Sveučilište Jurja Dobrile u Puli
Tehnički fakultet u Puli

Igor Marković

RAZVOJ WEB APLIKACIJE ZA POMOĆ PRI UČENJU

Završni rad

Pula, Srpanj 2024. godine

Sveučilište Jurja Dobrile u Puli
Tehnički fakultet u Puli

Igor Marković

RAZVOJ WEB APLIKACIJE ZA POMOĆ PRI UČENJU

Završni rad

JMBAG: 0303103057, redoviti student

Studijski smjer: Računarstvo

Predmet: Informacijske tehnologije i društvo

Znanstveno područje: Tehničke znanosti

Znanstveno polje: Računarstvo

Znanstvena grana: Web tehnologije

Mentor: izv. prof. Sven Maričić

Pula, Srpanj 2024.godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Igor Marković, kandidat za prvostupnika računarstva ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student
Igor Marković

U Puli, 8.7.2024.

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, Igor Marković dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj Završni rad pod nazivom "Razvoj web aplikacije za pomoć pri učenju"

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 8.7.2024.

Potpis

Igor Marković

Sadržaj

1. Uvod

U ovome završnom radu obradit će se tema izrade web aplikacije za pomoć pri učenju i upravljanjem obavezama. Kako u današnjem svijetu, tehnologije su ključne u ulozi razvoja alata koji će učenicima i studentima pomoći u upravljanju s vremenom i obavezama pa je tako došla ideja izrade web aplikacije. Kako u današnjem svijetu živimo užurbano, važno je napraviti plan općenito kako bi što produktivniji bili taj dan/tjedan/mjesec itd. I kako se kaže plan je pola posla. Pomodoro tehnika učenja smatra se jednom od najučinkovitijih tehnika učenja u današnjem svijetu i ona je popularna u cijelom svijetu. Ova aplikacija je zamišljena da bude zasnovana na Pomodoro tehnici i organiziranju zadataka (Mind map) što može poboljšati koncentraciju i učinkovitost učenika i studenata. „Pomodoro tehniku otkrio je Francesco Cirillo 90-ih godina prošlog stoljeća kada je on bio student. Pomodoro tehnika učenja dobila je ime po kuhinjskom tajmeru koji je on koristio kao tajmer, a bio je u obliku rajčice.“[1]

Aplikacija se bavi problemom s kojim se studenti i učenici susreću tijekom učenja da bi održali koncentraciju i učinkovitost. Tradicionalne metode učenja često dovode do toga da studenti budu nezainteresirani, i neučinkoviti, dok ova tehnika dokazano poboljšava njihovu koncentraciju i učinkovitost.

Hipoteza na kojoj se temelji ovaj završni rad je da web aplikacija koja je temeljena na Pomodoro tajmeru, Mind Mapu i 3D Mind Mapu znatno poboljšava koncentraciju učenika i studenata, čime se poboljšava njihov sveukupni akademski učinak.

Cijela aplikacija izrađena je u Visual Studio Code programskoj podršci kojeg je razvio Microsoft, gdje je upotrebljen HTML i CSS za dizajn i JavaScript za implementaciju funkcionalnosti. Aplikacija sadržava različite funkcionalnosti od postavljanja prilagođenog brojača, spremanje/očitavanje zadataka, dodavanje kategorija zadacima do toga da se može prebaciti u E-Ink mode, označavanje zadataka da su gotovi, itd...

[1] „Kako lakše učiti: Upoznajte Pomodoro metodu.“ (n. d.). Kreni Zdravo. Pristup: 23.6.2024. https://krenizdravo.dnevnik.hr/zdravlje/psihologija/kako-lakse-uciti-upoznajte-pomodoro-metodu

2. Što je Pomodoro tehnika učenja?

S Pomodoro tehnikom se upravlja s vremenom pri učenju, a osmislio ju je Francesco Cirillo 1990-ih godina u vremenu kada je i on bio student. Tehnika je osmišljena za poboljšanje produktivnosti podjelom rada ili učenja u kraće intervale. Oni traju 25 minuta, s kratkim pauzama. Ti se intervali nazivaju „pomodoros“, što dolazi od talijanske riječi rajčica samo u množini. A naziv je dobila prema kuhinjskom mjeraču vremena u obliku rajčice koji je i sam Cirillo koristio. Zašto zadatke uopće odvajati u cikluse? Zato što naš mozak ima vremenski ograničen fokus i ovo je jedna od najefektivnijih metoda učenja. Pomodoro tehnika može se opisati u pet koraka, a to su:

- Odaberete zadatak na kojem želite raditi.
- Postavite mjerač vremena na 25 minuta.
- Radite na tome zadatku dok brojač ne istekne.
- Napravite pauzu od 5 minuta.
- Tako ponovite cikluse 4 puta, gdje nakon četvrtog pomodoros ciklusa uzmite veću pauzu od 15 minuta.

Ova tehnika omogućuje da budemo što efikasniji i usredotočeni što više, a česti odmori poboljšavaju mentalnu agilnost. Prema autoru ove tehnike, Cirillu, jedan od glavnih ciljeva je smanjiti utjecaj unutarnjih i vanjskih prekida na fokus.

Nekoliko studija dokazalo je učinkovitost Pomodoro tehnike. „Pa tako istraživanje objavljeno u Journal of Applied Psychology sugerira da redovito uzimanje redovitih pauza tijekom duljeg radnog vremena može značajno poboljšati ukupnu izvedbu i koncentraciju.“[2] Tehniku naširoko koriste studenti, programeri i profesionalci koji žele optimizirati svoj tijek rada i održati visoku razinu produktivnosti.

Ukratko, tehnika Pomodoro ne znači samo naporniji rad, već i pametniji rad iskorištavanjem strategija upravljanja s vremenom za povećanje učinkovitosti i sprječavanje sagorijevanja. Potiče uravnotežen pristup radu i odmoru, promičući održivu produktivnost.

[2] Fritz et al. (2011). Highlighting continued uncertainty in global land cover maps for the user community. Institute of Physics (IOP) Publishing.

3. HTML

HTML je akronim za englesku riječ HyperText Markup Language. HTML je programski jezik koji se koristi za kreiranje i uređivanje dokumenata na *World Wide Webu* (WWW). HTML programski jezik je vrlo lagan za naučiti osnovne stvari, a kasnije su sve samo načini na koji će se nešto implementirati. Svaka web stranica koja postoji na Internetu je kreirana sa programskim jezikom HTML. On definira strukturu svake web stranice i njezin izgled. HTML koristi niz oznaka, tj. tzv. tag-ova. Tag je komanda koja govori pregledniku što i kako napraviti, tj. na koji način prikazati sadržaj stranice. Tag-ovi nisu osjetljivi na velika i mala slova, a pišu se unutar <“ „> znakova, samo bez navodnika. Početak svakog HTML dokumenta mora imati tag i to: <HTML> što pregledniku govori da je dokument koji treba očitati HTML i da će kao takvog prikazati na web browseru. Svaki HTML dokument mora imati zaglavlje i tijelo, što se označava kao <head> i <body>. U zaglavlju, tj. <head> se definiraju elementi koje korisnici ne vide. Također tu se definira ime dokumenta kako će se prikazivati na web browseru. Dok sve ostalo, se piše u tijelu HTML dokumenta. A to su na primjer: tekst, gumbi, slike, linkovi, grafikoni, itd…

Svaki HTML dokument započinje deklaracijom vrste dokumenta i HTML verzije, a primjer za to je: `*<!DOCTYPE html*`>. Većina ovih oznaka, tj. tag-ova imaju svoj početni i završni tag. Razlika je u tome što završni ima unutar <> kosu crtu - / i to ovako izgleda na primjer: </“html“> (samo bez navodnika). Postoje i elementi koji nemaju završni tag, kao što je na primjer
 što označava da se ide u novi red.

„Glavni elementi HTML dokumenta su:

- *<html> … </html>,*
- *<head> … </head>,*
- *<title> … </title>,*
- *<body> … </body>*.“[3]

Unutar tih elemenata, mogu se još otvarati neki elementi, kao što su:

- *<button> … </button>,*
- * … *,
- *<h1> … </h1>*, itd…

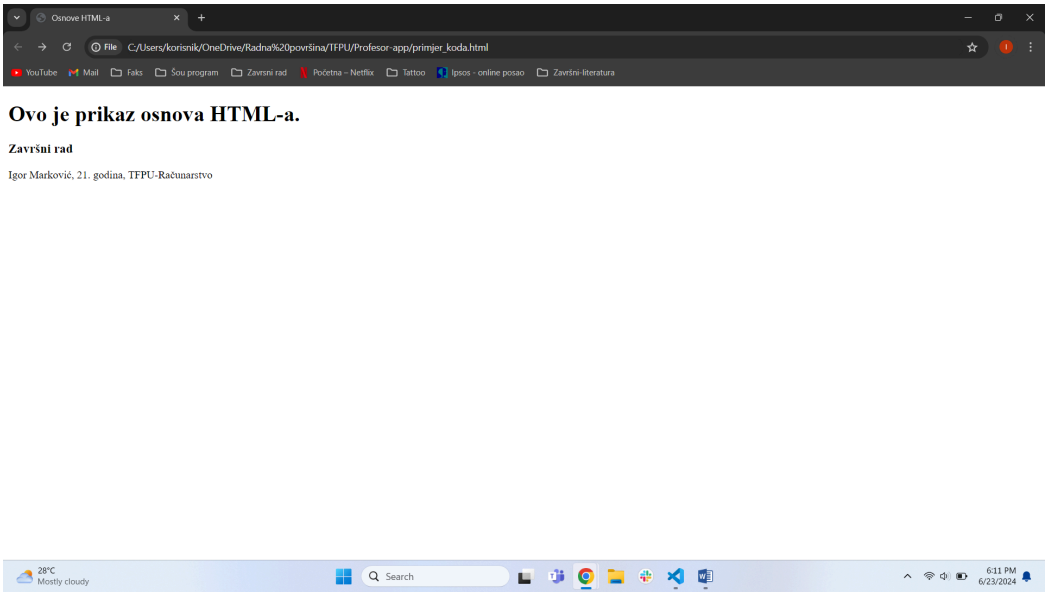
[3] Brooks D. R. (2007.). An Introduction to HTML and JavaScript: for Scientists and Engineers. (str. 1), London: Springer. ISBN-13: 978-1-84628-656-8

Sljedeće što će se prikazati u ovome radu je osnovni kod za vrlo jednostavan HTML dokument koji će se očitati na web browseru. Unutar njega odredit će se naziv dokumenta, i napisati jedan kratki tekst. Bit će dodano mu zaglavlje i tijelo, gdje će u zaglavlju pisati naslov, u tijelu tekst koji će biti prikazan na stranici.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Osnove HTML-a</title>
  </head>
  <body>
    <h1>Ovo je prikaz osnova HTML-a.</h1>
    <h3>Završni rad</h3>
    <p>Igor Marković, 21. godina, TFPU-Računarstvo</p>
  </body>
</html>
```

Primjer 1. Prikaz osnovnog koda u HTML-u

U primjeru 1. prikazano je da se u desetak linija koda napravi najjednostavnija stranica. Kao što je i gore navedeno, započeto je s definiranjem dokumenta komandom *<!DOCTYPE html>*, nakon toga otvoren je tag *<html>* što omota cijeli HTML sadržaj. Odjeljak *<head>* sadrži informacije o naslovu koji će se prikazivati na web pregledniku. Vidimo da mu je dodijeljeno ime *„Osnove HTML-a*“. Unutar tijela napisano je sve što želimo da se prikazuje na stranici, a to su nekakav osnovni tekst i prikaz da ima više vrsta teksta, h1 znači heading1 što je glavni tekst, tj. on definira najvažniji naslov i postoji sve do h6. <p> je skraćenica od engl. riječi paragraph što definira odlomak. Odlomak uvijek počinje u novom redu, a preglednici automatski dodaju razmak prije i poslije odlomka. I za kraj zatvoreno je tijelo HTML dokumenta i sam HTML dokument.



Slika 1. Prikaz osnovne stranice na web browseru

Na slici 1. prikazano je sve što je navedeno gore na prošloj stranici. Pa tako vidimo da je naziv „*Osnove HTML-a*“. Da unutar tijela prikazuju različite vrste teksta od najvećeg fonta, do jako malog. I vidimo da je prikazano u nekoliko linija kako napraviti jednostavnu HTML stranicu koja se prikazuje na web browseru.

4. CSS

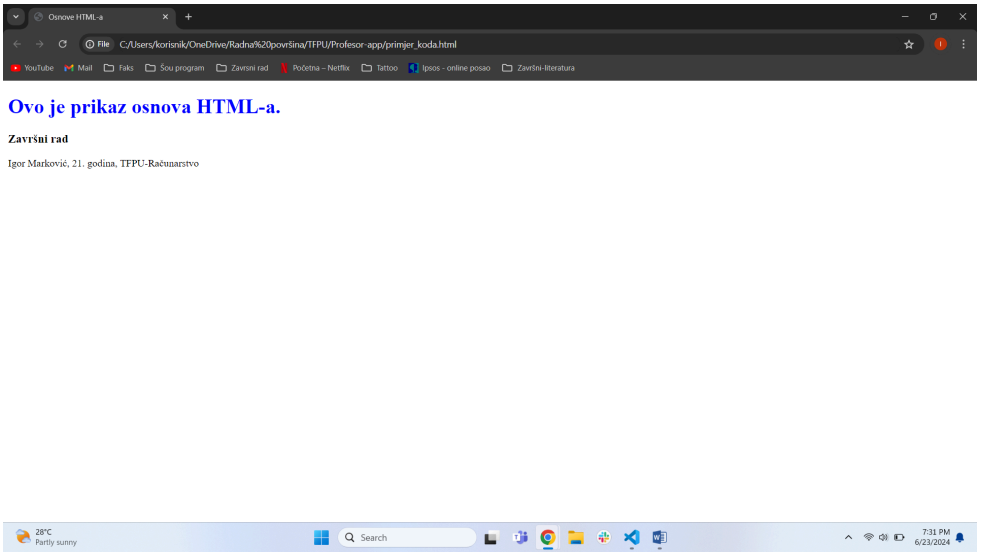
CSS, što dolazi od Cascading Style Sheets, pruža jednostavan način stiliziranja sadržaja na web browseru. CSS može izgledati kompliciranо korisnicima koji prvi puta vide kod u CSS-u. CSS kontrolira izgled, boje, fontove i cjelokupni vizualni izgled web stranice. Odvaja HTML od CSS-a, što omogućuje veću fleksibilnost i kontrolu u web dizajnu. CSS radi tako da povezuje stvari s HTML elementima, gdje svaka stavka ima ime i specifikacije. Ime se referencira na HTML, a stvari unutar CSS dodaju, kao što su veličina fonta, boja teksta, itd.

Na primjer ako pogledamo na slici 1. možemo npr. *h1* odrediti da tekst bude druge boje, npr. žute, a to bi napravili tako da prvo moramo unutar zaglavlja otvoriti element <style> i onda napisati:

