

**Politechnika Świętokrzyska w Kielcach**

Wydział Elektrotechniki, Automatyki i Informatyki

Politechnika Świętokrzyska  
Kielce University of Technology

**Inżynieria Systemów Informacyjnych**

**Projekt**

**Aplikacja do zarządzania siłownią**

**Skład zespołu:**

Rafał Grot

Filip Stępień

Bartłomiej Karkoszka

Mateusz Karbowniczak

Kierunek/specjalność: Systemy informacyjne

Studia: stacjonarne

Numer grupy: 3ID11B

Kielce, 18 czerwca 2025

# 1 Wprowadzenie

## 1.1 Krótki opis aplikacji

Celem projektu było stworzenie aplikacji służącej do zarządzania siłownią. System ten został zaprojektowany z myślą o czterech różnych typach użytkowników: kliencie, pracowniku, trenerze oraz menadżerze.

Aplikacja umożliwia m.in.:

- śledzenie postępów treningowych przez użytkownika,
- przegląd i edycję zaplanowanych sesji treningowych,
- zapisywanie przebiegu własnych treningów,
- odnawianie karnetu,
- wgląd w listę klientów i pracowników (dla kadry zarządzającej),
- zarządzanie salami treningowymi,
- tworzenie i edytowanie kont pracowników oraz klientów.

## 1.2 Wykorzystane technologie oraz narzędzia

Do stworzenia aplikacji wykorzystano następujące technologie i narzędzia:

- **TypeScript** – nadzbiór języka JavaScript umożliwiający statyczne typowanie,
- **Vite** – bundler i środowisko uruchomieniowe do aplikacji frontendowych,
- **Vitest** – framework testowy, zintegrowany z ekosystemem Vite,
- **React** – biblioteka JavaScript służąca do budowy interfejsów użytkownika,
- **React Router** – biblioteka do zarządzania nawigacją i routingiem w aplikacjach React,
- **Axios** – biblioteka do obsługi zapytań HTTP, wykorzystywana do komunikacji z backendem przez REST API,
- **OpenAPI** – standard do opisywania interfejsów REST API,
- **Orval** – narzędzie generujące klienta API na podstawie specyfikacji OpenAPI, ułatwiające komunikację z backendem,
- **Figma** – narzędzie do projektowania interfejsów użytkownika i prototypowania,
- **Task** – narzędzie do automatyzacji zadań deweloperskich,
- **Git** – system kontroli wersji wspierający zarządzanie kodem źródłowym oraz pracę zespołową,
- **GitHub** – serwis internetowy umożliwiający przechowywanie i zarządzanie repozytoriami Git,
- **GitHub Actions** – platforma CI/CD umożliwiająca automatyzację procesów związanych z budowaniem, testowaniem i wdrażaniem aplikacji,
- **Renovate** – bot do automatycznego aktualizowania zależności projektu,

- **Nix** – narzędzie do zarządzania środowiskiem programistycznym w systemie NixOS,
- **AntDesign** – biblioteka komponentów UI dla React,
- **Chart.js** – biblioteka do tworzenia dynamicznych wykresów,
- **Keycloak JS** – biblioteka służąca do integracji aplikacji frontendowej z systemem *Keycloak* do autoryzacji użytkowników,
- **Tailwind CSS** – biblioteka CSS do szybkiego stylowania interfejsów użytkownika,
- **Day.js** – biblioteka do obsługi dat i czasu.

## 2 Implementacja

### 2.1 Opis głównych funkcjonalności aplikacji

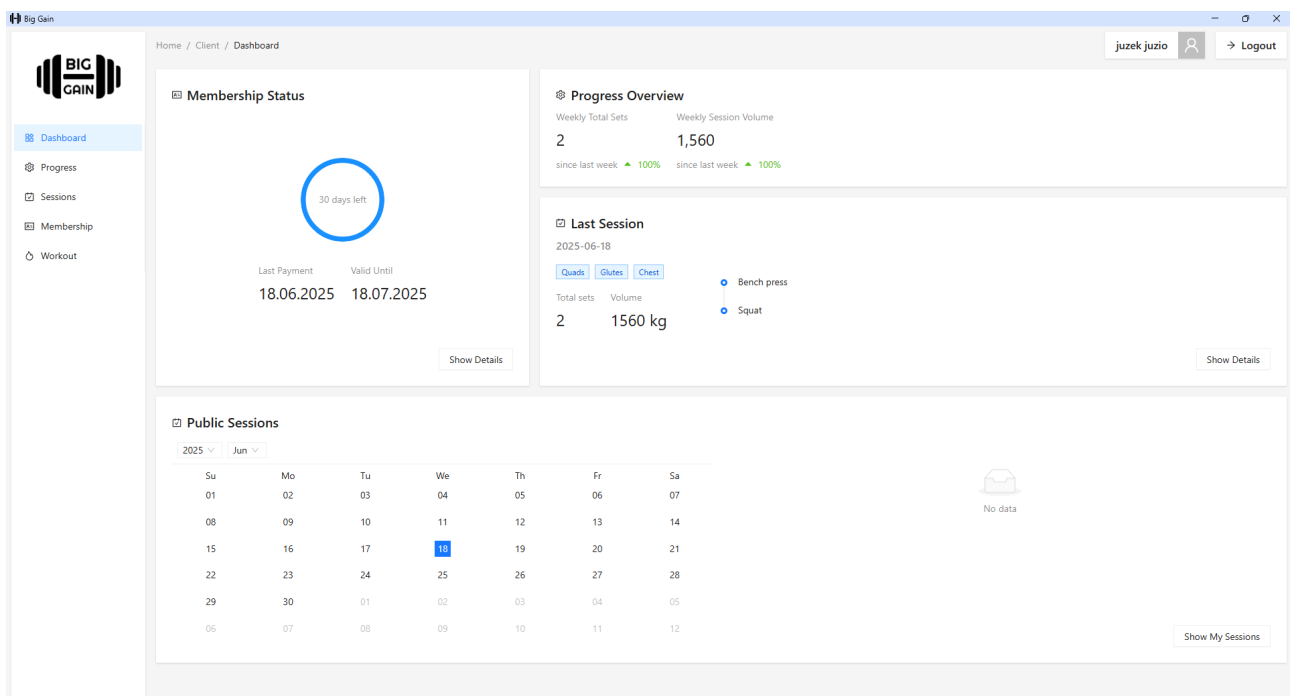
Aplikacja do zarządzania siłownią została zaprojektowana z myślą o różnych rolach użytkowników: klientach, pracownikach, trenerach oraz menadżerach. W zależności od przypisanej roli dostępne są różne funkcjonalności.

