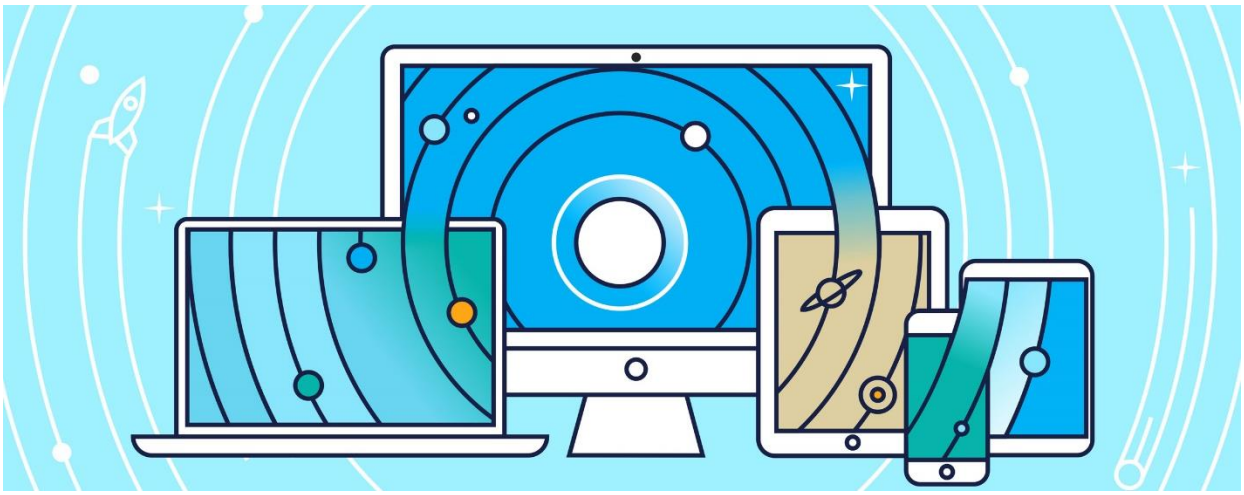
Проектна задача по предметот Напреден Веб дизајн

PROGRESSIVE WEB APPLICATIONS



Изработила:

Сара Митковска

Бр.на индекс: 173009

СОДРЖИНА

1. Општи карактеристики за Progressive Web App

Прогресивна веб-апликација(Progressive Web App) е тип на веб апликација која ги користи модерните веб-технологии за да им обезбеди на корисниците искуство слично на апликација директно преку нивните веб прелистувачи. PWA е дизајнирана така што го има земено најдоброто од веб и мобилните апликации, нудејќи функции како што се офлајн пристап, responsive (прилагодлив) дизајн кој се прилагодува на различни големини на екранот и уреди, нотификација за пристап итн.

Некои клучни карактеристики за PWA се следните:

1.1. **Responsive Design (прилагодлив дизајн):** PWA се изградени со принципи на дизајн, кои беспрекорно работат на различни уреди, вклучувајќи паметни телефони, таблети и десктоп компјутери.

1.2. **Офлајн пристап**:Ова е една од најистакнатите карактеристики на PWA со која може да се кешираат податоците и содржинити, овозможувајќи им на корисниците пристап до одредени функции и содржини дури и кога се офлајн или имаат бавна интернет конекција.Ова се постигнува преку технологии како што се Service Workers.Кога се офлајн или на бавна мрежа, кешираните средства може да се користат , обезбедувајќи непречено и доследно искуство.

1.3. **Корисничко искуство слично на апликации:**PWA имаат за цел да го имитираат изгледот на домашните мобилни апликации, обезбедувајќи непречено и извонредно корисничко искуство. Тие можат да се додадат на почетниот екран на уредот и да се стартуваат како традиционалните апликации.

1.4. **Push Notification:** PWA може да испраќаат push известувања до корисниците, дури и кога прелистувачот е затворен. Оваа функција помага да се вклучат корисниците и да се вратат во апликацијата.

1.5. **Подобрени перформанси:** PWA се дизајнирани да се вчитуваат брзо и да испорачуваат брзо корисничко искуство, благодарение на технологиите како што се lazy loading (мрзливо вчитување) и Service Workers.

1.6. **Безбедност:** PWA се опслужуваат преку HTTPS, со што се обезбедува приватност и безбедност на податоците.Ова е особено важно за апликациите што користат чувствителни информации.

1.7. **Self-Updating:** PWA може да се откријат преку пребарувачите, што ги прави лесно достапни за корисниците без потреба од инсталации апликации како што се Play Store, App Store итн.

1.8. **Компатибилност помеѓу платформи: PWA** се изградени со веб-технологии(HTML, CSS и JavaScript) и тие се компатибилни со повеќето современи веб прелистувачи.
1.9. **Само-ажурирање:** PWA може да се ажурираат себеси во позадина, обезбедувајќи корисниците секогаш да имаат пристап до најновата верзија на апликацијата.
1.10. **Поврзување:** PWA може да се споделуваат преко URL-адреси, исто како и традиционалните веб-локации, што го прави лесен за дистрибуција и пристап до нив.
1.11. **Пониски трошоци за развој и одржување:** Развивањето на еден PWA може да биде поисплатливо отколку да се создаваат посебни домашни апликации за различни платформи.

Прогресивните веб-апликации се здобиле со популарност поради нивната способност да обезбедат беспрекорно кориснично искуство низ уредите, да ја намалат сложеноста на развојот и да ја намалат бариерата за влез за развивачите на апликации.Тие се особено вредни за бизнисите кои сакаат да допрат до поширока публика со постојано и мобилно веб искуство.

За да создадеме успешен PWA, треба да се фокусираме на респонсивен дизајн, оптимизирајќи ги перформансите и треба да ги искористиме Service Workers за офлајн способности и push нотификациите. Прифаќањето на PWA денес е чекор кон градење на следната генерација на веб апликации кои ги исполнуваат постојано растечките очекувања на современите корисници.