```
<style>
h1 {
    color: blue;
}
</style>
```

Tu se može još dodavati razmak, zaobljenje nekakvih gumbova, podebljanje teksta, itd.

CSS je dobar alat za uređivanje za web dizajnere i programere koji omogućuje stvaranje vizualno privlačnih web stranica. Njegova sposobnost odvajanja sadržaja od prezentacije povećava mogućnost održavanja i ponovne upotrebe web stranice.



Slika 2. Prikaz web stranice kada dodamo jednostavni kod za CSS

Na slici 2. je prikazano kada na onaj kod koji je prikazan u primjeru 1. se doda jednostavan CSS kod za uređivanje boje određenog teksta, a samo je dodano da h1 bude plave boje. To je dodano unutar zaglavlja, gdje i cijeli kod za CSS treba biti.

5. JavaScript

JavaScript je programski jezik koji se koristi za kreiranje HTML stranica. Osnovala ga je tvrtka Netscape Corporation 1995. godine. Njegovo prvo ime bilo je LiveScript. „JavaScript je objektno orijentirani programski jezik, s korijenima u C/C++, koji je razvijen za korištenje s drugim alatima. JavaScript ne radi kao samostalni jezik, već je dizajniran da radi s HTML-om za stvaranje interaktivnih web stranica. JavaScript nije isto što i Java, jer je Java kompajlirani objektno orijentirani jezik, a JavaScript je interpretirani objektno orijentirani jezik.“[4] U početku se JavaScript koristio za generiranje koda na klijentskoj strani za nekakve klikove na stranici i podnošenje obrasca. Kasnije se on razvio, pa tako danas imamo platforme kao što su Node.js na primjer, što omogućuje da programeri koriste JavaScript i na serverskoj strani. JavaScript je potreban uz HTML i CSS dio za kreiranje web aplikacija, a koristi se za mijenjanje sadržaja dinamički na temelju onoga što korisnik unese ili nekakvog događaja u sustavu. Isto se može koristiti za dohvaćanje podataka, s poslužitelja, i omogućuje ažuriranje u stvarnom vremenu bez da se osvježava stranica. JavaScript ima dosta biblioteka koje ubrzavaju kod, a neke od njih su React, Vue, Angular i tako dalje.

[4] Brooks D. R. (2007.). An Introduction to HTML and JavaScript: for Scientists and Engineers (str. 3). London: Springer. ISBN-13: 978-1-84628-656-8

6. Programsko rješenje

U ovom poglavlju detaljno će se prikazati programski kod aplikacije koja pomaže pri učenju. Za izradu web aplikacije korišten je programski alat VS Code. U ovom poglavlju cilj je prikazati tehnologije koje su korištene i strukturu koda. Da odmah naglasimo da u ovom poglavlju će biti opisane najbitnije funkcionalnosti, neće se prikazati cijeli kod jer ga ima previše da se cijeli opiše.

6.1. Korištene tehnologije

Ova web aplikacija se sastoji od tri glavna programska jezika i to: HTML-a (HyperText Markup Language), CSS-a (Cascading Style Sheets) i JavaScript-a.

- HTML – koristi se za strukturu sadržaja,
- CSS – koristi se za uređivanje i prikaza web aplikacije,
- JavaScript – programski jezik koji se koristi za stvaranje dinamičkog sadržaja.

Bitno je za naglasiti da je cijeli kod za aplikaciju napisan u Visual Studio Code.

6.2. Struktura koda

U aplikaciji kod je podijeljen na tri dijela, odnosno po programskim jezicima. Prvi dio koda je CSS, drugi dio koda je HTML i treći dio koda je JavaScript.

Za početak je bilo bitno napraviti jednu mapu na računalu gdje će biti spremljeno sve što će se prikazivati na pregledniku. Za svaku web stranicu ili aplikaciju potreban je HTML dio, jer on je glavni za prikaz te stranice ili aplikacije. U njega se spremaju svi elementi koji će se prikazati na korisničkoj strani. Također u HTML-u se određuje kako će aplikacija izgledati i na koji način će ona funkcionirati. Ova aplikacija se sastoji od 3 glavna dijela i to od: Pomodoro tajmera, Mind Mapa i 3D Mind Mapa. S kodiranjem aplikacije krenuto je tako da se odredio tip dokumenta i to je prva linija koda (primjer 2.). S naredbom <*html lang=en*> se definira jezik sadržaja, a vidimo postavljeno je na engleski. Nakon toga, se otvara <*head*> element gdje se dodaje ime (*title*) koji će biti prikazan u pregledniku. Unutar <*head*> mogu se dodavati skripte koje treba očitati, linkovi na CSS datoteke, itd. Kao što je ranije rečeno svaka HTML datoteka treba sadržavati <!DOCTYPE html>, <head> i <body>. Uz pomoć tih elemenata definiraju se nekakvi opći znakovi koji se koriste za kodiranje.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Pomodoro Timer and Mind Map</title>
```

Primjer 2. Prikaz početnog koda

6.2.1. CSS dio

Nakon početnog imenovanja stranice i nekakvih osnovnih podataka započinje se s CSS-om, a to se ostvaruje tako da se upiše komanda <*style*> i </*style*> unutar kojih se upisuje CSS kod, sve to unutar <*head*> poglavlja. U <*head*> element se upisuju CSS podaci, dok u <*body*> sadržaj stranice. Kao što je i rečeno unutar <*head*> određujemo kako će aplikacija izgledati (font, debljina fonta, boja, pozadinska boja, veličina gumba,…). Sada će biti prikazano i objašnjeno neki od glavnih CSS stilova koji su korišteni u aplikaciji.

```
body,
html {
        margin: 0;
        padding: 0;
        height: 100%;
        display: flex;
        flex-direction: column;
        font-family: "Roboto", sans-serif;
        background-color: #121212;
        color: #e0e0e0;
        overflow: hidden;
}
```

Primjer 3. Prikaz osnovnog CSS koda

U primjeru 3. je prikazan osnovni prikaz cijelog tijela i HTML dokumenta kako će izgledati. Uklonjene su sve margine i padding na 0. Također je postavljena visina stranice na 100 %, određen font na „Roboto“ i određena pozadinska boja - tamno siva (HEX #121212) i boja teksta - svijetlo siva (HEX #e0e0e0). Sljedeći dio koda je `*app-container*` i `*app-section*` (primjer 4.).

```
.app-container {
        display: flex;
        width: 100%;
        height: 100%;
        padding: 20px;
        box-sizing: border-box;
      }

      .app-section {
        display: flex;
        align-items: center;
        justify-content: center;
        flex-direction: column;
        background: #1e1e1e;
        border-radius: 10px;
        box-shadow: 0 4px 20px rgba(0, 0, 0, 0.5);
        padding: 20px;
        box-sizing: border-box;
        height: 100%;
      }
```

Primjer 4. Prikaz app-containera i app-sectiona

U primjeru 4. definirano je kako će glavni kontejner `*app-container*` aplikacije izgledati. Određeno je da se elementi prikazuju horizontalno s naredbom `*display: flex*`. Također je određeno puna širina i visina aplikacije i unutarnje margine od 20px (piksela) i za kraj je postavljeno `*box-sizing: border-box*` da i bude uključen u širinu i visinu cijelog elementa. Dok vidimo da ima i `*app-section*` koji stilizira sekcije unutar tog kontejnera. Unutar njega je postavljeno da centrira sadržaj i vertikalno i horizontalno, pozadinska boja sekcija je siva (HEX #1e1e1e), isto tako su zaobljeni uglovi postavljeni na 10px (piksela), sjena (box-shadow) i margine (padding) na 20px.

```
#pomodoro-section {
        text-align: center;
        padding: 40px;
        background: #1e1e1e;
```

```
        border-radius: 10px;
        box-shadow: 0 5 20px rgba(0, 0, 0, 0.5);
        flex: 1;
       width: 100%;
        height: 100vh;
        overflow: hidden;
      }
#mindmap-section {
        display: flex;
        width: 100%;
        padding: 0;
        margin: 0;
        flex: 1;
        display: none;
        height: 100vh;
        overflow: hidden;
      }
#mindmap-3d-section {
        display: flex;
        width: 100%;
        padding: 0;
        margin: 0;
        flex: 1;
        display: none;
        height: 100vh;
        overflow: hidden;
      }
```

Primjer 5. Prikaz CSS-a za Pomodoro tajmer, Mind Map i 3D Mind Map

U primjeru 5. vidimo kako će izgledati sve tri sekcije u aplikaciji, točnije `*pomodoro-section*` određuje kako će izgledati Pomodoro timer, `*mindmap-section*` određuje kako će izgledati Mind Map, a `*mindmap-3d-section*` određuje kako će

izgledati 3D Mind Map u aplikaciji. Pa tako vidimo da u Pomodoro sekciji je postavljen tekst na centar s marginom od 40px, dok je pozadina kao i kod većine stvari tamno sive boje (HEX #1e1e1e), također aplikacija ima zaobljene rubove od 10px, sjenu i maksimalnu širinu i visinu na 100vh i skriva prolijevanje sadržaja izvan kontejnera. U `*mindmap-section*` i `*midmap-3d-section*` vidimo da je određeno `*display: flex;*` što znači da je ova sekcija horizontalno postavljena, sa 100 % širinom, bez margina i paddinga, a početno je skrivena da se ne vidi (to je postignuto naredbom `*display: none;*`). Isto tako je postavljena na maksimalnu visinu od 100vh i skriva prolijevanje sadržaja.

```
.timer-controls {
        margin-bottom: 20px;
      }
      .action-button,
      .popup-button {
        padding: 10px 20px;
        border: none;
        background-color: #007bff;
        color: white;
        margin: 5px;
        border-radius: 5px;
        cursor: pointer;
        font-size: 16px;
        transition: background-color 0.3s, transform 0.3s;
      }
      .action-button:hover,
      .popup-button:hover {
        background-color: #0056b3;
        transform: scale(1.05);
      }
```

Primjer 6. Prikaz CSS koda za action i popup buttone

U primjeru 6. vidimo da ima`*.timer-controls*` gdje je određen razmak ispod kontrola tajmera na 20px. `*.action-button*` i `*.popup-button*` postavljaju padding, pozadinsku boju na plavu s bijelim znakovima, margine su postavljene na 5px i zaobljenost isto 5px i veličina fonta je 16px. Ovdje još vidimo kada se dođe kursorom na taj gumb da

kursor postane pointer i da ima određenu transformaciju i da se povećava kada je kursor iznad ovih gumb.

```
.timer-display {
        font-size: 48px;
        margin: 20px 0;
        font-weight: bold;
      }
      input[type="text"],
      input[type="checkbox"],
      .popup,
      .confirm-popup {
        padding: 10px;
        border-radius: 5px;
        border: none;
        margin-bottom: 5px;
        width: 120px;
      }
      input[type="text"] {
        background-color: #252525;
        color: #e0e0e0;
        border: 1px solid #333;
        padding: 10px;
        border-radius: 5px;
      }
      .popup,
      .confirm-popup {
        position: fixed;
        width: 300px;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
        background-color: #333;
        box-shadow: 0 0 20px rgba(0, 0, 0, 0.7);
        display: none;
        flex-direction: column;
```

```
        padding: 20px;
        z-index: 1001;
      }
```

Primjer 7. Prikaz CSS-a za izgled tajmera, input polja i popup-ova

U primjeru 7. vidimo da je postavljeno kako će tajmer biti prikazan tj. njegov display. Vidimo da je postavljen font na 48px s marginama od 20px i da je font podebljan. Sljedeće što vidimo je da je određeno kako će izgledati input polja (npr. nekakav tekst i check boxovi) i popup prozori. Pa tako vidimo da će to imati margine od 10px s unutrašnje strane, zaobljenje popupova s 5px i da neće imati okvir. Dodana je margina od 5px da bude ispod elemenata i postavljena širina na 120px. Sljedeće što slijedi je specifični stil za input za tekst, tj. okvir za tekst. Tako je tu postavljeno pozadinska boja na tamno sivu (#252525) s bijelim znakovima teksta. Još je određeno da okvir bude tamno sivi (#333) što nam pruža blagi kontrast, određen je razmak zbog bolje preglednosti.

```
.tasks {
        list-style-type: none;
        text-align: center;
        padding: 0;
        width: 60%;
        max-height: 400px;
        overflow-y: auto;
        padding: 10px;
      }

      .tasks li {
        margin-bottom: 10px;
        display: flex;
        align-items: center;
        background-color: #2e2e2e;
        padding: 8px;
        border-radius: 8px;
        transition: background-color 0.3s, box-shadow 0.3s;
        box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
        white-space: normal;
```

```
        overflow: visible;
        font-size: 10px;
      }

      .tasks li:hover {
        background-color: #3d3d3d;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);
}
```

Primjer 8. Definiranje izgleda za prikaz liste zadataka

Sljedeće treba odrediti kako će lista zadataka izgledati (primjer 8.).

`.*tasks*` postavlja stilove za listu zadataka, što uključuje poravnanje teksta, padding, širinu, maksimalnu visinu i omogućava vertikalno prolijevanje.

Sljedeće je `.*tasks li*` gdje se određuju stilovi za stavke liste što uključuje margine, flex-box, poravnanje elemenata, pozadinsku boju na crnu (HEX #2e2e2e), sa zaobljenim rubovima od 8px, tranzicije za promjene boje i sjenu.

`*.tasks li:hover*` određuje pozadinsku boju liste zadataka na sivu (HEX #3d3d3d) i dodaje sjenu kada je kursor iznad stavke liste.

```
.task-manager {
        display: flex;
        text-align: center;
        flex-direction: column;
        align-items: center;
        width: 98%;
        height: calc(100vh - 160px);
        overflow-y: auto;
      }

      #task-input,
      #add-task {
        width: 70%;
        max-width: 500px;
        margin: 5px 2px;
      }
```

```
      #add-task {
        width: auto;
        margin-left: auto;
        margin-right: auto;
      }
```

Primjer 9. Prikaz polja za unos zadataka i gumb za dodavanje zadataka

U primjeru 9. su tri stvari, a to su `*.task-manager*`, `*#task-input*` i `*#add-task*`.

`*.task-manager*` definira kako će izgledati kontejner koji upravlja zadacima u aplikaciji. Pa tako vidimo da koristi flexbox za raspoređivanje elemenata unutar toga kontejnera, centrira teksta u sredinu, postavlja elemente unutar kontejnera u kolonu, postavlja horizontalno u centar elemente. Također, on ima širinu od 98 %, postavlja širinu i ako dođe do premašivanja visine kontejnera postavlja se vertikalno pomicanje kroz kontejner.

`*Task-input*` i `*add-task*` su za unos zadataka i gumb za isto. Pa tako vidimo da je postavljena širina na 70 % gdje je maksimalno 500px i koji imaju margine 5px vertikalno i 2px horizontalno. Dok sami `*#add-task*` je specifični stil za gumb za dodavanje novih zadataka. Tu je postavljeno da se širina postavlja automatski i da centrira gumb horizontalno.

```
.add-subtask {
        padding: 3px 6px;
        font-size: 12px;
        background-color: #007bff;
        color: white;
        border: none;
        border-radius: 5px;
        cursor: pointer;
        margin-left: 5px;
        transition: background-color 0.3s, transform 0.3s;
      }

      .add-subtask:hover {
        background-color: #0056b3;
        transform: scale(1.05);
```

```
      }


      .checked-task {
        color: #ffd700;
        text-decoration: line-through;
      }
```

Primjer 10. Prikaz CSS-a za dodavanje novih pod zadataka

Primjer 10. prikazuje dio koda za gumbove za dodavanje novih pod zadataka.

`*.add-subtask*` je gumb za dodavanje novog pod zadatka. Ovdje vidimo da je dodan padding, veličina fonta je 12px, pozadinska boja tj. koje će boje biti gumb. Isto tako određeno je da tekst bude bijele boje bez podebljanja, zaobljenje gumba je 5px i kada se dođe kursorom na gumb da kursor postane pointer. Još je dodana margina s lijeve strane od 5px da ima određenu tranziciju kada se dođe pokazivačem na gumb.

`*.checked-task*` označava kada je neki zadatak završen, a u Pomodoro tajmeru ga možemo označiti da je dovršen i taj tekst postane žute boje s linijom preko teksta, što je ovdje definirano.

```
svg {
        height: 100%;
        width: 100%;
      }
      .footer {
        position: absolute;
        bottom: 10px;
        right: 20px;
      }
      .footer .action-button {
        margin: 0 5px;
      }


      .round-button {
        position: absolute;
        top: 10px;
        left: 10px;
        width: 50px;
```

```
        height: 50px;
        background-color: rgba(255, 255, 255, 0.3);
        border: none;
        border-radius: 50%;
        cursor: pointer;
        z-index: 1001;
        display: flex;
        justify-content: center;
        align-items: center;
        font-size: 24px;
        color: #fff;
        opacity: 22%;
      }
```

Primjer 11. Prikaz elemenata SVG-a, footera i round buttona

U primjeru 11. detaljno je objašnjeno kako su uređeni SVG (engl. Scalable Vector Graphics)[5], footer i round button. SVG predstavlja širinu i visinu elemenata svojih kontejnera. Konkretno, SVG određuje kako će se Mind Map prikazivati. Na slici vidimo da je postavljena puna širina i duljina unutar kontejnera.

`*.footer*` postavlja poziciju na apsolutnu unutar svog nadređenog elementa, i to točno 10px od dna i 10px od desnog ruba. Također `*.footer*` elementi su gumbovi koji imaju margine 5px između sebe. Ovo konkretno služi za gumbove za prikazivanje sekcija Mind Mapa, Pomodoro tajmera i 3D Mind Mapa.

`*.round-button*` definira kako će izgledati gumb koji je okrugli i pozicioniran je u gornji lijevi kut. Vidimo da gumb ima visinu i širinu od 50px, s bijelom prozirnom pozadinskom bojom, bez obruba, sa zaobljenim rubovima od 50 %. Isto tako vidimo u kodu da je kursor pointer kada se dođe s kursorom na gumb i da je gumb prikazan iznad drugih elemenata, a tekst centriran horizontalno i vertikalno u centar unutar gumba.

