ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Онлайн-платформа для організації колективної діяльності

АВТОР:
СТ. ГР. КІУКІ-21-2
ТОПОРЕЦЬ ДМИТРО

Місія

Мета:

Створення єдиного настроюваного web-інтерфейсу високого ступеню інтеграції сторонніх сервісів для організації колективної діяльності.

Цілі сталого розвитку:

17) Співпраця заради досягнення цілей.

Проблеми:

- Відсутність єдиного узагальненого формату інтерфейсу для відстеження прогресу та дій учасників;
- Значні затрати технічних ресурсів та підтримання належного рівня безпеки передачі даних у реальному часі.

Учасники проекту

Користувачі (клієнти):

- Наукові групи;
- Навчальні заклади;
- Бізнес.

Персонал:

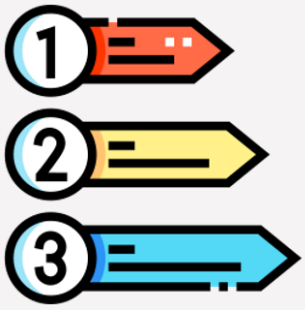
- Розробники;
- Адміністратори системи;
- Технічна підтримка.

Технічні партнери:

- Постачальники сторонніх сервісів.

3

Цінності



Для клієнтів:

- Економія часу завдяки спрощенню організації процесів;
- Простий доступ до сторонніх сервісів;
- Спрощена взаємодія з іншими клієнтами;
- Централізоване управління та контроль над проектами;
- Відстеження дій учасників у реальному часі;
- Покращена координація завдяки єдиній платформі;
- Можливість налаштування та масштабування платформи

Для адміністраторів системи:

- Легка масштабованість платформи;
- Передача частини обов'язків та відповідальності клієнтам;
- Простота в обслуговуванні та забезпеченні безпеки

Для партнерів:

- Нові можливості для співпраці

4

Архітектура платформи

Сервер-координатор:

- Забезпечення утримання та надання актуального списку робочих серверів.

Робочий сервер:

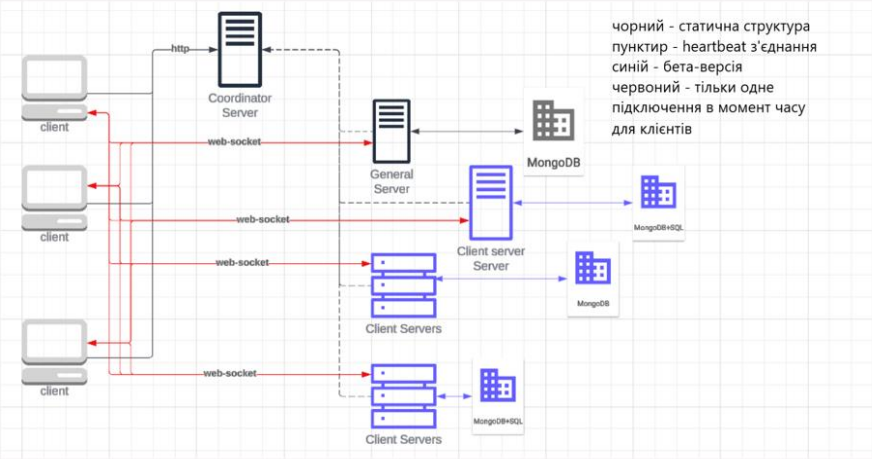
- Підтримка з'єднання з сервером-координатором;
- Обробка клієнтських HTTP-запитів;
- Робота з клієнтським WebSocket з'єднанням;
- Робота з MongoDB.