2.Развој на PWA (Progressive Web Application)

Со правилна примена на неколку чекори ние во целост може да креираме целосно функционална PWA кој ќе нуди неверојатно корисничко искуство на сите уреди.

Чекор 1: Планирање на апликација

Пред да започнеме со развој на апликацијата, треба да ги земеме во предвид целите на PWA, кои карактеристики сакаме да ги вклучиме, приоритетите и корисничкото искуство. Можете да ги креирате првите концепти за дизајн и рамки за да ги визуелизирате структурата и распоредот.

Чекор 2: Дизајнирањр на корисничкиот интерфејс

Откако ќе завршиме со планирањето, може да продолжиме со дизајнирање на интерфејсот на апликацијата. Во текот на оваа фаза се земаат во предвид работите како што се компатибилност со различки платформи ,одзивност итн.

Чекор 3: Развивање на Front-End

Користење на веб технологии како што се HTML, CSS, JavaScript и frameworks како што е Angular, React или Vue.js развиваат привлечен интерфејс за корисниците.

Чекор 4: Имплементација на Service Workers

Service Workers е клучна компонента за PWA. Тие се JavaScript-датотеки кои работат во позадина, овозможувајќи офлајн функционалност, кеширање итн.

Чекор 5: Додавање Push Notifications

Со искористување на Push API и Service Workers може да имплементираме push известувања.

Чекор 6: Оптимизирање на перформансите

Оптимизацијатаа е многу важен чекор во развојот воопшто. Ова е начинот на кој обезбедуваме беспрекорна искуство за корисниците , со тоа што ќе го намалите времето на вчитување, со искористување на техники како што се разделување код и кеширање, со што ќе може да постигнеме брза и ефикасна операција за PWA.

Чекор 7: Тестирање и дебагирање

На крајот треба да го тестираме PWA на различни уреди, прелистувачи и мрежна состојба за да бидеме сигурни дека се исполнети сите цели. Исто така треба да собереме и повратни информации од корисниците и да ги направите потребните подобрувања кога е потребно.

3.Опис и развивање на мојата PWA

Budget Manager е Прогресивна веб апликација со која може да имаме преглед во нашите трошоци, односно да имаме преглед во тоа на што и колку сме потрошиле, исто така добрата страна на оваа апликација е тоа што може да ја користиме и офлајн, бидејќи не секогаш сите имаат пристап до интернет. Погодно за користење во ситуации кога немаме интернет конекција.

Во продолжение ќе ги образложиме сите клучни чекори поединечно и кодот со помош на кои ја изградивме оваа мини апликација.Во овој проект користиме HTML,CSS и JavaScript.

1. **Најпрвин започнуваме со креирање на HTML код**

index.html > html > head > link

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Budget Manager</title>

    <link rel="stylesheet" href="all.min.css" />
    <link rel="stylesheet" href="style.css" />
    <link rel="manifest" href="/manifest.json" />
  </head>
  <body>
    <header>
      <br />
      <h1>BUDGET MANAGER</h1>
      <br />
    </header>
    <div class="wrapper">
      <div class="container">
        <div class="sub-container">
          <!-- Budget -->
          <div class="total-amount-container">
            <h3>Budget</h3>
            <p class="hide error" id="budget-error">
              Value cannot be empty or negative
            </p>
            <input
              type="number"
              id="total-amount"
              placeholder="Enter Total Month Amount"
            />
```

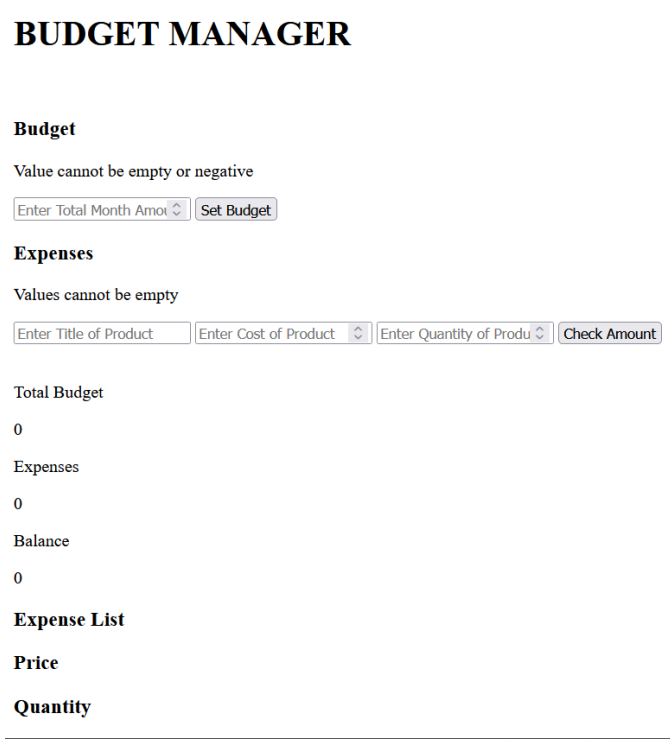
```
            <h3>Budget</h3>
            <p class="hide error" id="budget-error">
              Value cannot be empty or negative
            </p>
            <input
              type="number"
              id="total-amount"
              placeholder="Enter Total Month Amount"
            />
            <button class="submit" id="total-amount-button">Set Budget</button>
          </div>

          <div class="user-amount-container">
            <h3>Expenses</h3>
            <p class="hide error" id="product-title-error">
              Values cannot be empty
            </p>
            <input
              type="text"
              class="product-title"
              id="product-title"
              placeholder="Enter Title of Product"
            />
            <input
              type="number"
              id="user-amount"
              placeholder="Enter Cost of Product"
            />
```

