ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

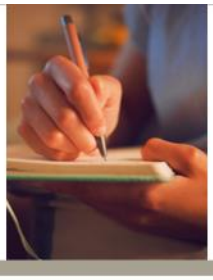
Кафедра ЕОМ

Кваліфікаційна робота на тему:

«Програмне забезпечення для ведення переліків бажань»

Здобувач:
Богдан РОГОЗЯНСЬКИЙ
КІУКІ-21-4

Керівник:
ас. Олег ЖУРИЛО



Актуальність

Аналіз існуючих рішень

	Сервіс	Платформа	Приватність	Платний доступ	Спеціалізація на бажаннях	Легкість опанування
	Listy	Мобільна	Часткова	Так (кастомізація)	Немає	Можливо
	Wisher	Мобільна	Відсутня	Ні	Так	Так
	Rewish	Мобільна + веб	Відсутня у free-версії	Так (для приватності)	Так	Так
	Notion	Усі	Повна (змінюється)	Ні (умовно, є платні плани)	Немає	Ні

Постановка задачі

- Створення/редагування/видалення переліків та бажань;
- Реєстрація та авторизація користувачів;
- Обмін переліками між друзями;
- Контроль доступу до переліків (приватні/публічні);
- Перегляд сповіщень про соціальні оновлення;
- Реалізувати зручний UI.

