

**Let's talk about  
Node.js**

---

# Hello!

## Nils Mehlhorn

---

freelance software engineer



[nils-mehlhorn.de](http://nils-mehlhorn.de)



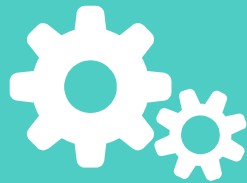
[@n\\_mehlhorn](https://twitter.com/n_mehlhorn)



[@n\\_mehlhorn](https://www.instagram.com/n_mehlhorn)



**What's  
Node.js?**



**Views &  
APIs**



**Real  
World**



# What's Node.js

Let's look at the  
basics

# Node.js is ...

---



... a JavaScript runtime

... based on Chrome's V8 engine

... designed for scalable network apps

... asynchronous event-driven ?

... using non-blocking IO ??

IO = Access to  
slow stuff

- network
- file system
- database

# Blocking IO

---

```
const fs = require('fs')

const options = {encoding: 'utf-8'}
// block
const content = fs.readFileSync('./file.txt', options)
console.log(content)
console.log('(other things your app does)')
```

Synchronous Call  
(JavaScript is single-threaded)

# Non-Blocking IO

---

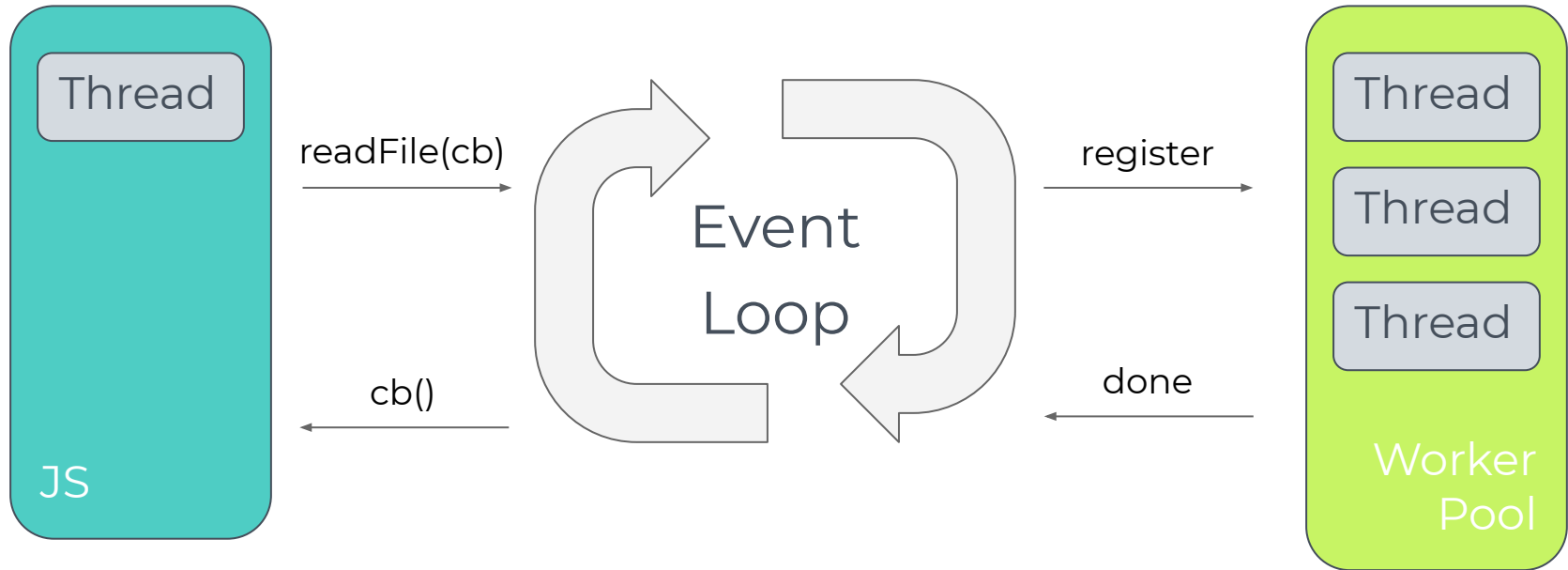
```
const fs = require('fs')

const options = {encoding: 'utf-8'}
// no block, but async
fs.readFile('./file.txt', options, (err, content) => {
  console.log(content)
})
console.log('(other things your app does)')
```