```
            <input
              type="number"
              id="user-quantity"
              placeholder="Enter Quantity of Product"
            />
            <button class="submit" id="check-amount">Check Amount</button>
          </div>
        </div>
        <br />
        <div class="output-container flex-space">
          <div>
            <p>Total Budget</p>
            <span id="amount">0</span>
          </div>
          <div>
            <p>Expenses</p>
            <span id="expenditure-value">0</span>
          </div>
          <div>
            <p>Balance</p>
            <span id="balance-amount">0</span>
          </div>
        </div>
      </div>
      <div class="list">
        <h3>Expense List</h3>
        <h3>Price</h3>
        <h3>Quantity</h3>
        <div class="list-container" id="list"></div>
```

```
          <div class="flex">
            <p class="product"></p>
            <p class="amount"></p>
            <p class="quantity"></p>
          </div>
        </div>
      </div>
      <script>
        if ("serviceWorker" in navigator) {
          navigator.serviceWorker.register("/sw.js", { scope: "/" });
        }
      </script>
      <script src="script.js"></script>
    </body>
  </html>
```

Во овој HTML ја дефинираме содржината на страната и ги дефинираме сите класи кои ни се подоцна потребни за изведување на потребната функционалност на страната како и за користење во CSS за дизајнирање на изгледот на страната.

И од горенаведениот код го добиваме следниот изглед на страната:



2.Потоа креираме CSS код кој го поврзуваме со HTML-от.

style.css > .total-amount-container

```css
* {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
  font-family: "Poppins", sans-serif;
}
h3 {
  display: inline;
  padding: 2em;
}
body {
  background-color: #38393a;
}
header {
  background-color: orange;
  text-align: center;
  color: #38393a;
}
.wrapper {
  width: 90%;
  font-size: 16px;
  max-width: 43.75em;
  margin: 1em auto;
}
.container {
  text-align: center;
  width: 100%;
  color: orange;
}
```

```
.sub-container {
  width: 100%;
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 3em;
}
.flex {
  display: flex;
  align-items: center;
}
.flex-space {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
.wrapper h3 {
  color: #363d55;
  font-weight: 500;
  margin-bottom: 0.6em;
}
.container input {
  display: block;
  width: 100%;
  padding: 0.6em 0.3em;
  border: 1px solid #d0d0d0;
  border-radius: 0.3em;
  color: #414a67;
  outline: none;
  font-weight: 400;
```

```
  margin-bottom: 0.6em;
}
.container input:focus {
  border-color: #fac898;
}
.total-amount-container,
.user-amount-container {
  background-color: #ffffff;
  padding: 1.25em 0.9em;
  border-radius: 0.6em;
  box-shadow: 0 0.6em 1.2em rgba(28, 0, 80, 0.06);
}
.output-container {
  background-color: orange;
  color: #535353;
  border-radius: 0.3em;
  box-shadow: 0 0.6em 1.2em rgba(28, 0, 80, 0.06);
  margin: 2em 0;
  padding: 1.2em;
}
.output-container p {
  font-weight: 500;
  margin-bottom: 0.6em;
}
.output-container span {
  display: block;
  text-align: center;
```

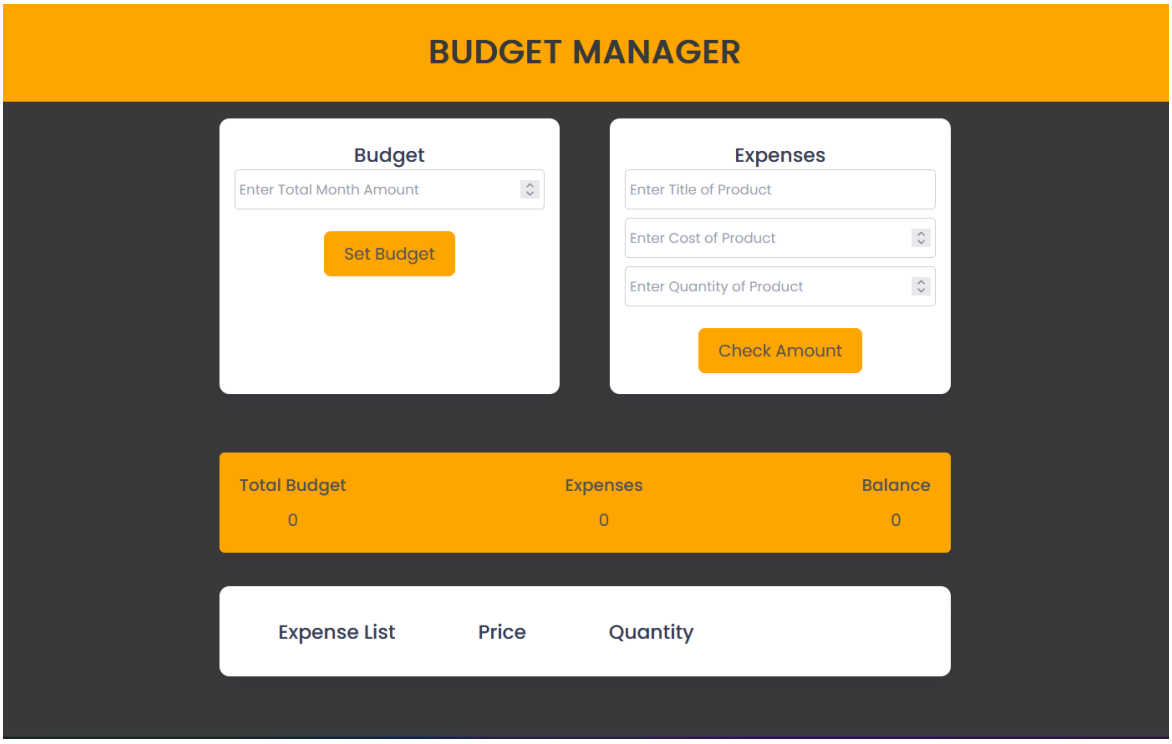
```
    font-weight: 400;
    color: #535353;
  }
  .submit {
    font-size: 1em;
    margin-top: 0.8em;
    background-color: orange;
    border: none;
    outline: none;
    color: #535353;
    padding: 0.6em 1.2em;
    border-radius: 0.4em;
    cursor: pointer;
  }
  .list {
    background-color: #ffffff;
    padding: 1.8em 1.2em;
    box-shadow: 0 0.6em 1.2em rgba(28, 0, 80, 0.06);
    border-radius: 0.6em;
  }
  .sublist-content {
    width: 100%;
    border-left: 0.3em solid orange;
    margin-bottom: 0.6em;
    padding: 0.5em 1em;
    display: grid;
    grid-template-columns: 3fr 2fr 1fr 1fr;
  }
```

