Szabaduljunk meg az unalmas feladatoktól

Az egyik legalapvetőbb és leghasznosabb dolog, amit a jelenlegi generatív MI-modellekkel tehetünk, az az, hogy olyan típusú kódot generálunk velük, amelyet a programozók előszeretettel neveznek angolul boilerplate kódnak, vagy magyarul egyszerűen sablonkódnak.


ELMÉLET

A *boilerplate* kifejezés (magyarul szó szerint kazánlemezt jelent) a 19. századból származik, amikor acéllemezeket használtak sablonként a gőzkazánok gyártásához.

A sablonkód olyasmi, ami minden fájlban vagy minden projektben ugyanolyan vagy majdnem ugyanolyan. Senki sem szeret sablonkódot írni, de meg kell írnunk, mert a kód, amit írni szeretnénk (az érdekes rész) nem fog működni a sablon nélkül.

Ismerjük fel az unalmas feladatokat

Az MI-vel automatizálható feladatok azonosításához gondolj azokra a feladatokra, amelyeket több projektben is újra és újra elvégzel. Előfordulhat például, hogy egy JavaScript-fejlesztő kijelöl egy elemet a böngészőablakban, például egy gombot, és hozzákapcsol egy eseményfigyelőt, hogy valamilyen műveletet hajtson végre, amikor a felhasználó interakcióba lép az elemmel. Vagy csinálhat valami bonyolultabbat, például megírhatja egy adatbázis-alapú alkalmazás kódját, amely létrehozási, olvasási, frissítési és törlési (angolul create, read, update, and delete, röviden CRUD) feladatokat kezel.

A webfejlesztők és webdizájnerek számára egy egyszerű példa az automatizálható sablonkódra egy HTML-oldal alapvető szerkezete. Minden érvényes HTML-oldal a dokumentumtípus deklarációjával kezdődik, amelyet egy HTML elem, majd egy head (fej) elem (benne számos belső elemmel, például a title (cím) taggel és metatagekkel), végül pedig egy body (törzs) elem követ.

Hagyjuk, hogy az MI megírja a sablont

Többféleképpen is minimálisra csökkenthetjük az ahhoz szükséges munkát, hogy megírjunk egy egyszerű HTML-sablont, például használhatunk egy kódrészletet, vagy kereshetünk egy példát az interneten. Ha azonban egy olyan egyéni sablont szeretnénk készíteni, amelynek van valamilyen szerkezete, és jó kiindulási alapot ad egy jövőbeli weboldal megtervezéséhez, akkor egyszerűen megkérdezhetjük az MI-t. Például a következő kérést adtam meg a ChatGPT-nek:

```
Kérlek, hozz létre egy HTML-sablont egy flexboxot használó style
elemmel egy háromoszlopos elrendezés létrehozásához, egy minta
felső navigációs sávval és egy lábléccel.
```

És itt van a kód, amelyet a ChatGPT generált:

