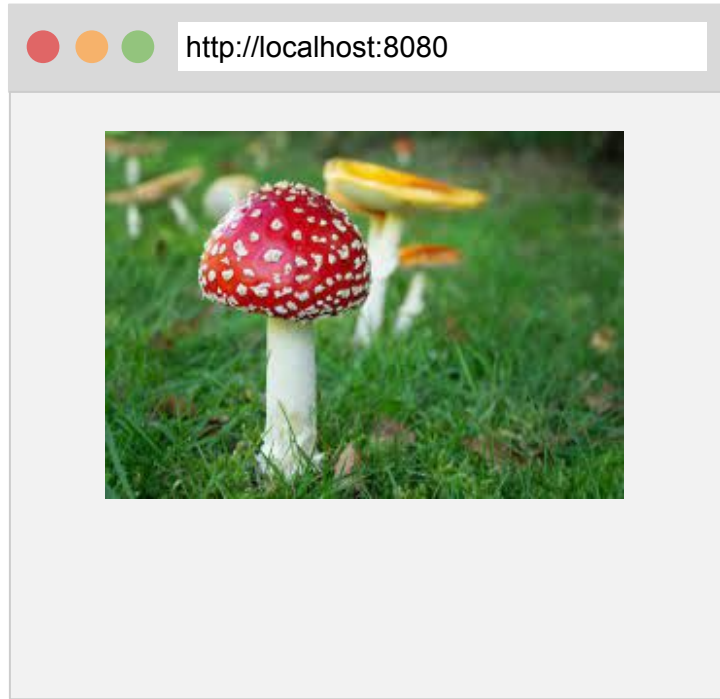


Browser



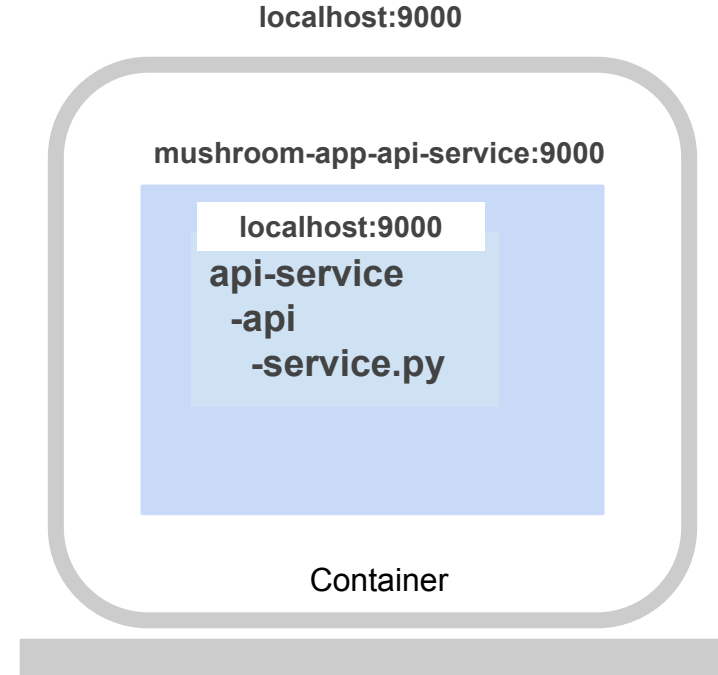
Local computer / Server

How does Model Serving work



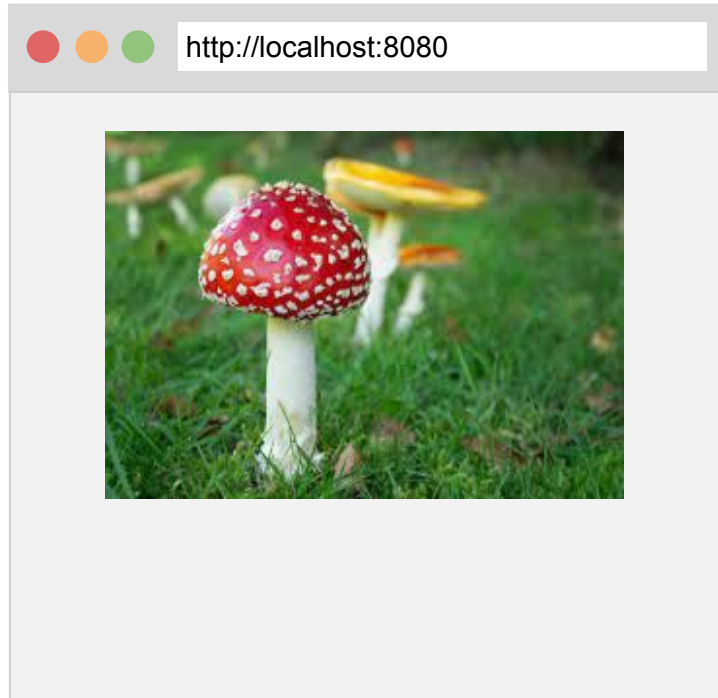
Browser

HTTP request made to
<http://localhost:9000/predict>



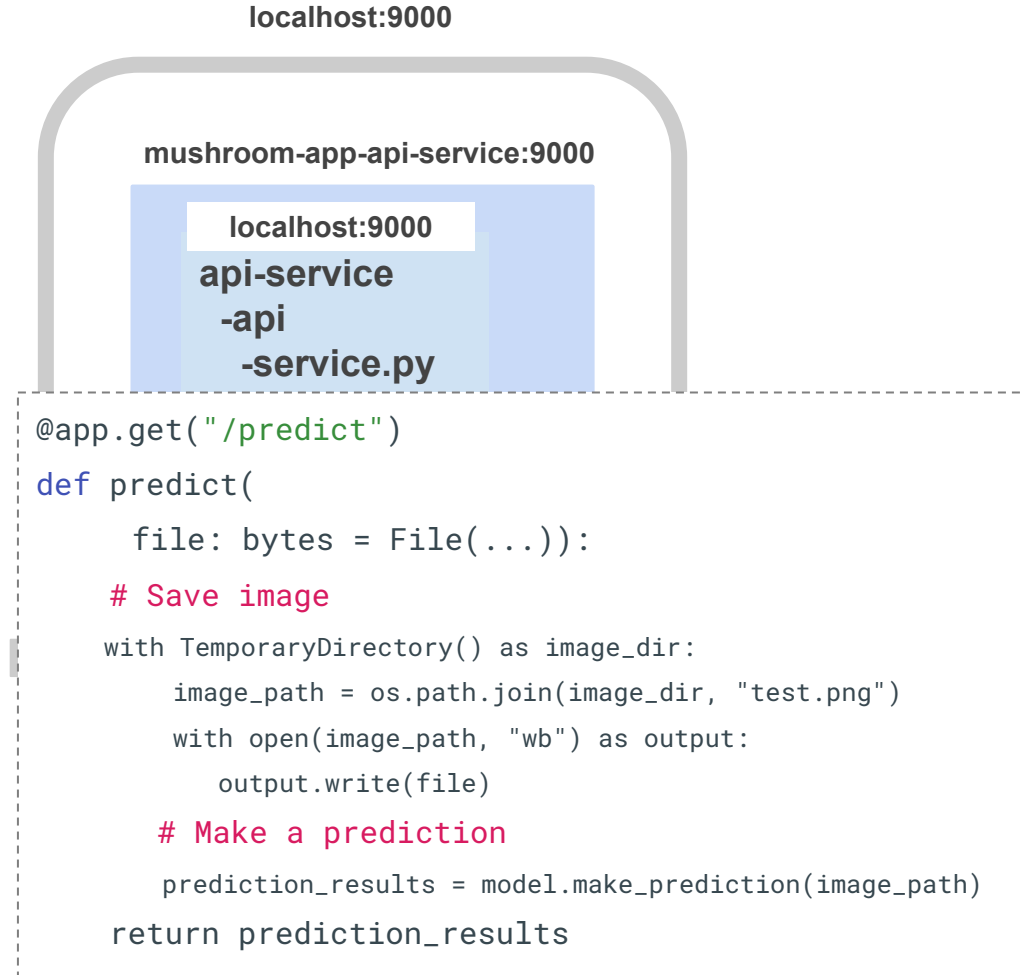
Local computer / Server

How does Model Serving work

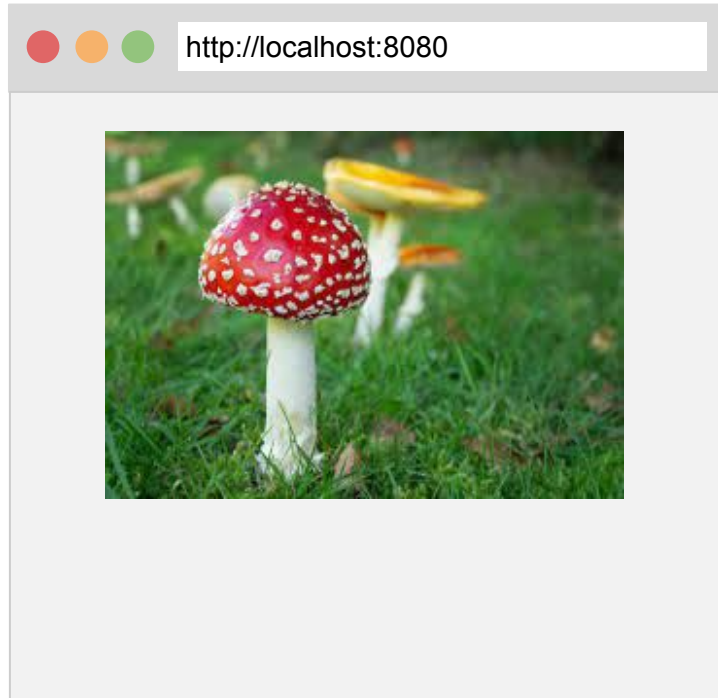


Browser

HTTP request made to
<http://localhost:9000/predict>



How does Model Serving work