```
  .product {
    font-weight: 500;
    color: #363d55;
    display: block;
    width: 25%;
    text-align: center;
  }
  .amount {
    color: #414a67;
    display: block;
    width: 25%;
    text-align: center;
  }
  .quantity {
    color: #363d55;
    display: block;
    width: 25%;
    text-align: center;
    margin-left: -2%;
  }
  .icons-container {
    width: 5em;
    margin: 1.2em;
    align-items: center;
  }
  .product-title {
    margin-bottom: 1em;
  }
```

```
.hide {
  display: none;
}
.error {
  color: #ff465a;
}
.edit {
  margin-left: auto;
}

.edit,
.delete {
  background: transparent;
  cursor: pointer;
  margin-right: 1.5em;
  border: none;
  color: #fac898;
}
.flex {
  flex-direction: row;
  flex: auto;
  width: 100% !important;
  display: flex;
}
@media screen and (max-width: 600px) {
  .wrapper {
    font-size: 14px;
  }
  .sub-container {
    grid-template-columns: 1fr;
      gap: 1em;
    }
  }

```

Со овој CSS фајл го добиваме изгледот на страната како и респонсивниот дизајн,односно страната може да се прилагодува на повеќе видови уреди.

Постигнатиот дизајн од кодот погоре е следниот:



2. **Креираме JavaScript код во кој ја правиме целата функционалност на апликацијата и го поврзуваме со HTML-от со што добиваме респонсивен дизајн односно прилагодлив на различни уреди.**

Во продолжение подетално ќе поминеме низ кодот во JS.

```
let totalAmount = document.getElementById("total-amount");
let userAmount = document.getElementById("user-amount");
let userQuantity = document.getElementById("user-quantity");
const checkAmountButton = document.getElementById("check-amount");
const totalAmountButton = document.getElementById("total-amount-button");
const productTitle = document.getElementById("product-title");
const errorMessage = document.getElementById("budget-error");
const productTitleError = document.getElementById("product-title-error");
const productCostError = document.getElementById("product-cost-error");
const amount = document.getElementById("amount");
const expenditureValue = document.getElementById("expenditure-value");
const balanceValue = document.getElementById("balance-amount");
const list = document.getElementById("list");
let tempAmount = 0;

```

Со овој код ги земаме објектите Element кои го претставуваат елементот чие id одговара на наведениот string.

```
totalAmountButton.addEventListener("click", () => {
  tempAmount = totalAmount.value;
  if (tempAmount === "" || tempAmount < 0) {
    errorMessage.classList.remove("hide");
  } else {
    errorMessage.classList.add("hide");
    amount.innerHTML = tempAmount;
    balanceValue.innerText = tempAmount - expenditureValue.innerText;
    totalAmount.value = "";
  }
});
```

Со овој код ја правиме функционалноста на Budget делот односно внесот на месечни примања.

```
const disableButtons = (bool) => {
  let editButtons = document.getElementsByClassName("edit");
  Array.from(editButtons).forEach((element) => {
    element.disabled = bool;
  });
};
```

Со овој код го оневозможуваме delete и edit копчето во случај кога немаме ниту еден продукт внесено.

```
const modifyElement = (element, edit = false) => {
  let parentDiv = element.parentElement;
  let currentBalance = balanceValue.innerText;
  let currentExpense = expenditureValue.innerText;
  let parentAmount = parentDiv.querySelector(".amount").innerText;
  let parentQuantity = parentDiv.querySelector(".quantity").innerText;
  if (edit) {
    let parentText = parentDiv.querySelector(".product").innerText;
    productTitle.value = parentText;
    userAmount.value = parentAmount;
    disableButtons(true);
  }
  balanceValue.innerText = parseInt(currentBalance) + parseInt(parentAmount);
  expenditureValue.innerText =
    parseInt(currentExpense) - parseInt(parentAmount);
  parentDiv.remove();
};
```

Го модификуваме последниот елемент.