Користувальницький додаток:

- Робота з проектами та активностями;
- Інтерактивний розклад;
- Система сповіщень;
- WebSocket-з'єднання з робочим сервером

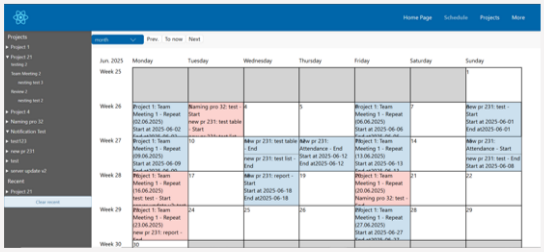
5

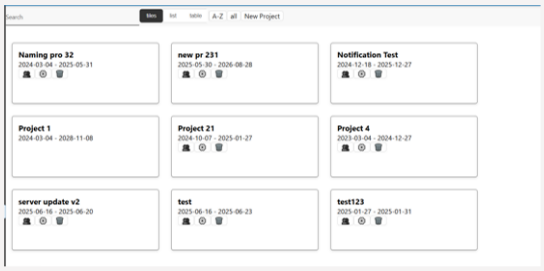
Топологія системи



6

Користувальницький додаток





Функціонал:

- Інтерактивний розклад
- Історія перегляду в межах сесії
- Редактор проектів та активностей
- Система сповіщень

Адміністраторський додаток

Функціонал:

Редагування прав користувачів

Реєстрація /видалення користувачів

бан користувачів

Перегляд логів серверу

name	middleName	surName	patronimic	genStatus	access	accessLevel
Dmytro		Toporets		root		root
testName		TestSurname		admin	X	user

Сервери

Функціонал:

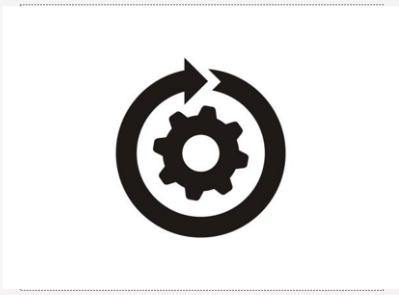
Оптимізована та захищена передача даних з Front-end

Запуск серверу через контейнер docker

Можливість оновлення через «docker pull»

Ліцензування екземплярів

Перспективи та плани розвитку



Плани:

- Розширення можливостей редактора проектів та активностей
- Розширення кастомізації інтерфейсу користувача
- Впровадження редактору шаблонів для адміністраторів

Перспективи:

- Відкрите тестування
- Система підтримки користувачів на основі ШІ
- Монетизація проекту

ДОДАТОК Б

Приклади реалізації шаблонів програмування

Б.1 Front-end

Б.1.1 «Підняття стану вгору» (State Lifting)

З компоненту App (корінь стану):

```
…
const [projects, setProjects] = useState([])
…
return (
 <Router>
  <div className="App">
   <AppHeader
    …
   />
   <div>
    {isLoggedIn &&
     <AppNav
      projects={projects}
      …
     />
    }
    <Routes>
     <Route path="*" element={
      <AppDynamicContent
       …
       projects={projects}
       activities={activities}
       setData={updateData} //used to update projects
       …
      />
     } />
    </Routes>
   </div>
  </div>
 …
</Router>
)
}
```

З компоненту AddEditItem:

```
const handleSubmit = (e) => {
    e.preventDefault()

    //validation
    …

    const project = Shared.getItemById(projects,
state.currentProject)
    const formattedFormValues = formatFormValues(formValues)

    const { item: parent } =
Shared.findItemWithParent(project?.activities || [], "_id",
containerId, project)

    let newItem = {
      …
    }

    // submit

    let item
    const action = currentItemId ? "edit" : "add"

    if (selectedType === "Project") {
      …
    }
    else if (selectedType === "Activity") {
      …
    }
    setData({ action, item }) //setter method for projects
    closeDialog()
  }
```

Б.1.2 «Container-Presentation»

З компоненту-контейнеру ItemTiles

```
const ItemTiles = ({
  …
}) => {

  const getItem = (item, index, dnd) => {
    const Component = (dnd) ? Shared.Item : Items.Project
    return (
      <Component
        key={item._id}
```

```
        …
        item={item}
        index={index}
        containerId={containerId}//
        containerType={containerType}//
        …
      />
    )
  }

  const noItems = itemsToDisplay.length === 0
  const noParent = !containerId
  const isInContainerItem = containerType === "Group"
  const isInListBasedItem = ["List", "Chat", "Attendance",
"Report"].includes(containerType)

  return (
    <>
      {(state.currentPage === "Projects") ?
        (
          itemsToDisplay.map((item, index) => (
            getItem(item, index, false)
          ))
        ) : (
          <div
            className="item-tiles"
          >
            <SortableContext items={itemsToDisplay.map((i) =>
i._id)}>
              {itemsToDisplay.map((item, index) => (
                getItem(item, index, true)
              ))}
              {noItems && (noParent || isInContainerItem ||
isInListBasedItem) &&
                getItem(true, 0, true)
              }
            </SortableContext>
          </div>
        )
      }
    </>
  )
}

export default ItemTiles
```

