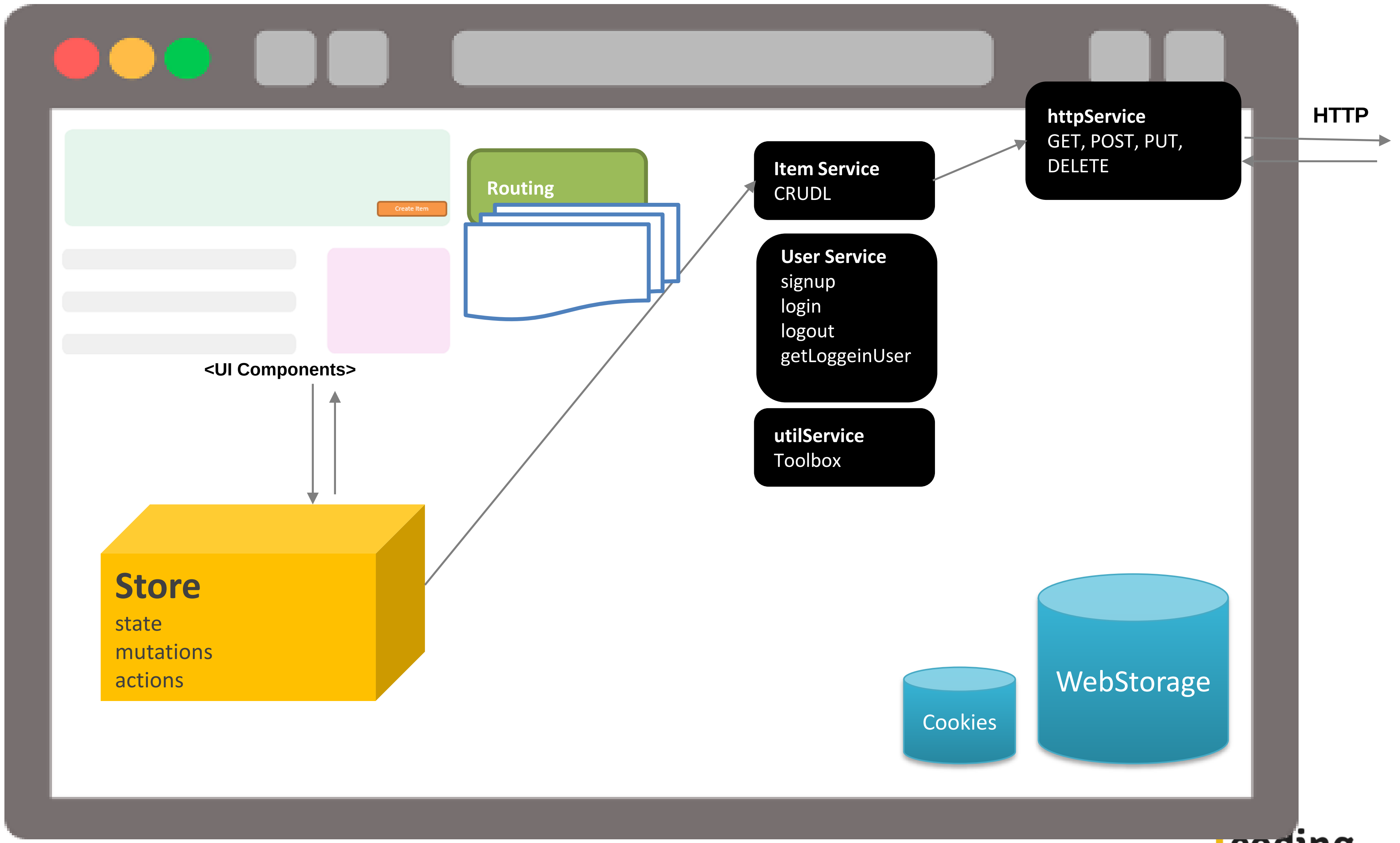
Node.js

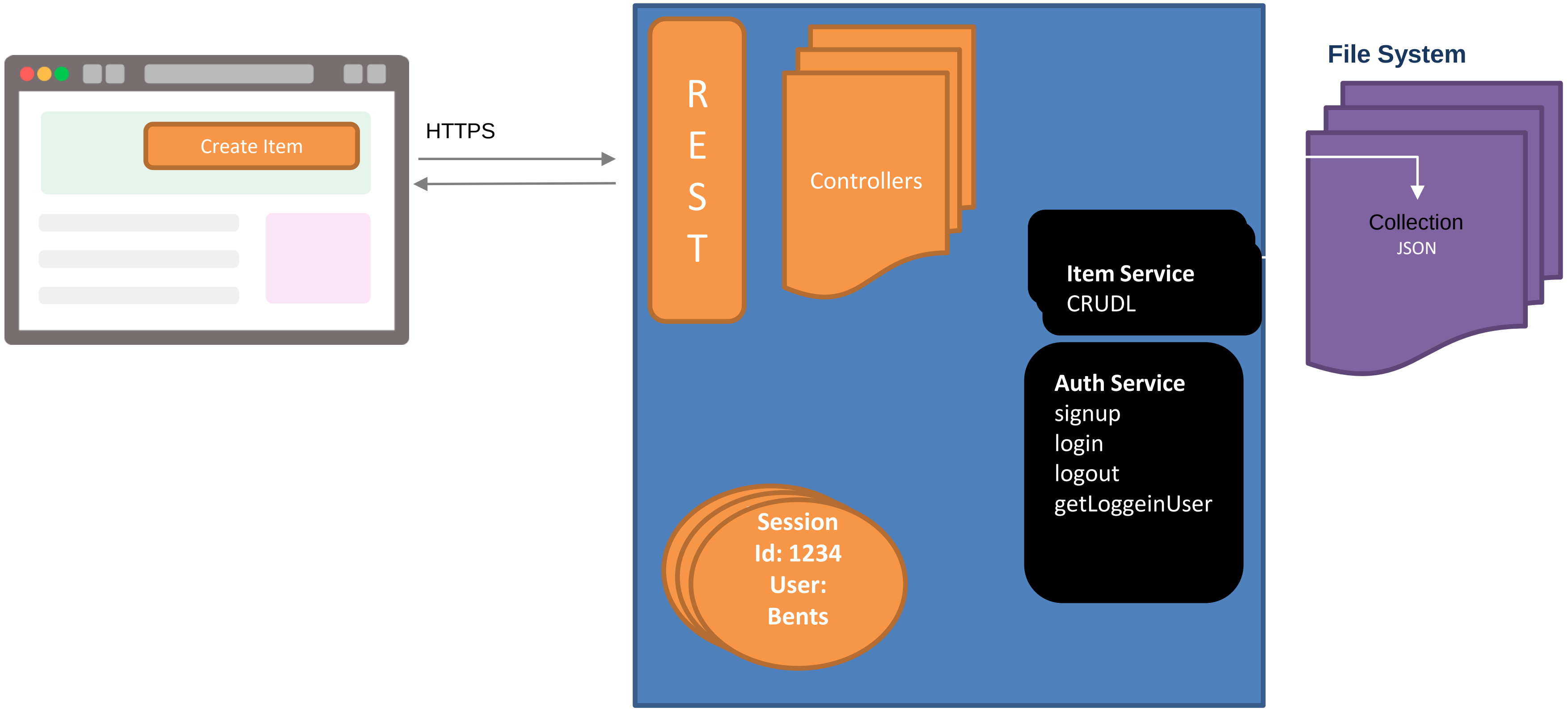
First, lets revise our current architecture and tools