Asynchronous Callback



# Node.js Event Loop



more: [What the heck is the event loop anyway?](#)

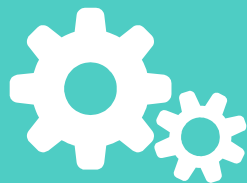
# Promises

- no “callback hell”
- less error-prone
- chainable

```
const fs = require('fs')
const options = {encoding: 'utf-8'}

function getJsonFile(path, callback) {
  fs.readFile(path, options, (err, content) => {
    if (err) {
      callback(err)
    }
    const json = JSON.parse(content)
    callback(null, json)
  })
}

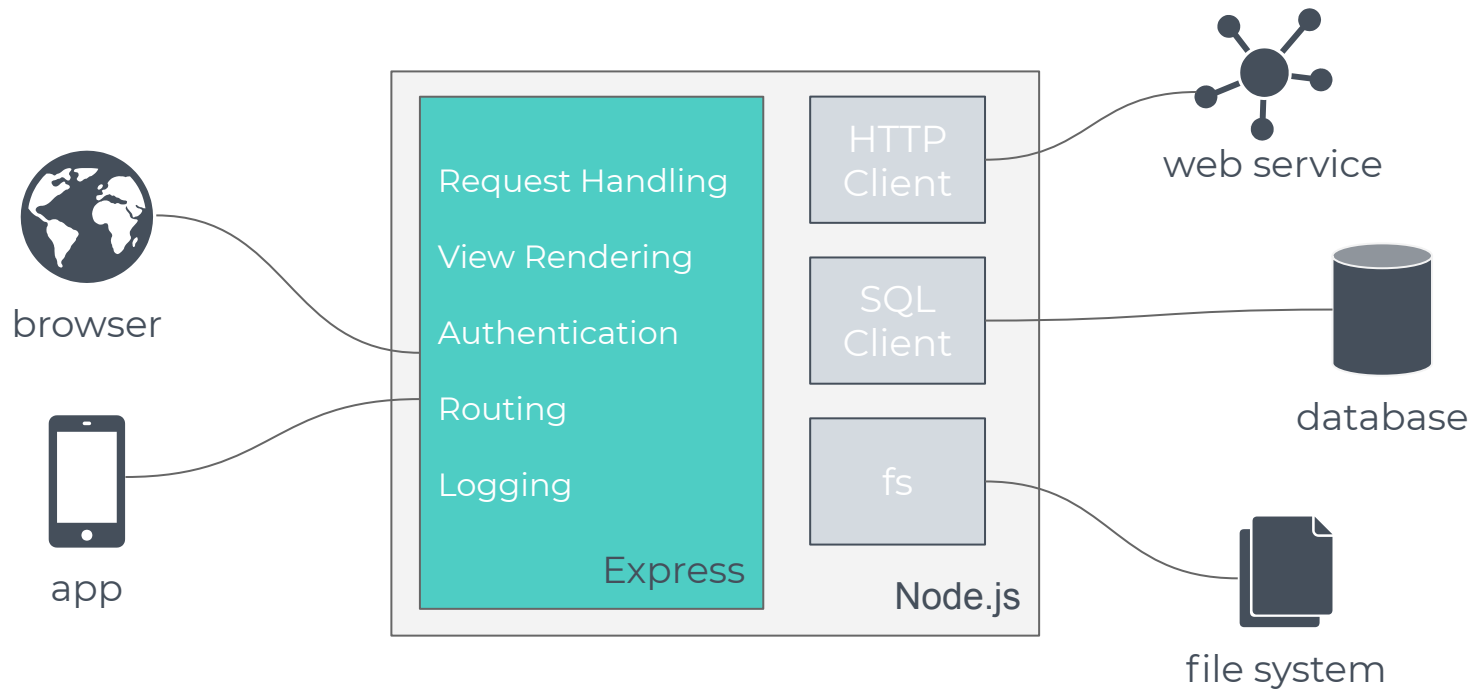
function getJsonFile(path) {
  return fs.promises.readFile(path, options)
    .then(content => JSON.parse(content))
}
```