```
const listCreator = (expenseName, expenseValue, expenseQuantity) => {
  let sublistContent = document.createElement("div");
  sublistContent.classList.add("flex");
  list.appendChild(sublistContent);
  sublistContent.innerHTML = `<p class="product">${expenseName}</p><p class="amount">${expenseValue}</p><p
  class="quantity">${expenseQuantity}</p>`;
  let editButton = document.createElement("button");
  editButton.classList.add("fa-solid", "fa-pen-to-square", "edit");
  editButton.style.fontSize = "1.2em";
  editButton.addEventListener("click", () => {
    modifyElement(editButton, true);
  });
  let deleteButton = document.createElement("button");
  deleteButton.classList.add("fa-solid", "fa-trash-can", "delete");
  deleteButton.style.fontSize = "1.2em";
  deleteButton.addEventListener("click", () => {
    modifyElement(deleteButton);
  });
  sublistContent.appendChild(editButton);
  sublistContent.appendChild(deleteButton);
  document.getElementById("list").appendChild(sublistContent);
};

```

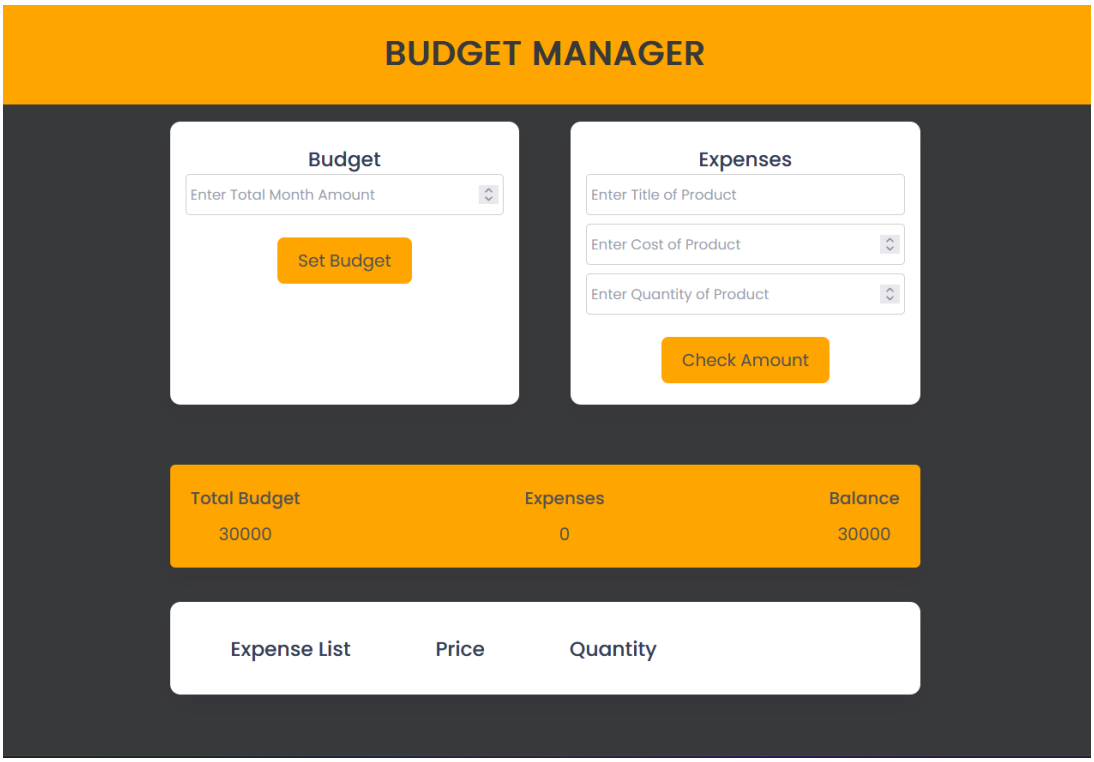
Додавање на продуктите во Expense list.