Blocks Diagram – Frontend



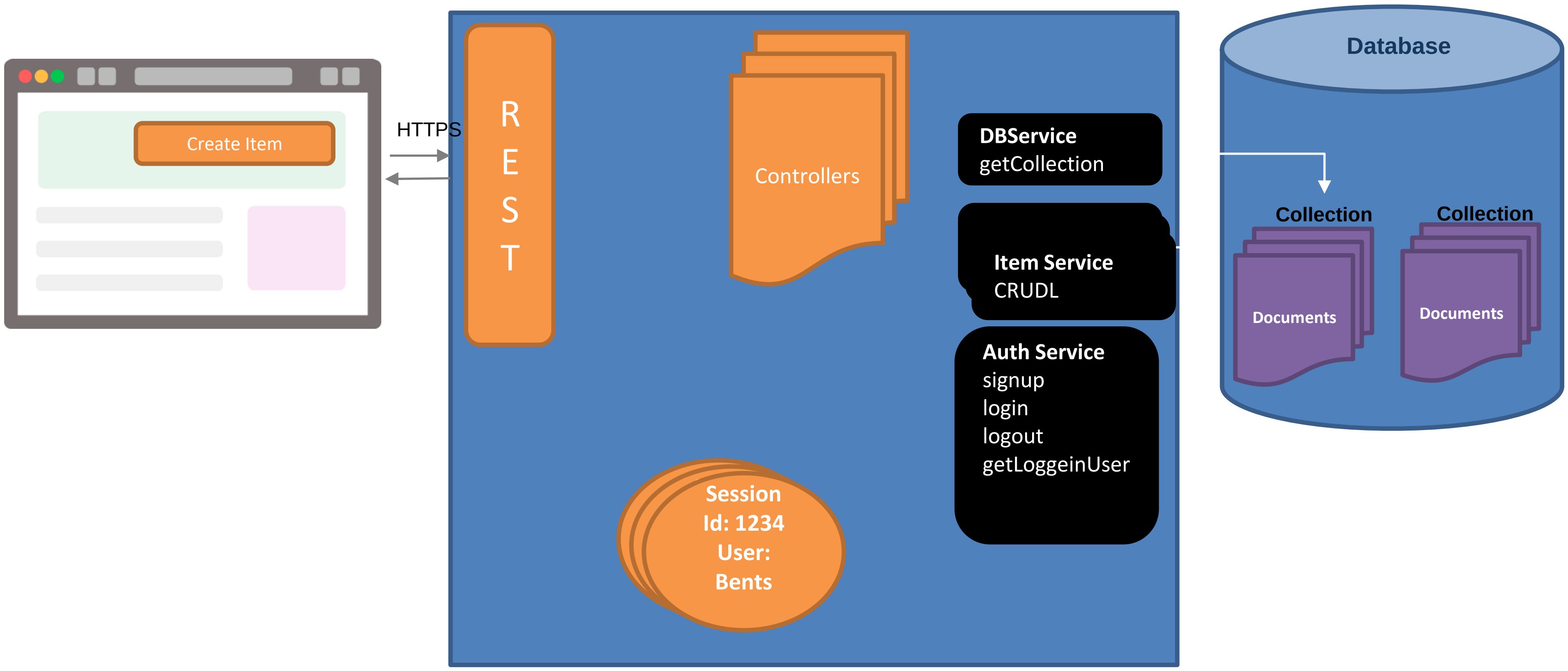
coding_ academy

Blocks Diagram – Backend With Files



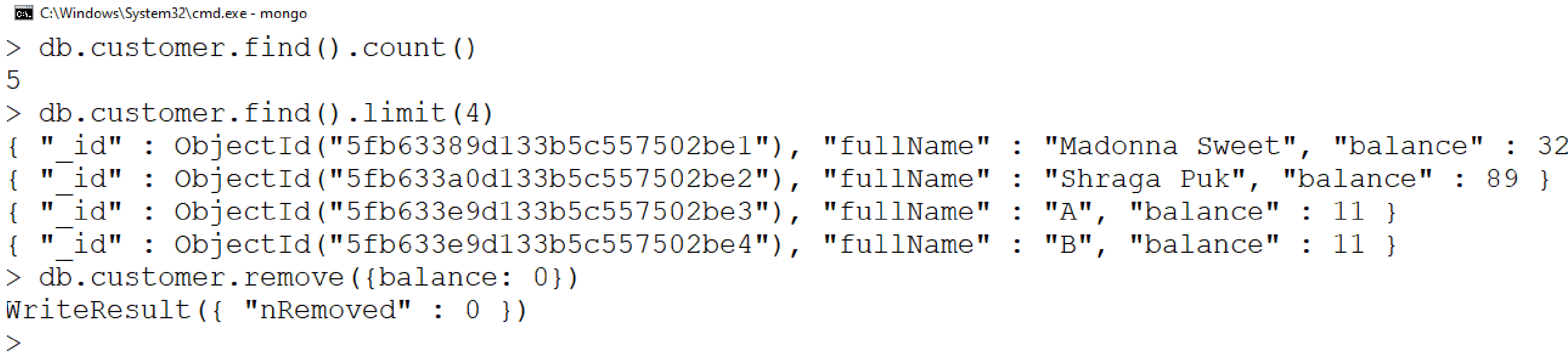
coding_ academy

Blocks Diagram – Backend With MongoDB



mongoDB

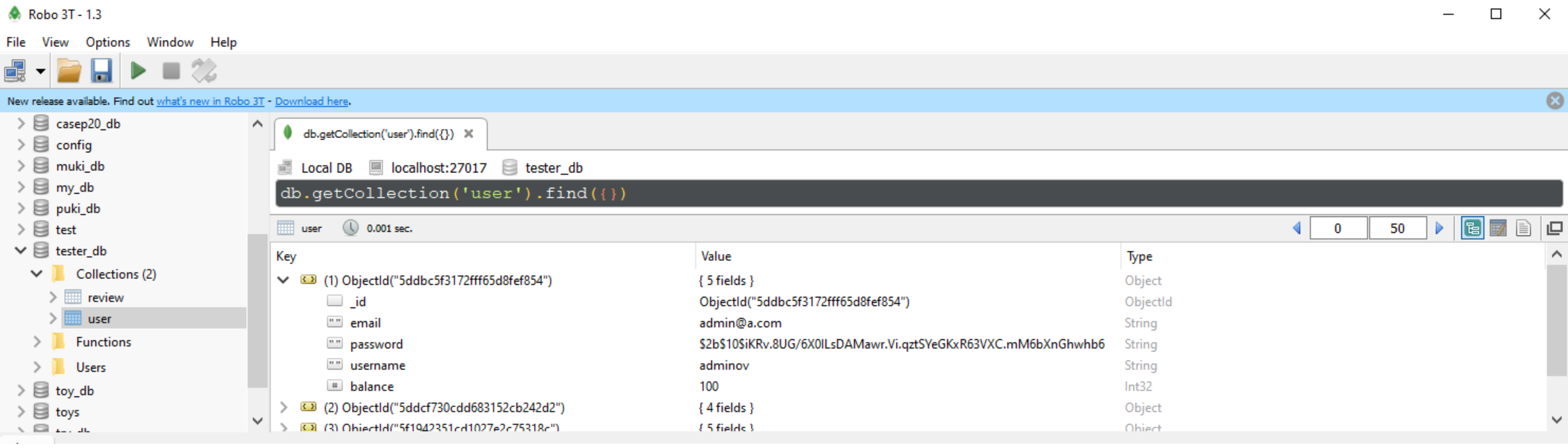
Accessing mongoDB



C:\Windows\System32\cmd.exe - mongo

```
> db.customer.find().count()
5
> db.customer.find().limit(4)
{ "_id" : ObjectId("5fb63389d133b5c557502be1"), "fullName" : "Madonna Sweet", "balance" : 32
{ "_id" : ObjectId("5fb633a0d133b5c557502be2"), "fullName" : "Shraga Puk", "balance" : 89 }
{ "_id" : ObjectId("5fb633e9d133b5c557502be3"), "fullName" : "A", "balance" : 11 }
{ "_id" : ObjectId("5fb633e9d133b5c557502be4"), "fullName" : "B", "balance" : 11 }
> db.customer.remove({balance: 0})
WriteResult({ "nRemoved" : 0 })
>
```

Accessing mongoDB



New release available. Find out what's new in Robo 3T - Download here.

db.getCollection('user').find({})

Local DB localhost:27017 tester_db

```
db.getCollection('user').find({})
```

user 0.001 sec. 0 50

Key	Value	Type
(1) ObjectId("5ddbc5f3172fff65d8fef854")	{ 5 fields }	Object
_id	ObjectId("5ddbc5f3172fff65d8fef854")	ObjectId
email	admin@a.com	String
password	$2b$10$iKRv.8UG/6X0ILsDAMawr.Vi.qztSYeGKxR63VXC.mM6bXnGhwhb6	String
username	adminov	String
balance	100	Int32
(2) ObjectId("5ddcf730cdd683152cb242d2")	{ 4 fields }	Object
(3) ObjectId("5f1942351cd1027e2c75318c")	{ 5 fields }	Object