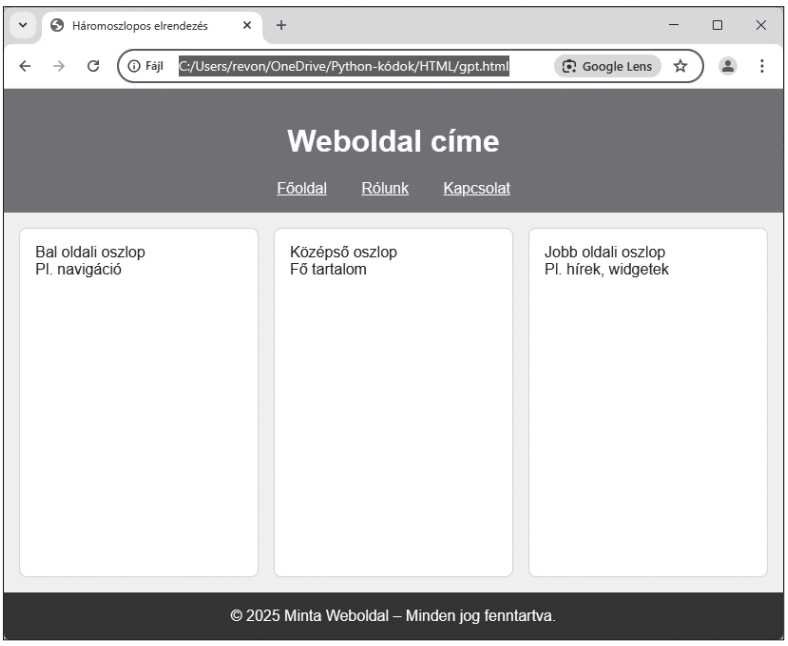
```
<!DOCTYPE html>
<html lang="hu">
<head>
  <meta charset="UTF-8">
  <title>Háromoszlopos elrendezés</title>
  <style>
    body {
      margin: 0;
      font-family: sans-serif;
      display: flex;
      flex-direction: column;
      min-height: 100vh;
    }

    header {
      background-color: #007acc;
      color: white;
      padding: 1rem;
      text-align: center;
    }

    nav {
      margin-top: 0.5rem;
    }

    .container {
      display: flex;
      flex: 1;
      padding: 1rem;
      gap: 1rem;
      background-color: #f4f4f4;
    }

    .column {
      flex: 1;
      background-color: white;
```

```
      padding: 1rem;
      border: 1px solid #ddd;
      border-radius: 0.5rem;
    }

    footer {
      background-color: #222;
      color: white;
      text-align: center;
      padding: 1rem;
    }
  </style>
</head>
<body>

  <header>
    <h1>Weboldal címe</h1>
    <nav>
      <a href="#" style="color: white; margin: 0
1rem;">Főoldal</a>
      <a href="#" style="color: white; margin: 0
1rem;">Rólunk</a>
      <a href="#" style="color: white; margin: 0
1rem;">Kapcsolat</a>
    </nav>
  </header>

  <div class="container">
    <div class="column">Bal oldali oszlop<br>Pl. navigáció</
div>
    <div class="column">Középső oszlop<br>Fő tartalom</div>
    <div class="column">Jobb oldali oszlop<br>Pl. hírek,
widgetek</div>
  </div>

  <footer>
    &copy; 2025 Minta Weboldal – Minden jog fenntartva.
  </footer>

</body>
</html>
```

A kérésem (más néven promptom) által generált HTML-kód elég jól megfelelt annak, amit én magam írtam volna kézzel. Az 1.1. ábrán láthatod az eredményül kapott weboldalt, ha lementjük a generált HTML-kódot egy fájlba, és módosítás nélkül megnyitjuk a böngészőben.

FONTOS

A könyvben használt összes kódot megtalálhatod a `https://panem.hu/720-tantusz-` konyvek oldalon.