# Views & APIs

Let's build for  
the web

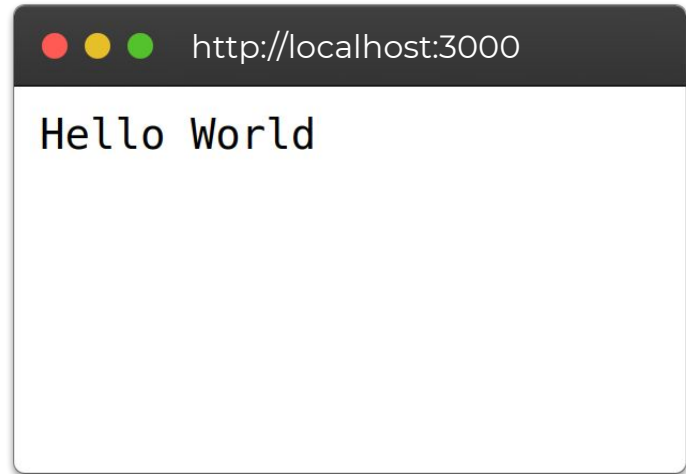
# Express



```
const express = require('express')
const app = express()

app.get('/', (req, res) => {
  res.send('Hello World')
})

const port = 3000
app.listen(port, () => {
  console.log('Server started')
})
```



---

Hello World with Express

# Views

```
app.set('views', './views')
app.set('view engine', 'ejs')

app.get('/greet', (req, res) => {
  const data = {
    title: 'iJS',
    person: req.query.person
  }
  res.render('greet', data)
})
```

greet.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Hey there, <%= person %>!</p>
  </body>
</html>
```

/greet?person=Delilah

iJS

Hey there, Delilah!

## Output

```
<title><%= title %></title>
```

## Control Flow

```
<ul>  
  <% users.forEach(user => { %>  
    <li><%= user.name %></li>  
  <% }) %>  
</ul>
```

## Includes

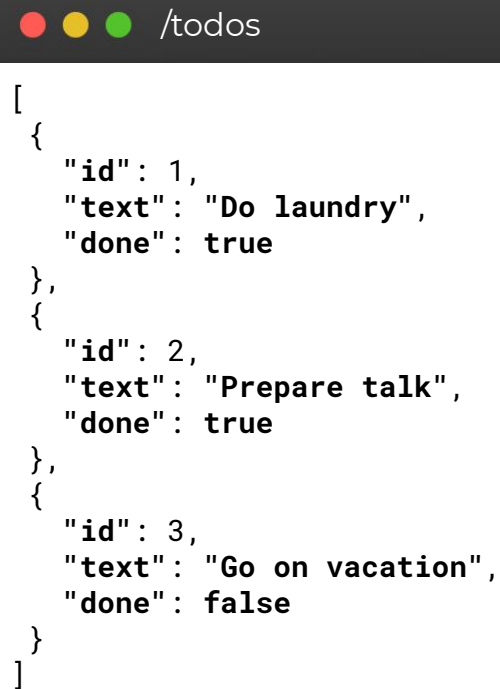
```
<html>  
<body>  
  