Browser

HTTP request made to
`http://localhost:9000/predict`

`/predict` was requested so the
results of the `/predict` will be sent
back to browser

```
{
  "input_image_shape": "(None, 224, 224, 3)",
  "prediction_shape": [
    1,
    3
  ],
  "prediction_label": "amanita",
  "prediction": [
    0.0015741393435746431,
    0.001133422483690083,
    0.9972924590110779
  ]
},
"accuracy": 99.73,
"poisonous": true
```

localhost:9000

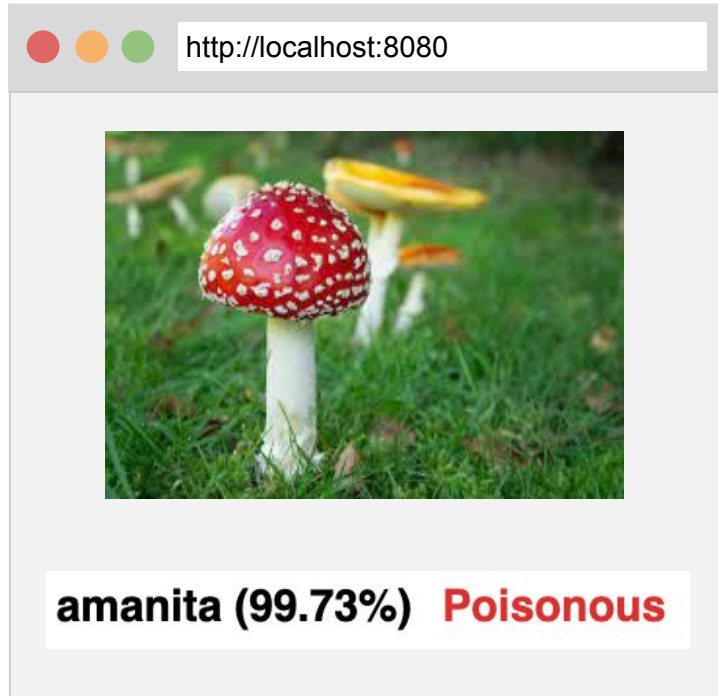
mushroom-app-api-service:9000

localhost:9000

api-service
-api
-service.py

```
@app.get("/predict")
def predict(
    file: bytes = File(...)):
    # Save image
    with TemporaryDirectory() as image_dir:
        image_path = os.path.join(image_dir, "test.png")
        with open(image_path, "wb") as output:
            output.write(file)
    # Make a prediction
    prediction_results = model.make_prediction(image_path)
    return prediction_results
```