- **Klient** może:
  - Przeglądać status swojego karnetu.
  - Śledzić swój postęp treningowy.
  - Przeglądać i zapisywać się na dostępne sesje treningowe.
  - Przeglądać historię treningów.
- **Menadżer** ma możliwość:
  - Tworzenia nowych pracowników poprzez formularz.
  - Przeglądania listy klientów i pracowników.
  - Zarządzania salami treningowymi oraz zadaniami serwisowymi.
- **Pracownik** ma możliwość:
  - Przeglądania listy klientów.
  - Sprawdzania ważności karnetów.
  - Zgłaszania usterek i tworzenia zadań serwisowych.
- **Trener** może:
  - Przeglądać przypisanych podopiecznych.
  - Tworzyć i edytować plany treningowe.
  - Zarządzać kalendarzem sesji treningowych.

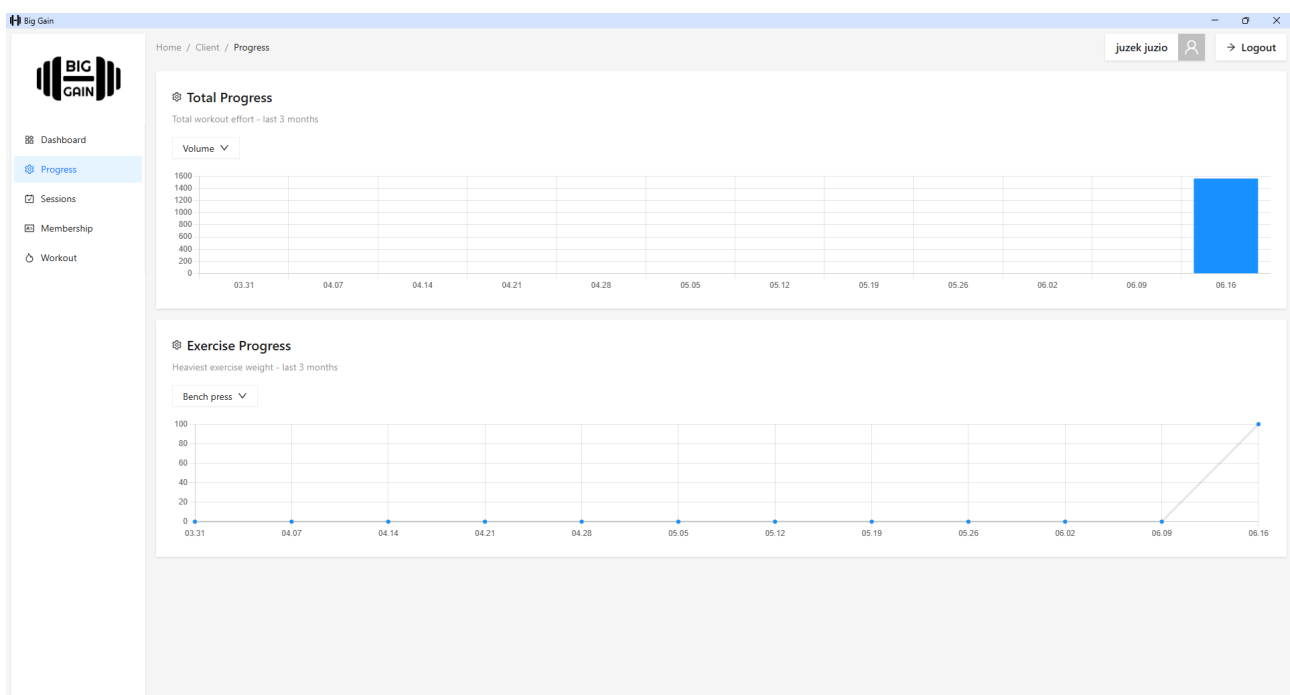
Aplikacja obsługuje różne widoki dla użytkowników w zależności od ich roli i zapewnia prosty oraz intuicyjny interfejs użytkownika, co przedstawiono na poniższych zrzutach ekranu.

*Ze względu na dużą ilość zakładek pokazane zostały najczęściej używane ekrany.*

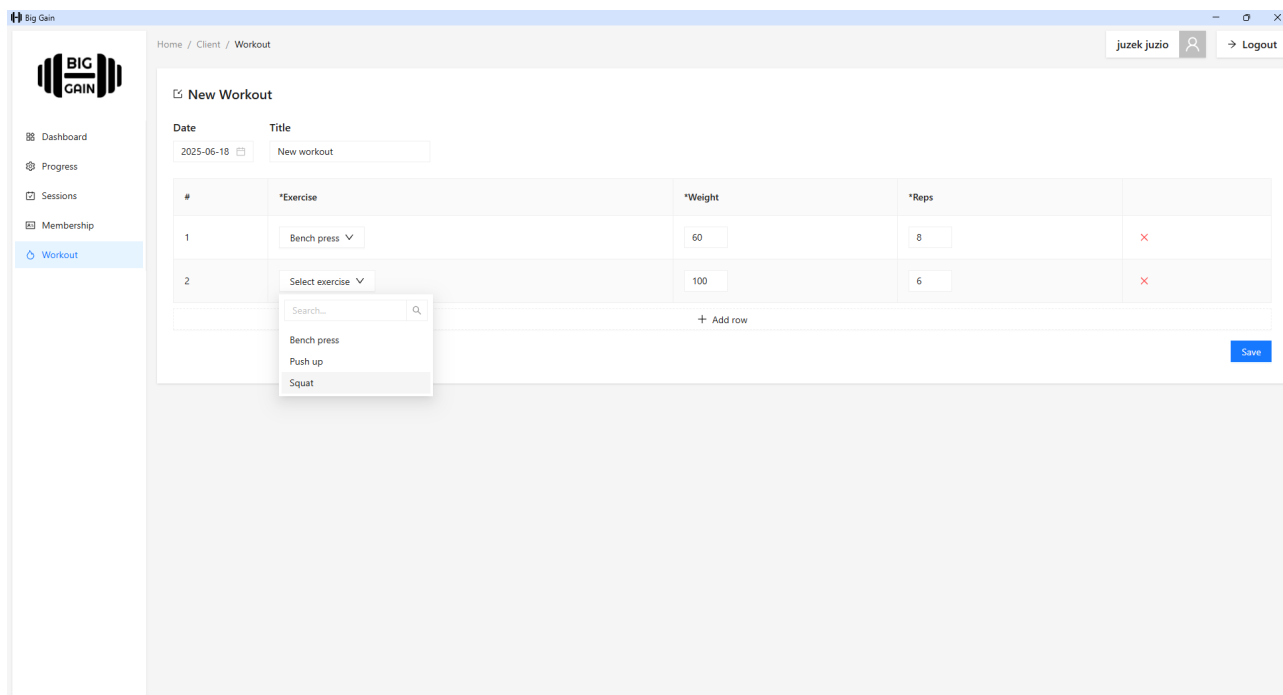
## 2.2 Prezentacja zrzutów ekranu (screeny) prezentujących działanie aplikacji