  <% include(../header, {title}) %>  
  
  <main>Hello</main>  
  
  <% include ../footer %>  
  
</body>  
</html>
```

---

Embedded JavaScript Templates

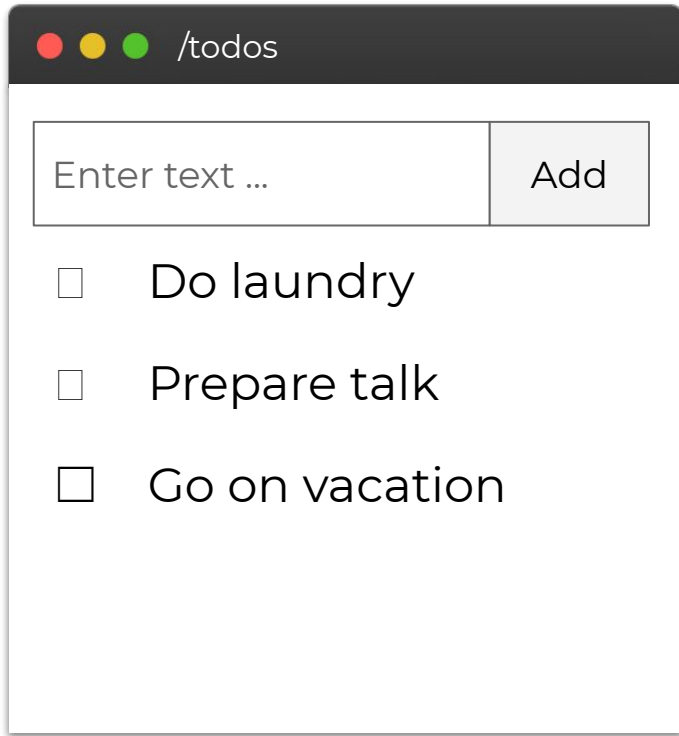
# APIs

```
let todos = [  
  {id: 1, text: "Do laundry", done: true},  
  {id: 2, text: "Prepare talk", done: true},  
  {id: 3, text: "Go on vacation", done: false}  
]  
  
app.get('/todos', (req, res) => {  
  res.json(notes)  
}))
```



```
[  
  {  
    "id": 1,  
    "text": "Do laundry",  
    "done": true  
  },  
  {  
    "id": 2,  
    "text": "Prepare talk",  
    "done": true  
  },  
  {  
    "id": 3,  
    "text": "Go on vacation",  
    "done": false  
  }  
]
```





/todos

Add

- ☐ Do laundry
- ☐ Prepare talk
- ☐ Go on vacation


---

Back-End API

🔴 🟡 🟢 /todos

Add

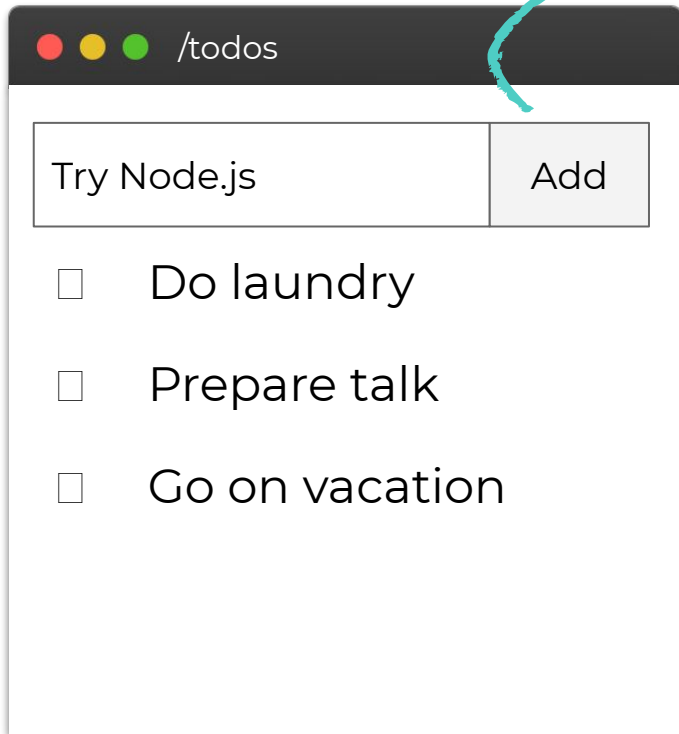
- ☐ Do laundry
- ☐ Prepare talk
- ☐ Go on vacation



PATCH /todos/3

```
let todos = [...]  
  
app.patch('/todos/:id', (req, res) => {  
  const id = req.params.id  
  const todo = todos  
    .find(todo => todo.id === id)  
  todo.done = !todo.done  
  res.sendStatus(200)  
})
```

Back-End API



Try Node.js Add

- ☐ Do laundry
- ☐ Prepare talk
- ☐ Go on vacation

POST /todos  
{ "text": "Try Node.js" }

```
let todos = [...]  
  