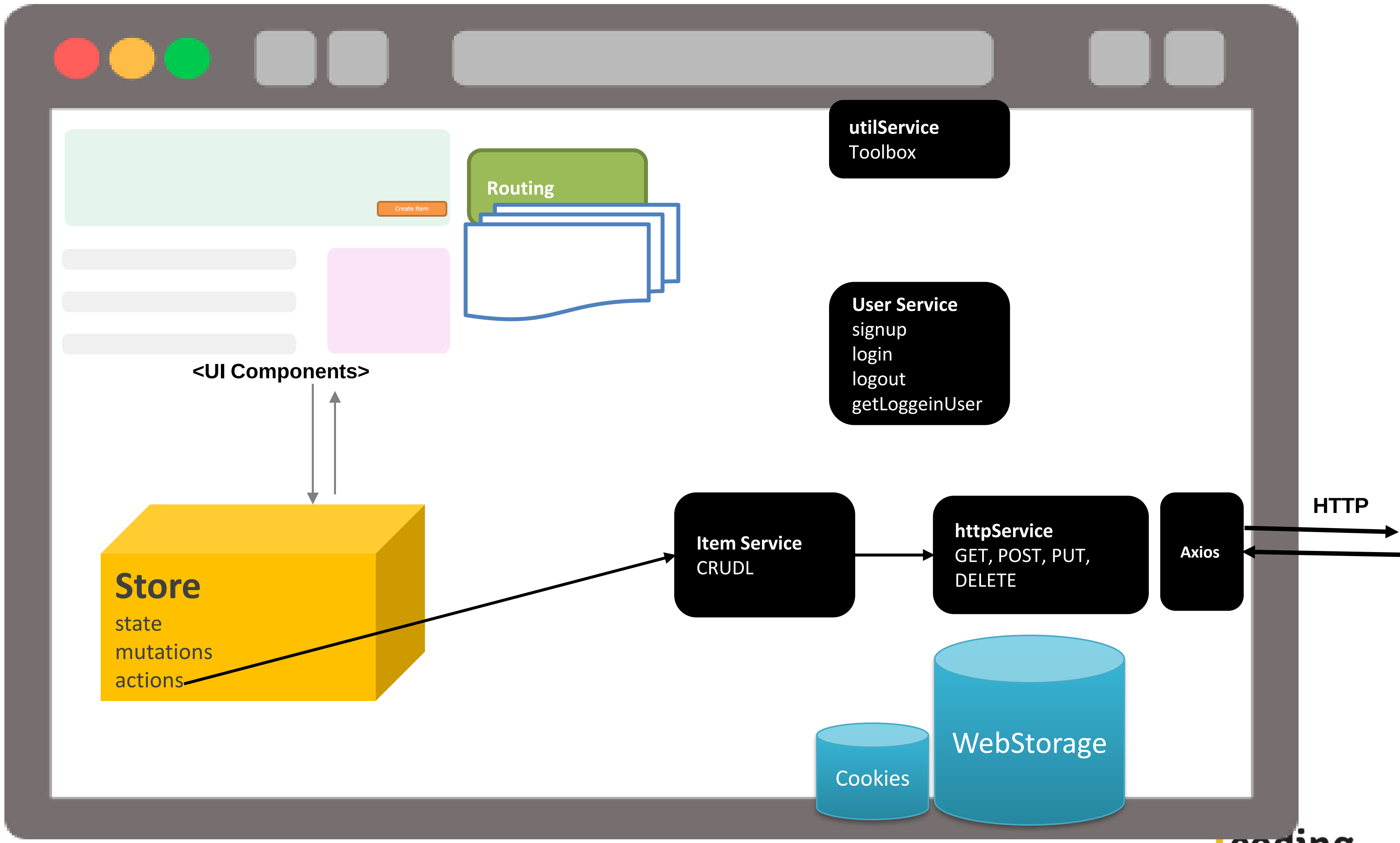
coding_ academy

NodeJS and MongoDB

```
const MongoClient = require('mongodb').MongoClient
const url = 'mongodb://localhost:27017'
const dbName = 'tester_db'
'

    MongoClient.connect(url, (err, client) => {
        if (err) return console.log('Cannot connect to DB')
        console.log("Connected successfully to server")
        const db = client.db(dbName)
        const collection = db.collection('customer')
        // Find some documents
        collection.find({ balance: { $gte: 10 } }).toArray((err, docs) => {
            if (err) return console.log('Cannot find customers')
            console.log("Found the following records")
            console.log(docs)
        })
        client.close()
    })
```

