JEGYZŐKÖNYV

Web technológiák I.

WatchOut karóra ismertető oldal

Készítette: **Szalóczy Krisztián**

Neptunkód**: Y4O4X0**

Dátum: 2024. December 11

Miskolc, 2024

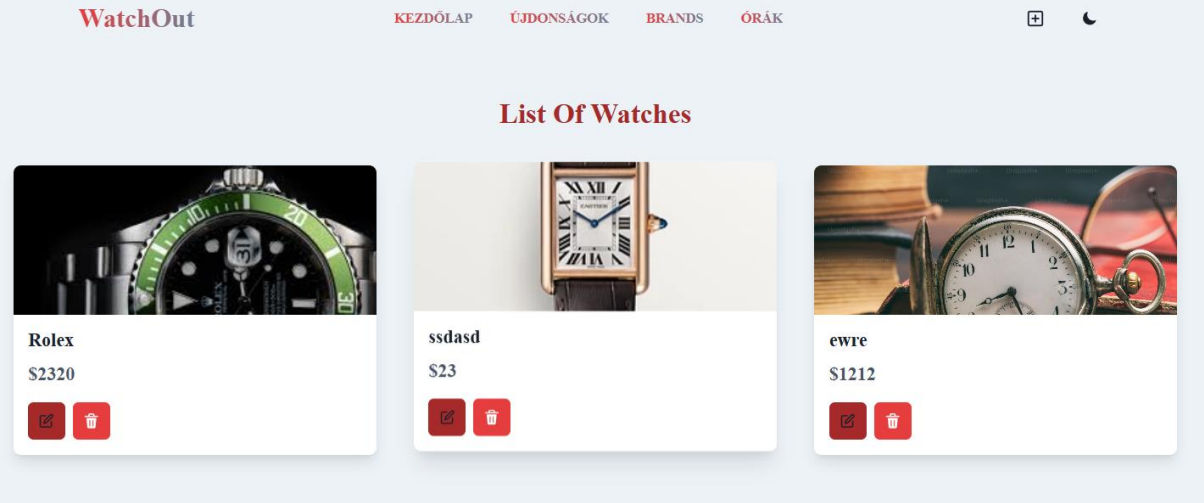
Tartalomjegyzék

Bevezetés

Az általam készített projektben a karórák világát bemutató weboldal létrehozását választottam. Ez a döntésem arra vezethető vissza, hogy érdeklődéssel figyelem a divat és technológiai fejlődés találkozását, ami a karórák világában különösen izgalmasnak ígérkezik.

A karórák többek mint pusztán időmérő eszközök: egyúttal stílust és személyiséget is kifejeznek. Évszázadok óta hű társai az embereknek, és a technológiai előre lépesekkel párhuzamosan folyamatosan alakultak és fejlődtek. Ez a weboldal lehetőséget kínál arra, hogy átfogó képet kapjunk erről a fascináló világról, beleértve a történetüket, technikai jellemzőiket és a híres gyártóikat.

Az oldal célja, hogy könnyed, szórakoztató és informatív módon mutassa be a karórák világát. Szándékom, hogy a látogatókat magával ragadó élménnyel ajándékozzam meg, és egyben lehetőséget biztosítsak számukra arra, hogy mélyebben elmerüljenek ebben a témában. Így nem csupán egy egyszerű ismertető oldalt készítek, hanem egy olyan platformot, ahol a karórák szerelmesei felfedezhetik az újabbnál újabb karórákat.



A weboldal fejlesztése során különböző technológiákat és eszközöket alkalmaztam annak érdekében, hogy egy modern, dinamikus felhasználói élményt biztosítsak. A front-end fejlesztéshez a React keretrendszert és a Chakra UI komponenstárat használtam, hogy reszponzív és stílusos felületet hozzak létre. A back-end oldalon pedig a Node.js és az Express.js keretrendszerek segítségével biztosítottam a szerveroldali funkcionalitást.

1. Mappa struktúra

Fő könyvtár: **WatchOut**

- **backend**
  - **config**
  - **controllers**
  - **models**
  - **routes**
  - **server.js**
- **frontend**
  - **src**

        - **assets**
        - **components**
        - **pages**
        - **store**
        - **App.jsx**
        - **index.css**
        - **main.jsx**
    - **index.html**
    - **eslint.config.js**
    - **vite.config.js**
- **.env**

eslint.config.js: fájl az **ESLint** konfigurációs fájlja, amely meghatározza a szabályokat és beállításokat a kód minőségének és stílusának ellenőrzésére. Ez a fájl tartalmazza, hogy milyen szabályokat használjon az ESLint (pl. milyen hibákat figyeljen, milyen kódformázási irányelveket kövessen), és segít egységesíteni a kódstílust a projektben.

vite.config.js: fájl a **Vite** projekt konfigurációs fájlja. Ebben a fájlban lehet beállítani a **Vite** build eszköz működését, például az útvonalakat, plugineket, eszközöket vagy speciális build opciókat. Ezzel szabályozhatod, hogy hogyan történjen az alkalmazás fejlesztése és összeállítása (pl. hot-reload, aliasok, optimizálás).