6.2.2. HTML dio

Sljedeće što slijedi u kodu je HTML dio. To je glavni dio kako će aplikacija izgledati, tj. ono što će se prikazati, ono što korisnik aplikacije vidi zapravo.

```
    <div
```

[5] SVG – značenje (https://hr.wikipedia.org/wiki/SVG)

```
    class="overlay"
    onclick="toggleSettings(false); toggleEInkPopup(false);"></div>
  <div class="app-container">
    <div id="pomodoro-section" class="app-section">
      <div class="timer-controls">
        <button class="action-button" id="pomodoro">Pomodoro</button>
        <button class="action-button" id="short-break">Short Break</button>
        <button class="action-button" id="long-break">Long Break</button>
        <div class="timer-display" id="time-left">25:00</div>
        <button class="action-button" id="start-stop">START</button>
        <button class="action-button" id="reset">RESET</button>
        <buttonclass="action-button" id="settings">Settings</button>
      </div>
```

Primjer 12. Prikaz dijela HTML koda za početnu stranicu

Primjer 12. prikazuje dio koda za početnu stranicu. Tako vidimo prvo „*overlay*“ element koji se koristi za prikaz tamnog načina na stranici kada su neki prozori (popup-i) otvoreni, što omogućuje bolju fokusiranost na te popup-e. `*App-container*` određuje glavni kontejner koji sadrži sve sekcije u aplikaciji.

`*.app-section*` određuje različite sekcije aplikacije, a u ovom slučaju vidimo da je to prikaz pomodoro-sectiona gdje imamo gumbove za Pomodoro koji traje 25 minuta, Short Break koji traje 5 minuta i Long Break koji traje 15 minuta, što određuje postavljanje različitih vremenskih intervala, prikaz brojača tj. tajmer-displaya i gumbova za start, reset i otvaranje popupa `*Settings*`.

```
      <div class="popup" id="settings-popup">
        <span class="close-btn" onclick="toggleSettings(false)">&times;</span>
        <button class="action-button" onclick="setCustomTimer()">
          Custom Timer
        </button>
        <button class="action-button" onclick="saveTasksToFile()">
          Save Tasks
        </button>
        <button class="action-button" onclick="loadTasksFromFile()">
          Load Tasks
        </button>
```

```
        <button class="action-button" onclick="testSound()">Sound</button>
        <button
          class="action-button"
          id="e-ink-button"
          onclick="toggleEInkPopup(true)">
          e-Ink
        </button>
        <div id="category-container">
          <input type="text" id="category1" placeholder="Category 1 Name" />
        </div>
        <button class="action-button" id="add-category">Add Category</button>
        <button class="action-button" onclick="updateCategories()">
          Update Categories
        </button>
        <button class="action-button" id="fullscreen-toggle">
          Full Screen
        </button>
      </div>
```

Primjer 13. Prikaz HTML-a za Settings popup

U primjeru 13. prikazan je HTML kod za prikaz Settings popupa. On se otvara ako na početnoj stranici se klikne na gumb „*Settings“* preko početne stranice. Vidimo da u Settings popupu postoje različite stvari, kao što su postavljanje prilagođenog brojača, gdje se može postaviti tajmer brojača po našoj želji za svaku vrstu, spremanje i ponovno očitavanje zadataka, gdje se podaci spremaju u json datoteku. Također, određivanje posebnog zvuka brojača i testiranje istog. Isto tako, u postavkama postoji mogućnost aktivacije E-Inka, dodavanje kategorija za zadatke, gumbovi za ažuriranje kategorija i za kraj omogućeno je ako korisnik želi da aplikaciju može postaviti preko cijelog zaslona.

```
      <div class="confirm-popup" id="e-ink-confirm-popup">
        <p>Apply grayscale filter?</p>
        <button class="action-button" onclick="applyEInkMode(true)">
          Yes
        </button>
        <button class="action-button" onclick="applyEInkMode(false)">
          No
```

```
        </button>
      </div>
      <div class="task-manager">
        <input
          type="text"
          id="task-input"
          placeholder="Enter tasks separated by ; (e.g., Task1;Task2)" />
        <button class="action-button" id="add-task">Add Tasks</button>
        <ul class="tasks" id="tasks"></ul>
      </div>
    </div>
```

Primjer 14. Prikaz popupa za E-Ink i upravljanje zadacima

U primjeru 14. je prikazan popup prozor koji se pojavi kada korisnik želi unutar `*Settings*` aktivirati E-Ink. Gdje se otvori prozor s paragrafom *„Apply grayscale filter?“* i dva gumba „*Yes*“ ili „*No*“ za potvrdu ili poništavanje. E-Ink je zapravo sivi filter koji pomaže u smanjenju naprezanja očiju. Također na slici vidimo `*div*` koji prikazuje upravljanje zadacima. Korisnici mogu unijeti podatke u polje za unos zadataka koji ima placeholder *„Enter tasks separated by; (e.g., Task1;Task2)“* gdje ih može upisati više od jednom samo je potrebna točka-zarez(;) između tih zadataka i kliknuti na gumb *„Add Tasks“* kako bi te iste zadatke dodali na listu. Nakon toga se podaci prikazuju na listi ispod u aplikaciji.

```
    <div id="mindmap-section" class="app-section">
      <svg id="mindmap"></svg>
    </div>
    <div id="mindmap-3d-section" class="app-section" style="display: none">
      <input type="file" id="fileInput" style="display: none" />
      <button id="changeDescriptionButton">
        Change Central Node Description
      </button>
      <canvas id="3d-mindmap-canvas"></canvas>
    </div>
  </div>
  <div class="footer">
    <button class="action-button" onclick="switchTo('pomodoro')">
```

```
      Pomodoro Timer
    </button>
    <button class="action-button" onclick="switchTo('mindmap')">
      Mind Map
    </button>
    <button class="action-button" onclick="switchTo('mindmap-3d')">
      3D Mind Map
    </button>
  </div>
```

Primjer 15. Prikaz Mind map-a i footera za Mind Map, Pomodoro tajmer i 3D Mind Map

U primjeru 15. je prikazan dio koji definira `*mindmap-section*` i `mindmap-3d-section` koji su početno skriveni i samo se prikazuje Pomodoro tajmer preko cijelog zaslona što je prikazano gore u CSS dijelu. Sljedeće je footer klasa gdje su gumbovi *„Pomodoro Timer“ „Mind Map“* ili *„3D Mind Map“* gdje ako se klikne na *„Pomodoro Timer“* poziva se funkcija `*switchTo(„pomodoro“)*` otvara se samo Pomodoro tajmer gdje se on prikazuje preko cijelog zaslona, a ako se klikne na gumb *„Mind Map“* poziva se funkcija `*switchTo(„mindmap“)*` gdje se otvara Mind Map s prikazom pola-pola tj. 50 % zaslona je Pomodoro tajmer, a 50 % je Mind Map. Ako se klikne na gumb *„3D Mind Map“* isto kao i kod običnog Mind Mapa, otvara se pola zaslona Pomodoro tajmer, a druga polovica 3D Mind Map. Ako se korisnik odluči da želi da mu bude samo ekran s Pomodoro tajmer opet može kliknut gumb „Pomodoro Timer“. Neovisno je li otvoren samo Pomodoro tajmer ili i Mind Map sve se automatski sinkronizira između sekcija.

6.2.3. JavaScript dio

Sljedeće što slijedi za objasniti je JavaScript dio koda. Na početku JavaScript koda su dodane dvije biblioteke - D3.js i Markmap koji omogućavaju vizualizaciju podataka i prikaz mind map-a. Sljedeće što je u kodu su globalne varijable kao što su kategorije, preimenovani zadaci i pod zadaci kako bi im se pristupilo u aplikaciji. Isto tako JavaScript je podijeljen u tri dijela, tj. tri modula, radi lakšeg snalaženja u kodu, a to su:

1.) Pomodoro Timer Module
2.) Mind Map Module
3.) 3D Mind Map Module

U ovom dijelu će biti opisano samo ukratko u nekoliko rečenica što koja funkcija radi, dok nakon ovoga dijeli slijedi prikaz funkcionalnosti baš na stranici s primjerima gdje će sve detaljnije biti objašnjeno. Isto tako u ovom dijelu se neće prikazati baš sve funkcije i cijeli kod, jer je prevelik.

```
// Function to set the timer duration
      function setTimer(type) {
        switch (type) {
          case "pomodoro":
            timerDuration = 25 * 60;
            break;
          case "short-break":
            timerDuration = 5 * 60;
            break;
          case "long-break":
            timerDuration = 15 * 60;
            break;
        }
        document.getElementById("time-left").textContent =
          formatTime(timerDuration);
        document.title = `${formatTime(timerDuration)} - Pomodoro Timer`;
      }

      // Function to set a custom timer duration
      function setCustomTimer() {
        let customTime = parseInt(prompt("Enter custom minutes:", "25"));
        if (!customTime) return;
        timerDuration = customTime * 60;
        document.getElementById("time-left").textContent =
          formatTime(timerDuration);
        document.title = `${formatTime(timerDuration)} - Pomodoro Timer`;
      }
```

Primjer 16. Funkcije za postavljanje trajanja tajmera i prilagođeni tajmer

Primjer 16. prikazuje dvije funkcije. Prva funkcija je `*setTimer*` koja postavlja trajanje vremena za 3 slučaja i to za Pomodoro, kratku i dugu pauzu. Pomodoro traje 25

minuta, kratka pauza traje 5 minuta i duga pauza traje 15 minuta. S time da se tajmer postavlja na odgovarajuće trajanje. Ova funkcija omogućava korisnicima da upravljaju svojim vremenom rada i odmora. Funkcija `*setCustomTimer*` omogućava korisnicima da postave tajmer na prilagođeno vrijeme po njihovoj želji. Tu funkciju je moguće pozvati unutar „*Settings*“ kada se klikne na gumb „*Custom Timer*“ gdje se otvara prompt s tekstom „*Enter custom minutes:*“, gdje početno bude postavljeno na 25 minuta.

```
// Function to test the sound settings
function testSound() {
  const type = prompt(
    "Enter sound type (sine, square, triangle, sawtooth):",
    currentSound.type
  );

  if (type === null) return;

  const frequencyInput = prompt(
    "Enter frequency (Hz):",
    currentSound.frequency
  );

  if (frequencyInput === null) return;

  const frequency = parseInt(frequencyInput);

  if (isNaN(frequency)) {
    alert("Invalid frequency input");
    return;
  }

  const repeatCountInput = prompt(
    "Enter repeat count (1-5):",
```

```
        repeatCount
      );

      if (repeatCountInput === null) return;

      const repeatCountValue = parseInt(repeatCountInput);

      if (isNaN(repeatCountValue)) {
        alert("Invalid repeat count input");
        return;
      }

      repeatCount = Math.max(1, Math.min(5, repeatCountValue));
      currentSound = { type, frequency };
      beep(repeatCount, currentSound.type, currentSound.frequency);
    }
```

Primjer 17. Funkcija testSound

Primjer 17. prikazuje funkciju `*testSound*` koja radi to da nakon što se otvori „*Settings*“ postoji gumb „*Sound*“ gdje nakon što se stisne taj gumb otvori se novi prompt gdje pita da se unese vrsta zvuka tj. kako će tajmer zvoniti kada istekne, a postavljene su 4 vrste (sine, square, triangle i sawtooth), gdje nakon što se to upiše otvara se novi prozor gdje pita korisnika da unese frekvenciju u Hz i nakon što to korisnik upiše otvara se još jedan prozor gdje pita koliko puta želimo da se zvuk ponavlja u rasponu od 1 do 5. Početno je postavljeno da je zvuk square 440 Hz i broj ponavljanja je 2 puta.

```
    // Function to play a beep sound
    function beep(times = 2, type = "square", frequency = 440) {
      return new Promise((resolve) => {
        const audioContext = new (window.AudioContext ||
          window.webkitAudioContext)();
        function beepOnce(timesRemaining) {
          const oscillator = audioContext.createOscillator();
          const gainNode = audioContext.createGain();
```

```
            oscillator.connect(gainNode);
            gainNode.connect(audioContext.destination);
            gainNode.gain.value = 0.1;
            oscillator.frequency.value = frequency;
            oscillator.type = type;
            oscillator.start();
            setTimeout(() => {
              oscillator.stop();
              if (timesRemaining > 1) {
                setTimeout(() => beepOnce(timesRemaining - 1), 200);
              } else {
                resolve();
              }
            }, 200);
          }
          beepOnce(times);
        });
      }

      // Function to stop the beep sound
      function stopBeep() {
        if (oscillator) {
          oscillator.stop();
          oscillator.disconnect();
          gainNode.disconnect();
          oscillator = null;
          gainNode = null;
        }
      }
```

Primjer 18. Funkcije za reprodukciju zvuka i za zaustavljanje zvuka

Primjer 18. prikazuje dvije funkcije `*beep*` i `*stopBeep*`, gdje funkcija `*beep*` proizvodi zvuk nakon što tajmer istekne. Koristi Web Audio API za proizvodnju zvuka, koji se oglasi nekoliko puta kako bi korisnik bio obaviješten da je određena sesija završila. Ova funkcionalnost omogućava korisniku da kada nije otvoren prozor u kojem je korisnik da čuje da je sesija završila. Dok funkcija `*stopBeep*` prekida zvuk tajmera

što i samo ime te funkcije kaže. Bez te funkcije bi se zvuk stalno čuo i ne bi se nikad ugasio.

```
// Function to toggle the timer state
      function toggleTimer() {
        if (isTimerRunning) {
          clearInterval(timer);
          isTimerRunning = false;
          document.getElementById("start-stop").textContent = "START";
        } else {
          timer = setInterval(() => {
            if (timerDuration > 0) {
              timerDuration--;
              document.getElementById("time-left").textContent =
                formatTime(timerDuration);
              document.title = `${formatTime(timerDuration)} - Pomodoro Timer`;
            } else {
              clearInterval(timer);
              isTimerRunning = false;
              document.getElementById("start-stop").textContent = "START";
              document.title = "Time's up! - Pomodoro Timer";
              beep(repeatCount, currentSound.type, currentSound.frequency).then(
                () => {
                  alert("Time's up!");
                }
              );
            }
          }, 1000);
          isTimerRunning = true;
          document.getElementById("start-stop").textContent = "STOP";
        }
      }
```

Primjer 19. Funkcija toggleTimer

Primjer 19. prikazuje funkciju `*toggleTimer*` i ona upravlja pokretanjem i zaustavljanjem brojača Pomodoro tajmera. Ako je tajmer pokrenut ova funkcija koristi

`*setInterval*` za odbrojavanje vremena u intervalima od 1 sekunde i svaku sekundu smanjuje. Ako je tajmer zaustavljen interval se briše pomoću `*clearInterval*`. A nakon što tajmer istekne prikazuje se alert *„Time's up“* i čuje se zvuk tajmera da je istekao. Također kako je rečeno, ako je korisnik u drugom prozoru, u web browseru, njemu stoji u imenu prozora u kojem mu je otvorena Pomodoro aplikacija koliko je vremena ostalo i to se ažurira svaku sekundu, i ako istekne tajmer stoji u nazivu „*Time's up-Pomodoro Timer“* što je omogućeno u funkciji `*toggleTimer*`. Tako da korisnik zna u svakom trenutku koliko još ima vremena za određenu sesiju ili dali je ona istekla da se može maksimalno koncentrirati na zadatke ili učenje.