How does Model Serving work



Browser

HTTP request made to
`http://localhost:9000/predict`

`/predict` was requested so the
results of the `/predict` will be sent
back to browser

```
{
  "input_image_shape": "(None, 224, 224, 3)",
  "prediction_shape": [
    1,
    3
  ],
  "prediction_label": "amanita",
  "prediction": [
    0.0015741393435746431,
    0.001133422483690083,
    0.9972924590110779
  ]
},
"accuracy": 99.73,
"poisonous": true
```

localhost:9000

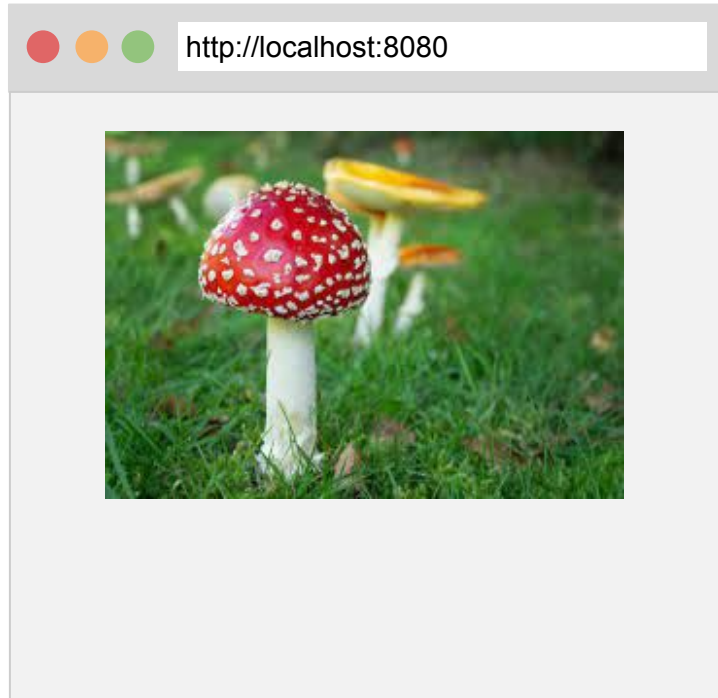
mushroom-app-api-service:9000

localhost:9000

api-service
-api
-service.py

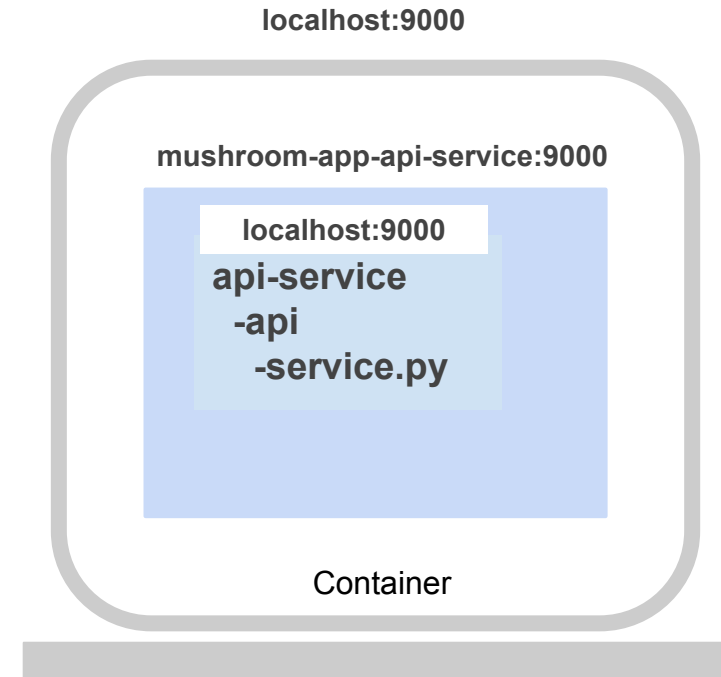
```
@app.get("/predict")
def predict(
    file: bytes = File(...)):
    # Save image
    with TemporaryDirectory() as image_dir:
        image_path = os.path.join(image_dir, "test.png")
        with open(image_path, "wb") as output:
            output.write(file)
    # Make a prediction
    prediction_results = model.make_prediction(image_path)
    return prediction_results
```