.env: fájl célja, hogy érzékeny információkat és környezeti változókat biztonságosan tároljon és elérhetővé tegyen az alkalmazás számára. Ilyen változók lehetnek például az adatbázis kapcsolódási adatai, az API-kulcsok, a portszám vagy más konfigurációs értékek.

2. Backend API

A weboldalam rendelkezik egy backend résszel, amely a **Node.js** és az **Express.js** keretrendszerekre épül. Ez a szerveroldali logika biztosítja az adatkezelést, az **API**-kat és a dinamikus funkcionalitásokat az alkalmazás számára.

2.1 Server.js

```
import express from "express";
import dotenv from "dotenv";
import { connectDB } from "./config/db.js";
import watchRoutes from "./routes/watch.route.js";

dotenv.config();

const app = express();
const PORT = process.env.PORT;

app.use(express.json());
app.use("/api/watches", watchRoutes)

app.listen(PORT, () => {
    connectDB();
    console.log("Server started at http://localhost:" + PORT);
});
```

Express Server Node.js-ben

1. Modulok importálása:
Az **express, dotenv**, és az adatbázis kapcsolódásához szükséges fájlok importálása történik.

2. Környezeti változók beállítása:
A **dotenv** modul segítségével a környezeti változók (pl. PORT) betöltése a **.env** fájlból.

3. Express alkalmazás létrehozása:
Az Express.js szerver inicializálása az `app` változóban.

4. JSON kezelés:
Az alkalmazás képes JSON adatok fogadására a `express.json()` middleware használatával.

5. Útvonalak kezelése:
A `/api/watches` útvonalhoz a `watchRoutes` modul kapcsolódik, ami külön útvonalakat biztosít.

6. Szerver indítása:
A szerver a megadott porton indul, és a `connectDB()` meghívásával csatlakozik az adatbázishoz.

2.2 db.js

```
import mongoose from 'mongoose';


export const connectDB = async () => {
    try {
        const conn = await
mongoose.connect(process.env.MONGO_URI);
        console.log(`MongoDB connected: ${conn.connection.host}`);

    } catch (error) {
        console.error(`Error: ${error.message}`);
        process.exit(1);
    }
}
```

1. Mongoose importálása:
A mongoose egy népszerű ODM (Object Data Modeling) könyvtár a MongoDB-hez, amely megkönnyíti az adatmodellezést és az adatbázis-interakciókat JavaScript/Node.js alkalmazásokban.

2. Aszinkron adatbázis kapcsolat létrehozása:
A connectDB függvény aszinkron módon csatlakozik a MongoDB adatbázishoz a környezeti változóban (process.env.MONGO_URI) tárolt URL segítségével.

3. Sikeres kapcsolat:
Ha a kapcsolat sikeres, a conn.connection.host tartalmazza a MongoDB hoszt nevét, amit a konzolra kiír.

4. Hiba kezelés:
Ha hiba történik a csatlakozás során, a hibaüzenet megjelenik, és a folyamat leáll (process.exit(1)), hogy elkerüljük a további futást hibás kapcsolat esetén.

2.3 MongoDb model Mongoose-zal

```
import mongoose from "mongoose";

const watchSchema = new mongoose.Schema({
    name:{
        type: String,
        required: true
    },
    price:{
        type: Number,
        required: true,
    },
    image:{
        type: String,
        required: true
    }
}, {
    timestamps: true
});

const Watch = mongoose.model('Watch', watchSchema);

export default Watch;
```

1. **Séma létrehozása:** A watchSchema definiálja a "Watch" objektum szerkezetét: név, ár, kép (mind kötelező).
2. **Timestamps:** Automatikusan hozzáadja a createdAt és updatedAt mezőket.
3. **Modell létrehozása:** A Watch modell a séma alapján készült, és a MongoDB-ben kezeli az órákat.
4. **Exportálás:** A Watch modellt más fájlok is használhatják.