U primjeru 20. je prikazana funkcija `*saveTaskToFile*` koja kao što i samo ime kaže sprema trenutni popis zadataka i pod zadataka i umnu mapu u .JSON datoteku. Korisnici mogu s prečacem *„CTRL + S“* na tipkovnici lakše spremiti tu datoteku. Također unutar „*Settings*“ korisnici mogu stisnuti gumb *„Save Tasks“* koji će im istu stvar napraviti, tj. preuzeti trenutne zadatke, umnu mapu i 3D umnu mapu kako bi ju kasnije mogli očitati. Ova funkcija omogućava jednostavno spremanje i kasnije ponovno očitavanje zadataka da ništa ne propuste napraviti. Također ova funkcija omogućava da se spremi glavni tekst, ako ima kakvih pod zadataka, dali je zadatak izvršen ili nije, kategorije i vrijeme u koje je kreiran, sprema koordinate čvorova i pod čvorova u 3D Mind mapu. Spremljeni dokument ima naziv `*all_data*` s ekstenzijom .JSON.

```
// Function to save tasks to a file
      function saveTasksToFile() {
        const tasksList = document.getElementById("tasks");
        const tasks = [];
        tasksList.childNodes.forEach((task) => {
          const taskText = task.querySelector(".task-text").textContent;
          const completed = task.querySelector(
            "input[type='checkbox']"
          ).checked;
          const subtasks = Array.from(
            task.querySelector(".subtasks").children
          ).map(
            (subtask) => subtask.querySelector(".subtask-text").textContent
          );
```

```
      const category = task.querySelector(".task-category").value;
      const timestamp = task.querySelector(".task-timestamp").textContent;
      tasks.push({
        text: taskText,
        completed: completed,
        subtasks: subtasks,
        category: category,
        timestamp: timestamp,
      });
    });

    const data = {
      tasks: tasks,
      categories: categories,
      mindMap: rootNode,
      nodes: nodes.map((node) => ({
        x: node.x,
        y: node.y,
        z: node.z,
        text: node.text,
        parent: node.parent
          ? nodes.indexOf(nodes.find((n) => n.mesh === node.parent))
          : null,
        color: node.color,
        size: node.size,
      })),
      lines: lines.map((line) => ({
        startNode: nodes.indexOf(nodes.find((n) => n.mesh === line.node1)),
        endNode: nodes.indexOf(nodes.find((n) => n.mesh === line.node2)),
        color: line.line.material.color.getHex(),
      })),
    };

    const jsonString = JSON.stringify(data);
    const blob = new Blob([jsonString], { type: "application/json" });
```

```
      const link = document.createElement("a");
      link.href = URL.createObjectURL(blob);
      link.download = "all_data.json";
      link.click();
    }
```

Primjer 20. Funkcija za spremanje zadataka

Isto tako ima funkcija za očitavanje zadataka `*loadTaskFromFile*` koja omogućava korisnicima da očitavaju zadatke i umnu mapu iz .JSON datoteke (primjer 21.). Ova funkcija omogućava da korisnici mogu ponovno učitati zadatke koje možda nisu izvršili ili su ih ostavili za kasnije. Ta funkcija očitava kategorije zadataka ako su dodane, dali je neki zadatak izvršen i njegove pod zadatke ako ih ima. Također kao i što ima prečac za spremanje zadataka, tako postoji i prečac za očitavanje zadataka a to se postiže pritiskom na tipkovnici *„CTRL + O“* ili unutar „*Settings*“ ima gumb „*Load Task*s“ gdje kada se klikne na njega može se očitati na isti način kao i sa prečacem, određene zadatke i umnu mapu.

```
// Function to load tasks from a file
   function loadTasksFromFile() {
     const input = document.createElement("input");
     input.type = "file";
     input.accept = ".json";
     input.onchange = (e) => {
       const file = e.target.files[0];
       const reader = new FileReader();
       reader.onload = (event) => {
         const data = JSON.parse(event.target.result);

         // Load categories
         const newCategories = data.categories || {};
         categories = { ...categories, ...newCategories };
         const categoryContainer =
           document.getElementById("category-container");
         categoryContainer.innerHTML = "";
         for (let category in categories) {
           const categoryInput = document.createElement("input");
```

```
    categoryInput.type = "text";
    categoryInput.value = category;
    categoryContainer.appendChild(categoryInput);
  }
  const categoryInput = document.createElement("input");
  categoryInput.type = "text";
  categoryInput.placeholder = "Add new category";
  categoryContainer.appendChild(categoryInput);

  // Load tasks
  const tasksList = document.getElementById("tasks");
  data.tasks.forEach((task) => {
    const existingTask = Array.from(tasksList.childNodes).find(
      (li) => li.querySelector(".task-text").textContent === task.text
    );
    if (!existingTask) {
      const formattedTimestamp = task.timestamp
        ? ` ${task.timestamp.replace(" ", "")}`
        : "";
      createTaskElement(
        task.text,
        tasksList,
        task.completed,
        formattedTimestamp,
        task.category
      );
      task.subtasks.forEach((subtask) => {
        const parentLi = Array.from(tasksList.childNodes).find(
          (li) =>
            li.querySelector(".task-text").textContent === task.text
        );
        if (parentLi) {
          createSubtaskElement(
            subtask,
            parentLi.querySelector(".subtasks")
```

```
                );
              }
            });
          }
        });

        // Load 2D mind map
        rootNode = data.mindMap || rootNode;
        if (mindMapInitialized) {
          mm.setData(rootNode);
          mm.fit();
        }

        // Load 3D mind map
        loadScene(data.nodes, data.lines);

        updateTaskCategoryDropdowns();
      };
      reader.readAsText(file);
    };
    input.click();
  }
```

Primjer 21. Funkcija za očitavanje zadataka i pod zadataka

U primjeru 22. su prikazane dvije funkcije `*addTask*` i `*addSubtask*`. Funkcija `*addTask*` dodaje nove zadatke u listu koja je unutar Pomodoro sekcije i poziva funkciju koja sinkronizira zadatke. Zadaci se mogu dodati odjednom tako da zadatke samo razdvojimo s točkom-zarez (;). Funkcija `*addSubtask*` dodaje pod zadatke u listu tako da se otvori novi prozor u kojem pita *„Enter subtask*“ i otvara se polje za unos teksta, nakon što se unese tekst klikne se samo „Enter“ na tipkovnici i on će se dodati. Ako korisnik pokuša dodati pod zadatak koji već postoji, pojavit će se alert *„Subtask already exist.*“.

```
  // Function to add tasks to the task list
  function addTasks(taskInput) {
    const tasksList = document.getElementById("tasks");
```

```
    const tasks = taskInput.split(";");
    tasks.forEach((task) => {
      if (task.trim() !== "") {
        synchronizeTask(task.trim(), "pomodoro");
      }
    });
    document.getElementById("task-input").value = "";
  }

  // Function to add a subtask to a task
  function addSubtask(subtasksList, parentTask) {
    const subtaskText = prompt("Enter subtask:");
    if (!subtaskText) return;

    // Check if the parentTask is the "Central Node" or a subtask itself
    if (parentTask === rootNode.content) {
      alert("You cannot add subtasks to the Central Node.");
      return;
    }
    const parentTaskElement = Array.from(
      document.querySelectorAll(".task-text")
    ).find((task) => task.textContent === parentTask);
    const isSubtask =
      parentTaskElement && parentTaskElement.closest(".subtask-item");

    if (isSubtask) {
      alert("Subtasks cannot have new subnodes.");
      return;
    }

    // Check if subtask already exists
    const existingSubtask = Array.from(subtasksList.children).find(
      (subtask) =>
        subtask.querySelector(".subtask-text").textContent === subtaskText
    );
```

```
      if (existingSubtask) {
        alert("Subtask already exists.");
        return;
      }

      createSubtaskElement(subtaskText, subtasksList);

      parentTask = getUpdatedTaskDescription(parentTask);
      const parentNode = findNodeByContent(rootNode, parentTask);
      if (parentNode) {
        const newSubchild = { content: subtaskText, children: [] };
        parentNode.children.push(newSubchild);
        if (mindMapInitialized) {
          mm.setData(rootNode);
          mm.fit();
        }
      }

      // Prevent adding the same task to the mind map
      const existingNode = findNodeByContent(rootNode, subtaskText);
      if (!existingNode) {
        synchronizeTask(subtaskText, "pomodoro", parentTask);
      }

      // Synchronize with 3D Mind Map
      synchronize3DMindMap(subtaskText, parentTask);
    }
```

Primjer 22. Funkcije za dodavanje zadataka i pod zadataka

Nakon toga imamo funkcije `*createTaskElement*` i `*createSubtaskElement*`, koje neće biti prikazane u kodu jer su prevelike funkcije, nego će se samo opisati u nekoliko rečenica što one rade. Prva je na redu `*createTaskElement* ` ona dinamički stvara HTML elemente za zadatke i pod zadatke, također ona u sebi ima okvir za status zadatka dali je on izvršen ili nije, opis zadatka, vrijeme u koje je kreiran i kategorije koje se prikazuju u padajućem izborniku. Funkcija još sadrži gumbove kao

što su strelice za pomicanje zadataka gore/dole, brisanje zadataka i uređivanje što je omogućeno duplim klikom na tekst.

Isto tako funkcija ` *createSubtaskElement*` ima za cilj pravilno dodati i strukturirati pod zadatak, da se on može uređivati i brisati.

```
   //Shows or hides a settings popup and an overlay layer
      function toggleSettings(show) {
        const settingsPopup = document.getElementById("settings-popup");
        const overlay = document.querySelector(".overlay");
        if (show) {
          settingsPopup.style.display = "flex";
          overlay.style.display = "block";
        } else {
          settingsPopup.style.display = "none";
          overlay.style.display = "none";
        }
      }


       //Displays or hides an E-Ink confirmation popup and its corresponding overlay
layer
      function toggleEInkPopup(show) {
        const eInkPopup = document.getElementById("e-ink-confirm-popup");
        const overlay = document.querySelector(".overlay");
        if (show) {
          eInkPopup.style.display = "flex";
          overlay.style.display = "block";
        } else {
          eInkPopup.style.display = "none";
          overlay.style.display = "none";
        }
      }
```

Primjer 23. Funkcije za prikaz popupa za postavke i E-Ink

Sljedeće što vidimo (primjer 23.) su funkcije za prikazivanje ili skrivanje popupova za postavke i E-Ink. Obadvoje je početno skriveno. Funkcija `*toggleSettings*` prekriva ili

skriva popup prozor postavki, dok funkcija `*toggleElnkPopup*` omogućuje da korisnik vidi skočni prozor za aktivaciju E-Inka.

Mogu se još spomenuti funkcije kao što su `*addCategory*`, `*updateCategories*` i `*updateTaskCategoryDropdowns*`. Funkcija `*addCategory*` dodaje novo polje za unos kategorija, a funkcije `*updateCategories*` i `*updateTaskCategoryDropdowns*` omogućavaju da je popis kategorija stalno ažuriran i da se prikazuje u padajućem izborniku.

```
// Function to switch between Pomodoro, Mind Map, and 3D Mind Map sections
function switchTo(section) {
  const pomodoroSection = document.getElementById("pomodoro-section");
  const mindmapSection = document.getElementById("mindmap-section");
  const mindmap3DSection = document.getElementById("mindmap-3d-section");

  // Hide all sections initially
  pomodoroSection.style.display = "none";
  mindmapSection.style.display = "none";
  mindmap3DSection.style.display = "none";

  pomodoroSection.style.width = "100%";

  if (section === "mindmap") {
    mindmapSection.style.display = "flex";
    pomodoroSection.style.display = "flex";
    pomodoroSection.style.width = "50%";
    mindmapSection.style.width = "50%";
    initializeMindMap();
    mindMapInitialized = true;
  } else if (section === "pomodoro") {
    pomodoroSection.style.display = "flex";
  } else if (section === "mindmap-3d") {
    mindmap3DSection.style.display = "flex";
    pomodoroSection.style.display = "flex";
    pomodoroSection.style.width = "50%";
    mindmap3DSection.style.width = "50%";
```

```
        initialize3DMindMap();
        mindmap3dInitialized = true;
      }
    }
```

Primjer 24. Funkcija za prebacivanje između modula

Primjer 24. prikazuje funkciju `*switchTo*` koja je napravljena da se može prebacivati između modula web stranice. Potreban je parametar koji se naziva `*section*` i on određuje kako će se određeni modul prikazivati. Na početku skriva sve module, osim Pomodoro tajmera i on je postavljen da mu je početna širina 100 %. Ako se klikne na `*section*` „mindmap“ onda se prikazuju Mind map i Pomodoro tajmer gdje im je širina 50 % za svaki modul. Ako je `*section*` Pomodoro onda se prikazuje samo Pomodoro tajmer, a ako je `*section*` „mindmap-3d“ onda se prikazuju i 3D Mind Map i Pomodoro tajmer gdje je širina svakog modula 50 %. Također ova funkcija inicijalizira i Mind Map i 3D Mind Map i postavlja njihove oznake na `*true*`.

Modul Mind Map je važan za aplikaciju jer korisnicima ove aplikacije omogućava da imaju pregled svojih zadataka u Mind Mapu. Ovaj dio koda, tj. modul sadržava funkcije za pokretanje, ažuriranje i sinkronizaciju s Pomodoro tajmerom i 3D Mind Map. Nakon kratkog uvoda, prikazat će se ključne funkcije i opisati iste.

```
// Mind Map Module


// Initialize root node for mind map
let rootNode = {
  content: "Mind Map Root",
  children: [],
};
let mm;
let mindMapInitialized = false;
```

Primjer 25. Inicijalizacija Mind Map-a

Primjer 25. prikazuje kako je inicijaliziran Mind Map da bi se ispravno prikazivao u aplikaciji i postavio početni čvor koji se naziva *„Mind Map Root“* koji nema nikakve podređene čvorove na početku. Deklarirana je varijabla `*mm*` koja će sadržavati instancu Markmapa i varijabla `*mindMapInitialized*` prati dali je Mind Map inicijaliziran ili nije, a početno je postavljeno da nije.

Sljedeće što treba je inicijalizirati Mind Map uz pomoć Markmapa. Markmap je biblioteka koja omogućava izradu Mind Mapa. Funkcija `*initializeMindMap*` prvo briše sav sadržaj koji postoji u SVG elementu, a onda nakon toga stvara instancu Mind Mapa (primjer 26.). Unutar funkcije dodani su slušatelji događaja (engl. *event listener*) uz pomoć kojih se dodaju novi čvorovi (CTRL + lijevi klik miša unutar Mind Mapa) i pod čvorovi (desni klik miša na određeni čvor za koji želimo dodati pod čvor gdje se za oba slučaja otvaraju prozori gdje se može upisati tekst.

```
// Function to initialize mind map
function initializeMindMap() {
  const markmap = window.markmap;
  const svg = document.querySelector("svg#mindmap");

  while (svg.firstChild) {
    svg.removeChild(svg.firstChild);
  }

  mm = markmap.Markmap.create(
    "svg#mindmap",
    markmap.deriveOptions(null),
    rootNode
  );

  const mindmapSvg = document.getElementById("mindmap");

  mindmapSvg.addEventListener("click", function (event) {
    if (event.ctrlKey) {
      const newNodeContent = prompt("Enter new node content:");
      if (newNodeContent && newNodeContent.trim() !== "") {
        const newNode = {
          content: newNodeContent,
          children: [],
        };
        rootNode.children.push(newNode);
        mm.setData(rootNode);
        mm.fit();
```

```
            synchronizeTask(newNodeContent, "mindmap");
            event.stopImmediatePropagation(); // Prevent further handling
          }
        }
      });

      mindmapSvg.addEventListener("contextmenu", function (event) {
        event.preventDefault();
        const target = event.target;
        if (
          target.tagName.toLowerCase() === "div" &&
          target.parentNode.tagName.toLowerCase() === "foreignobject"
        ) {
          const selectedNode = target.textContent;
          const selectedParent = rootNode.children.find(
            (node) => node.content === selectedNode
          );
          if (selectedParent) {
            const newSubchildContent = prompt(
              `Enter new subchild content for ${selectedNode}:`
            );
            if (newSubchildContent && newSubchildContent.trim() !== "") {
              const newSubchild = {
                content: newSubchildContent,
                children: [],
              };
              selectedParent.children.push(newSubchild);
              mm.setData(rootNode);
              mm.fit();
              synchronizeTask(newSubchildContent, "mindmap", selectedNode);
              event.stopImmediatePropagation(); // Prevent further handling
            }
          }
        }
      });
```

```
    }
```

Primjer 26. Funkcija za inicijalizaciju Mind mapa

U primjeru 27. je funkcija `*synchronizeTask*` koja omogućuje sinkronizaciju između sva tri modula, odnosno između Pomodoro tajmera, Mind Mapa i 3D Mind Mapa. Kada se neki zadatak kreira nebitno u kojem modulu on dodaje datum i vrijeme kreiranja zadatka unutar Pomodoro tajmera. Nakon toga provjerava dali taj zadatak već postoji u listi unutar Pomodoro tajmera, osigurava da je 3D Mind Map inicijaliziran, ako nije onda ga inicijalizira. Isto tako provjerava za sve dali već zadatak postoji, ako postoji neće ga opet dodati. Ako ne postoji onda taj zadatak dodaje u sva tri modula.

```
// Function to synchronize tasks between mind map and task list
    function synchronizeTask(task, source, parentTask = null) {
      if (!mindMap3DInitialized) {
        initialize3DMindMap();
      }

      const tasksList = document.getElementById("tasks");
      const currentTime = formatDate(new Date());

      // Check if task already exists in the tasks list
      const existingTask = Array.from(tasksList.childNodes).find(
        (li) => li.querySelector(".task-text").textContent === task
      );

      // Add task to the Pomodoro timer only if it doesn't exist
      if (!existingTask) {
        if (parentTask) {
          parentTask = renamedTasks[parentTask] || parentTask;
          const parentTaskElement = Array.from(tasksList.childNodes).find(
            (li) => li.querySelector(".task-text").textContent === parentTask
          );
          if (parentTaskElement) {
            const subtasksList = parentTaskElement.querySelector(".subtasks");
            createSubtaskElement(task, subtasksList);
```

```
      }
    } else {
      createTaskElement(task, tasksList, false, currentTime);
    }
  }

  // Add task to the regular Mind Map only if it doesn't exist
  const existingNode = findNodeByContent(rootNode, task);
  if (!existingNode) {
    const newNodeContent = task;
    const newNode = { content: newNodeContent, children: [] };
    if (parentTask) {
      parentTask = renamedTasks[parentTask] || parentTask;
      const parentNode = findNodeByContent(rootNode, parentTask);
      if (parentNode) {
        parentNode.children.push(newNode);
      }
    } else {
      rootNode.children.push(newNode);
    }
    if (mindMapInitialized) {
      mm.setData(rootNode);
      mm.fit();
    }
  }

  // Add task to the 3D Mind Map only if it doesn't exist
  const existing3DNode = nodes.find((node) => node.text === task);
  if (!existing3DNode) {
    addNodeTo3DMindMap(task, parentTask);
    if (mindMap3DInitialized) {
      animate();
    }
  }
```

```
}
```

