TP-12

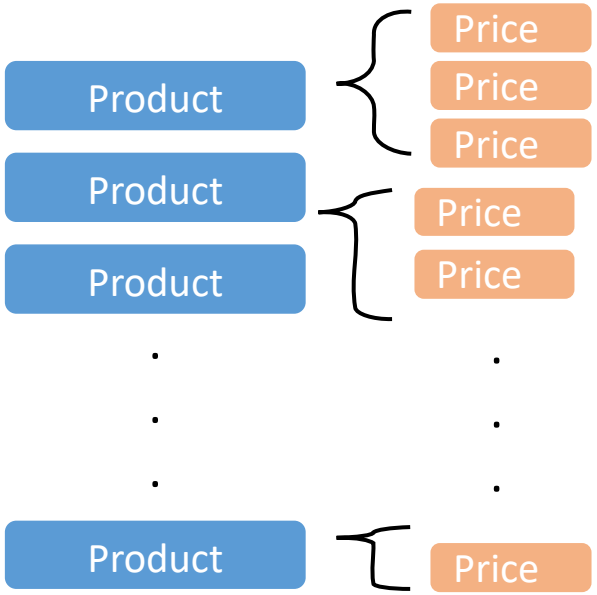
VueJS, NodeJS

CRUD APIs, Mongoose, Admin dashboard

TP12 Exercise

TP12.1: ExpressJS

Create an API for product prices.



Makes sure:

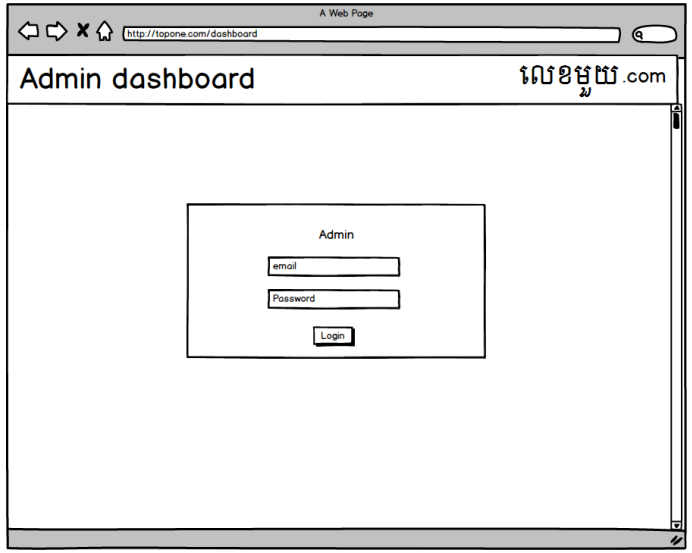
👉 You have CRUD APIs for product price

TP11 Exercise

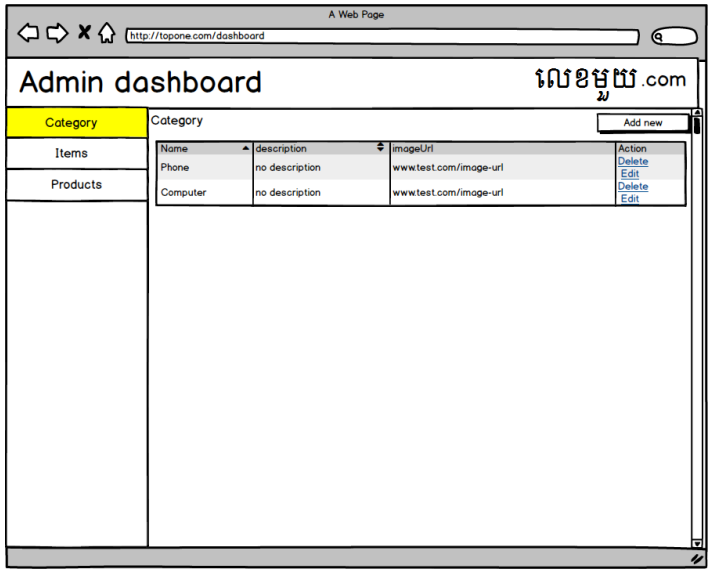
TP12.2: VueJS

Design a basic admin dashboard to manage a dynamic website.