1.1. ábra. Egy ChatGPT által generált HTML-sablon

ChatGPT

Készítsünk CRUD-kódot MI-vel

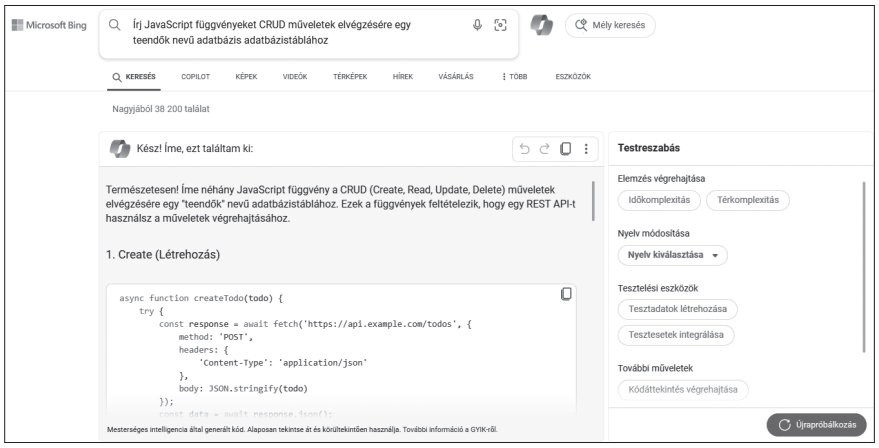
Bármilyen számítógépes program egyik leggyakoribb feladata, hogy elérjük az adatforrást, és hogy függvényeket írjunk az adatforrással történő műveletek elvégzésére. Az alapvető műveletek, amelyeket bármely adatforrással elvégezhetünk, az a rekordok létrehozása, a rekordok olvasása, a rekordok frissítése és a rekordok törlése. Arra a kódra, amely ezeket a műveleteket lehetővé teszi, használjuk gyűjtőnéven a csodálatosan sokatmondó CRUD betűszót. A legtöbben nem szeretnek CRUD-kódot írni.

Ebben a szakaszban a generatív MI-vel – kevesebbet munkával – generálunk CRUD-kódot. Legelőször is hozzáféréssel kell rendelkezned valamelyik felülethez, amelyen egy generatív MI-modellel, például a ChatGPT-vel, a Google Geminivel vagy a Microsoft Copilottal tudsz csevegni (utóbbit a Bing keresőben is használhatod). Ha még egyiknél sincs fiókod, akkor a ChatGPT-re való regisztrációhoz kövesd a 4. fejezetben található utasításokat.

Ezután, amint megjelenik előtted a csevegőablak, először küldj egy rövid kérést a generatív modellnek, hogy megnézd, mit csinál. Ha a szolgáltatás, amelyet használsz, nem hajlandó vagy nem képes teljesíteni a kérésedet, akkor legalább nem vesztegettél sok időt a részletes kérés kidolgozására. Amikor például arra kértem a Binget, hogy generáljon CRUD-kódot, hozzákezdett, de aztán azt mondta, hogy kérjek valami mást. Korábban ez gyakran előfordult, de a Copilot ma már megoldja ezt a feladatot, ahogy azt az 1.2. ábrán is láthatod.

Ekkor el kell döntened, hogy a válasz ígéretes-e. Ha nem kaptál választ, próbálj meg egy másik generatív MI-modellt használni.

Íme a válasz eleje, amelyet a kezdeti kérésemre („Írj JavaScript-függvényeket CRUD műveletek elvégzésére a teendők nevű adatbázis táblán.”) kaptam a ChatGPT-től a GPT-3.5 modellt használva:



1.2. ábra. A Bing korábban nem volt hajlandó CRUD-kódot generálni, de a Copilot ma már megoldja a dolgot.

Microsoft Corporation

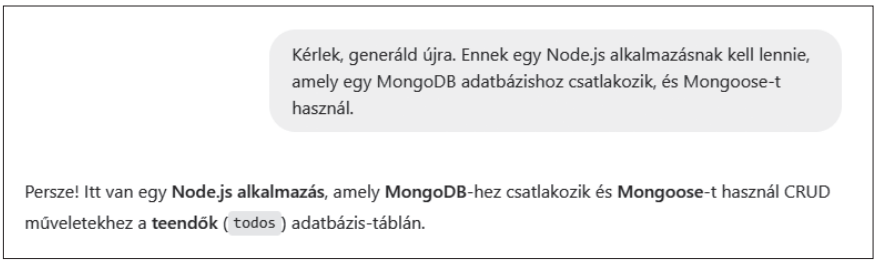
```
// Szimulált memóriában tárolt adatbázis a teendőkhöz
const teendokDB = [];

// Függvény új teendő létrehozására
function createTodoItem(title, description) {
  const newItem = {
    id: teendokDB.length + 1,
    title,
    description,
    completed: false,
  };
  teendok.push(newItem);
  return newItem;
}
```

A válasz megfelelő, de mivel nem adtam meg elég információt a kívánt alkalmazásról, a ChatGPT egy tömböt használt úgy, mintha az egy adatbázis lenne, és nem valószínű, hogy itt erre lesz szükség.