Primjer 27. Funkcija za sinkronizaciju zadataka

Funkcija `synchronizeTaskUpdate` (primjer 28.) sinkronizira i ažurira opise zadataka za sva tri modula i osigurava dosljednost.

```
// Function to synchronize updates to a task's description
function synchronizeTaskUpdate(oldTask, newTask) {
  renamedTasks[oldTask] = newTask;

  // Update task in the Pomodoro task list
  const tasksList = document.getElementById("tasks");
  const taskItem = Array.from(tasksList.childNodes).find(
    (li) => li.querySelector(".task-text").textContent === oldTask
  );
  if (taskItem) {
    taskItem.querySelector(".task-text").textContent = newTask;
    taskItem.querySelector(".task-text").ondblclick = () =>
      editTask(newTask, taskItem, taskItem.querySelector(".task-text"));
  }

  // Update task in the regular Mind Map
  function updateNodeContent(node, oldContent, newContent) {
    if (node.content === oldContent) {
      node.content = newContent;
      return true;
    }
    for (const child of node.children) {
      if (updateNodeContent(child, oldContent, newContent)) {
        return true;
      }
    }
    return false;
```

```
    }
    if (
      updateNodeContent(rootNode, oldTask, newTask) &&
      mindMapInitialized
    ) {
      mm.setData(rootNode);
      mm.fit();
    }

    // Update task in the 3D Mind Map
    const node = nodes.find((n) => n.text === oldTask);
    if (node) {
      node.text = newTask;
      node.label.element.textContent = newTask;
    }
  }
```

Primjer 28. Funkcija za ažuriranje opisa zadataka

Isto tako ima funkcija `*synchronizeSubtaskUpdate*` koja sve isto radi, kao i `*synchronizeTaskUpdate*` samo za pod zadatke.

```
  // Function to delete a node from the mind map
  function deleteMindMapNode(content) {
    function deleteNode(node, content) {
      const index = node.children.findIndex(
        (child) => child.content === content
      );
      if (index !== -1) {
        node.children.splice(index, 1);
        return true;
      }
      for (const child of node.children) {
        if (deleteNode(child, content)) {
          return true;
        }
```

```
      }
      return false;
    }

    if (deleteNode(rootNode, content) && mindMapInitialized) {
      mm.setData(rootNode);
      mm.fit();
    }

    // Also delete the node in the 3D Mind Map
    delete3DNode(content);
    // Also delete the task in the Pomodoro
    deleteTaskInPomodoro(content, true);
  }
```

Primjer 29. Funkcija za brisanje čvorova unutar Mind Mapa

Primjer 29. prikazuje funkciju `*deleteMindMapNode*` koja omogućava brisanje čvorova iz Mind Mapa. Ona rekurzivno traži čvor s navedenim opisom, briše ga iz Mind Mapa, Pomodoro tajmera i 3D Mind Mapa i ažurira prikaz.

U primjeru 30. je prikazana funkcija `*initialize3DMindMap()*` koja postavlja 3D Mind Map. Ona inicijalizira sve elemente koji su ključni, pa tako inicijalizira kameru, scenu, renderer i kontrole, postavlja rasvjetu i kreira glavni čvor ako još uvijek ne postoji. Isto tako ovdje su postavljeni slušatelji događaja (*event listeneri*) koji služe za dodavanje ili brisanje čvorova unutar 3D Mind Mapa. Ova funkcija osigurava da se 3D Mind Map inicijalizira samo jednom, gdje postavlja `*mindMap3DInitialized*` na `*true*` nakon što se funkcija izvrši.

```
  //Initializes the 3D mind map
  function initialize3DMindMap() {
    if (mindMap3DInitialized) return; // Initialize only once

    const canvas = document.getElementById("3d-mindmap-canvas");
    scene = new THREE.Scene();
    camera = new THREE.PerspectiveCamera(
      75,
      canvas.clientWidth / canvas.clientHeight,
```

```
    0.1,
    100
  );
  renderer = new THREE.WebGLRenderer({ canvas: canvas });
  renderer.setSize(canvas.clientWidth, canvas.clientHeight);

  labelRenderer = new THREE.CSS2DRenderer();
  labelRenderer.setSize(canvas.clientWidth, canvas.clientHeight);
  labelRenderer.domElement.style.position = "absolute";
  labelRenderer.domElement.style.top = "0px";
  labelRenderer.domElement.style.pointerEvents = "none";
  document
    .getElementById("mindmap-3d-section")
    .appendChild(labelRenderer.domElement);

  const centralNodePosition = new THREE.Vector3(0, 0, 0);
  camera.position.set(0, 0, 10);
  const ambientLight = new THREE.AmbientLight(0x404040);
  scene.add(ambientLight);
  const directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);
  directionalLight.position.set(0, 1, 1);
  scene.add(directionalLight);

  controls = new THREE.OrbitControls(camera, renderer.domElement);
  controls.target.copy(centralNodePosition);
  controls.enableDamping = true;
  controls.dampingFactor = 0.05;

  if (!nodes.some((node) => node.text === "Central Node")) {
    addNode(0, 0, 0, rootNode.content, null, 0xff0000, {
      width: 1.5,
      height: 1.5,
      depth: 1.5,
    });
  }
```

```
renderer.domElement.addEventListener("contextmenu", (event) => {
  event.preventDefault();
});

// Event listener for 3D Mind Map to add nodes and subnodes
renderer.domElement.addEventListener("click", (event) => {
  const selectedNode = findObjectUnderMouse(event);
  const parentNode = selectedNode
    ? nodes.find((node) => node.mesh === selectedNode)
    : null;

  if (event.altKey && event.button === 0) {
    if (parentNode) {
      // Check if the parent node is the "Central Node"
      if (parentNode.text === rootNode.content) {
        alert("You cannot add subnodes to the Central Node.");
        return;
      }

      const text = prompt("Enter subnode content:");
      if (!text) return;

      // Check if the parent node is a subnode itself
      const isSubnode = parentNode && parentNode.parent !== null;
      if (isSubnode) {
        alert("Subnodes cannot have new subnodes.");
        return;
      }

      const position = calculateNewPosition(parentNode);
      addNode(
        position.x,
        position.y,
        position.z,
```

```
            text,
            selectedNode,
            Math.random() * 0xffffff,
            { width: 1, height: 1, depth: 1 }
          );

          synchronizeTask(text, "3dmindmap", parentNode.text);
        }
      } else if (event.ctrlKey && event.button === 0 && !parentNode) {
        const text = prompt("Enter node content:");
        if (!text) return;

        const position = calculateNewPosition(null);
        addNode(position.x, position.y, position.z, text);

        synchronizeTask(text, "3dmindmap");
      } else if (event.shiftKey) {
        if (parentNode && parentNode.text !== rootNode.content) {
          removeNode(parentNode);
          deleteTaskInPomodoro(parentNode.text);
        }
      }
    });

    animate();
    mindMap3DInitialized = true; // Set to true after initialization
  }
```

Primjer 30. Funkcija za inicijalizaciju 3D Mind Mapa

Unutar funkcije `*initialize3DMindMap*` je isto tako i slušatelj događaja koji upravlja klikovima unutar 3D Mind Mapa (na primjer: ALT, CTRL, SHIFT,…), a oni dodavaju nove čvorove, pod čvorove ili brišu iste.

Primjer 31. prikazuje funkciju `*addNode*` koja dodaje nove čvorove u 3D Mind Map, gdje stvara oblik koji je 3D i dodaje ga unutar scene i dodaje tekst tome čvoru. Ako čvor ima „roditelja“ onda dodaje liniju koja povezuje čvor s „roditeljem“.

```
//Adds a node to the 3D scene.
function addNode(
  x,
  y,
  z,
  text,
  parent = null,
  color = Math.random() * 0xffffff,
  size = { width: 1.5, height: 1, depth: 0.5 }
) {
  const geometry = new THREE.BoxGeometry(
    size.width,
    size.height,
    size.depth
  );
  const material = new THREE.MeshPhongMaterial({ color: color });
  const node = new THREE.Mesh(geometry, material);
  node.position.set(x, y, z);
  scene.add(node);

  const div = document.createElement("div");
  div.className = "label";
  div.textContent = text;
  const label = new THREE.CSS2DObject(div);
  label.position.set(0, size.height / 2 + 0.1, 0);
  node.add(label);

  nodes.push({
    mesh: node,
    x,
    y,
    z,
    text,
    parent,
    color,
```

```
        label,
        size,
      });

      if (parent) {
        const parentNode = nodes.find((n) => n.mesh === parent);
        addLine(parentNode.mesh, node, 0xffff00); // Yellow for subnodes
      } else {
        const centralNode = nodes.find((n) => n.text === rootNode.content);
        if (centralNode) {
          addLine(centralNode.mesh, node, 0x0000ff); // Blue for main nodes
        }
      }
    }
```

Primjer 31. Funkcija za dodavanje čvorova

Funkcija `*addLine*` u primjeru 32. dodaje linije između dva čvora u 3D Mind Mapu. Ona stvara liniju s bojom koja se mijenja ovisno dali je čvor ili pod čvor. Ova funkcija je omogućena zbog vizualnog povezivanja čvorova, gdje označuje odnose između čvorova.

```
//Adds a line between two nodes in the 3D scene.
    function addLine(node1, node2, color) {
      const material = new THREE.LineBasicMaterial({ color: color });
      const points = [
        new THREE.Vector3(
          node1.position.x,
          node1.position.y,
          node1.position.z
        ),
        new THREE.Vector3(
          node2.position.x,
          node2.position.y,
          node2.position.z
        ),
      ];
      const geometry = new THREE.BufferGeometry().setFromPoints(points);
```

```
        const line = new THREE.Line(geometry, material);
        scene.add(line);
        lines.push({ line, node1, node2 });
      }

        //Removes a node from the 3D scene, including its subnodes and associated
lines.
      function removeNode(node) {
          if (node.text === "Central Node") return; // Prevent deleting the central
node

        const subnodes = nodes.filter((n) => n.parent === node.mesh);
        subnodes.forEach((subnode) => removeNode(subnode));

        for (let i = lines.length - 1; i >= 0; i--) {
          if (lines[i].node1 === node.mesh || lines[i].node2 === node.mesh) {
            scene.remove(lines[i].line);
            lines.splice(i, 1);
          }
        }

        node.mesh.remove(node.label);
        scene.remove(node.mesh);
        const nodeIndex = nodes.findIndex((n) => n.mesh === node.mesh);
        if (nodeIndex !== -1) {
          nodes.splice(nodeIndex, 1);
        }
      }
```

Primjer 32. Funkcija za dodavanje linija među čvorovima

Primjer 33. prikazuju funkciju `*removeNode*` koja uklanja čvorove iz 3D Mind Mapa, zajedno s njegovim linijama i pod čvorovima ako ih ima. Isto tako ova funkcija osigurava da glavni čvor u 3D Mind Mapu se ne može obrisati. Pod čvorovi se rekurzivno uklanjaju, a sve linije koje su povezane s čvorom se isto uklanjaju. Ova funkcija važna je za upravljanje strukturom 3D Mind Mapa.

```
        //Removes a node from the 3D scene, including its subnodes and associated
lines.
```

```
      function removeNode(node) {
          if (node.text === "Central Node") return; // Prevent deleting the central
node


        const subnodes = nodes.filter((n) => n.parent === node.mesh);
        subnodes.forEach((subnode) => removeNode(subnode));


        for (let i = lines.length - 1; i >= 0; i--) {
          if (lines[i].node1 === node.mesh || lines[i].node2 === node.mesh) {
            scene.remove(lines[i].line);
            lines.splice(i, 1);
          }
        }


        node.mesh.remove(node.label);
        scene.remove(node.mesh);
        const nodeIndex = nodes.findIndex((n) => n.mesh === node.mesh);
        if (nodeIndex !== -1) {
          nodes.splice(nodeIndex, 1);
        }
      }
```

Primjer 33. Funkcija za uklanjanje čvorova u 3D Mind Mapu

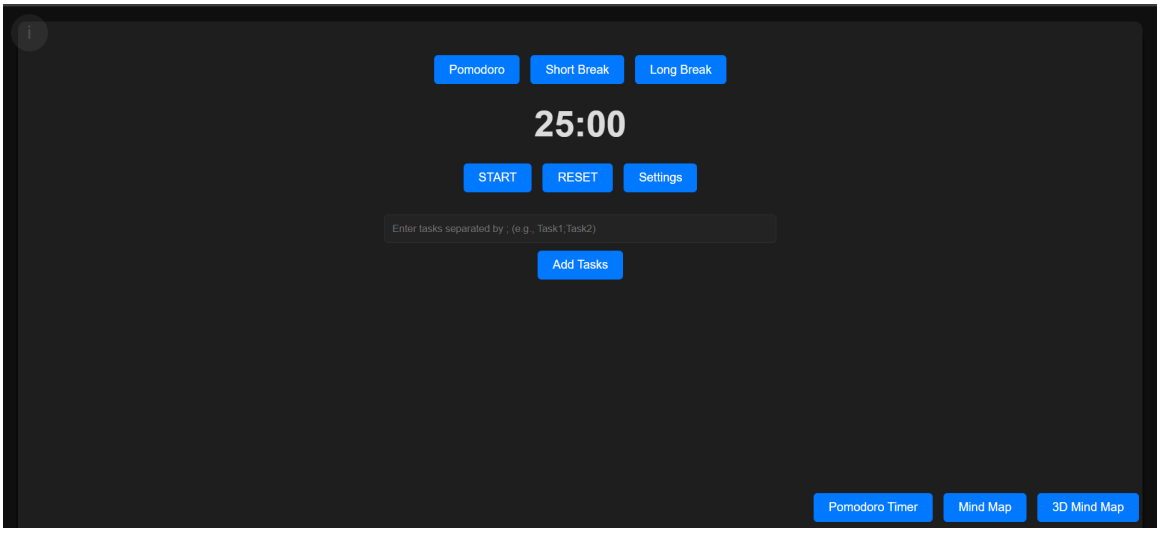
Funkcija `*animate*` ažurira prikaz u 3D Mind Mapu kako bi se se sve glatko ažuriralo u 3D Mind Map, a za to koristi `*requestAnimationFrame*` (primjer 34.).

```
      //Animates the scene by continuously rendering it
      function animate() {
        requestAnimationFrame(animate);
        controls.update();
        renderer.render(scene, camera);
        labelRenderer.render(scene, camera);
      }
```

Primjer 34. Funkcija za interakcije

6.3. Prikaz izgleda aplikacije

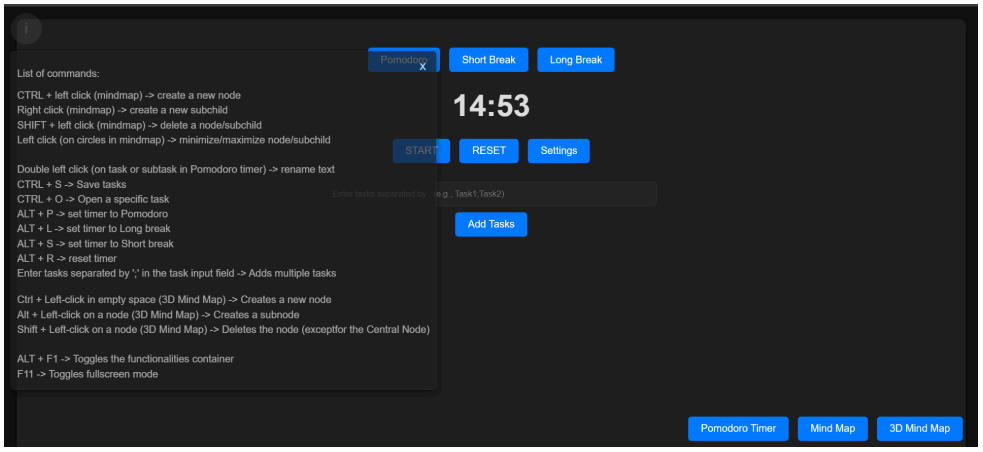
Kada se tek otvori aplikacija vidljiva je samo Pomodoro sekcija (slika 3.) Gdje se unutar Pomodoro sekcije prikazuju 3 gumba, a to su za Pomodoro, kratku i dugu pauzu i njih smo definirali unutar HTML dijela kao `*action-button*. Ispod toga je prikazan displej za odbrojavanje vremena koji se unutar HTML dijela naziva `*timer-display*`. Poslije displeja su tri gumba za *START/STOP, RESET i Settings.* Kada se klikne na gumb *Settings* otvara se novi prozor iznad sekcija, gdje se mogu prilagođavati stvari kako osoba želi, a one će biti kasnije opisane. Kada se klikne na gumb *Settings* poziva se funkcija `*toggleSettings*`. Onda tako ispod njih je polje za unos zadataka, gdje se mogu upisivati zadaci, a ako se upisuje više zadataka samo između zadataka se postavi „točka zarez“(;). To polje za unos podataka je definirano kao `*task-input*` unutar HTML i CSS koda. Ispod njega je *gumb „Add Tasks“* koji omogućava kada se klikne da se zadaci dodaju na listu. Da se ne mora kliknuti gumb, može se pritisnuti samo *„ENTER*“ na tipkovnici i zadaci će se dodati na listu. U desnom donjem rubu postoje 3 gumba za premještanje između modula. Prvi gumb je *„Pomodoro Timer“* i on je početno postavljen da bude prikazan, nakon njega *„Mind Map“* koji kada se klikne na njega poziva funkciju `*switchTo(mindmap)*` i onda se zaslon podijeli na 50 % Pomodoro tajmera, a 50 % Mind Map. Na lijevoj strani će se prikazati Pomodoro tajmer, a na desnoj Mind Map. I isto tako postoji i gumb „*3D Mind Map*“ koji poziva funkciju `*switchTo(mindmap-3d)*` i isto kao i Mind Map, 50 % na lijevoj strani zaslona je Pomodoro tajmer, a drugih 50 % zaslona je prikazan 3D Mind Map i to desno. I za kraj u gornjem lijevom kutu je gumb za prikaz svih funkcionalnosti i prečaca da se korisnik može lakše snaći, a njega se može otvoriti i pomoću prečaca na tipkovnici koristeći *„ALT + F1“*. On je definiran kao `*round-button*` unutar HTML i CSS dijela koda koji su ranije objašnjeni detaljno. I njega možemo bilo kada otvoriti, nebitno u kojem smo modulu (slika 4.).