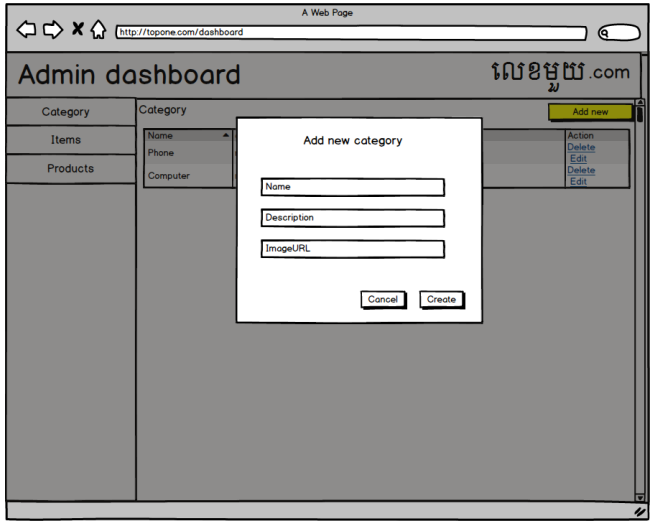
Login (/dashboard)



Management



Creation



Make sure to:

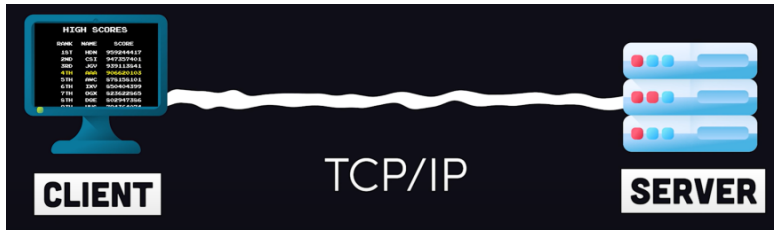
☞ Apply category, item, and product APIs with the frontend UI designs

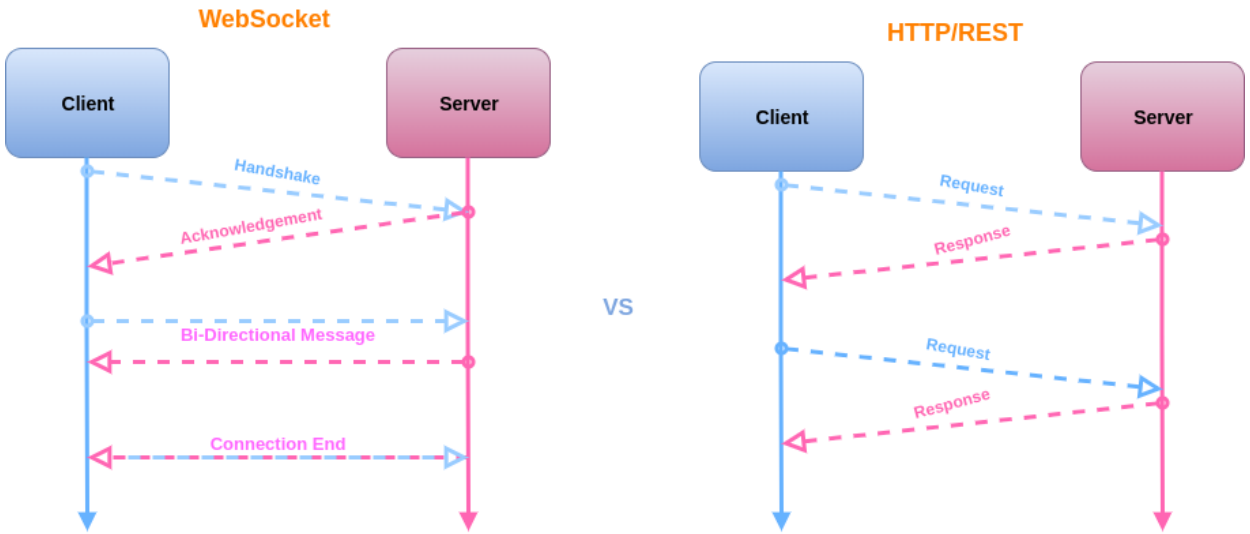
☞ Be able to perform CRUD process for category, item, product, and product price.