// parse JSON from HTTP body  
app.use(express.json())  
  
app.post('/todos', (req, res) => {  
  const todo = req.body  
  todo.id = getNextId()  
  todo.done = false  
  todos.push(todo)  
  res.json(todo)  
})
```

Back-End API

# Middleware

---

- ▣ Parsing
- ▣ Routing
- ▣ Static Files
- ▣ Error Handling
- ▣ Authentication
- ▣ Logging

```
const express = require('express')
const path = require('path')

// serve static files from ./public
app.use(express.static(
  path.join(__dirname, 'public')
))

/* request handlers */

// handle unmapped routes
app.use((req, res) => {
  res.status(404).send('Not Found')
})
```

order



# Project Structure

```
// install generator
npm -g install express-generator

// generate project
express --view=ejs my-app
```

Configures:

- Error handling
- View rendering
- Static files
- Cookie parsing
- Logging



# package.json

---

```
{  
  "name": "ijs-express",  
  "version": "0.0.1",  
  "scripts": {  
    "start": "node index.js",  
    "watch": "nodemon index.js",  
    "start-ts": "ts-node index.ts"  
  },  
  "dependencies": {...},  
  "devDependencies": {  
    "nodemon": "^1.19.4",  
    "ts-node": "^8.4.1"  
  }  
}
```



# Real World

Node.js is a tool

# Opportunities

---

npm is  
biggest registry

deployment options

## **Ecosystem**

tool support

community

no context switch

hiring

## **Full Stack JS**

onboarding

code sharing

multitude of  
requests

less concurrency  
concerns

## **IO Performance**

connect  