Slika 3. Početna stranica aplikacije

Početno je vrijeme brojača postavljeno na 25 minuta, a klikom na gumbove iznad tajmera, će se ažurirati i vrijeme na displeju. Za promjenu vremena isto se mogu koristiti prečaci na tipkovnici, a to su:

- ALT + P -> postavlja se brojač na Pomodoro, tj. na 25 minuta.
- ALT + L -> postavlja se brojač na dužu pauzu, tj. na 15 minuta.
- ALT + S -> postavlja se brojač na kratku pauzu, tj. na 5 minuta.
- ALT + R -> resetiranje brojača.



Slika 4. Prikaz liste funkcionalnosti

Klikom na gumb *„Settings“* otvara se skočni prozor za promjenu postavki (slika 5.).

Unutar toga skočnog prozora je gumb *„Custom Timer“* gdje se otvara novi skočni prozor gdje postavlja pitanje da se unese specifično vrijeme trajanja neke sesije, gdje

je početno postavljeno na 25 minuta, ali se može promijeniti kako bi to korisnik htio. Klikom na ovaj gumb poziva se funkcija *`setCustomTimer()`*.

Nakon ovoga gumba ima gumb *„Save Tasks“* uz pomoć kojeg se mogu spremiti zadaci i pod zadaci. Ovaj gumb poziva funkciju *`saveTasksToFile()`* koja sprema zadatke u JSON datoteku i sprema sve što je opet potrebno pri očitavanju tih zadataka. Da bi te zadatke i pod zadatke mogli očitati postoji gumb *„Load Tasks“* koji poziva funkciju *`loadTasksFromFile()´* koja očitava sve zadatke, pod zadatke, njihove kategorije, vrijeme u koje su kreirani, dali su izvršeni ili nisu i njihov poredak. Kao što je prikazano na slici 4. postoje prečaci na tipkovnici za spremanje i učitavanje zadataka, a to su:

- Za spremanje zadataka -> *„CTRL + S“*
- Za očitavanje zadataka -> *„CTRL + O“*

Ako neki zadatak ima isti tekst ili pod zadatak a ponovno je očitan, on se neće dodati nakon očitavanja, jer funkcija za očitavanje provjerava dali takav isti zadatak već postoji.

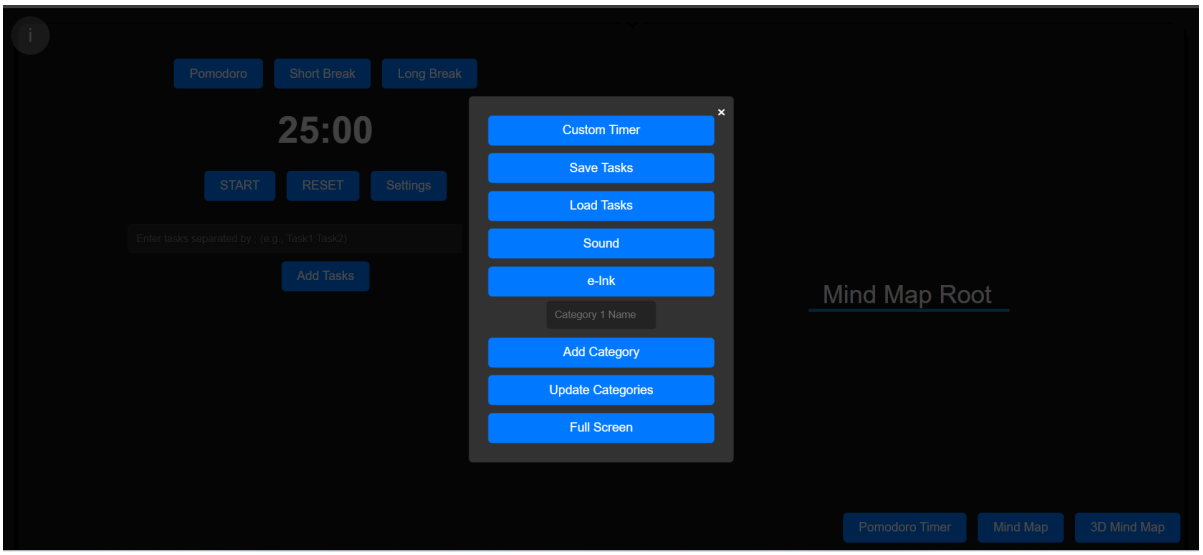
Sljedeće po redu je postavljanje zvuka koji će se oglasiti kada tajmer završi, a za to je gumb *„Sound“* koji poziva funkciju *`testSound`* nakon čega se otvara skočni prozor gdje dolazi tekst *„Enter sound type (sine, square, triangle, sawtooth):“* unutar kojeg se može odabrati između 4 vrste zvuka, nakon što se odabere vrsta zvuka, odabire se koliku frekvenciju želimo *„Enter frequency (Hz):“* i na kraju pita koliko puta korisnik želi da se zvuk ponovi (između 1 i 5 puta) – *„Enter repeat count (1-5):“.* Gdje je početno postavljeno na square, 440 Hz i 2 ponavljanja.

Nakon toga je gumb za omogućavanje E-Ink moda, koji cijelu stranicu postavi u sivu boju, da korisnik ne napreže oči. Ova metoda je jako popularna kada se čita duže vrijeme nešto.

Dalje imamo polje za unos kategorija za zadatke, gdje se mogu upisivati kategorije zadataka koje imamo, kao što su na primjer: *„fakultet“, „posao“, „ispiti“,* itd. Polje za unos nove kategorije se može otvoriti tako da se klikne na gumb odmah ispod polja za kategorije koji se naziva *„Add Category“* ili klikom na tipkovnici *„TAB“* gdje nakon svakog novog dodanog polja premješta u to polje da korisnik ne mora kliknuti mišem. Nakon polja za unos kategorija i gumba za dodavanje kategorija je još jedan gumb

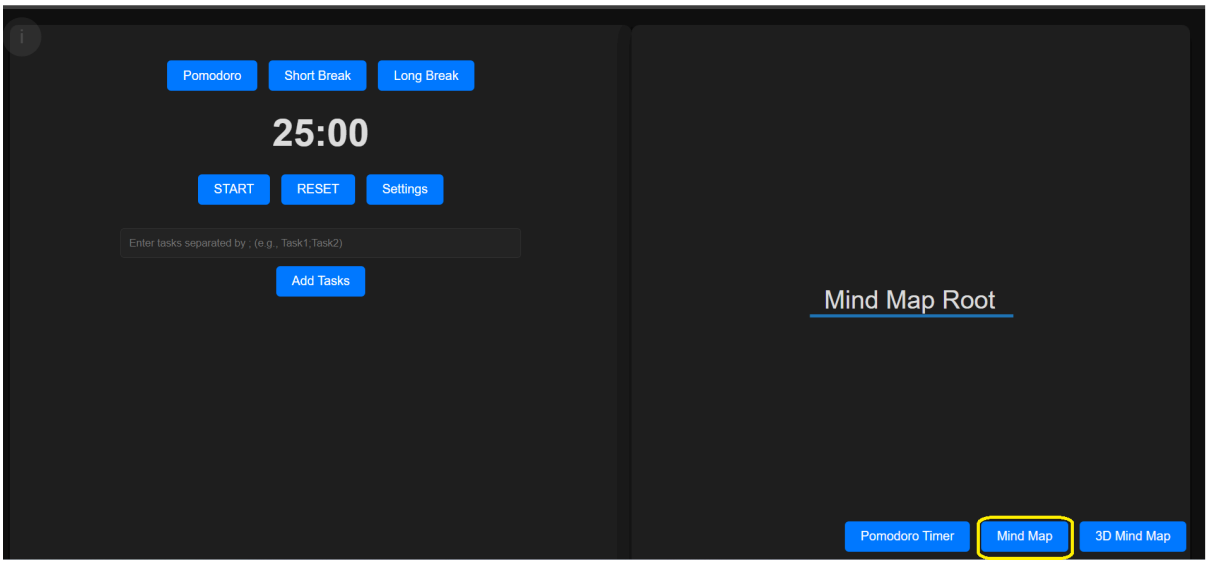
za kategorije, a to je *„Update Categories“* i on omogućava da se one ažuriraju i da se prikažu u listi Pomodoro tajmera. Ovaj gumb poziva funkciju *`updateCategories()`*.

I za kraj ostaje gumb *„Full Screen“* koji omogućava da aplikacija budu preko cijelog zaslona.



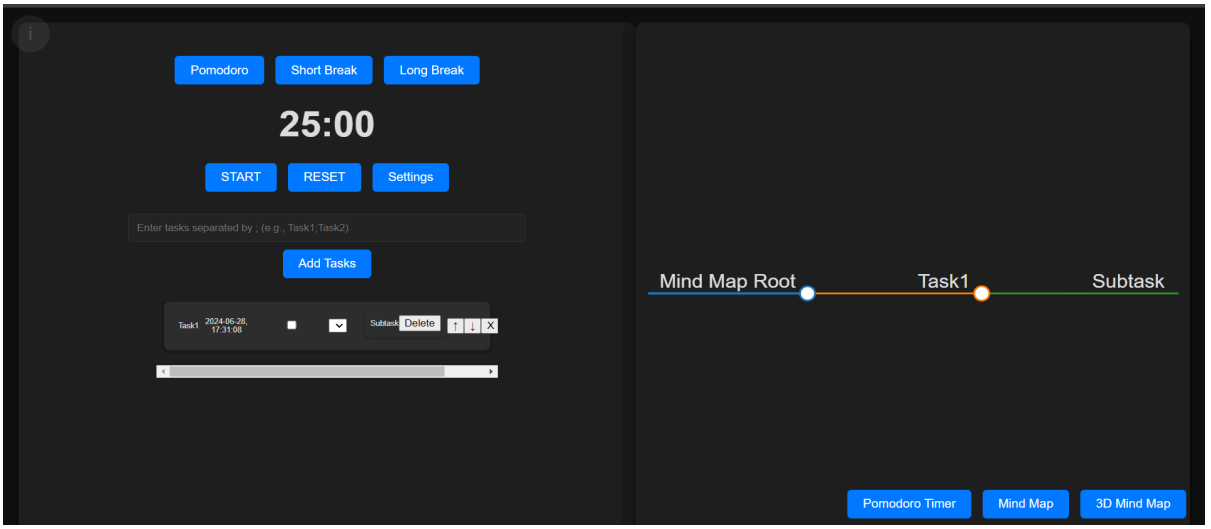
Slika 5. Skočni prozor za postavke

Kada se klikne na gumb u donjem desnom kutu na *„Mind Map“* kao što je ranije rečeno otvara se pola ekrana aplikacije Pomodoro tajmer, a druga polovica ekrana je Mind Map (slika 6.).Gumb je označen žutom bojom. Mind Map služi kao umna mapa umjesto liste zadataka što imamo unutar Pomodoro tajmera. Unutar tog Mind Mapa se mogu isto tako dodavati zadaci i pod zadaci i obrisati ih. Početno ime Mind Mapa je postavljeno na *„Mind Map Root“*.



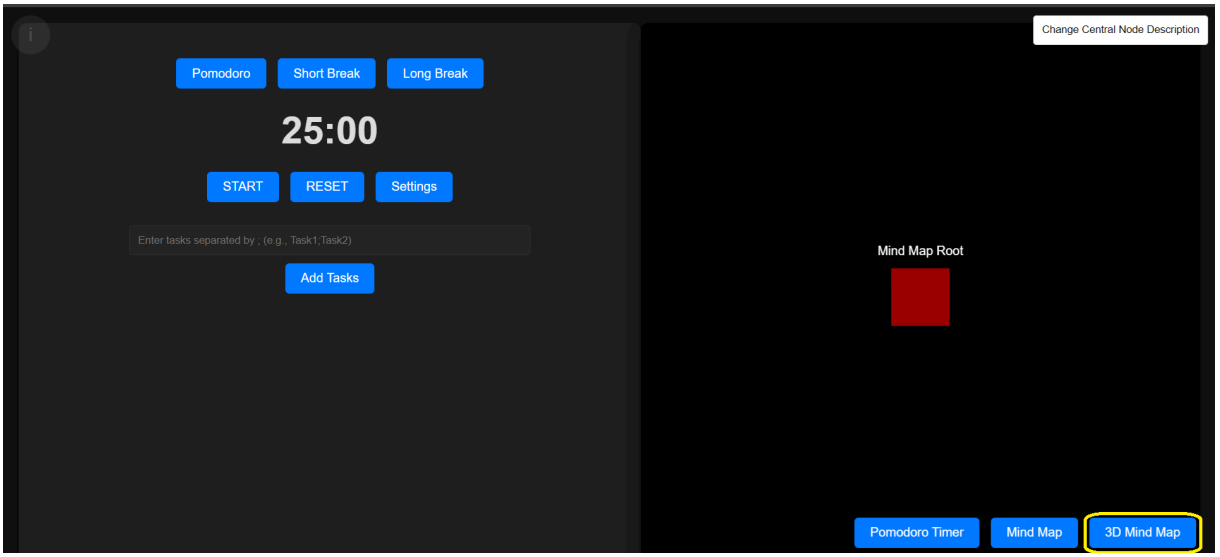
Slika 6. Mind Map

Zadaci se dodaju tako da bilo gdje unutar sekcije Mind Map korisnik klikne na tipkovnici *„CTRL + left mouse click“* gdje se nakon toga otvara skočni prozor gdje pita *„Enter new node content:“* nakon što se taj zadatak unese on se dodaje i u Mind Map, Pomodoro tajmer i u 3D Mind Map. Isto tako ako osoba klikne na taj zadatak/čvor desnim klikom onda se otvara novi skočni prozor gdje pita da se upiše ime pod zadatka za taj zadatak, pa tako ako na primjer imamo zadatak *„Task1“* i za njega osoba želi dodati pod zadatak otvorit će se skočni prozor *„Enter new subchild content for Task1:“* i onda ako se upiše neki tekst on će se dodati (slika 7.), a radi primjera dodat ćemo tekst za pod zadatak: *„Subtask“.*



Slika 7. Primjer dodavanja zadatka i pod zadatka u Mind Mapu

Na slici 7. također vidimo da su se taj zadatak i pod zadatak dodali i u Mind Map, i u Pomodoro tajmer. Isto ako bi otvorili 3D Mind Map i tamo bi bio dodan, ali to će kasnije biti prikazano. Također Mind Map se može povećavati, smanjivati, pomicati gore, dole, lijevo, desno. S prečacem na tipkovnici *„SHIFT + left mouse click“* na zadatak ili pod zadatak se brišu. Klikom na gumb u desnom donjem rubu *„3D Mind Map“* otvara se umjesto 2D Mind Map, 3D Mind Map. Ovaj gumb označen je žutom bojom na slici 8.



Slika 8. Prikaz 3D Mind Mapa

Na slici 8. vidimo da kada se otvorio 3D Mind Map umjesto 2D Mind Map, dok je Pomodoro tajmer i dalje ostao isti. Razliku između 2D i 3D Mind Mapa vidjet ćemo kasnije, kada ćemo prikazati jedan primjer svih funkcionalnosti koje imaju ova 3 modula.