2.4 API hívások

A következő kódrészlet bemutat egy **GET** metódust, amit arra használunk, hogy lekérdezzük az adatbázisban tárol óra objektumokat.

```
export const getWatches = async (req, res) => {
```

```
    try {
        const watches = await Watch.find({});
        res.status(200).json({success:true, data: watches});
    } catch (error) {
        console.log("error in fetching watches:", error.message);
        res.status(500).json({success:false, msg: "Server
Error"});
    }
}
```

PUT metódus amit arra használunk, hogy módosítást végezzünk egy adatbázisban levő óra objektumon

```
export const updateWatch = async (req, res) => {
    const {id} = req.params;

    const watch = req.body;

    if (!mongoose.Types.ObjectId.isValid(id)) {
        return res.status(404).json({success: false, msg:"Invalid
Watch Id"});
    }

    try {
        const updatedWatch = await Watch.findByIdAndUpdate(id,
watch, {new:true});
        res.status(200).json({success:true, data:updatedWatch});
    } catch (error) {
        console.log("Error in updating watch", error.message);
        res.status(500).json({success: false, msg: "Server
Error"});
    }
}
```

3. Frontend oldal

A frontend megvalósításához a **React** könyvtárat használtam, amely lehetővé tette a gyors és dinamikus komponens alapú fejlesztést. Az **UI dizájnhoz a Chakra UI** könyvtárat választottam, amely elegáns és testreszabható komponenseket kínál, így egyszerűen megvalósíthattam a reszponzív, felhasználóbarát felületet.

3.1 Használt komponensek és útvonalak

Az **App.jsx** a fő komponens, amelyben tisztán láthatóak milyen útvonalak érhetőek el az alkalmazásban.

```
function App() {
```

```
  return (
        <Box minH={"100vh"} bg={useColorModeValue("gray.100",
"gray.900")}>
          <Navbar />
            <Routes>
              <Route path="/" element={<HomePage />}> </Route>
              <Route path="/create" element={<CreatePage
/>}></Route>
              <Route path="/watches" element={<Watches/>}> </Route>
              <Route path="/news" element={<NewsPage/>}></Route>
              <Route path="/blog" element={<BrandsPage/>}></Route>
            </Routes>
        </Box>
  );}
```

3.2 Részlet a WatchCard komponensből

Ebben a komponensben számos **Chakra-UI** komponenst használta az oldal struktúrálása és reszponzivitása érdekében. Ilynek például a **Box, Image, Heading, Text, HStack, Modal** stb..

```
return (
    <Box shadow={"lg"} rounded={"lg"} overflow={"hidden"}
      transition={"all 0.3s"}
      _hover={{ transform: "translateY(-5px)", shadow: "xl" }}
      bg={bg}
    >
      <Image src={watch.image} alt={watch.name} h={40}
        w="full"
        objectFit="cover"
      ></Image>

      <Box p={4}>
        <Heading as="h3" size="md" mb={2} fontFamily={"serif"}>
          {watch.name}
        </Heading>

        <Text fontWeight={"bold"} fontSize={"xl"}
color={textColor} mb={4}>
          ${watch.price}
        </Text>
. . .
```

3.3 frontend oldali kérések

Ez a fájl egy "watch store"-t hoz létre a frontend oldalon a **zustand** állapotkezelő könyvtár használatával, amely lehetővé teszi a karórákkal kapcsolatos műveletek végrehajtását (pl. hozzáadás, módosítás, törlés és lekérdezés).

UseWatchstore Hook: Ez a hook a karórák állapotát kezeli az alkalmazáson belül, lehetővé téve az aszinkron műveletek végrehajtását, mint például adatok lekérdezése a szerverről vagy új óra hozzáadása.

```
import { create } from "zustand";
```

```
export const useWatchStore = create((set) => ({
    watches: [],
    setWatches: (watches) => set({ watches }),

    createWatch: async (newWatch) => {
        if (!newWatch.name || !newWatch.price || !newWatch.image)
{
            return {success: false, msg: "Please fill in all
fields"}
        }
        const res = await fetch("/api/watches", {
            method:"POST",
            headers:{
                "Content-Type":"application/json",
            },
            body:JSON.stringify(newWatch),
        });
        const data = await res.json();
        set((state) => ({ watches: [...state.watches, data.data
]}));
        return {success: true, msg:"Watch created successfully"};
    },
```

4. Források

https://unsplash.com/s/photos/watch

https://www.chrono24.hu/

https://www.rolex.com/watches/Cosmograph-Daytona?gclid=Cj0KCQjwj4K5BhDYARIsAD1Ly2qeByQFCIiPEAu0MQor8on6Zf0hJbVZ7I3Z18I-ERDZKCvT6IlS6jUaAr6dEALw_wcB&ef_id=Cj0KCQjwj4K5BhDYARIsAD1Ly2qeByQFCIiPEAu0MQor8on6Zf0hJbVZ7I3Z18I-ERDZKCvT6IlS6jUaAr6dEALw_wcB:G:s&s_kwcid=AL!141!3!656478705078!e!!g!!rolex%20daytona!867197436 9!88692517562&gad_source=1

https://www.chrono24.hu/omega/speedmaster--mod74.htm

https://www.hodinkee.com/

https://www.justwatches.com/?srsltid=AfmBOorXn7Op59TvV6Pl35d-7Gp54-D6HhDSei-AmwvaY1fB3dTAECy-