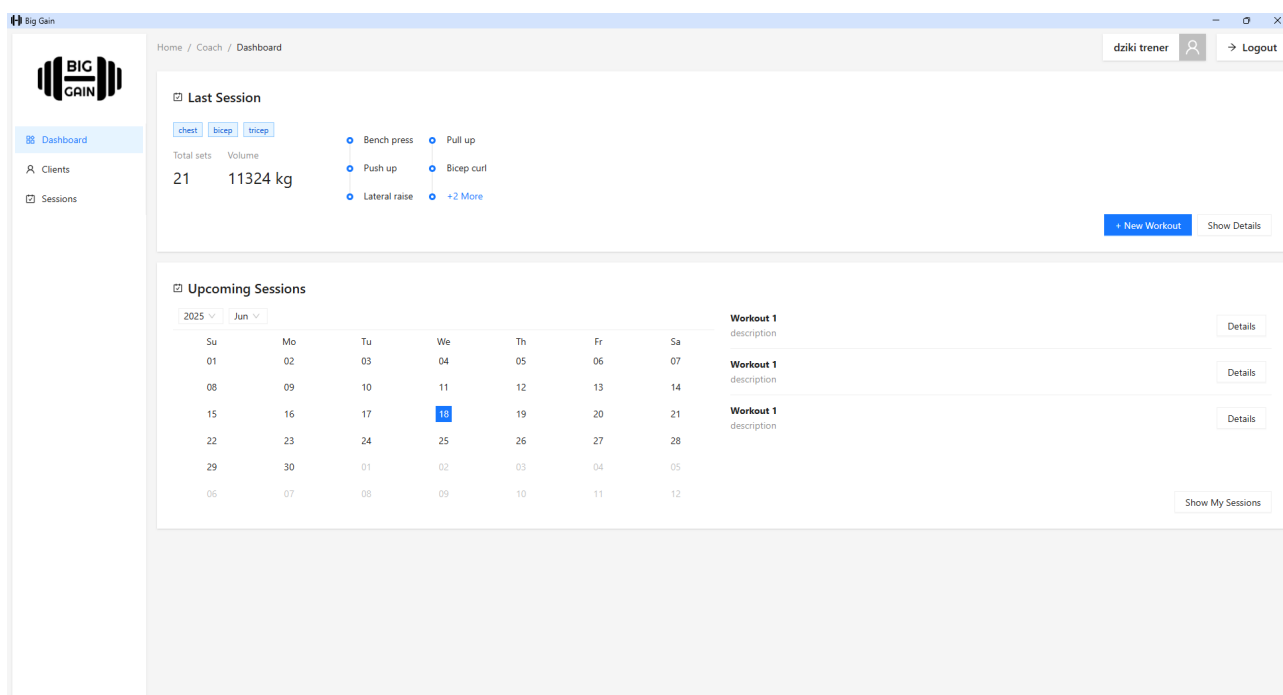
Rysunek 1: Panel klienta – przegląd statusu członkostwa, progresu oraz kalendarz sesji.



Rysunek 2: Klient może śledzić swoje statystyki.



Rysunek 3: Nowy trening - klient może tworzyć własne sesje.



Rysunek 4: Panel trenera - ostatni trening, kalendarz nadchodzących sesji.



Rysunek 5: Również trener może przeglądać statystyki treningowe klienta.