downstream  
services



# Challenges

---

## Security

audits

package

corruption

best practices

code injection

linting

## Maintainability

architecture

NestJS

no typing

TypeScript

testing

Jest

## CPU Performance

worker threads

# Use Cases

---



## Web

- Server-side pages
- APIs



## Data-intensive

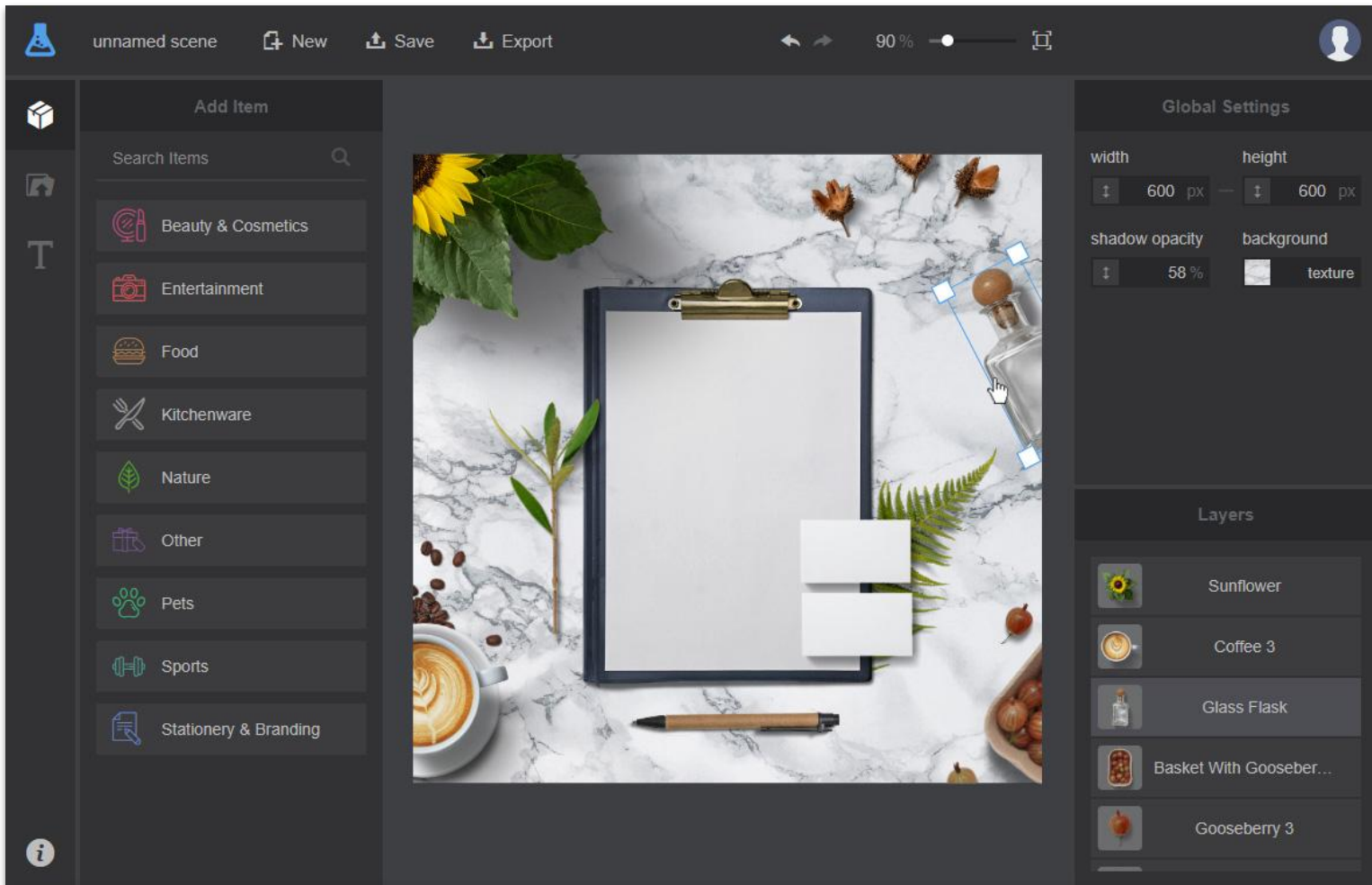
- Internet of Things
- Streaming



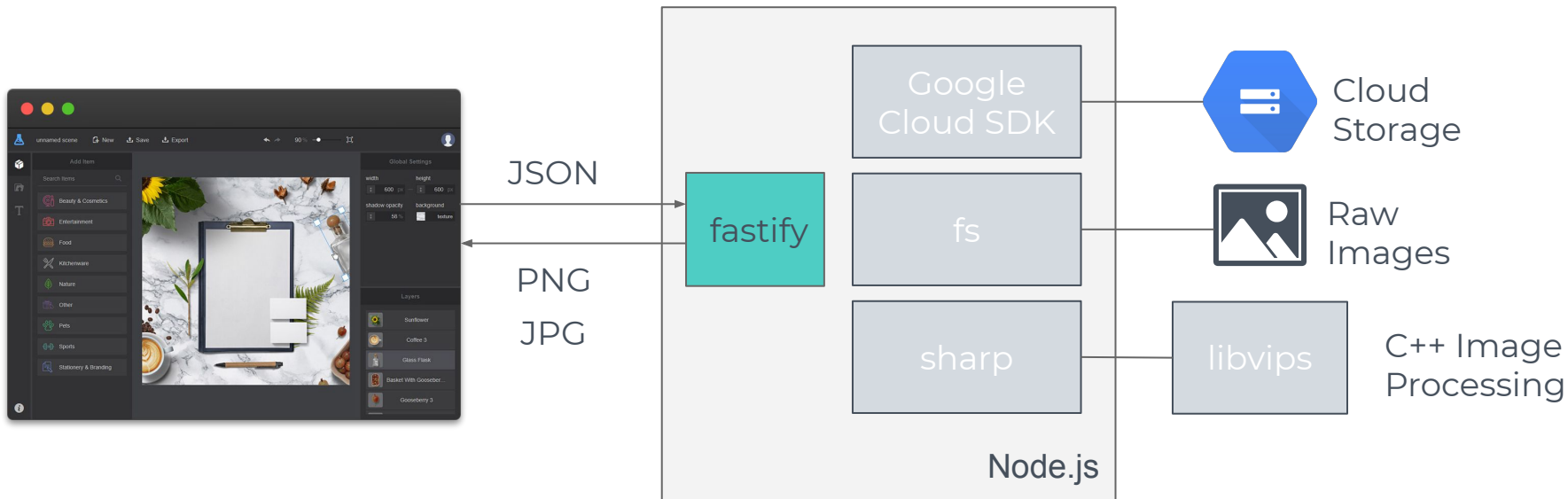
## Realtime

- Chat
- Gaming

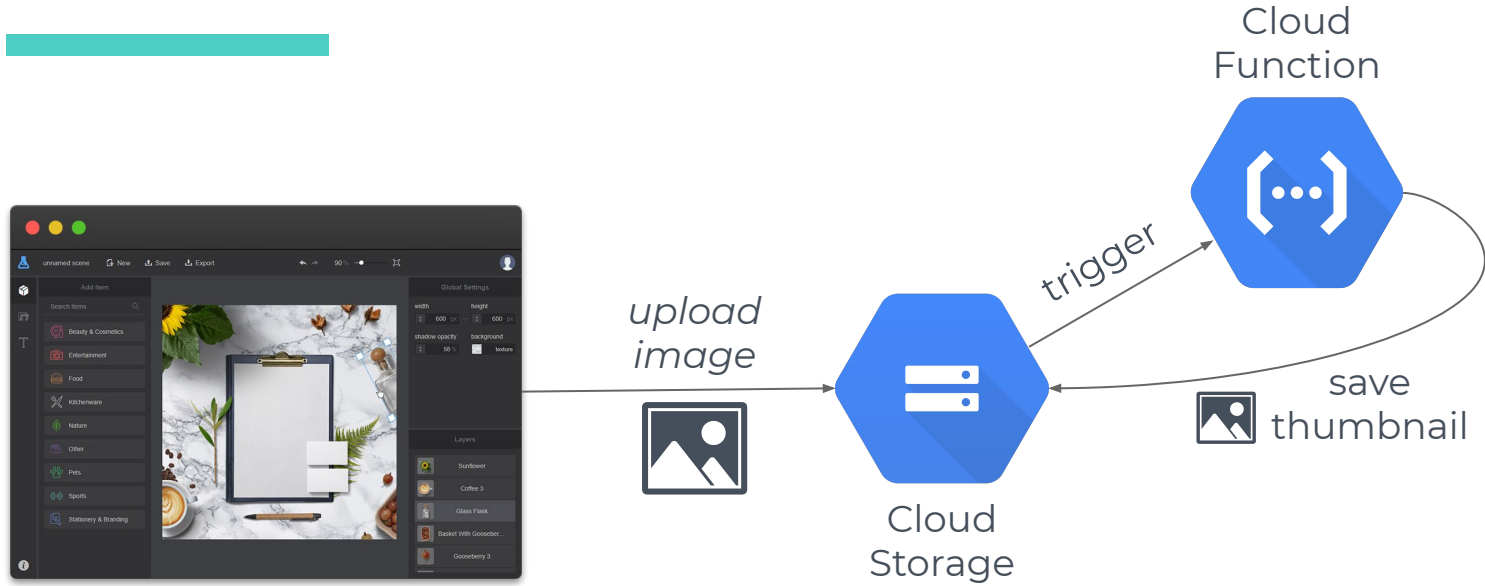
***You're most effective with what you have - until you're not***



# Server-side Image Rendering



# Thumbnail Generation



# Thanks!

## Nils Mehlhorn

freelance software engineer



nils-mehlhorn.de



@n\_mehlhorn



@n\_mehlhorn