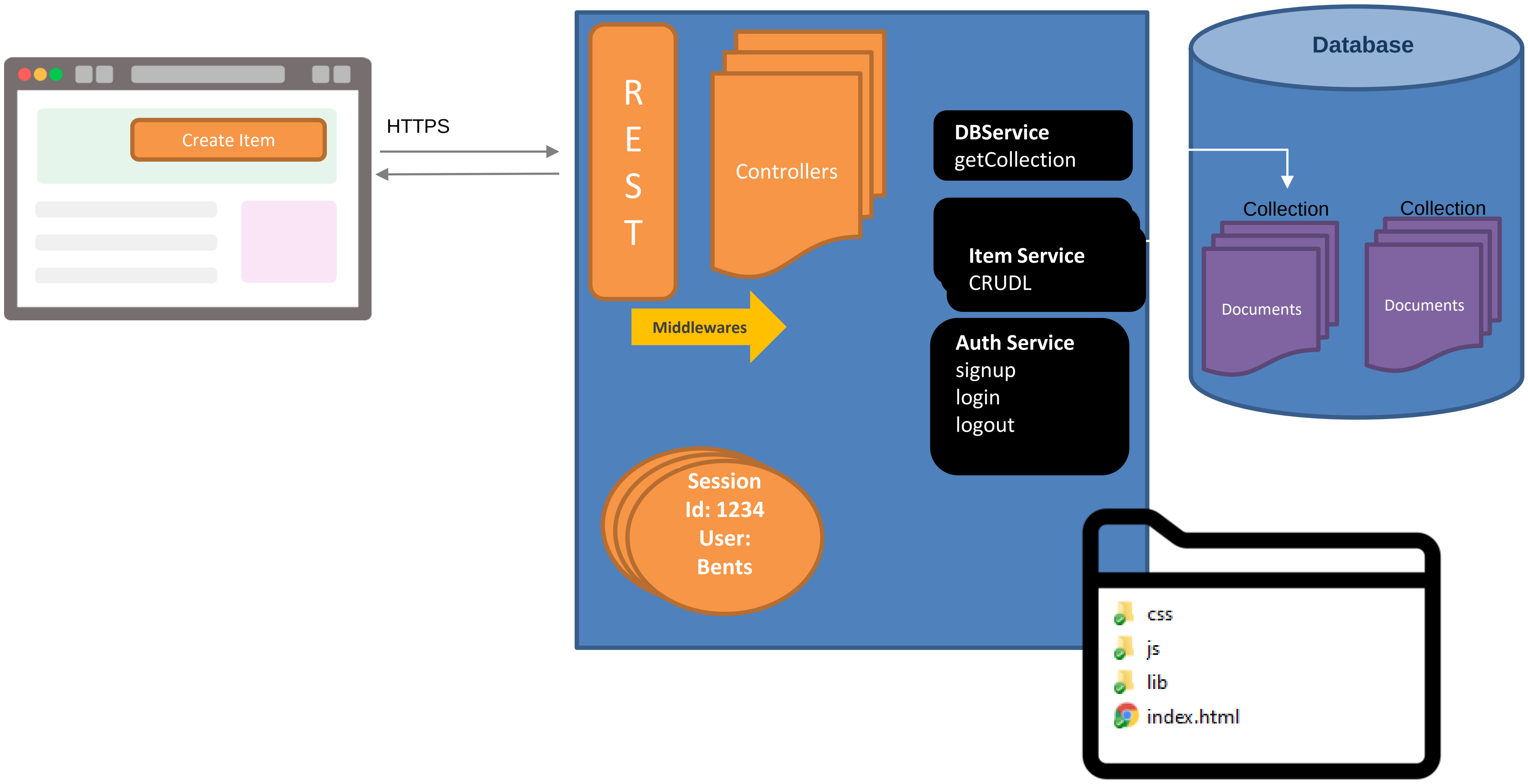
Production Ready Node setup

Blocks Diagram – Frontend



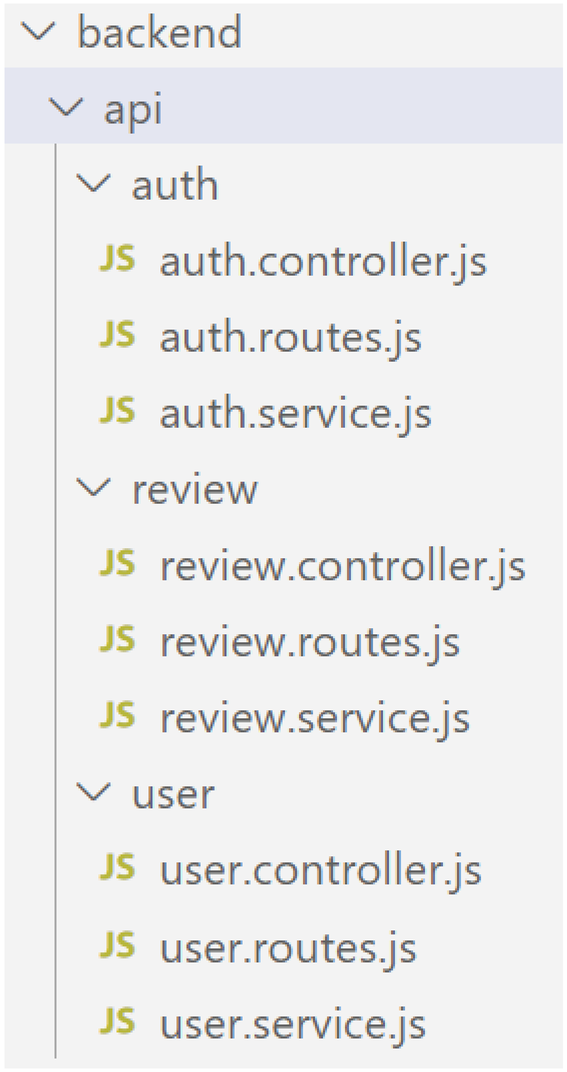
coding_ academy

Blocks Diagram – Backend



Route Splitting

```
// routes
app.use('/auth', authRoutes)
app.use('/review', reviewRoutes)
app.use('/user', userRoutes)
```



```
backend
  api
    auth
      auth.controller.js
      auth.routes.js
      auth.service.js
    review
      review.controller.js
      review.routes.js
      review.service.js
    user
      user.controller.js
      user.routes.js