Ha ígéretes választ kapsz a kezdeti promptodra, adj meg további részleteket a modellnek. A következő lépésünk például az lehet, hogy megmondjuk a ChatGPT-nek, hogy ez egy Node.js-alkalmazás, amelyben egy MongoDB-adatbázist fogunk használni, és a Mongoose-t kell használnia.

FONTOS

Azt a folyamatot, amely során finomítjuk az adott GenAI-modellnek adott utasításainkat, valamint megadjuk a kontextust és a példákat, *prompttervezésnek* nevezzük (az eredeti angol kifejezés erre a *prompt engineering*). A prompttervezés folyamatáról a 4. fejezetben olvashatsz részletesebben. Az 1.3. ábrán a második általam készített promptot láthatod.



ChatGPT

1.3. ábra. A kontextus megadása, hogy jobb választ kapjunk.

Ezúttal a ChatGPT egy teljes Node.js-alkalmazást adott meg válaszként, amely egy helyi adatbázishoz csatlakozik, és képes CRUD-műveleteket végrehajtani. Az újragenerált kódot az 1.1. listában láthatod.

1.1. LISTA **A ChatGPT második próbálkozása**

```
const mongoose = require('mongoose');

// Csatlakozás a MongoDB-hez
mongoose.connect('mongodb://localhost/todo-list', {
   useNewUrlParser: true,
   useUnifiedTopology: true,
});

// TodoItem séma definiálása
const todoItemSchema = new mongoose.Schema({
   title: String,
   description: String,
   completed: Boolean,
});
```

```
// TodoItem modell létrehozása
const TodoItem = mongoose.model('TodoItem', todoItemSchema);

// Függvény új teendő létrehozására
async function createTodoItem(title, description) {
    const newItem = new TodoItem({
      title,
      description,
      completed: false,
    });

    try {
      await
      newItem.save();
      return newItem;
    } catch (error)
      { throw error;
    }
}

// Függvény az összes teendő olvasására
async function getAllTodoItems() {
    try {
      const todoItems = await TodoItem.find();
      return todoItems;
    } catch (error) {
      throw error;
    }
}

// Függvény adott teendő olvasására azonosító szerint
async function getTodoItemById(id) {
    try {
      const todoItem = await TodoItem.findById(id);
      return todoItem || null;
    } catch (error) {
      throw error;
    }
}

// Függvény teendő frissítésére azonosító szerint
async function updateTodoItem(id, updatedData) {
```

```
    try {
      const updatedItem = await TodoItem.findByIdAndUpdate(id,
        updatedData, { new: true });
      return updatedItem;
    } catch (error) {
      throw error;
    }
}

// Függvény teendő törlésére azonosító szerint
async function deleteTodoItem(id) {
    try {
      const deletedItem = await TodoItem.findByIdAndRemove(id);
      return deletedItem || null;
    } catch (error) {
      throw error;
    }
}
```

Ez egyáltalán nem rossz. Tetszik, hogy a ChatGPT (többnyire) a legújabb JavaScript-szintaxist használta, megfelelően kezelte az esetlegesen előforduló hibákat, és jó megjegyzéseket írt. Láttam már ennél sokkal rosszabb kódot, amit ember írt.

Az igazi teszt azonban az, hogy a kód fut-e. Miután a ChatGPT generálta a CRUD-függvényeket, néhány példával illusztrálta az 1.2. listában a használatukat.