Home / Coach / New-session

dziki trener → Logout

**New Session**

Title: Workout session Date: 2025-06-17

Training Hall: 2A Time: 05:06

Description: kocham psk

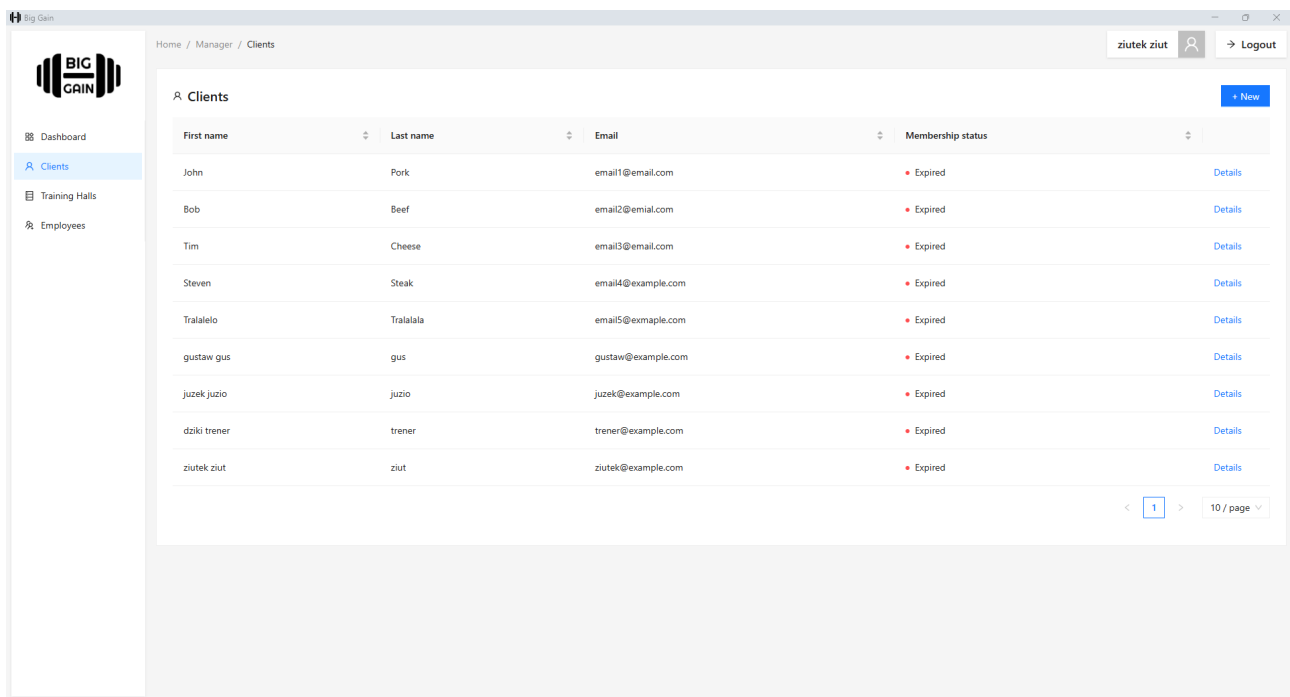
Exercises

#	*Exercise	*Weight	*Reps	
1	Bench press ▾	60	5	×
2	Squat ▾	100	6	×

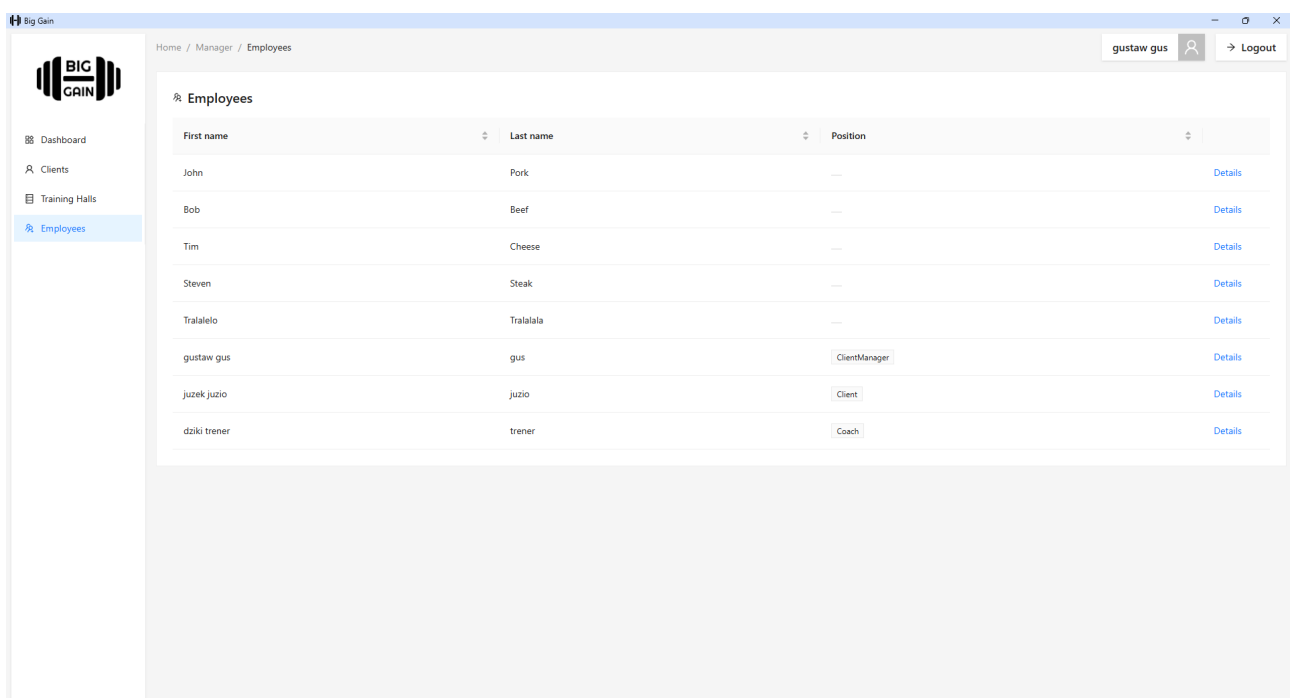
+ Add row

Create

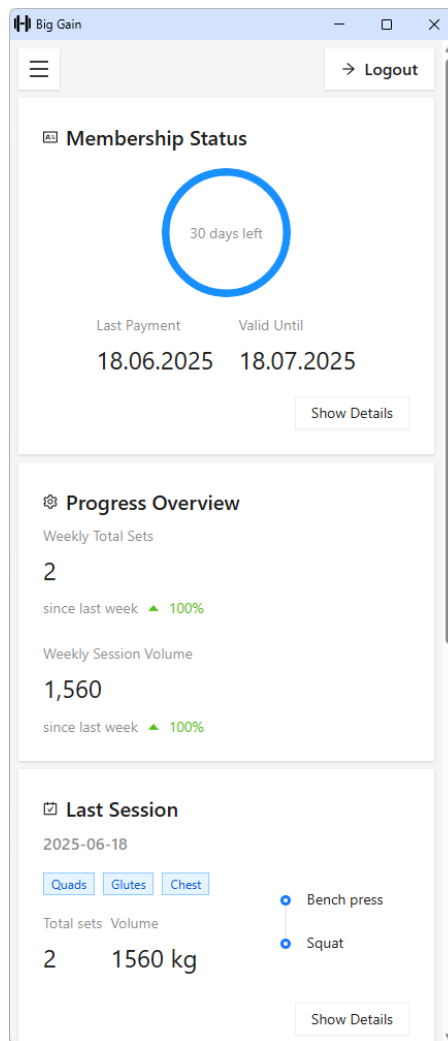
Rysunek 6: Tworzenie nowej sesji treningowej przez trenera.



Rysunek 7: Manager jak i pracownik mają dostęp do wszystkich klientów.



Rysunek 8: Manager może przeglądać dane pracowników.



Rysunek 9: Panel użytkownika na urządzeniu mobilnym - cała aplikacja jest responsywna.

## 2.3 Wybrane fragmenty kodu z kluczowymi funkcjonalnościami

```

1  function getValidityLabel(lastPayment?: Dayjs, validUntil?: Dayjs) {
2    if (!lastPayment || !validUntil) {
3      return 'Expired';
4    }
5
6    const now = dayjs();
7    const until = dayjs(validUntil);
8
9    if (until.isBefore(now)) {
10     return 'Expired';
11    }
12
13    const diffDays = until.diff(now, 'day');
14    const diffDaysWithToday = diffDays + 1;
15
16    if (diffDays >= 1) {
17     return `${diffDaysWithToday}_day${diffDaysWithToday===1?'':'s'}_left`;
18    }
19
20    const diffHours = until.diff(now, 'hour');
21    return `${diffHours}_hour${diffHours===1?'':'s'}_left`;
22  }
23
24  function getCirclePercentage(lastPayment?: Dayjs, validUntil?: Dayjs): number {
25    if (!lastPayment || !validUntil) return 0;
26
27    const now = dayjs();
28
29    if (now.isAfter(validUntil, 'day')) return 0;
30    if (now.isBefore(lastPayment, 'day')) return 100;
31
32    const total = validUntil.diff(lastPayment, 'days');
33    const elapsed = now.diff(lastPayment, 'days');
34
35    return Math.min(100, Math.max(0, (1 - elapsed / total) * 100));
36  }
37
38  // ...