How does Model Serving work - Vertex AI



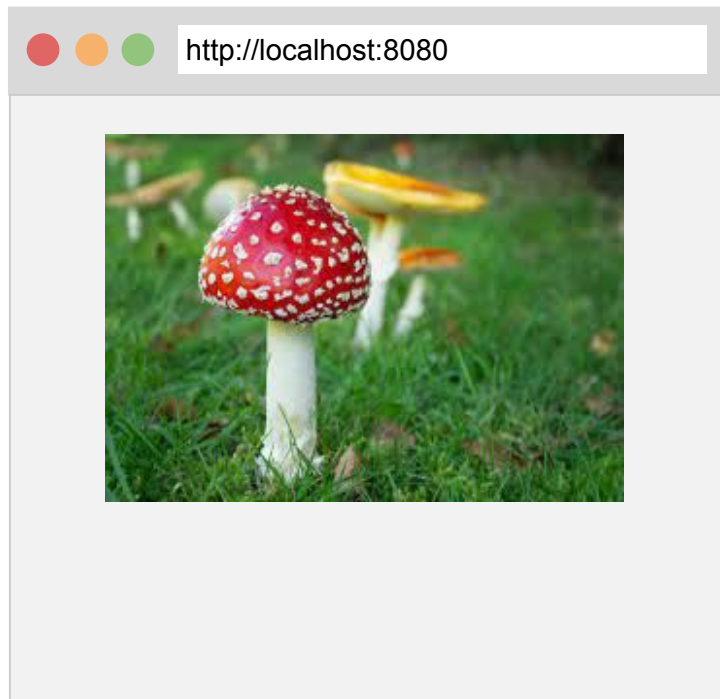
Browser

HTTP request made to
<http://localhost:9000/predict>



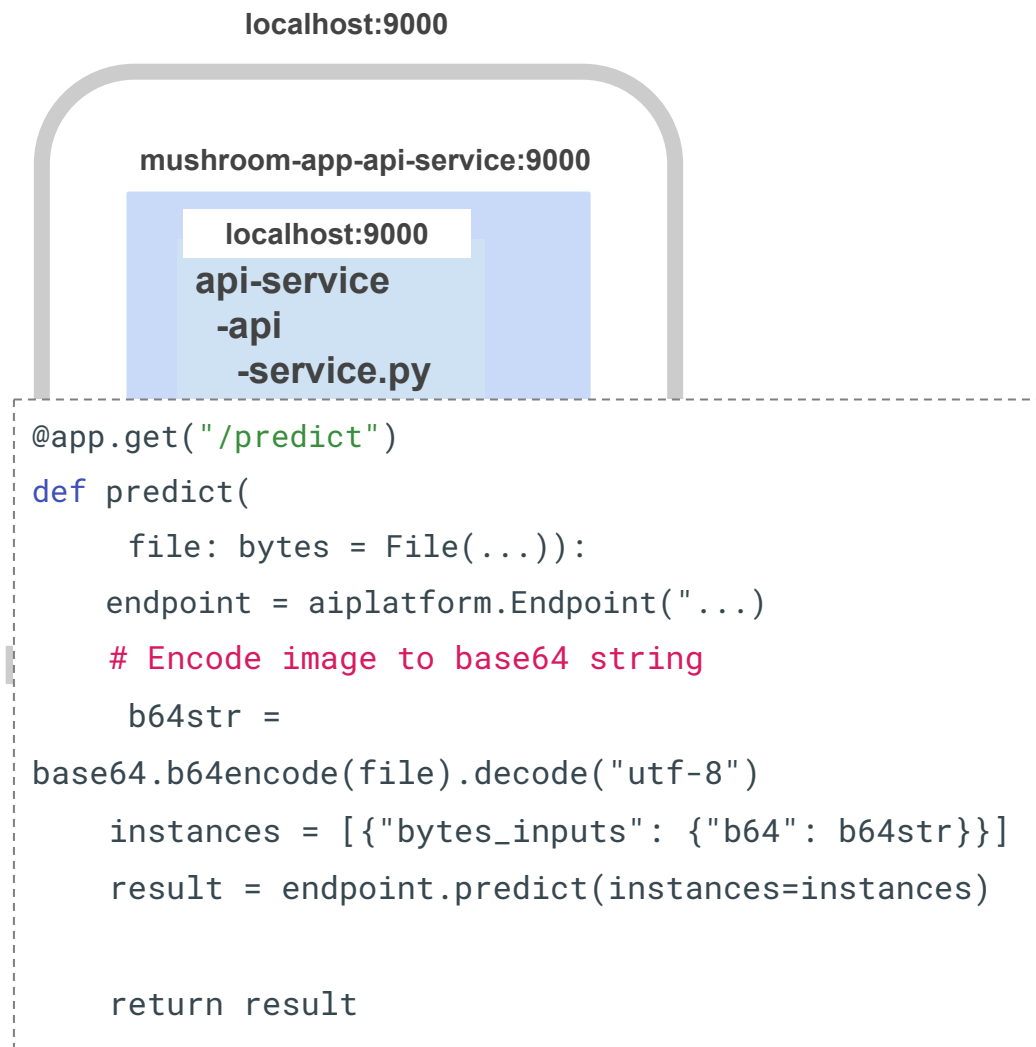
Local computer / Server

How does Model Serving work - Vertex AI

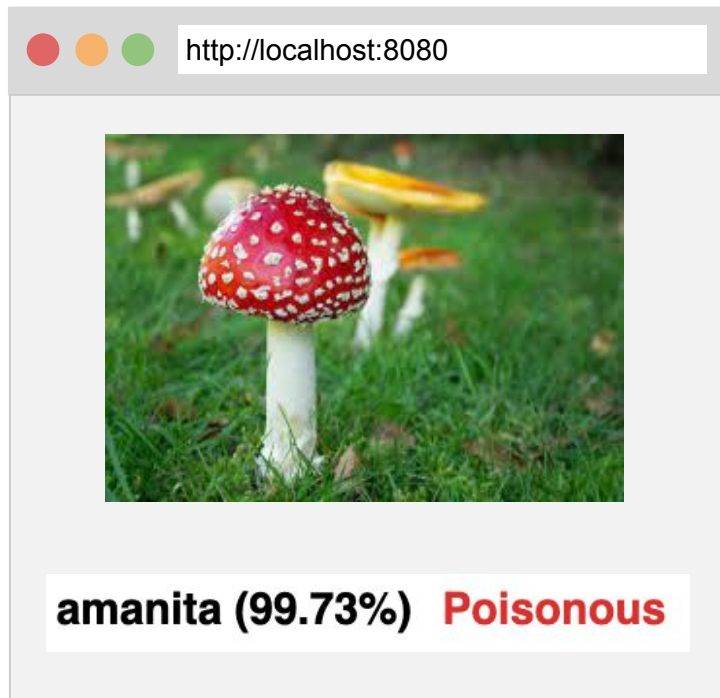


Browser

HTTP request made to
`http://localhost:9000/predict`



How does Model Serving work - Vertex AI



Browser

HTTP request made to
`http://localhost:9000/predict`



`/predict` was requested so the
results of the `/predict` will be sent
back to browser

```
{
  "input_image_shape": "(None, 224, 224, 3)",
  "prediction_shape": [
    1,
    3
  ],
  "prediction_label": "amanita",
  "prediction": [
    0.0015741393435746431,
    0.001133422483690083,
    0.9972924590110779
  ]
},
"accuracy": 99.73,
"poisonous": true
```