      user.service.js
```

```
// Last fallback
app.get('*', (req, res) => {
   res.sendFile(path.resolve(__dirname, 'public', 'index.html'));
});
```

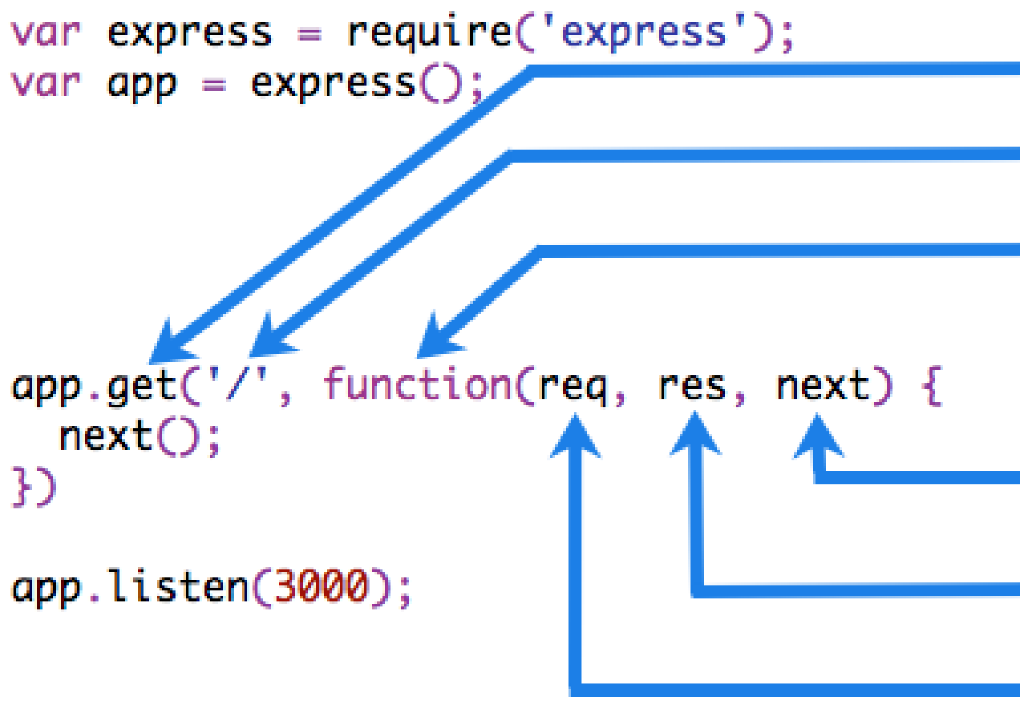
account routes

```
const express = require('express')
const requireAuth = require('../../middlewares/requireAuth.middleware')
const {getUser, getUsers} = require('./user.controller')
const router = express.Router()

// middleware that is specific to this router
router.use(requireAuth)

router.get('/', getUsers)
router.get('/:id', getUser)

module.exports = router
```