1.2. LISTA **Példák a ChatGPT-től kapott kód használatára**

```
// Használati példa:
(async () => {
    try {
      const newItem = await createTodoItem("Bevásárlás", "Tej,
        tojás, kenyér");
      console.log("Teendő létrehozva:", newItem);

      const allItems = await getAllTodoItems();
      console.log("Összes teendő:", allItems);

      const itemToUpdate = await getTodoItemById(newItem._id);
      if (itemToUpdate) {
```

```
        const updatedItem = await
          updateTodoItem(itemToUpdate._id,
          { completed: true });
        console.log("Teendő frissítve:", updatedItem);
      }

      const deletedItem = await deleteTodoItem(newItem._id);
      console.log("Teendő törölve:", deletedItem);

    } catch (error) {
      console.error("Hiba:", error);
    } finally {
      mongoose.disconnect();
    }
})();
```

Ha a Node.js és a MongoDB telepítve van a fejlesztéshez használt gépeden, ki is próbálhatod ezt a kódot, úgy, hogy az 1.1. és az 1.2. listát bemásolod egy fájlba, és `.js` kiterjesztéssel elmented.

Az alkalmazás futtatása előtt inicializálnod kell a `.js` fájlt tartalmazó könyvtárat Node-csomagként a következő parancs kiadásával egy terminálablakban:

```
npm init -y
```

Ezután telepítsd a Mongoose-t a következő paranccsal:

```
npm install mongoose
```

Következő lépésként futtasd a programot a **`node`** parancs kiadásával, utána megadva a fájlnevet, valahogy így:

```
node lista0102.js
```

Az 1.4. ábrán láthatod, hogy mi történt a program futtatásakor.

Annak ellenőrzéséhez, hogy a ChatGPT kódja működik-e, megjegyzésbe helyeztem a létrehozott rekordot törlő kódot, újra lefuttattam a Node.js alkalmazást, majd elindítottam a Mongo shellt, és megnéztem a `todo-list` (teendők listája) gyűjtemény tartalmát, ahogy azt az 1.5. ábrán is láthatod.

```
● (base) laci@A_VAS fejezet01 % node lista0103.js
  Created item: {
    title: 'Bevásárlás',
    description: 'Tej, tojás, kenyér',
    completed: false,
    _id: new ObjectId("650c4885564f597926a10ac0"),
    __v: 0
  }
  All items: [
    {
      _id: new ObjectId("650c4885564f597926a10ac0"),
      title: 'Bevásárlás',
      description: 'Tej, tojás, kenyér',
      completed: false,
      __v: 0
    }
  ]
  Updated item: {
    _id: new ObjectId("650c4885564f597926a10ac0"),
    title: 'Bevásárlás',
    description: 'Tej, tojás, kenyér',
```

1.4. ábra. A Node.js alkalmazás futtatása.

```
test> use todo-list
switched to db todo-list
todo-list> show collections
todoitems
todo-list> db.todoitems.find()
[
  {
    _id: ObjectId("650c49f3acefaa817b939047"),
    title: 'Bevásárlás',
    description: 'Tej, tojás, kenyér',
    completed: true,
    __v: 0
  }
]
todo-list>
```

1.5. ábra. A gyűjtemény tartalmának megtekintése a MongoDB-ben.

Segítség a szintaxishoz

A számítógépes programozással járó munka jelentős részét az teszi ki, hogy egyszerűen felidézed vagy megkeresed az adott programozási nyelv szerkezetét meghatározó szabályokat, más néven a szintaxist. Minden nyelvnek vagy kódtárnak megvannak a maga módszerei a dolgokra. Ha már ismered egy programozási nyelv működésének alapjait (például hogy hogyan hozhatsz létre függvényeket, hogyan használhatod az alapvető operátorokat és hogyan írhatsz ciklusokat), tudnod kell, hogy milyen beépített függvények állnak rendelkezésre a környezetedben (legyen az akár egy böngésző vagy egy mobil operációs rendszer), és azok milyen paramétereket és adattípusokat várnak. Ez már elég sok információ, amit észben kell tartani, még egy programozó sem tud meg-