З компоненту-контейнеру Item

```
return (
 <div
  className={'activity-item'}
  ref={setNodeRef}
  {...attributes}
  style={style}
 >
  {!noActions.includes(containerType) &&
   <div
    className='item-actions'
   >
    {/* ➕ Add below button */}
    {Shared.getDialogButton(…)}
    {isItem && accessCheck() && /* ⭕ DRAG HANDLE (6-dots) */
     <div
      …
     >
      ⁞⁞
     </div>
    }
   </div>
  }
  {isItem && accessCheck() &&
   <>
    {!noActions.includes(containerType) &&
     <Shared.ItemActions
      …
      item={item}
      …
     />
    }
    <ItemComponent
     key={item._id}
     …
     item={item}
     index={index}
     containerId={containerId}
     containerType={containerType}
     …
    />
   </>
  }
 </div>
)
```

З компоненту представлення ItemActions

```
 return (
  <>
   {!item.deleted &&
rights.edit.includes(Shared.getAccess(accessibleItem, userData))
&& (
   <div className={wrapperClass}>
    {Shared.getDialogButton(
     setState,
     buttonClass,
     "AddEditUserList",
     [item._id],
     "👥",
     true
    )}
    {(parts?.length < 3) &&
    Shared.getDialogButton(
     setState,
     buttonClass,
     "AddEditItem",
     [item, item._id],
     "⚙️",
     true
    )}
    <button className={buttonClass}
     onClick={(e) => {
      e.stopPropagation()
      Shared.deleteItem(projects, setData, item._id)
     }}
    >
     🗑️
    </button>
   </div>
   )}
  </>
 )
```

Б.1.3 «Введення залежностей»

Метод із залежністю «getInput»:

```
const getInput = (fieldName, type, value, checked = false,
handler, disabled = false, section = undefined, width = 50) => {
 return (
  <div
   className="input-group field"
```

```
    style={{
     width: `${width}%`
    }}
   >
    <input
     type={type}
     className="input-field"
     placeholder={fieldName}
     name={fieldName}
     id={fieldName}
     data-section={section}
     disabled={disabled}
     value={value}
     checked={checked}
     onChange={handler}
   />
    <label
     htmlFor={fieldName}
     className="input-label"
    >
     {fieldName}
    </label>
   </div>
 )
}
export default getInput
```

Вставлення залежності:

```
//AddEditContent

{Shared.getInput(
  "Field name",
  "text",
  newField.name,
  undefined,
  e => setNewField(prev => ({ ...prev, name: e.target.value })),
  false,
  null,
  50
  )}

//AddEditItem

{Shared.getInput(
  formatLabel(key),
  currentStructure[key],
  currentStructure[key] !== 'checkbox' ? formValues[key] :
undefined,
  currentStructure[key] === 'checkbox' ? formValues[key] :
false,
```

```
  handleInputChange,
  false,
  null,
  60,
)}

//LogIn

{Shared.getInput(
  "login", "text",
  formValues.login, false,
  handleInputChange,
  false, null, 60
)}
{Shared.getInput(
  "password", "text",
  formValues.password, false,
  handleInputChange,
  false, null, 60
)}
```

Б.1.4 «Middleware» (Chain of Responsibility)

Проміжний компонент для мапи розкладу «Schedule»:

```
const Schedule = ({…}) => {
 …
 return (
  <>
   <Shared.ControlPanel
    …
   />
   <div className='schedule-container page-wrapper'>
    {currentScale === "week" &&
     <p className='warning'>Not timed events are not displayed
at weekly scale</p>
    }
    <p className='current-map'>
    {currentScale!=='year' && months[intervalAnchor.getMonth()]}
{intervalAnchor.getFullYear()}
    </p>
    <div
     className="schedule-v-map"
     style={{
      display: 'grid',
      gridTemplateColumns: scheduleBoard.gridTemplateColumns
     }}>
     {scheduleVMap}
    </div>
```

```
    <div
     className='schedule-scrollable'
    >
     <div
      className="schedule-h-map"
      style={{
       display: 'grid',
       gridTemplateRows: scheduleBoard.gridTemplateRows
      }}
     >
      {scheduleHMap}
     </div>
     <ScheduleBoard
      …
     />
    </div>
   </div>
  </>
 )}

export default Schedule
```