```



```

39 const { lastPayment, validUntil } = props;
40 const lastPaymentDate = dayjs(lastPayment);
41 const validUntilDate = dayjs(validUntil);
42
43 <Progress
44   type='circle'
45   percent={
46     lastPayment && validUntilDate
47     ? getCirclePercentage(lastPaymentDate, validUntilDate)
48     : 0
49   }
50   format={() => (
51     <div className='text-font-secondary_text-sm'>
52       {getValidityLabel(lastPaymentDate, validUntilDate)}
53     </div>
54   )}
55   size={125}
56 >
57 // ...

```

Listing 1: Przykład logiki obliczeniowej w komponencie. Funkcje służą do wyznaczania wskaźników statusu karnetu dla komponentu *Progress* z *Ant Design*.

```

1  export function Chart(props: ChartProps) {
2    const { chartData, type, className, dropdownType } = props;
3
4    const initialChartDataEntry = chartData?.data?.at(0);
5
6    const [chartDataEntry, setChartDataEntry] = useState<ChartEntry | undefined>(
7      initialChartDataEntry
8    );
9
10   useEffect(() => {
11     setChartDataEntry(initialChartDataEntry);
12   }, [initialChartDataEntry]);
13
14   if (!chartData) {
15     return <Empty />;
16   }
17
18   const menuItems = chartData.data.map(({ title }) => ({ key: title, label: title }));
19
20   const dataExists =
21     chartDataEntry &&
22     chartDataEntry.timeSeries.labels.length > 0 &&
23     chartDataEntry.timeSeries.values.length > 0;
24
25   const chartComponentData = {
26     labels: chartDataEntry?.timeSeries.labels,
27     datasets: [
28       {
29         data: chartDataEntry?.timeSeries.values,
30         backgroundColor: getCSSVariable('--color-primary')
31       }
32     ]
33   };
34
35   const handleMenuItemSelect = (item: { key: string; label: string }) => {
36     const dataEntry = chartData.data.find(({ title }) => title === item.key);
37     setChartDataEntry(dataEntry);
38   };
39
40   const menu =
41     dropdownType === 'search' ? (
42       <SearchDropdown
43         placeholder={initialChartDataEntry?.title ?? 'Select'}
44         menuItems={menuItems}
45         onSelect={handleMenuItemSelect}
46       />
47     ) : (
48       <Dropdown
49         placeholder={initialChartDataEntry?.title ?? 'Select'}
50         menuItems={menuItems}
51         onSelect={handleMenuItemSelect}
52       />
53     );
54
55   const chart =
56     type === 'line' ? (
57       <Line options={chartComponentOptions} data={chartComponentData} />
58     ) : (
59       <Bar options={chartComponentOptions} data={chartComponentData} />
60     );
61
62   return (
63     <Flex vertical>
64       <Text className='text-font-secondary_mb-middle'>{chartData.description}</Text>
65       {menu}
66       <div className={`pt-middle_min-h-50_${className}`}>
67         {dataExists ? chart : <Empty />}
68       </div>
69     </Flex>
70   );
71 }

```

Listing 2: Przykład tworzenia wykresów. Komponent generuje wykres *Chart.js* na podstawie przekazanych parametrów. Możliwe jest wybranie typu wykresu oraz rozwijanej listy z kategoriami.

```

1  const columns: TableColumnsType<DataType> = [
2    {
3      title: 'First_name',
4      dataIndex: 'firstName',
5      key: 'firstName',
6      fixed: 'left',
7      sorter: (a, b) => a.firstName.localeCompare(b.firstName)
8    },
9    {
10     title: 'Last_name',
11     dataIndex: 'lastName',
12     key: 'lastName',
13     sorter: (a, b) => a.lastName.localeCompare(b.lastName)
14   },
15   {
16     title: 'Position',
17     dataIndex: 'position',
18     key: 'position',
19     sorter: (a, b) => a.position.localeCompare(b.position),
20     render: (position: string) => <Tag>{position}</Tag>
21   },
22   {
23     key: 'detailsHref',
24     dataIndex: 'detailsHref',
25     fixed: 'right',
26     width: 100,
27     render: (href: string) => <Link to={href}>Details</Link>
28   }
29 ];
30
31
32 export function EmployeesTableCard({ employees = [], newEmployeeHref }: EmployeesTableCardProps) {
33   return (
34     <Card className='w-full'>
35       <Flex vertical className='gap-layout'>
36         <Flex justify='space-between'>
37           <CardTitle title='Employees' icon='employees' />
38           {newEmployeeHref && (
39             <ActionButton primary href={newEmployeeHref}>
40               + New
41             </ActionButton>
42           )}
43         </Flex>
44         <Table<DataType>
45           pagination={false}
46           columns={columns}
47           dataSource={employees.map((e, i) => ({ key: i, ...e } ))}
48           scroll={{ x: 'max-content' }}
49         />
50       </Flex>
51     </Card>
52   );
53 }

```

Listing 3: Przykład generowania tabeli. Tworzenie tabeli w każdym przypadku odbywa się za pośrednictwem komponentu *Table* z *Ant Design*.

```

1  export function HallCreationCard({ hallTypes = [], onCreate = () => {} }: HallCreationCardProps) {
2    const [form] = Form.useForm();
3
4    return (
5      <Card>
6        <CardTitle title='Create_Hall' icon='training-halls' />
7        <Form
8          form={form}
9          layout='vertical'
10         onFinish={onCreate}
11         requiredMark={label => <span>{label}</span>}
12         className='pt-small'
13       >
14         <Flex className='gap-layout'>
15           <Flex vertical className='w-full'>
16             <Form.Item
17               label='Hall_Number'
18               name='hallNumber'
19               rules={[{ required: true, message: '' }]}
20             >
21               <Input placeholder='Enter_hall_number' />
22             </Form.Item>
23
24             <Form.Item
25               label='Type'
26               name='hallType'
27               rules={[{ required: true, message: '' }]}
28             >
29               <Select placeholder='Select_hall_type'>
30                 {hallTypes.filter(Boolean).map(type => (
31                   <Select.Option key={type.name} value={type.uuid}>
32                     {type.name}
33                   </Select.Option>
34                 ))}
35               </Select>
36             </Form.Item>
37           </Flex>
38
39           <Flex className='w-full' vertical>
40             <Form.Item
41               label='Description'
42               name='hallDescription'
43               rules={[{ required: true, message: '' }]}