```
checkAmountButton.addEventListener("click", () => {
  if (!userAmount.value || !productTitle.value) {
    productTitleError.classList.remove("hide");
    return false;
  }
  disableButtons(false);
  let expenditure = parseInt(userAmount.value) * parseInt(userQuantity.value);
  let sum = parseInt(expenditureValue.innerText) + expenditure;
  expenditureValue.innerText = sum;
  const totalBalance = tempAmount - sum;
  balanceValue.innerText = totalBalance;
  listCreator(productTitle.value, userAmount.value, userQuantity.value);
  productTitle.value = "";
  userAmount.value = "";
  userQuantity.value = "";
});

```

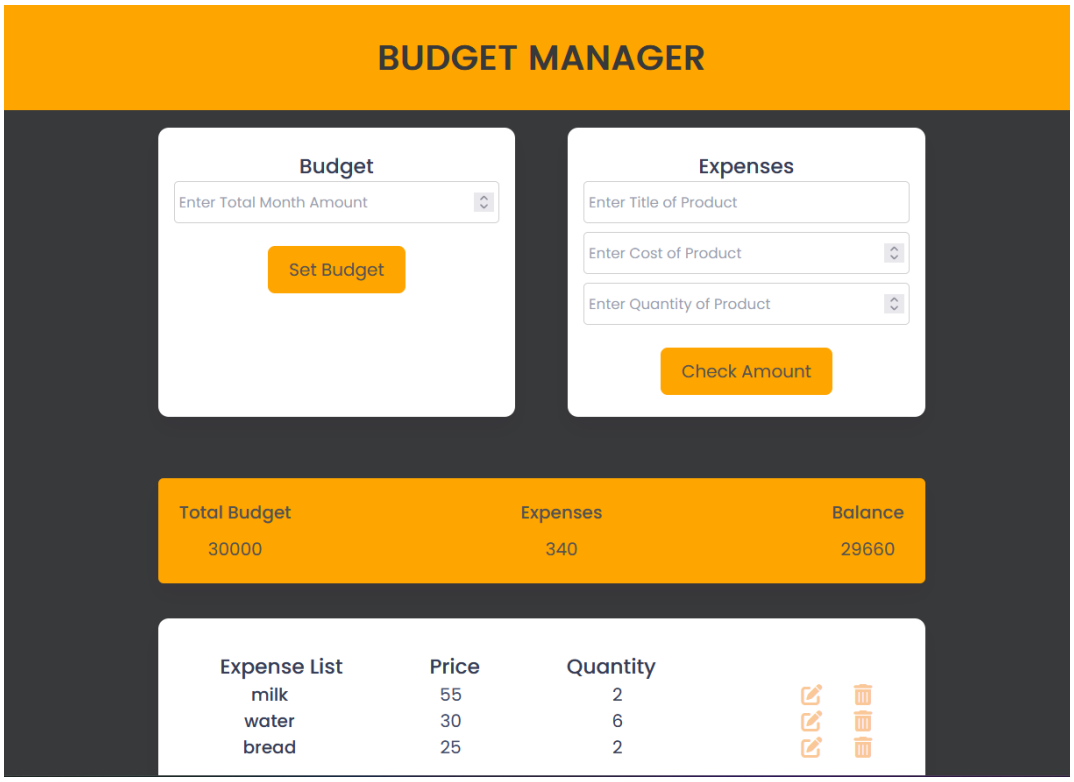
Со оваа функција најпрво додаваме Expenses, disableButtons го седираме на false,

Вршиме пресметка на цената на продуктот помножено со количината, сето тоа го одземаме од вкупниот износ и името на продуктот , цената и количината ги прикажуваме во Expenses List и productTitle, userAmount, userQuantity ги правиме празни стрингови за следното внесување на нов продукт.

Со сето ова горенаведено ја добиваме потребната функционалност на страната односно, може да внесеме месечни примања и продукти/работи на кои сме потрошиле пари количината, колку и да имаме увид во своите месечни трошоци.

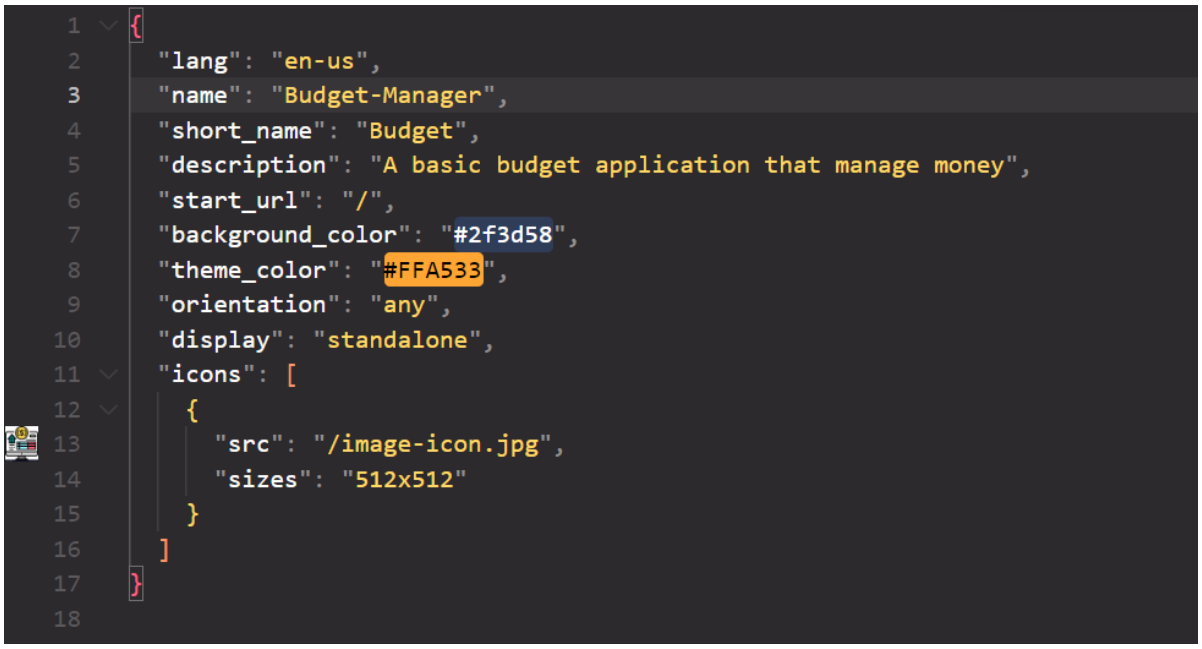


Пример на внесен месечен приход, понатаму внесуваме продукти на кои сме потрошиле пари.



Со внесените продукти автоматски ни се покажува колку пари имаме потрошено и уште колку имаме на располагање , а во долниот дел добиваме листа од продукти на кои сме потрошиле пари. Исто така, ако се случи да погрешиме многу лесно може да поправиме со кликање на копчето за промена или пак на копчето за бришење на целиот продукт.

3. **Креираме Manifest.json ги дефинира својствата на PWA, како што се име,иконите и боите на темите.**



```
{
  "lang": "en-us",
  "name": "Budget-Manager",
  "short_name": "Budget",
  "description": "A basic budget application that manage money",
  "start_url": "/",
  "background_color": "#2f3d58",
  "theme_color": "#FFA533",
  "orientation": "any",
  "display": "standalone",
  "icons": [
    {
      "src": "/image-icon.jpg",
      "sizes": "512x512"
    }
  ]
}
```

Manifest датотеката обезбедува важни информации за компјутерската програма и проект, содржи информации за името на нашиот проект , краток опис на проектот, верзија на проект итн.

4. **Креираме Service Worker кој овозможува на PWA да функционира офлајн и да ги кешираа средствата,вооедно ова е главната карактеристика на Progressive Web Application.**

```
const CACHE_NAME = `budget-manager`;

// Use the install event to pre-cache all initial resources.
self.addEventListener("install", (event) => {
  event.waitUntil(
    (async () => {
      const cache = await caches.open(CACHE_NAME);
      cache.addAll(["/image-icon.jpg", "/script.js", "/style.css", "/"]);
    })()
  );
});

self.addEventListener("fetch", (event) => {
  event.respondWith(
    (async () => {
      const cache = await caches.open(CACHE_NAME);

      // Get the resource from the cache.
      const cachedResponse = await cache.match(event.request);
      if (cachedResponse) {
        return cachedResponse;
      } else {
        try {
          // If the resource was not in the cache, try the network.
          const fetchResponse = await fetch(event.request);

          // Save the resource in the cache and return it.
          cache.put(event.request, fetchResponse.clone());
          return fetchResponse;
```

```
        } catch (e) {
          // The network failed.
        }
      }
    })()
  );
});
```

5. **Во нашиот HTML код ставаме референци до датотеката на Manifest.json и Service Worker и ги дефинираме стратегии за кеширање.**

Откако ќе го искуцаме кодот за sw.js потребно е да го регистрираме во HTML со следниот код **.**