localhost:9000

mushroom-app-api-service:9000

localhost:9000

api-service
-api
-service.py

```
@app.get("/predict")
def predict(
    file: bytes = File(...)):
    endpoint = aiplatform.Endpoint(...)
    # Encode image to base64 string
    b64str =
base64.b64encode(file).decode("utf-8")
    instances = [{"bytes_inputs": {"b64": b64str}}]
    result = endpoint.predict(instances=instances)

    return result
```

Tutorial: Model Serving

Steps to run Mushroom App **Model Serving**:

- <https://github.com/dlops-io/mushroom-app-v2#model-serving>

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
- experiments.html
- model.html

Frontend

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

- index.html
- experiments.html
- model.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

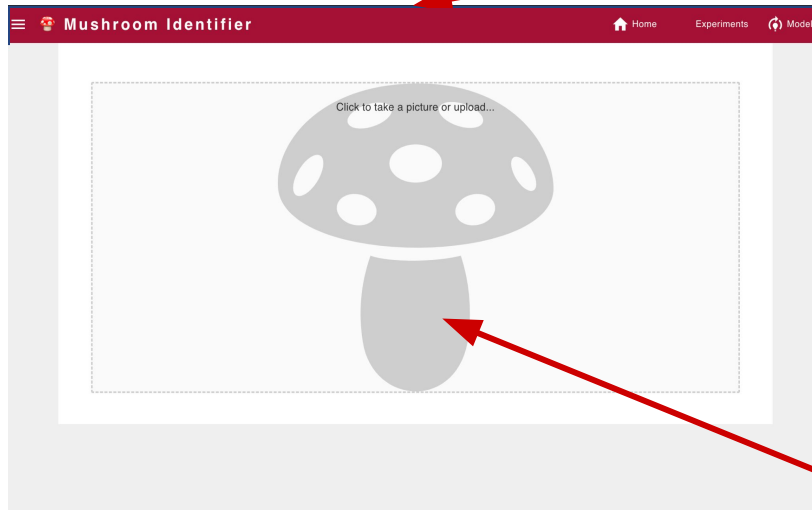
Header

Content

Footer

React App

Header defined only once



	Model	Trainable Parameters	Training Time (mins)	Loss	Accuracy	Model Size (Mb)	Learning Rate	Batch Size	Epochs	Optimizer
1	tthub_mobilenetv2_train_base_True	2,306,051	2.51	42.7	92.73%	9.60	0.001	32	10	SGD
2	mobilenetv2_train_base_True	2,388,227	3.74	70.6	92.73%	9.91	0.001	32	15	SGD
3	tthub_mobilenetv2_train_base_False	82,179	2.20	42.5	89.70%	9.60	0.001	32	10	SGD
4	mobilenetv2_train_base_False	164,355	3.67	70.4	89.70%	9.91	0.001	32	15	SGD
5	tthub_mobilenetv2_train_base_True	2,306,051	2.77	42.7	89.09%	9.60	0.001	32	10	SGD
6	mobilenetv2_train_base_True	2,388,227	3.93	70.8	88.48%	9.91	0.001	32	15	SGD
7	mobilenetv2_train_base_True	2,388,227	4.09	70.9	87.88%	9.91	0.001	32	15	SGD
8	tthub_mobilenetv2_train_base_False	82,179	2.48	42.8	87.88%	9.60	0.001	32	10	SGD

Current Model Details

Name	tthub_mobilenetv2_train_base_True
Trainable Parameters	2,306,051
Training Time (mins)	2.51
Loss	42.7
Accuracy	92.73%
Model Size (mb)	9.60
Learning Rate	0.001
Batch Size	32
Epochs	10

Content block switched for each page

Tutorial: React Frontend

Steps to run Mushroom App **React Frontend**:

- <https://github.com/dlops-io/mushroom-app-v2#frontend-app-react>

THANK YOU