Middleware functions

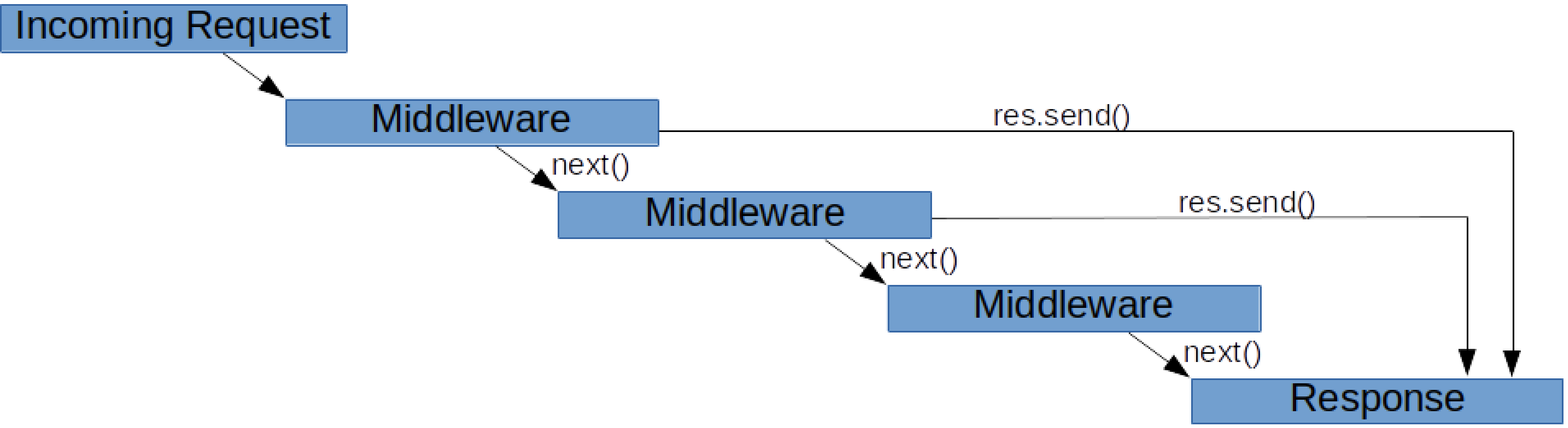
Middleware functions are functions that have access to:

- the request object (req),
- the response object (res)
- and the next middleware function in the application's request-response cycle.



```
var express = require('express');
var app = express();

app.get('/', function(req, res, next) {
  next();
})

app.listen(3000);
```

Middleware functions



Middleware functions can perform the following tasks:

- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware function in the stack.

* If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

coding_ academy

Auth Middleware

```
Async function requireAuth(req, res, next) {
    if (!req.session.user) {
        res.status(401).end('Unauthorized');
        return;
    }
    next();
}
```

user controller

```
const userService = require('./user.service')

async function getUser(req, res) {
   const user = await userService.getById(req.params.id)
   res.send(user)
}
async function getUsers(req, res) => {
   const users = await userService.query()
   res.send(users)
}

module.exports = {
   getUser,
   getUsers
}
```

auth routes

```
const requireAuth = require('../../middlewares/requireAuth.middleware')
const {login, signup, logout} = require('./auth.controller')

const router = express.Router()

router.post('/login', login)
router.post('/signup', signup)
router.post('/logout', requireAuth, logout)

module.exports = router
```

auth controller

```
async function login(req, res) {
    const { username, password } = req.body
    try {
        const user = await authService.login(username, password)
        req.session.user = user
        res.json(user)
    } catch (err) {
        logger.error('Failed to Login ' + err)
        res.status(401).send({ err: 'Failed to Login' })
    }
}
```

auth controller

```
async function signup(req, res) {
    try {
        const { username, password, fullname } = req.body
        const account = await authService.signup(username, password, fullname)
        const user = await authService.login(username, password)
        req.session.user = user
        res.json(user)
    } catch (err) {
        logger.error('Failed to signup ' + err)
        res.status(500).send({ err: 'Failed to signup' })
    }
}

async function logout(req, res){
    try {
        req.session.destroy()
        res.send({ msg: 'Logged out successfully' })
    } catch (err) {
        res.status(500).send({ err: 'Failed to logout' })
    }
}
```

The Auth Service

```
async function login(username, password) {

    const user = await userService.getByUsername(username)
    if (!user) return Promise.reject('Invalid username or password’)

    const match = await bcrypt.compare(password, user.password)
    if (!match) return Promise.reject('Invalid username or password')

    delete user.password
    return user
}

async function signup(username, password, fullname) {
    const saltRounds = 10
    const hash = await bcrypt.hash(password, saltRounds)
    return userService.add({ username, password: hash, fullname })
}
```