```

```

44         >
45         <TextArea rows={5} placeholder='Enter_description' allowClear />
46       </Form.Item>
47     </Flex>
48   </Flex>
49
50   <Flex justify='end'>
51     <Button type='primary' htmlType='submit'>
52       Create
53     </Button>
54   </Flex>
55 </Form>
56 </Card>
57
58   );
59 }

```

Listing 4: Przykład obsługi formularza. Wykorzystywany jest *hook useForm* z *Ant Design* który pozwala na łatwą walidację i przesyłanie wartości z formularza.

```

1  export function UserProvider({ children }: { children: JSX.Element }) {
2    const [userDetails, setUserDetails] = useState<UserDetails>();
3
4    const updateUser = (updates: Partial<UserDetails>) => {
5      setUserDetails(prev => (prev ? { ...prev, ...updates } : prev));
6    };
7
8    useEffect(() => {
9      async function initUser() {
10        const initialized = await keycloak.init({ onLoad: 'login-required' });
11        if (!initialized) return;
12
13        initializeAxios(keycloak);
14
15        const userProfile = await keycloak.loadUserProfile();
16        const userRoles = keycloak.tokenParsed?.['roles'] as UserRole[];
17        console.log(keycloak.tokenParsed);
18        const significantRole = userRoles
19          .sort((a: UserRole, b: UserRole) => rolesPriority[b] - rolesPriority[a])
20          .at(0);
21
22        const whoAmIData = await whoAmI()
23          .then(data => data?.data)
24          .catch(() => {
25            console.error('failed_to_fetch_user_data._Is_backend_API_online?');
26            return undefined;
27          });
28
29        const userDetails: UserDetails = {
30          firstName: userProfile.firstName as string,
31          lastName: userProfile.lastName as string,
32          email: userProfile.email as string,
33          role: significantRole ?? 'client',
34          id: whoAmIData?.uuid,
35          hasValidMembership: dayjs().isBefore(dayjs(whoAmIData?.membership?.validUntil))
36        };
37
38        setUserDetails(userDetails);
39      }
40
41      if (import.meta.env.VITE_AUTH_ENABLED === 'true') {
42        initUser();
43      }
44    }, []);
45
46    return (
47      <UserContext.Provider value={{ user: userDetails, updateUser }}>
48        {children}
49      </UserContext.Provider>
50    );
51 }

```

Listing 5: Przykład globalnego stanu aplikacji. *UserProvider* dostarcza innym komponentom kontekst użytkownika pobrany z serwisu *Keycloak*.

```

1  export function useHTMLElementResizeObserver<T extends HTMLElement = HTMLDivElement>() {
2    const ref = useRef<T>(null);
3    const [width, setWidth] = useState(0);
4
5    useEffect(() => {
6      if (!ref.current) return;
7      const observer = new ResizeObserver(entries => {
8        for (const entry of entries) {
9          const { width } = entry.contentRect;
10          setWidth(width);
11        }
12      });
13
14      observer.observe(ref.current);
15      return () => observer.disconnect();
16    }, []);
17
18    return [ref, Math.floor(width)];
19 }

```

Listing 6: Przykład niestandardowego *hooka*. Służy do obserwacji szerokości elementu i jest wykorzystany przy obsłudze responsywności.

```

1 export function MembershipStatusCardWithData({ horizontal, showDetails }: MembershipStatusCardWithDataProps) {
2   const { user } = useUser();
3   const [membership, setMembership] = useState<Membership>();
4
5   useEffect(() => {
6     async function getMembershipStatus() {
7       if (!user?.id) return;
8       const userMembership = (await getUser(user.id)).data.membership;
9       setMembership(userMembership);
10    }
11
12    getMembershipStatus();
13  }, [user?.id]);
14
15  return (
16    <MembershipStatusCard
17      detailsHref={showDetails ? `/${user?.role}/membership` : undefined}
18      lastPayment={membership?.validFrom}
19      validUntil={membership?.validUntil}
20      horizontal={horizontal}
21    />
22  );
23 }

```

Listing 7: Przykład komponentu pobierającego dane z zewnętrznego API. Wykorzystywane są funkcje generowane przez *Orval* co eliminuje konieczność ręcznego korzystania z biblioteki *axios*.

```

1 export function Router() {
2   return (
3     <BrowserRouter>
4       <AuthGuard>
5         <Routes>
6           <Route path='renew-membership' element=<ClientRenewMembershipPage /> />
7           <Route
8             path={rolesConfig['client'].routePrefix}
9             element=<AuthenticatedLayout renderChat={false} role='client' />
10          >
11             <Route path='dashboard' element=<ClientDashboardPage /> />
12             <Route path='progress' element=<ClientProgressPage /> />
13             <Route path='sessions' element=<SessionsPage /> />
14             <Route path='membership' element=<ClientMembershipPage /> />
15             <Route path='workout' element=<ClientWorkoutPage /> />
16             <Route path='workout/:id' element=<WorkoutDetailsPage /> />
17           </Route>
18
19           <Route
20             path={rolesConfig['employee'].routePrefix}
21             element=<AuthenticatedLayout renderChat={false} role='employee' />
22          >
23             <Route path='dashboard' element=<EmployeeDashboardPage /> />
24             <Route path='clients' element=<ClientsPage /> />
25             <Route path='training-halls' element=<HallsPage /> />
26             <Route path='create-membership' element=<MembershipCreationPage /> />
27             <Route path='clients/:id' element=<ClientDetailsPage /> />
28             <Route path='training-halls/:id' element=<EmployeeHallDetailsPage /> />
29           </Route>
30
31           <Route
32             path={rolesConfig['coach'].routePrefix}
33             element=<AuthenticatedLayout renderChat={false} role='coach' />
34          >
35             <Route path='dashboard' element=<CoachDashboardPage /> />
36             <Route path='clients' element=<ClientsPage /> />
37             <Route path='sessions' element=<SessionsPage /> />
38             <Route path='workout/:id' element=<WorkoutDetailsPage /> />
39             <Route path='clients/details/:id' element=<CoachClientDetailsPage /> />
40             <Route path='new-session' element=<CoachNewSessionPage /> />
41           </Route>
42
43           <Route
44             path={rolesConfig['manager'].routePrefix}
45             element=<AuthenticatedLayout renderChat={false} role='manager' />
46          >
47             <Route path='dashboard' element=<ManagerDashboardPage /> />
48             <Route path='create-membership' element=<MembershipCreationPage /> />
49             <Route path='clients' element=<ClientsPage /> />
50             <Route path='clients/:id' element=<ClientDetailsPage /> />
51             <Route path='employees' element=<ManagerEmployeesPage /> />
52             <Route path='create-employee' element=<ManagerEmployeeCreationPage /> />
53             <Route path='employees/:id' element=<ManagerEmployeeDetailsPage /> />
54             <Route path='client-details' element=<ClientDetailsPage /> />
55             <Route path='training-halls' element=<HallsPage /> />
56             <Route path='training-halls/details/:id' element=<div></div> />
57             <Route path='create-hall' element=<ManagerHallCreationPage /> />
58           </Route>
59         </Routes>
60       </AuthGuard>
61     </BrowserRouter>
62   );
63 }