Обрані технології

Фреймворк, мова

TypeScript

NEXT.JS 15

CSS та компоненти

tailwindcss

shadcn/ui

Авторизація та автентифікація

Auth.js

JWT JSON WEB Token

Засіб розгортки та контейнеризації

docker

Управління БД

PostgreSQL

Prisma

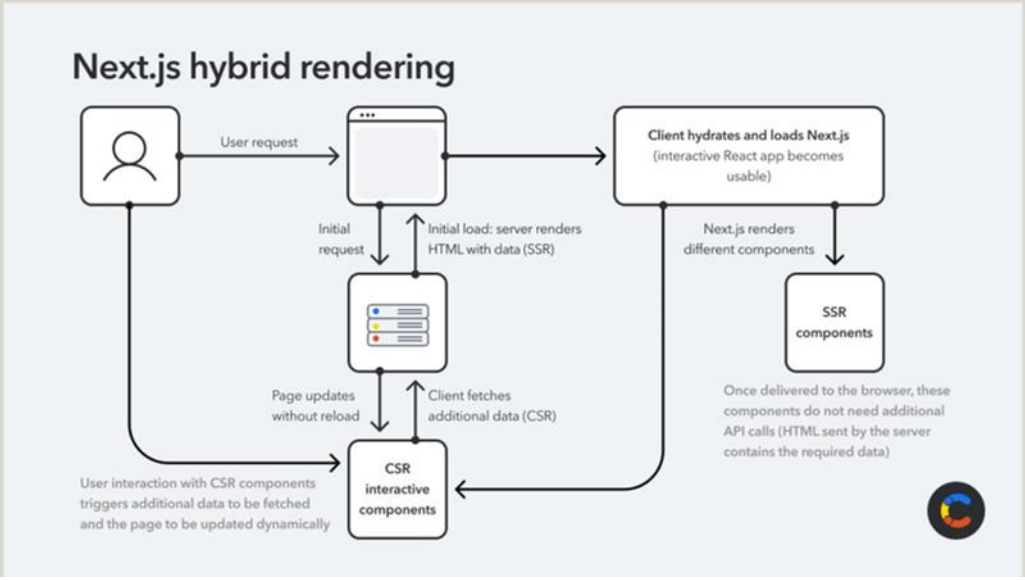
Розгортання та хостинг БД

Vercel

supabase

5

Архітектура застосунку

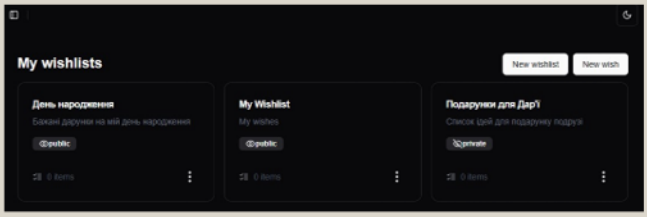


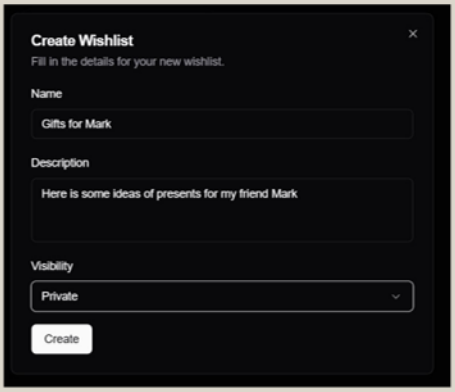
6

Схема бази даних

7

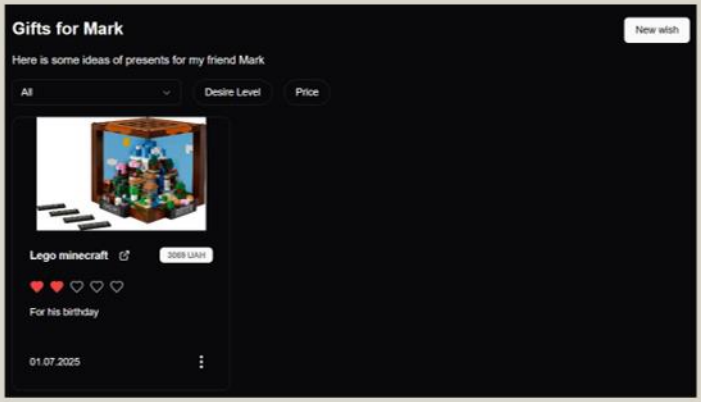
Створення переліків бажань

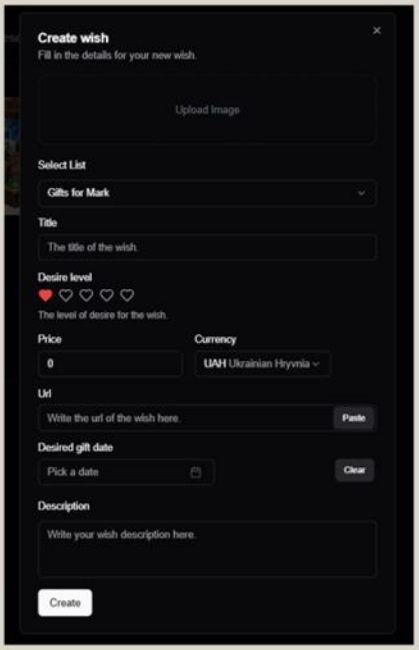




8

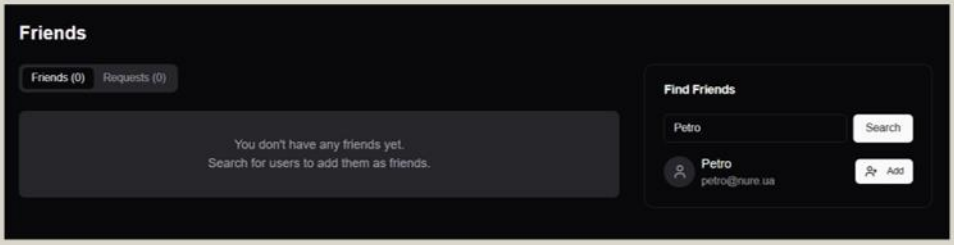
Створення бажання

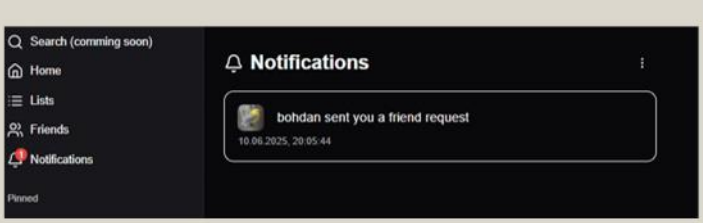


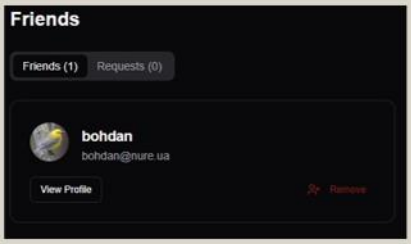


9

Надсилання запиту у друзі







10

Висновки

- Проведено аналіз ринку схожих застосунків;
- Розроблено вебзастосунок для ведення переліків бажань;
- Реалізовано зручний та адаптивний інтерфейс;
- Реалізовано соціальну взаємодію;
- Система підтримує розгортання локально (Docker) та в мережі (за допомогою Vercel);
- За результатами роботи були опубліковані тези на 29-му Міжнародному молодіжному форумі «Радіоелектроніка і молодь в XXI столітті».

ДОДАТОК Б

Вихідний код застосунку

Б.1 Конфігурація NextAuth

```
export const { handlers, signIn, signOut, auth } = NextAuth({
  adapter: PrismaAdapter(prisma),
  pages: {
    signIn: "/sign-in",
  },
  session: {
    strategy: "jwt",
  },
  providers: [
    Credentials({
      name: "credentials",
      credentials: {
        email: { label: "email", type: "text" },
        password: { label: "password", type: "password" },
      },
      async authorize(credentials) {
        const validateCredentials = z
          .object({
            email: z.string(),
            password: z.string(),
          })
          .parse(credentials);

        const user = await prisma.user.findUnique({
          where: {
            email: validateCredentials.email,
          },
        });

        if (!user || !user.password) return null;

        const passwordsMatch = await compare(
          validateCredentials.password,
          user.password,
        );

        if (passwordsMatch) return user;

        console.log("Invalid credentials");
        return null;
      },
    }),
```

```
    GitHub,
  ],
  callbacks: {
    async jwt({ token, user, trigger, session }) {
      if (user) {
        token.id = user.id;
        token.name = user.name;
        token.image = user.image;
      }
      if (trigger === "update" && session?.name &&
session?.email) {
        token.name = session.name;
        token.email = session.email;
      }
      if (trigger === "update" && session?.image) {
        token.image = session.image;
      }
      return token;
    },
    async session({ session, token }) {
      if (session.user) {
        session.user.id = token.id as string;
        session.user.name = token.name as string;
        session.user.image = token.image as string;
      }
      return session;
    },
    async redirect({ url, baseUrl }) {
      return "/";
    },
  },
});
```