Handle Auth - Frontend

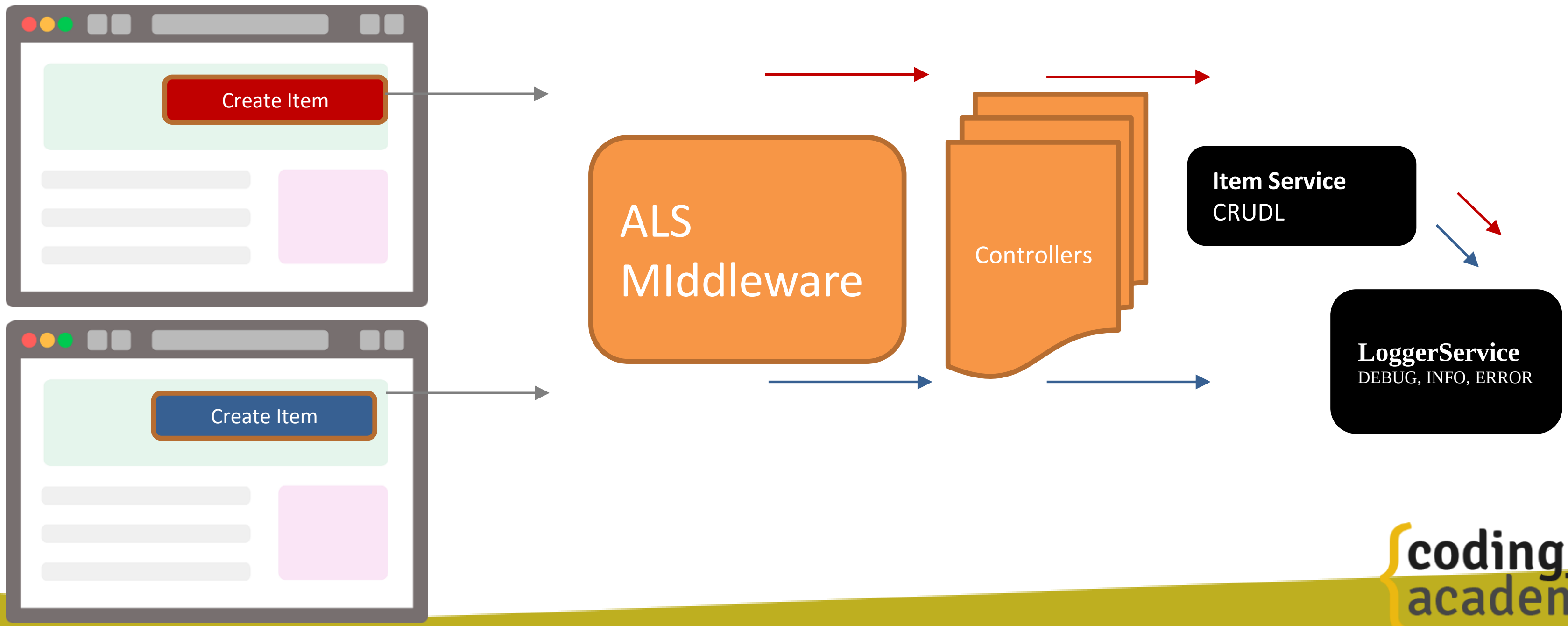
```
// frontend user service
export default {
    login,
    logout,
    getLoggedInUser,
}

var loggedInUser = JSON.parse(localStorage.getItem('loggedInUser'))
```

NPM Scripts

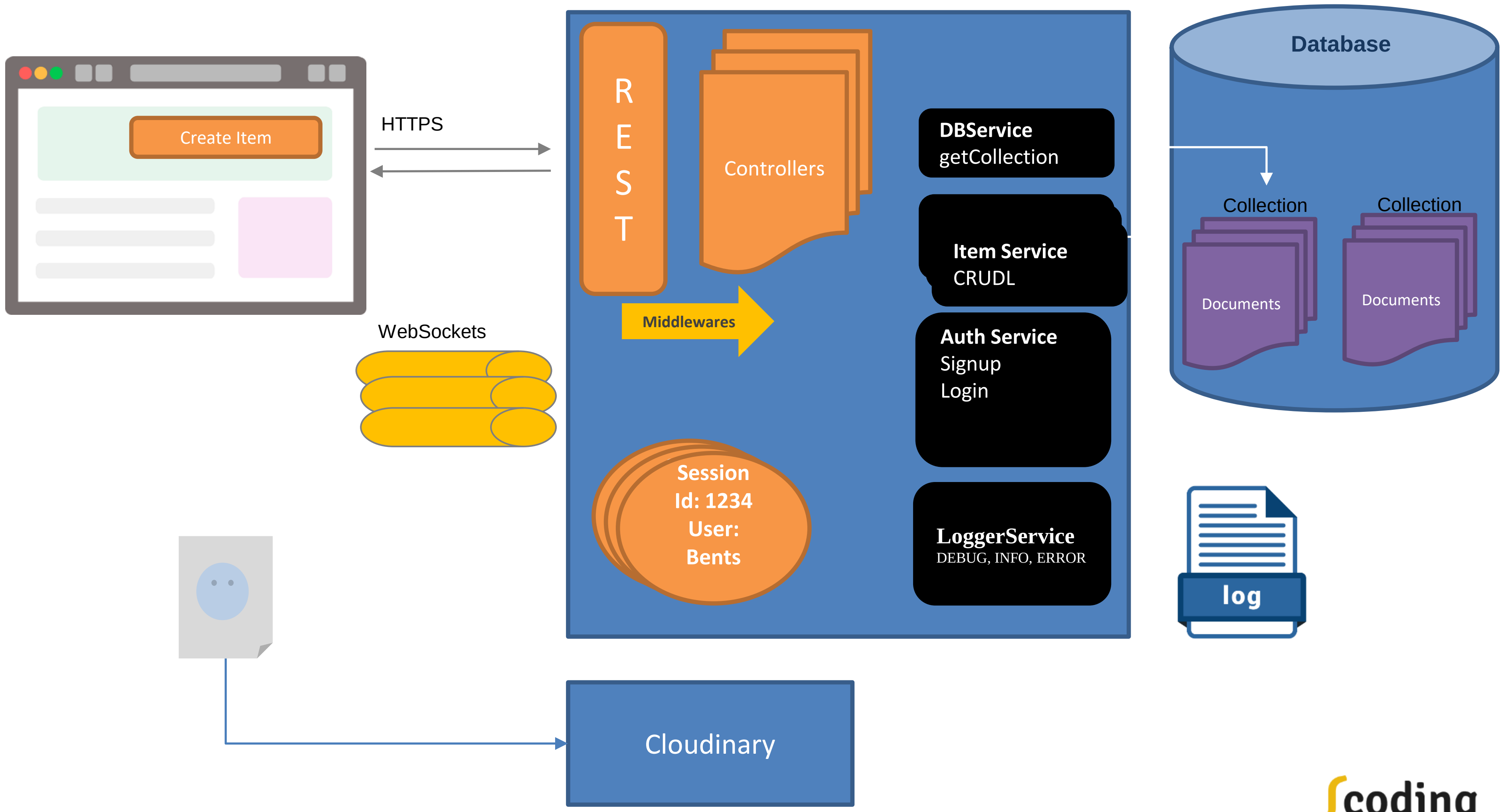
- We will later see that NPM Scripts are used to run pre defined tasks on the project
- Such as:
  - Run dev environment
    compile es6, lint, sass, etc.
  - Run tests
  - Build for production
    Minify all, Concat files
- Lets play with it for a bit