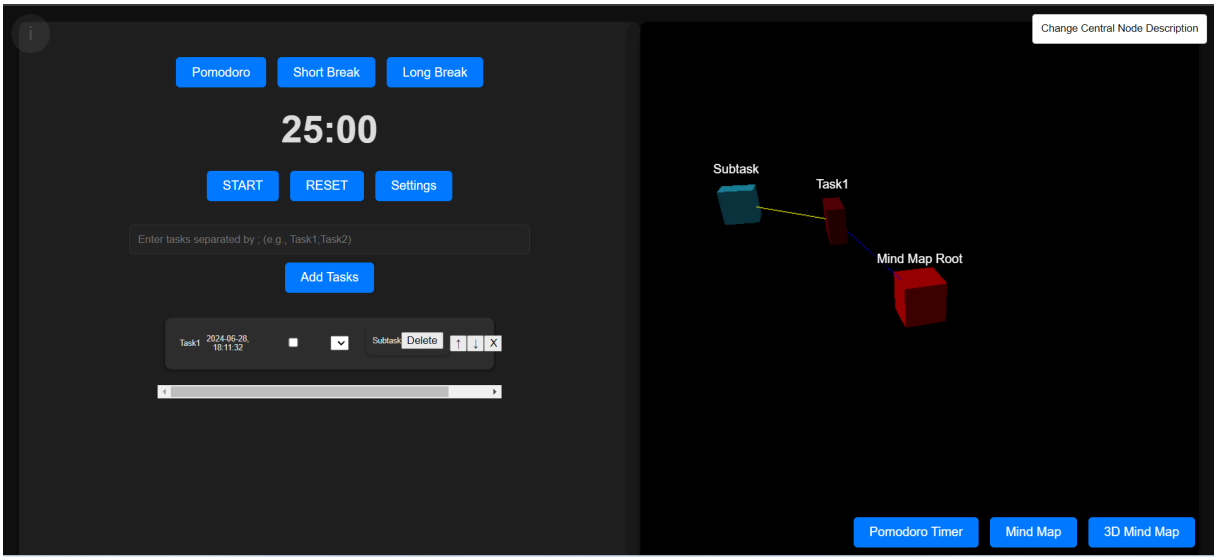
Za početak vidimo da glavna ili početna kocka ima naziv kao i u 2D Mind Map, a to je *„Mind Map Root“,* isto tako vidimo da u gornjem desnom kutu imamo gumb *„Change Central Node Description“* uz pomoć kojega se može promijeniti ime centralnog čvora, ako osoba to želim. On se sinkronizira, tj. promjeni ime i u 3D Mind Map, ali i u 2D Mind Map.

Na slici 9. vidimo da da je isti opis kao i za prošli primjer koji je prikazan u 2D Mind Mapu, sada smo prikazali kako to izgleda i u 3D Mind Map.

3D Mind Map se pomiče u Z os s lijevim klikom miša, a u X i Y os s desnim tako da se pomiče u onom smjeru u kojem korisnik želi.

Funkcije `*addNode*`, `*addLine*` i `*addNodeTo3DMindMap*` omogućavaju da se nasumično u prostoru dodaju oblici za zadatke i pod zadatke. Također ove funkcije osiguravaju da razlikujemo što je zadatak, a što pod zadatak, tako da je linija između zadatka i centralnog oblika plava, a između zadatka i pod zadatka žuta. Ove funkcije osiguravaju da se ne može obrisati centralni čvor.

Zadaci se dodaju tako da se bilo gdje u prostoru unutar 3D umne mape klikne na tipkovnici *„CTRL + left mouse click“*, a klikom na zadatak s *„ALT + left mouse click“* dodaje se pod zadatak. Brisanje je omogućeno s *„SHIFT + left mouse click“* na čvor.



Slika 9. Prikaz zadataka i pod zadataka unutar 3D umne mape

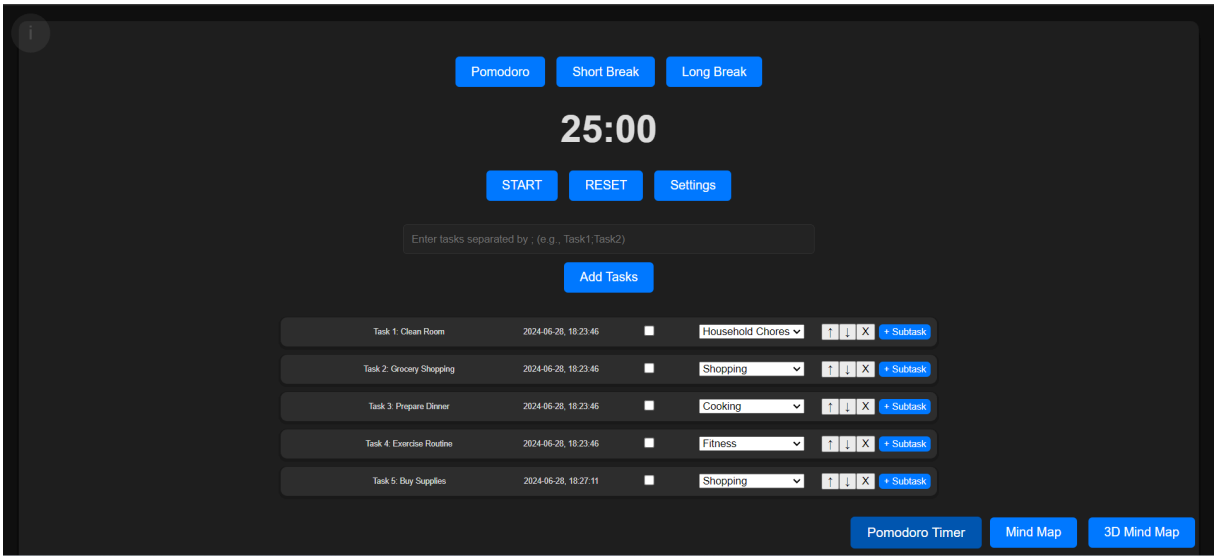
Sada će biti prikazane sve moguće funkcionalnosti koje prije nisu spomenute. Bit će dodano više zadataka i pod zadataka kako bi bolje bile prikazane sve funkcionalnosti.

Na slici 10. vidimo da je dodano 5 zadataka i to:

- Task 1: Clean Room
- Task 2: Grocery Shopping
- Task 3: Prepare Dinner
- Task 4: Exercise Routine
- Task 5: Buy Supplies

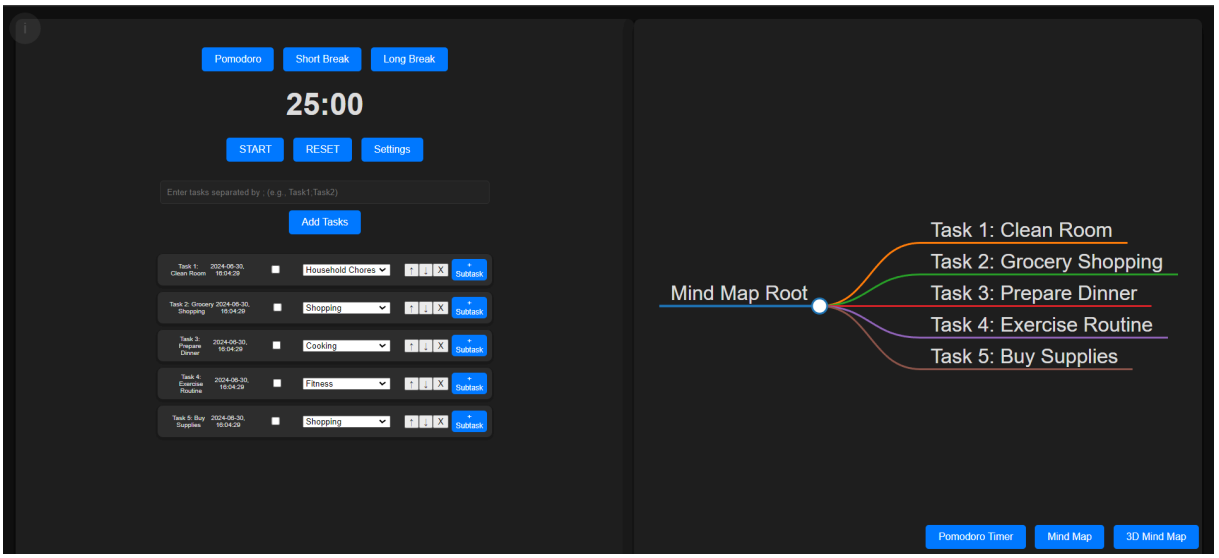
Dodane su 4 kategorije da svaki zadatak spada pod neku kategoriju. Prvo će biti prikazano kako ti zadaci izgledaju samo kada je otvoren Pomodoro tajmer, a kasnije

će biti prikazano kako to izgleda u 3D Mind Mapu, a i u 2D Mind Map. Također vidimo da za svaki zadatak ima vrijeme kada je dodan i kućica da se može označiti dali je izvršen ili nije. Isto tako postoje i strelice da ih možemo pomicati gore/dole kroz listu.



Slika 10. Prikaz zadataka u Pomodoro tajmeru

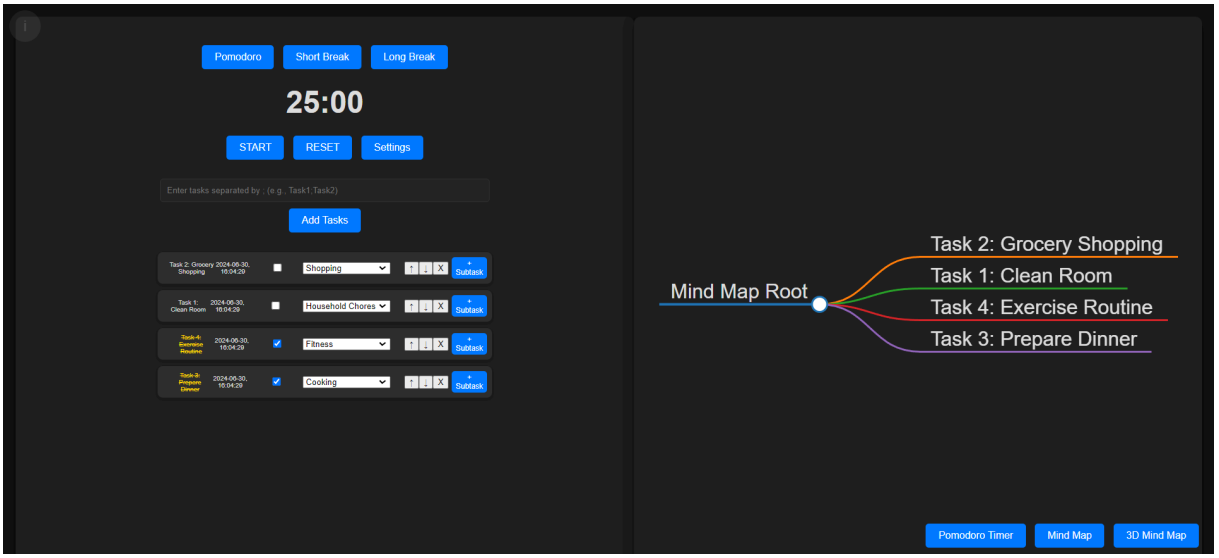
Sada ćemo na sljedećoj slici vidjeti kako to izgleda kada se otvori „*Mind Map*“ (slika 11.). Ako postoji puno teksta, ima klizač s kojim se može ako treba pomicati listu zadataka.



Slika 11. Prikaz zadataka u Pomodoro tajmeru i Mind mapu

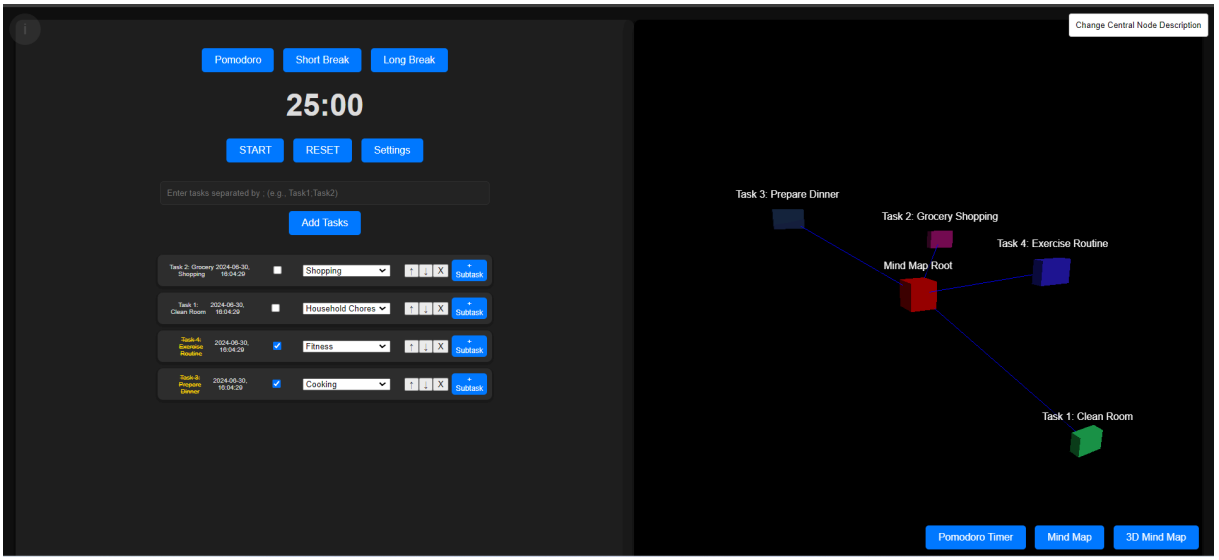
Na sljedećoj slici će biti izmijenjen redoslijed zadatak uz pomoć strelica će biti označeni neki od zadataka koji su izvršeni. Ti će se zadaci sinkronizirati i na listi od Pomodoro tajmera i na listi od Mind Mapa i svi će biti jednako poslagani (slika 12.).

Na slici 12. vidimo da je jedan zadatak obrisan, a to je zadatak broj 5. bio, njega više nema. Vidimo da su zadaci 3 i 4 označeni da su završeni i oni su prebačeni na kraj liste.



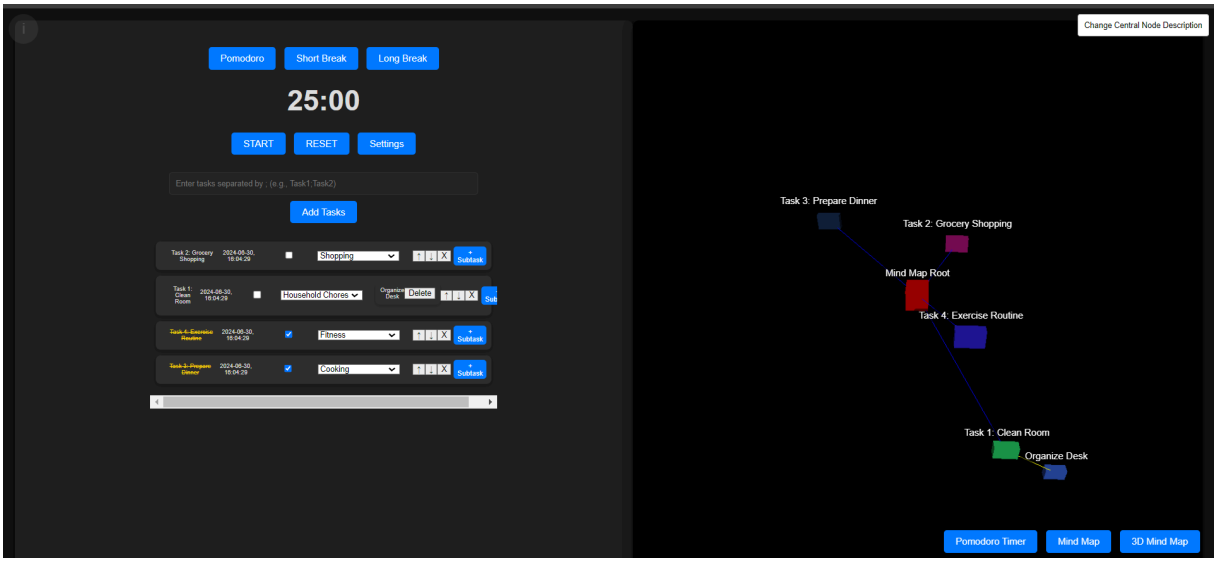
Slika 12. Izgled zadataka nakon promjene

Sada ako se ode pogledati kako izgleda 3D Mind Map, on će se prikazivati isto kao Pomodoro tajmer i Mind Map, samo u 3D obliku (slika 13.).