Getting to learn another
new Thing 🤤

“WebSocket (Real-time data communication)”

Why WebSocket?? 🤔





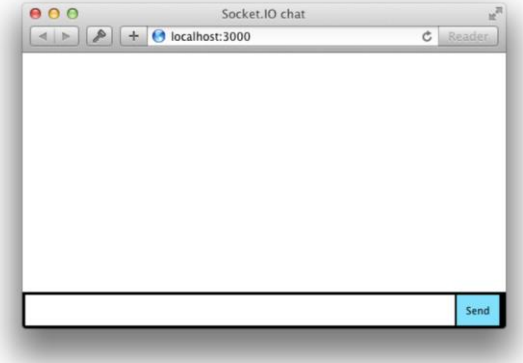
Some Most Amazing Use Cases of Websockets

1. Real-time Feeds
2. Real-time Multiplayer Gaming
3. Real-time Collaborative Editing
4. Real-time Data Visualization
5. Real-time Multimedia Chat
6. Audio / Video Chat with WebRTC
7. E-Learning Applications
8. Real-time Location Apps
9. Real-time User Behavior
10. Real-time Sports / Event Updates

Get started with a basic understanding

https://socket.io/get-started/chat
https://javascript.info/websocket#data-transfer

```
npm install express@4
```



Set up our nodeJS application

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);

app.get('/', (req, res) => {
  res.send('<h1>Hello world</h1>');
});

server.listen(3000, () => {
  console.log('listening on *:3000');
});
```

Serving HTML

```
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Socket.IO chat</title>
    <style>
      body { margin: 0; padding-bottom: 3rem; font-family: -apple-system, BlinkMacSystemFont, "Segoe

      #form { background: rgba(0, 0, 0, 0.15); padding: 0.25rem; position: fixed; bottom: 0; left: 0
      #input { border: none; padding: 0 1rem; flex-grow: 1; border-radius: 2rem; margin: 0.25rem; }
      #input:focus { outline: none; }
      #form > button { background: #333; border: none; padding: 0 1rem; margin: 0.25rem; border-radi

      #messages { list-style-type: none; margin: 0; padding: 0; }
      #messages > li { padding: 0.5rem 1rem; }
      #messages > li:nth-child(odd) { background: #efefef; }
    </style>
  </head>
  <body>
    <ul id="messages"></ul>
    <form id="form" action="">
      <input id="input" autocomplete="off" /><button>Send</button>
    </form>
  </body>
</html>
```

Integrating Socket.IO

Socket.IO is composed of two parts:

- A server that integrates with (or mounts on) the Node.JS HTTP Server socket.io
- A client library that loads on the browser side socket.io-client

During development, socket.io serves the client automatically for us, as we'll see, so for now we only have to install one module:

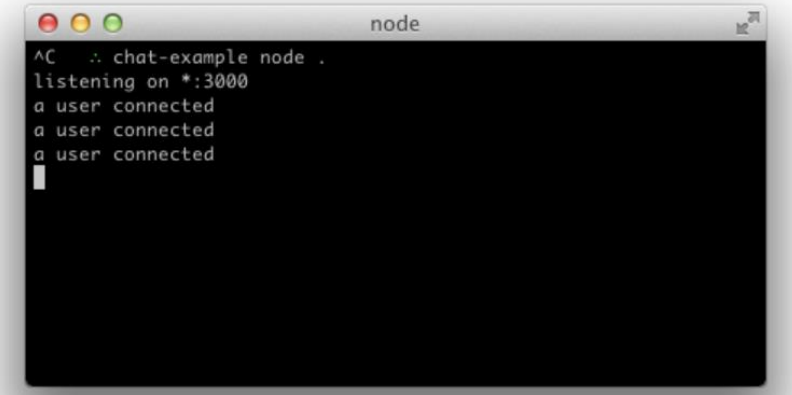
```
npm install socket.io
```

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);
const { Server } = require("socket.io");
const io = new Server(server);

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});

io.on('connection', (socket) => {
  console.log('a user connected');
});

server.listen(3000, () => {
  console.log('listening on *:3000');
});
```