Проміжний компонент для відображення списку елементів «»:

```
const ItemTiles = ({…}) => {

 const getItem = (item, index, dnd) => {

  const Component = (dnd) ? Shared.Item : Items.Project

  return (
   <Component
    key={item._id}

    …
    item={item}
    index={index}
    containerId={containerId}//
    containerType={containerType}//
    …
   />
  )
 }

 const noItems = itemsToDisplay.length === 0
 const noParent = !containerId

 const isInContainerItem = containerType === "Group"
 const isInListBasedItem = ["List", "Chat", "Attendance",
"Report"].includes(containerType)
```

```
 return (
  <>
   {(state.currentPage === "Projects") ?
    (
     itemsToDisplay.map((item, index) => (
      getItem(item, index, false)
     ))
    ) : (
     <div
      className="item-tiles"
     >
      <SortableContext
       items={itemsToDisplay.map((i) => i._id)}
      >
       {itemsToDisplay.map((item, index) => (
        getItem(item, index, true)
       ))}

       {noItems && (noParent || isInContainerItem ||
isInListBasedItem) &&
        getItem(true, 0, true)
       }
      </SortableContext>
     </div>
    )
   }
  </>
 )
}

export default ItemTiles
```

Б.2 Back-end

Б.2.1 «Singleton»

Елемент в одиничному екземплярі з файлу «db.js»:

```
const Collections = {
  user: User,
  project: Project,
  organisation: Organisation,
  activity: Activity
}

module.exports = { Collections, organisationId, ObjectId }
```

З кореневого файлу «App»:

```
require("dotenv").config()

const createError = require("http-errors")
const express = require("express")
const path = require('path')
const cookieParser = require("cookie-parser")
const logger = require("morgan")

const mongoose = require("mongoose")
const { v4: uuidv4 } = require('uuid')
const fs = require('fs')
const axios = require("axios")
```

Б.2.2 «Observer»:

```
const { getWebSocketByToken } = require("../sockets/websockets")
const { getAdminWebSocketByToken } =
require("../sockets/adminWebsockets")
```

Б.2.3 «DAO»:

```
const User = require("../models/User")
await User.findOne(...)
await User.findByIdAndUpdate(...)
await User.deleteOne(...)
```

Б.2.4 «Factory»:

```
const getActivityContent = (type) => {
 let content = {}

 switch(type) {
  case "Group": {
   content = {
    name: type,
    currentSettings: {}
   }
   break
  }
  case "Text": {
   content = {
    currentSettings: {},
    text: "text"
```

```
 }
 break
}
case "List": {
 content = {
  name: type,
  currentSettings: {
    type: "ul"
  },
  listItems: [],
  liStructure: {
   text: "text"
  }
 }
 break
}
case "Report": {
 content = {
  name: type,
  currentSettings: {
    markable: true
  },
  listItems: [],
  liStructure: {
   markable: "markable"
  }
 }
 break
}
case "Attendance": {
 content = {
  name: type,
  currentSettings: {
    markable: true
  },
  listItems: [],
  liStructure: {
   markable: "markable"
  }
 }
 break
}
case "Table": {
 content = {
  name: type,
  currentSettings: {},
  listItems: [],
  liStructure: {
   text: "text"
  }
 }
 break
}
```

```
  case "Chat": {
   content = {
    name: type,
    currentSettings: {},
    messageCount: 0,
    listItems: [],
    liStructure: {
     sender: "plain",
     content: "plain",
     dateTime: "plain"
    }
   }
   break
  }
  // case "": {
  //  content = {
  //   currentSettings: {}
  //  }
  //  break
  // }
  default: {
   content = {
    currentSettings: {}
   }
  }
 }
 return content
}
```