```
95    <script>
96      if ("serviceWorker" in navigator) {
97        navigator.serviceWorker.register("/sw.js", { scope: "/" });
98      }
99    </script>
```

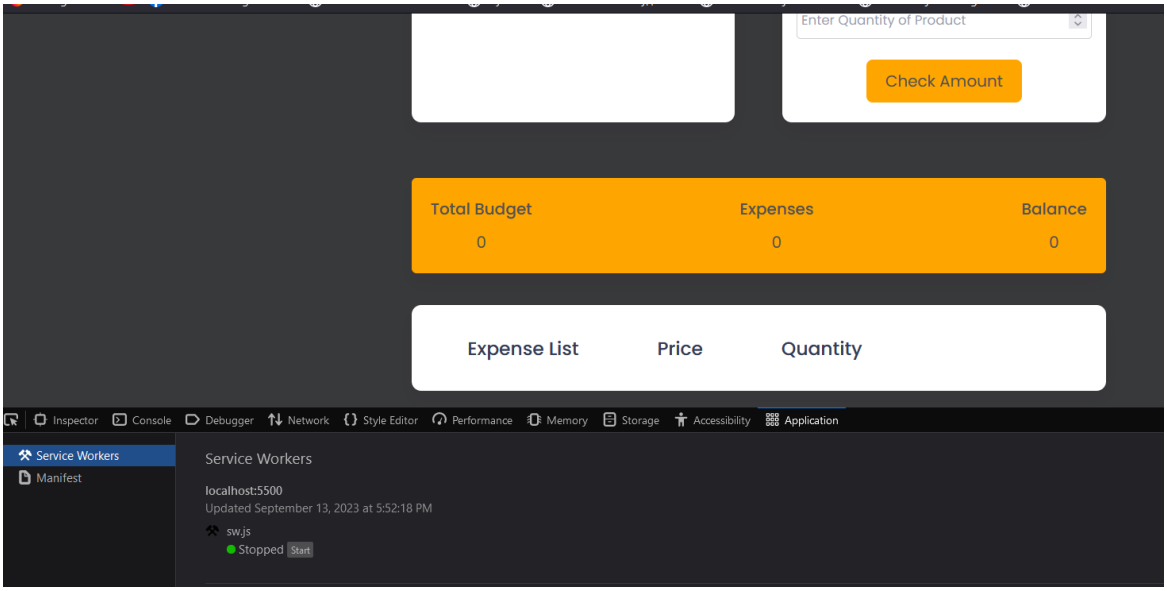
Исто така и manifest го поврзуваме со HTML со следниот код:

```
18    <link rel="manifest" href="/manifest.json" />
```

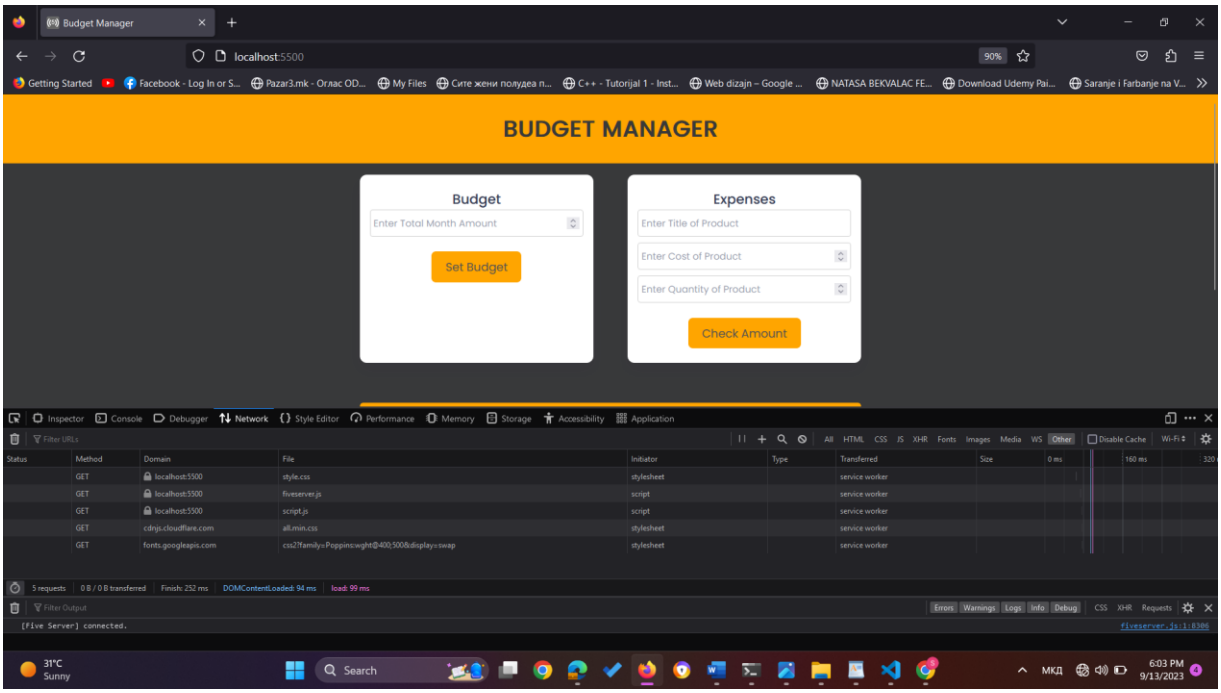
Тестирање на функционалност на апликацијата

Во продолжение ќе прикажеме неколку слики од тестирањето на SW ,односно на тоа дали апликацијата ќе функционира офлајн.

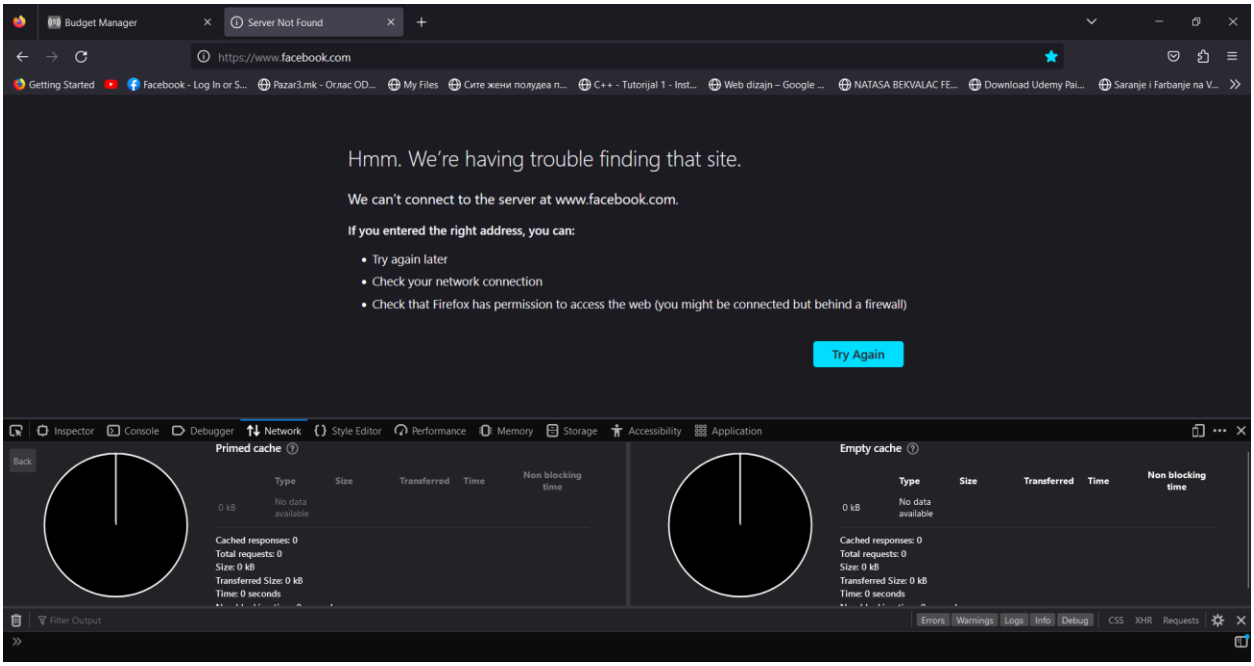
Најпрвин одиме inspect потоа application , service workers и проверуваме дали функционира.



Потоа одиме во Network подесуваме на Wi-Fi , ја гасиме нашата wi-fi конекција и одиме refresh и проверуваме дали нашата страна ќе работи офлајн.



Доколку пробаме да отвориме некоја друга страна која не е достапна офлајн ќе добиеме следно:



Со ова горенаведено добиваме една мини апликација лесна за користење и најбитно достапна офлајн и прилагодлива за сите уреди што е предност на Progressive Web Application.

4.Разлики помеѓу традиционален веб-сајт и PWA

Прогресивните веб апликации и традиционалните веб сајтови имаат неколку клучни разлики кои се вртат околу нивниот дизајн, функционалност и корисничко искуство. Некои главни разлики помеѓу нив се :

- **Офлајн функционалност**:

Кога станува збор за PWA можат да работат офлајн или во услови на ниска мрежа благодарение на service workers, кои ги кешираат основните ресурси. Корисниците можат да продолжат да користат одредени функции и да пристапуваат до содржината дури и кога не се поврзани на интернет.

Од друга страна традиционална веб страница не работат офлајн. На корисниците им треба активна интернет конекција за пристап и користење на веб сајтовите.

- **Респонсивен дизајн :**

PWA се дизајнирани да реагираат и да се прилагодуваат на различни големини и ориентации на екранот, обезбедувајќи постојано искуство на различни уреди. Додека традиционалната веб страница можат да одговорат, но можеби нема да имаат приоритет на оптимазацијата за мобилни телефони во иста мера како и PWA.

- **Push известувања:**

PWA подржуваат push известувања, дозволувајќи им на бизнисите да ги ангажираат корисниците со навремени ажурирања и пораки дури и кога апликацијата не е отворена.

Кај традиционалните веб страници обично не нудат push известувања освен ако не се имплементирани со помош на дополнителни технологии.

- **Поврзливост**

PWA лесно се споделуваат преку URL-адреси, што ги прави откриени и споделени на интернет. Ова ја поедноставува дистрибуцијата и маркетингот. Траддиционална веб страница се исто така достапни преку URL-адреси , но тие може да не го нудат истото ниво на кориснички ангажман и офлајн функционалност како PWA.

- **Инсталација и ажурирање:**

PWA може да се инсталираат директно од веб прелистувачите без потреба од продавница за апликации. Тие исто така можат автоматски да се ажурираат во позадина. Традиционалните веб страници немаат инсталации или автоматски ажурирања. Корисниците мора повторно да ја посетат веб страница за да пристапат до ажурирана содржина.

- **Безбедност:**

PWA мора да се опслужуваат преку HTTPS за да обезбеди безбедност и приватност на податоците, што ги прави стандардно безбедни. Традиционалните веб страници исто така може да користат HTTPS за безбедност, но тој може да не се спроведува на ист степен како кај PWA.

5.Заклучок

Накратко, PWA нудат подобрена функционалност, офлајн поддршка и искуство слично на апликацијата во споредба со традиционалните веб страници. Сепак, изборот помеѓу PWA и традиционална веб страница зависи од специфичните потреби и цели на проектот. PWA се добро прилагодени за сценарија каде што офлајн пристапот, push известувањата и високо привлечното корисничко искуство се важни, додека традиционалните веб страници може да бидат доволни за поедноставните информативни или промотивни цели.

Прогресивните веб-апликации го револуционизираа веб-развојот нудејќи брзи, сигурни и привлечни кориснички искуства. Тие го премостуваат јазот помеѓу native апликациите и веб, со што им олеснува на бизнисите и програмерите да допрат до поширока публика, истовремено обезбедувајќи им на корисниците непречено искуство на сите уреди.