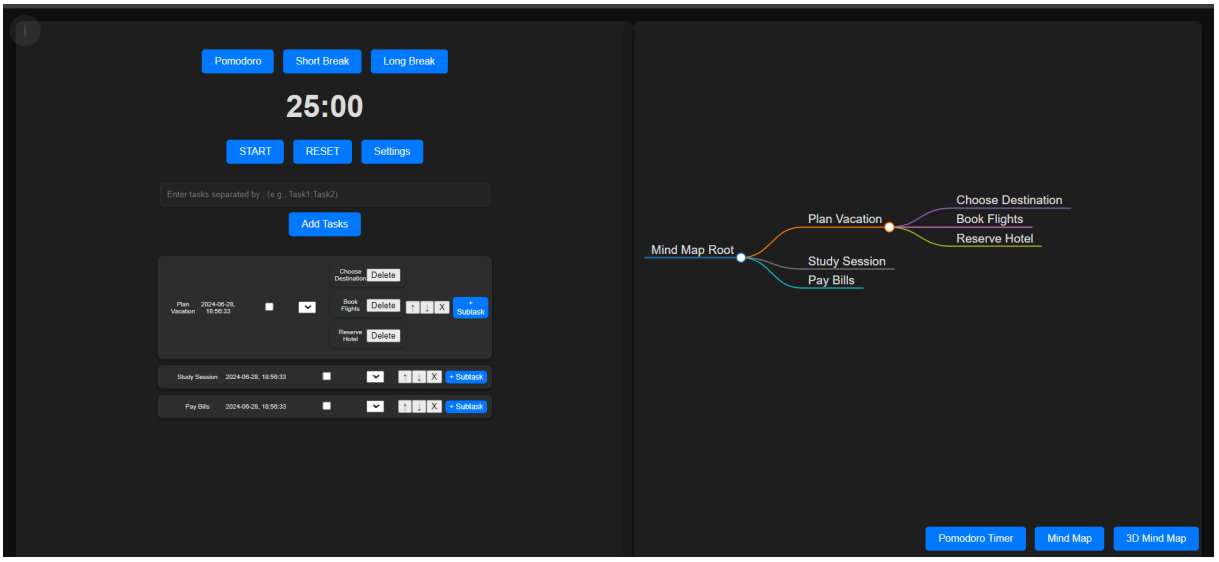
Slika 13. Prikaz zadataka unutar 3D Mind Mapa

Nakon toga dodan je jedan pod zadatak za *„Task1: Cleaning Room“*, a on će biti *„Organize Desk“ (slika 14.)*. Vidimo da je taj pod zadatak dodan i linija ide iz onog čvora koji mu je „roditelj“.



Slika 14. Dodavanje pod zadatka

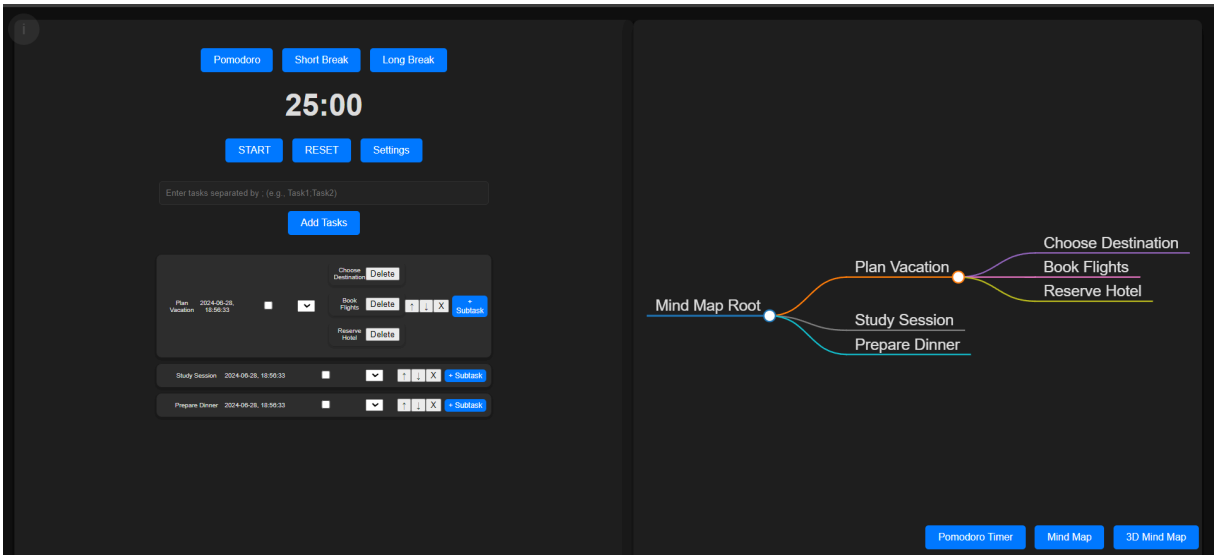
Sljedeće što će se napraviti jje obrisati sve zadatke unutar Pomodoro tajmera i vidjet ćemo da će se svugdje sinkronizirati, tj. obrisat će se u sva tri modula svi zadaci i bit će dodani novi zadaci i pod zadaci (slika 15.). Također na slici 15. vidimo da ne moramo dodati kategorije, može ostati i prazno.



Slika 15. Prikaz novih zadataka i pod zadataka

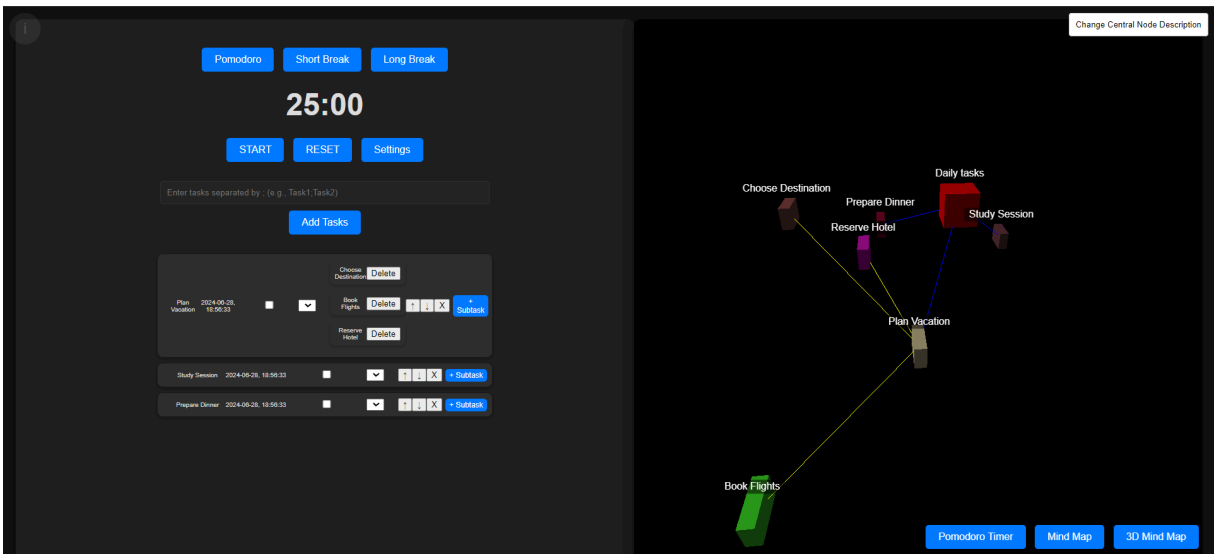
Ako osoba želi promijeniti opis nekog zadatka ili pod zadatka, samo se klikne duplim lijevim klikom miša na taj zadatak i otvori se novi skočni prozor koji pita da se promijeni ime tog zadatka ili pod zadatka gdje unutar polja bude već upisano trenutno ime toga zadatka ili pod zadatka. Pa tako ako na primjer želimo promijeniti opis zadatka *„Pay Bills“* u *„Prepare Dinner“* samo kliknemo lijevi dupli klik miša na taj

zadatak i promijenimo mu opis i to će se ažurirati u sva tri modula (slika 16.). A isto tako možemo i mijenjati opis pod zadataka isto će se ažurirati.



Slika 16. Promjena opisa zadatka

Ako se korisnik odluči da želi promijeniti ime glavnog čvora čije je ime početno „*Mind Map Root*“, može otići u sekciju gdje je 3D Mind Map i kliknuti na gumb koji se nalazi u kutu gore desno i promijeniti ime po želji, a ime će se ažurirati u 3D Mind mapu i regularnom Mind mapu. Pa tako na primjer ako promijenimo ime glavnog čvora u „*Daily tasks*“ opis će se ažurirati u obje umne mape (slika 17.).



Slika 17. Promjena opisa glavnog čvora

7. Zaključak

U ovome radu detaljno su obrađene sve ključne stvari da bi se napravila jedna web aplikacija, gdje je objašnjeno ukratko u teoriji tehnologije koje su korištene, njihova primjena i izazovi. Isto tako u ovome završnom radu utvrđeno je da je Pomodoro tehnika izuzetno efikasna u poboljšanju koncentracije. Ova metoda je jako popularna među učenicima i studentima, a zapravo nju je koristio i sam Francesco Cirillo, za svoje potrebe tijekom studentskih dana kako bi bio što efikasniji. Za potrebe izrade ove aplikacije korištene su tehnologije kao što su HTML, CSS i JavaScript. U ovome radu prikazano je kako kreirati web aplikaciju od osnovne strukture u HTML-u, gdje je postupno poboljšanje vizualnog izgleda aplikacije omogućeno uz pomoć CSS-a, te dodavanje funkcija uz pomoć JavaScripta kako bi mogli pravilno koristiti aplikaciju. Svaka ova tehnologija ima važnu ulogu u ovoj aplikaciji.

HTML daje temeljnu strukturu, gdje se definiraju dijelovi kako će stranica izgledati. Unutar HTML-a su definirane neke stvari, kao što su naslov, gumbovi, sekcije, itd…,

CSS je važan jer unutar njega se uređuje nešto što će se vidjeti unutar aplikacije. Odvajanjem sadržaja od dizajna, CSS omogućuje veću fleksibilnost i kontrolu nad izgledom.

JavaScript je omogućio interaktivnost i dinamičnost aplikacije. A upravo korištenjem ovih triju tehnologija omogućeno je stvaranje funkcionalne aplikacije.

Ova aplikacija sastoji se od 3 dijela, a to su Pomodoro timer, Mind Map i 3D Mind Map. Pomodoro tajmer je ključan dio aplikacije koji omogućava korisnicima da upravljaju svojim vremenom i obavezama na strukturiran način, gdje mogu pratiti koliko vremena imaju za određenu sesiju, spremati podatke i ponovno ih očitavati.

Ova aplikacije ima i neke dodatne stvari kao što su 3D Mind Map, unutar kojeg se mogu dodavati zadaci i pod zadaci i obrisati ih, E-Ink koji smanjuje naprezanje očiju i personalizirane tajmere, gdje korisnik može postaviti personalizirano vrijeme za određenu sesiju.

U ovome završnom radu prikazano je da tehnologija može utjecati na poboljšanje svakodnevnog života, a posebno u obrazovanju. Za razvoj ove aplikacije bilo je potrebno znanje iz programiranja, dizajniranja i psihologije učenja.

Neka od mogućih budućih unapređenja aplikacije su sinkronizacija s Google računom, kalendarom i slično. Iako ova aplikacija je već vrlo korisna, sve buduće nadogradnje će još unaprijediti njezinu uporabu.

Za kraj se zahvaljujem svim suradnicima, a posebno mentoru izv. prof. S. Maričiću koji je bio podrška tijekom izrade ove aplikacije i pomogao. Posebno bih želio zahvaliti mu se na ustupanju koda za mind map, koji je jedan od ključnih za razvoj ove aplikacije. Htio bih se zahvaliti i Vama svima korisnicima, čije su povratne informacije bile veoma važne za unapređenje ove aplikacije.

Literatura

1. Brooks D. R. (2007.): *An Introduction to HTML and JavaScript: for Scientists and Engineers*. London: Springer.
2. „*Uvod u HTML: Vodič kroz HTML: Moj Web dizajn*“ Pristupljeno: [20.6.2024.] Dostupno na: https://www.mojwebdizajn.net/skriptni-jezici/vodic/html/uvod-u-html.php,
3. „*Uvod u HTML“,* Portal Alfa, Pristupljeno [29.6.2024.] Dostupno na: https://www.portalalfa.com/1/Html/uvod.htm ,
4. „*Uvod u HTML“*, Nacionalni portal za učenje na daljinu „Nikola Tesla“, pristupljeno [24.6.2024.], dostupno na: https://tesla.carnet.hr/mod/book/view.php?id=5430&chapterid=887 ,
5. Michaud T. (2014.) : *Foundations of Web Design: Introduction to HTML & CSS*, USA: New Riders,
6. *„Kako lakše učiti? – upoznajte Pomodoro metodu“* Pristupljeno: [21.6.2024.] Dostupno na: https://krenizdravo.dnevnik.hr/zdravlje/psihologija/kako-lakse-uciti-upoznajte-pomodoro-metodu
7. Hasani, N. K., & Umamah, N. (2023). *Pomodoro Technique Analysis in Zoom-Based Classrooms. Journal of English Education and Linguistics Studies*, 10(1), 1-12. Pristupljeno: [2.7.2024.], Dostupno na: https://jurnalfaktarbiyah.iainkediri.ac.id/index.php/jeels/article/view/475/358
8. Wiradhany, W., & Biwer, F. (2023*). Understanding effort regulation: Comparing ‘Pomodoro’ breaks and self-regulated breaks*. *British Journal of Educational Psychology*. Pristupljeno: [1.7.2024.], Dostupno na: https://doi.org/10.1111/bjep.12593
9. Ranjan, A., Sinha, A., & Battewad, R. (2020). *JavaScript for Modern Web Development: Building a Web Application Using HTML, CSS, and JavaScript*. BPB Publications.
10. Ruvalcaba, Z., & Boehm, A. (2020). *Murach's HTML5 and CSS3*. Mike Murach & Associates.

Popis slika

Popis primjera

Sažetak

U ovome završnom radu prikazana je izrada web aplikacije koja pomaže pri učenju i upravlja obavezama, gdje su korištene tehnologije kao što su CSS, HTML i JavaScript. Još je važno napomenuti da je cijela aplikacija izrađena u Visual Studio Code okruženju. Ova aplikacija se temelji na Pomodoro tehnici, koja je jedna od dokazano najučinkovitijih tehnika za učenje danas. Pomodoro tehnika je poznata po tome što ima cikluse koji traju u intervalima, gdje prvi ciklus traje 25 minuta, nakon toga kratka pauza od 5 minuta i tako tri puta, nakon toga, ide ciklus od 25 minuta koji je još poznati i kao pomodoro ciklus, i onda duža pauza od 15 minuta i tako stalno u krug. Tako je nakon svakog 4. ciklusa ide duža pauza od 15 minuta. Aplikacija je napravljena da bude korisna i da korisnicima omogući da se mogu maksimalno koncentrirati na svoje zadatke. Također, u aplikaciji se može vidjeti cijeli popis i u Mind Mapu, ali i u 3D Mind Mapu, ako to korisnik želi. Aplikacija sadrži različite funkcionalnosti, a neke od njih su: prilagođavanje brojača po želji korisnika, spremanje i očitavanje zadataka, dodavanje kategorija, E-Ink, itd.

Kroz aplikaciju očekuje se značajno poboljšanje koncentracije i ukupnog akademskog učinka svih učenika i studenata koji budu koristili ovu aplikaciju, ali ovu aplikaciju mogu koristiti svi, a ne samo studenti i učenici. Kroz detaljno objašnjene funkcionalnosti i primjere, ovaj rad služi za razumijevanje razvoja jedne takve web aplikacije te primjenu tehnologije za unapređenje procesa učenja.

Summary

This paper presents the creation of a web application that helps with learning and manages tasks, where technologies such as CSS, HTML and JavaScript were used. It is also important to note that the entire application was created in the Visual Studio Code environment. This app is based on the Pomodoro technique, which is one of the most proven effective learning techniques today. The Pomodoro technique is known for having cycles that last in intervals, where the first cycle lasts 25 minutes, followed by a short break of 5 minutes and so three times, after that, there is a cycle of 25 minutes, which is also known as the pomodoro cycle, and then a longer break of 15 minutes and so on and on. That is, after every 4th cycle there is a longer break of 15 minutes. The application is made to be useful and to allow users to concentrate on their tasks as much as possible. Also, in the application, the entire list can be seen in the Mind Map, but also in the 3D Mind Map, if the user wishes. The application contains various functionalities, and some of them are: adjusting the counter according to the user's wishes, saving and reading tasks, adding categories, E-Ink, etc.

The application is expected to significantly improve the concentration and overall academic performance of all students who will use this application, but this application can be used by everyone, not only students and students. Through detailed explanations of functionality and examples, this paper serves to understand the development of such a web application and the application of technology to improve the learning process.