ALS - Async Local Storage in Node.js v14

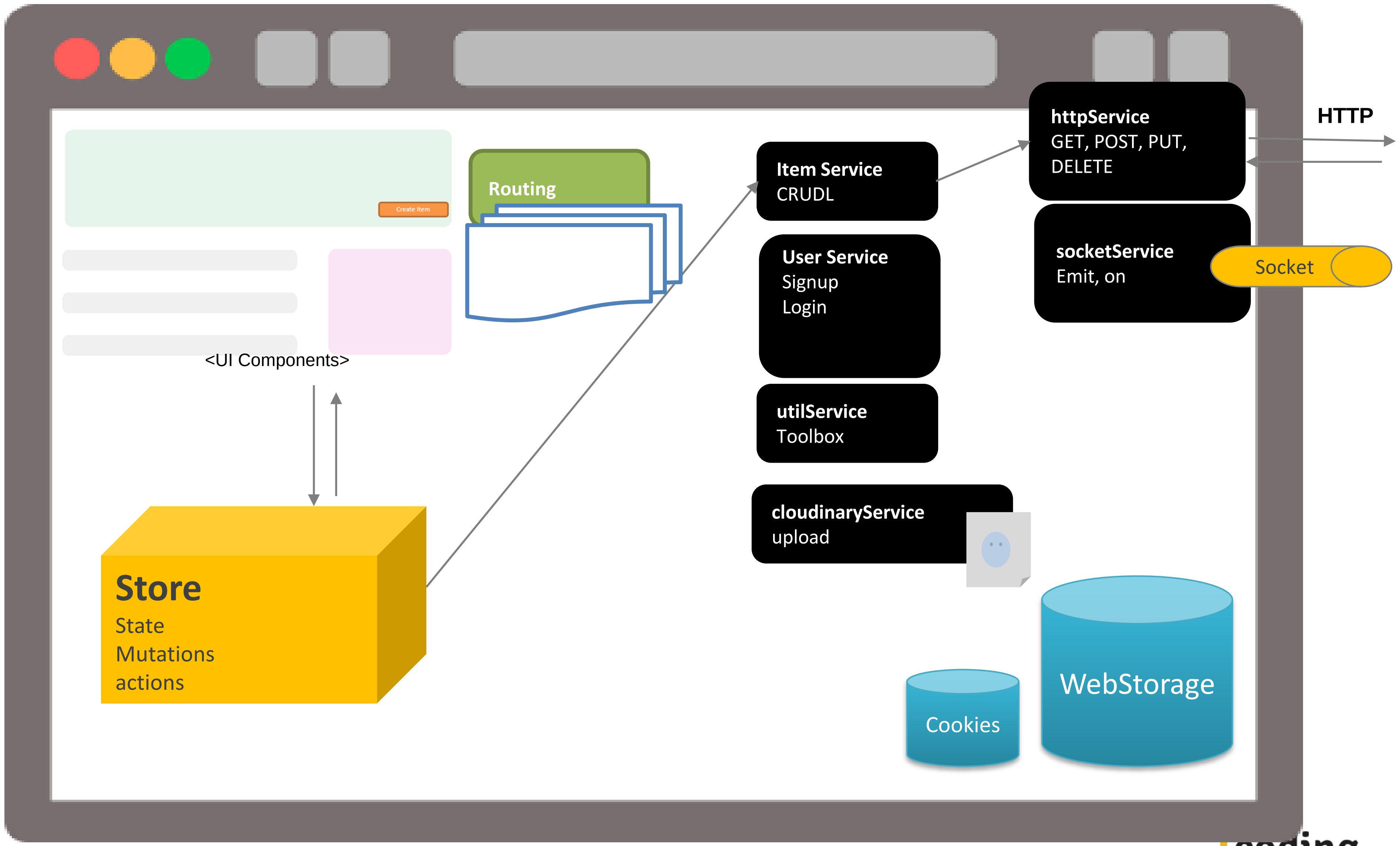
- It’s a Storage for asynchronous tasks
- Imagine two users hitting the server: Red and Blue
- ALS is a place where we can store information per-client-request and access it throughout the app



Blocks Diagram – Backend



Blocks Diagram – Frontend



coding_ academy

You are ready to build an End 2 End App