```

Listing 8: Komponent realizujący *routing* w aplikacji. Definiuje ścieżki i komponenty generujące poszczególne ekrany.

```

1 export function AuthGuard({ children }: AuthGuardProps) {
2   const { user } = useUser();
3   const location = useLocation();
4   const [loading, setLoading] = useState(false);
5
6   useEffect(() => {
7     setLoading(user === undefined);
8   }, [user]);
9
10  const isPublicPath = PUBLIC_PATHS.some(path => location.pathname.startsWith(path));
11
12  if (import.meta.env.VITE_AUTH_ENABLED === 'false' || isPublicPath) {
13    return children;
14  }
15
16  if (loading || !user) {
17    return <Loader />;
18  }
19
20  const { routePrefix, defaultRoute } = rolesConfig[user?.role];
21  const isPathAllowed = location.pathname.startsWith(routePrefix);
22
23  if (!isPathAllowed) {
24    return <Navigate to={routePrefix + defaultRoute} replace />;
25  } else if (isPathAllowed && user.role === 'client' && !user.hasValidMembership) {
26    return <Navigate to='/renew-membership' />;
27  } else {
28    return children;
29  }
30 }

```

Listing 9: Komponent chroniący podstrony - zapewnia przekierowania i wymusza logowanie użytkownika.

## 3 Testy

### 3.1 Opis metod testowania (np. testy manualne i automatyczne)

W projekcie zastosowano kombinację testów manualnych i automatycznych w celu zapewnienia jakości oprogramowania. Testy automatyczne skupiają się na komponentach interfejsu użytkownika, sprawdzając ich zachowanie w różnych scenariuszach. Testy manualne wykorzystano do weryfikacji ogólnej funkcjonalności systemu oraz integracji między modułami.

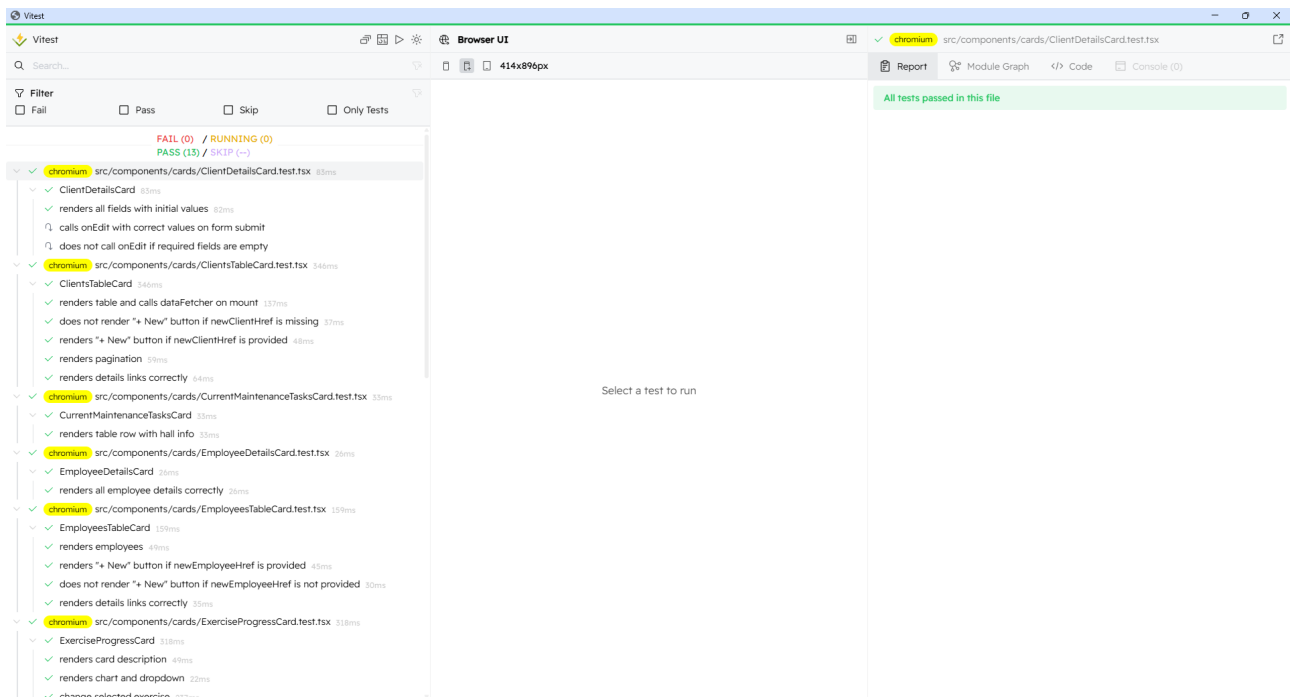
### 3.2 Wyniki testów, napotkane błędy oraz zastosowane rozwiązania

Przeprowadzone testy objęły wszystkie karty wykorzystywane w interfejsie użytkownika. Weryfikowano m.in. poprawność renderowania pól z określonymi wartościami, wywołania funkcji po wysłaniu formularza oraz widoczność elementów generowanych warunkowo.