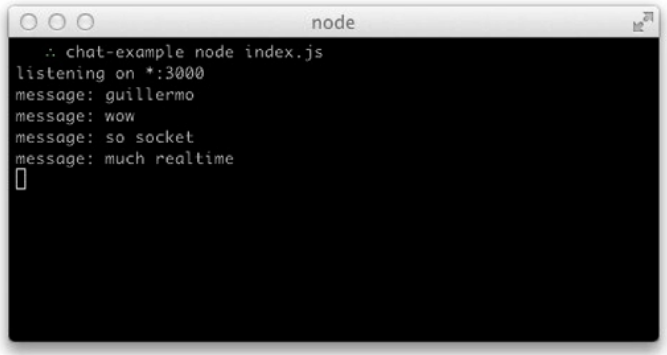
Client Side

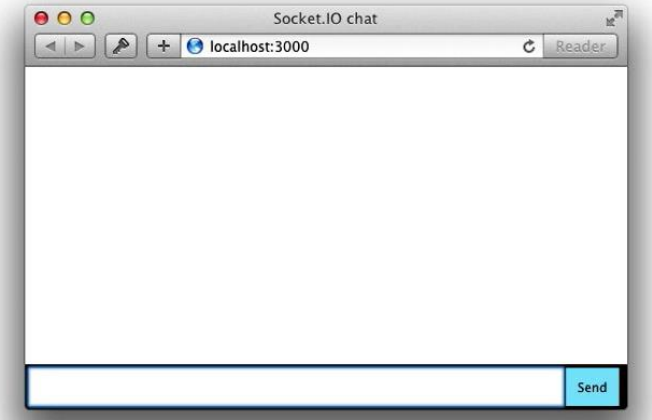
```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io();
</script>
```

Integrating Socket.IO

Server Side

```
io.on('connection', (socket) => {
  socket.on('chat message', (msg) => {
    console.log('message: ' + msg);
  });
});
```





Client Side

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io();

  var form = document.getElementById('form');
  var input = document.getElementById('input');

  form.addEventListener('submit', function(e) {
    e.preventDefault();
    if (input.value) {
      socket.emit('chat message', input.value);
      input.value = '';
    }
  });
</script>
```

Broadcasting

Server Side

In order to send an event to everyone, Socket.IO gives us the io.emit() method

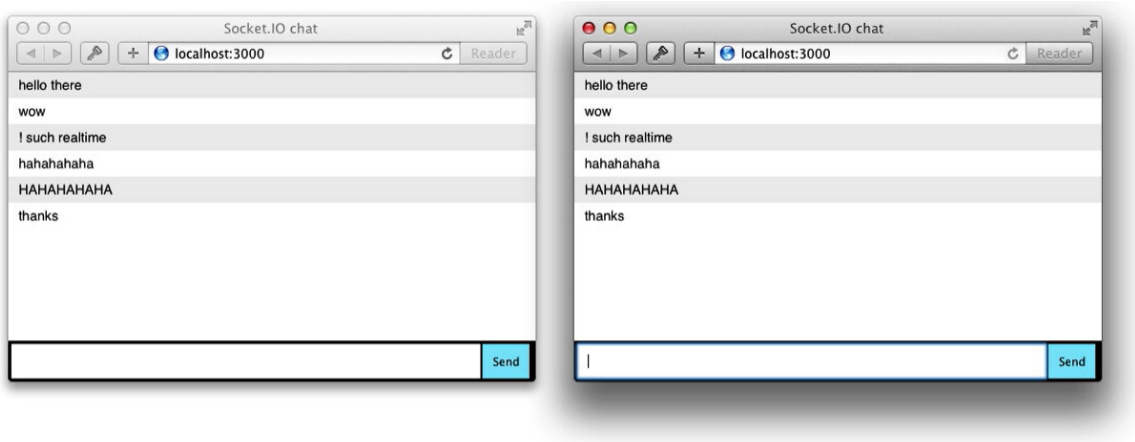
```
io.emit('some event', { someProperty: 'some value', otherProperty: 'other value' });
```

If you want to send a message to everyone except for a certain emitting socket, we have the broadcast flag for emitting from that socket:

```
io.on('connection', (socket) => {
  socket.broadcast.emit('hi');
});
```

In this case, for the sake of simplicity we'll send the message to everyone, including the sender.

```
io.on('connection', (socket) => {
  socket.on('chat message', (msg) => {
    io.emit('chat message', msg);
  });
});
```



Client Side

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io();

  var messages = document.getElementById('messages');
  var form = document.getElementById('form');
  var input = document.getElementById('input');

  form.addEventListener('submit', function(e) {
    e.preventDefault();
    if (input.value) {
      socket.emit('chat message', input.value);
      input.value = '';
    }
  });

  socket.on('chat message', function(msg) {
    var item = document.createElement('li');
    item.textContent = msg;
    messages.appendChild(item);
    window.scrollTo(0, document.body.scrollHeight);
  });
</script>
```

Good luck 👌

Practice:: Create an API to create a new category

routes\category.js

```
const categoryService = require('../services/category');
```

```
router.post('/create', auth.ensureSignedIn, async (req, res, next) => {
  const { name, desc, imageUrl } = req.body;
  const result = await categoryService.create({ name, desc, imageUrl })
  res.json(result);
})
```

services\category.js

```
const Categories = require("../models/categories")
```

```
const create = async (newCategory) => {
  // to do
  const createdCate = await Categories.create(newCategory);
  return createdCate;
}
```

Practice:: Create an API to create a new item (sub-category)

routes\item.js

```
const itemService = require('../services/item');
```

```
router.post('/create', auth.ensureSignedIn, async (req, res, next) => {
  const { name, desc, category } = req.body;
  const result = await itemService.create({ name, desc, category })
  res.json(result);
})
```

services\item.js

```
const Items = require("../models/items");
```

```
const create = async (newItem) => {
  // to do
  const createdItem = await Items.create(newItem);
  return createdItem;
}
```

Practice:: Create an API to create a new product

routes\product.js

```
const productService = require('../services/product');
```

```
router.post('/create', auth.ensureSignedIn, async (req, res,) => {
  const { title, category, item, user, imageUrl, desc, } = req.body;

  const result = await productService.create({
    title,
    category,
    item,
    user,
    imageUrl,
    desc,
  })
  res.json(result);
})
```

services\product.js

```
const Products = require("../models/products")
```

```
const create = async (newProduct) => {
  const createdProduct = await Products.create(newProduct);
  return createdProduct;
}
```

Practice:: Create an API to get category list along with item (sub-category)

routes\category.js

```
// Categorized items
router.get('/categorized-items', async (req, res) => {
  const result = await categoryService.findCategorizedItems()
  res.json(result);
})
```

services\category.js

```
const findCategorizedItems = async () => {
  return await Categories.aggregate([
    {
      $lookup: {
        from: "items",
        localField: "_id",
        foreignField: "category",
        as: "items"
      }
    },
  ])
}
```

```
const findCategorizedItems = async () => {
  return await Categories.aggregate([
    {
      $lookup: {
        from: "items",
        localField: "_id",
        foreignField: "category",
        as: "items"
      }
    },
    {
      $project: {
        _id: 1,
        name: 1,
        desc: 1,
        imageUrl: 1,
        items: {
          _id: 1,
          name: 1,
          category: 1,
          desc: 1,
        }
      }
    }
  ])
}
```

Project your wanted fields