Do testowania wykorzystano framework *Vitest* oraz tryb przeglądarkowy, umożliwiający podgląd na żywo przebiegu testów. Próby użycia trybu *headless* (bez interfejsu graficznego) skutkowały fałszywymi błędami, wynikającymi m.in. z braku interakcji z oknem (np. kliknięcia) lub problemów z mockowaniem natywnych funkcji przeglądarki, ponieważ testy w tym trybie uruchamiane są w środowisku *Node.js*. Wprowadzenie testowania w trybie przeglądarkowym wyeliminowało te problemy.

Podczas testów nie stwierdzono błędów w działaniu interfejsu – wszystkie kluczowe funkcjonalności działały zgodnie z oczekiwaniami. Jedyną napotkaną trudnością było połączenie interfejsu z API, co wymagało refaktoryzacji nazw atrybutów komponentów oraz ich dostosowania do przypadków, w których dane mogą nie zostać pobrane.

Znaczącym wyzwaniem okazała się również obsługa autoryzacji. Konieczne było utworzenie kontekstu użytkownika zintegrowanego z danymi pobieranymi z serwisu *Keycloak* oraz implementacja *interceptora* dla biblioteki *axios*, umożliwiającego prawidłowe odświeżanie sesji użytkownika. Dodatkowo potrzebny okazał się komponent odpowiedzialny za kontrolę dostępu do chronionych ścieżek i przekierowywanie niezalogowanych użytkowników. Ostatecznie proces logowania został pomyślnie wdrożony i działa poprawnie.



Rysunek 10: Wyniki testów interfejsu w środowisku Vitest.

## 4 Podsumowanie

### 4.1 Wnioski z realizacji projektu

Realizacja aplikacji do zarządzania siłownią pozwoliła zdobyć praktyczne doświadczenie w projektowaniu i implementacji złożonego interfejsu użytkownika z wykorzystaniem nowoczesnych rozwiązań frontendowych. W trakcie prac możliwe było zastosowanie w praktyce wiedzy dotyczącej architektury aplikacji typu SPA (*Single Page Application*), obsługi routingu, stanu aplikacji oraz integracji z zewnętrznymi usługami. Projekt umożliwił także lepsze zrozumienie zasad projektowania komponentów, tworzenia struktury kodu możliwej do wykorzystania w różnych częściach aplikacji oraz skutecznego testowania aplikacji — wcześniej wydawało się, że pisanie testów w aplikacjach *frontendowych* jest mało przydatne lub trudne do zrealizowania. Okazało się, że każdy z tych aspektów jest kluczowy dla prawidłowego działania aplikacji.

### 4.2 Ocena osiągniętych rezultatów i refleksje na temat procesu implementacji

Zrealizowana aplikacja spełnia założone wymagania funkcjonalne i techniczne. Interfejs użytkownika został zaprojektowany w sposób intuicyjny, a podział na role użytkowników (klient, pracownik, trener, menadżer) pozwala na klarowną separację uprawnień i dostępnych funkcji. Największym wyzwaniem okazała się integracja z backendem (spodziewaliśmy się, że pójdzie sprawniej), szczególnie podczas tworzenia i walidacji formularzy.

Pomimo relatywnie niewielkiej liczby funkcjonalności, projekt okazał się trudny i wyjątkowo czasochłonny. Każdy etap — od zaprojektowania interfejsu, przez jego wdrożenie i przetestowanie, aż po integrację z API — wymagał dużo zaangażowania i dokładności. Cały proces jest złożony i wymaga starannego planowania, co pokazuje, jak czasochłonne jest tworzenie nawet pozornie prostych aplikacji *frontendowych*.

### 4.3 Propozycje usprawnień lub dalszego rozwoju aplikacji

Aplikacja ma duży potencjał dalszego rozwoju. Do możliwych usprawnień należą:

- Wprowadzenie systemu powiadomień dla użytkowników (np. przypomnienia o wygasającym karnecie).
- Umożliwienie komunikacji pomiędzy trenerem a klientem (chat, wiadomości).
- Rozbudowa sekcji analitycznej dla menadżera, np. o raporty frekwencji czy obłożenia sal.
- Obsługa różnych rodzajów karnetów.
- Dodanie możliwości tworzenia i zarządzania kontami użytkowników bezpośrednio w aplikacji, zamiast manipulowania nimi w systemie *Keycloak*.
- Implementacja systemu ocen i opinii na temat trenerów i zajęć.

## 5 Podział pracy

- Filip Stępień – projekt graficzny aplikacji, przygotowanie i konfiguracja środowiska *developmenterskiego* oraz struktury projektu, implementacja komponentów, zapewnienie responsywności interfejsu, integracja z *backendem*, testy, dokumentacja.
- Rafał Grot – wdrożenie procesu automatyzacji (klient API), integracja z *backendem*, implementacja mechanizmów autoryzacji.
- Bartłomiej Karkoszka – implementacja komponentów, integracja z *backendem*, dokumentacja.

## 6 Literatura

- React – dokumentacja oficjalna, <https://react.dev/>, dostęp: 18.06.2025.
- Vite – dokumentacja oficjalna, <https://vite.dev/>, dostęp: 18.06.2025.
- Vitest – dokumentacja oficjalna, <https://vitest.dev/>, dostęp: 18.06.2025.
- React Router – dokumentacja oficjalna, <https://reactrouter.com/>, dostęp: 18.06.2025.
- Orval – dokumentacja oficjalna, <https://orval.dev>, dostęp: 18.06.2025.
- AntDesign – dokumentacja oficjalna, <https://ant.design/>, dostęp: 18.06.2025.
- Chart.js – dokumentacja oficjalna, <https://www.chartjs.org/>, dostęp: 18.06.2025.
- Keycloak JS – dokumentacja oficjalna, <https://www.keycloak.org/securing-apps/javascript-adapter>, dostęp: 18.06.2025.
- Tailwind CSS – dokumentacja oficjalna, <https://tailwindcss.com/>, dostęp: 18.06.2025.
- Day.js – dokumentacja oficjalna, <https://day.js.org/>, dostęp: 18.06.2025.
- Pawełkiwicz, Mateusz – wykłady z przedmiotu *Zaawansowane aplikacje frontendowe*, Politechnika Świętokrzyska, semestr letni 2025:
  - *Wprowadzenie do Frontend Developmentu,*
  - *Narzędzia i Frameworki,*
  - *Stylowanie i układ strony,*
  - *Zaawansowane techniki CSS,*
  - *Podstawy JavaScript,*
  - *Nowoczesny JavaScript, narzędzia i programowanie asynchroniczne,*
  - *Frameworki i Biblioteki Frontendowe – React, Angular, Vue.js,*
  - *Testowanie aplikacji frontendowych.*