Б.2 React-компонент WishDialog

```
interface WishDialogProps {
  wish: Wish | null;
  listId?: string;
  onClose: () => void;
  onSubmitAction: (
    data: WishSchemaType,
    id: string,
  ) => Promise<{ success: boolean; error?: string }>;
}

/**
 * A dialog to create or edit a wish. Pass `null` for `wish` to
create a new wish. Pass `wish` to edit an existing wish.
 *
 * For `listId`, pass the ID of the list if you are editing a
wish. If creating a new wish, `listId` is not required.
```

```
 *
 * @example
 * <WishDialog wish={wish} listId={listId} onClose={onClose}
onSubmitAction={editWishAction} />
 *
 * @example
 * <WishDialog wish={null} onClose={onClose}
onSubmitAction={createWishAction} />
 */
export function WishDialog({
  wish = null,
  listId,
  onClose,
  onSubmitAction,
}: WishDialogProps) {
  const { toast } = useToast();
  const { data: session } = useSession();
  const isEditing = !!wish; // isEditing is true, so wish and
listId are not null
  const [compressedFile, setCompressedFile] = useState<File |
null>(null);
  const [isCompressing, setIsCompressing] = useState(false);

  const [lists, setLists] = useState<{ id: string; name: string
}[]>([]);

  useEffect(() => {
    async function loadLists() {
      const fetchedLists = await getListsOptions();
      if (fetchedLists.success) {
        setLists(fetchedLists.data);
      }
    }
    loadLists();
  }, []);

  const form = useForm<WishSchemaType>({
    resolver: zodResolver(wishSchema),
    defaultValues: {
      listId,
      title: wish?.title || "",
      desireLvl: wish?.desireLvl || 1,
      price: wish?.price || 0,
      currency: wish?.currency || "UAH",
      url: wish?.url || "",
      description: wish?.description || "",
      desiredGiftDate: wish?.desiredGiftDate
        ? new Date(wish.desiredGiftDate)
        : undefined,
      imageUrl: wish?.imageUrl || DEFAULT_IMAGE_URL,
    },
    mode: "onTouched",
  });
```

```
  const onSubmit = async (values: WishSchemaType) => {
    const entityId = isEditing ? wish!.id : "";
    const userId = session?.user?.id!;

    const imageUrl =
      compressedFile && userId
        ? await fileUploadService.uploadFile(
            compressedFile,
            userId,
            BUCKETS.IMAGES,
          )
        : values.imageUrl;

    const { success, error } = await onSubmitAction(
      { ...values, imageUrl },
      entityId,
    );

    toast({
      title: success ? "Wish edited successfully!" : "Failed
editing wish",
      description: success ? undefined : error,
      variant: success ? undefined : "destructive",
    });

    if (success) onClose();
  };

  return (
    <Dialog open={true} onOpenChange={onClose}>
      <DialogContent className="max-h-[90vh] overflow-y-auto
sm:max-w-lg">
        <DialogHeader>
          <DialogTitle>{isEditing ? "Edit wish" : "Create
wish"}</DialogTitle>
          <DialogDescription>
            {isEditing
              ? "Make changes to your wish here."
              : "Fill in the details for your new wish."}
          </DialogDescription>
        </DialogHeader>

        <Form {...form}>
          <form onSubmit={form.handleSubmit(onSubmit)}
className="space-y-4">
            <ImageUploader
              onCompressed={(file) => setCompressedFile(file)}
              imageUrl={form.watch("imageUrl") as string}
              deleteImage={() => {
                setCompressedFile(null);
                form.setValue("imageUrl", DEFAULT_IMAGE_URL);
                setIsCompressing(false);
```

```
              }}
              isCompressing={isCompressing}
              setIsCompressing={setIsCompressing}
            />

            <FormField
              control={form.control}
              name="listId"
              render={({ field }) => (
                <FormItem>
                  <FormLabel>Select List</FormLabel>
                  <Select
                    onValueChange={field.onChange}
                    defaultValue={field.value}
                  >
                    <FormControl>
                      <SelectTrigger>
                        <SelectValue placeholder="Select a list"
/>
                      </SelectTrigger>
                    </FormControl>
                    <SelectContent>
                      {lists.map(({ id, name }) => (
                        <SelectItem key={id} value={id}
disabled={isEditing}>
                          {name}
                        </SelectItem>
                      ))}
                    </SelectContent>
                  </Select>
                  <FormMessage />
                </FormItem>
              )}
            />

            <FormField
              control={form.control}
              name="title"
              render={({ field }) => (
                <FormItem className="flex flex-col">
                  <FormLabel>Title</FormLabel>
                  <FormControl>
                    <Input {...field} placeholder="The title of
the wish." />
                  </FormControl>
                  <FormMessage />
                </FormItem>
              )}
            />

            <FormField
              control={form.control}
              name="desireLvl"
```

```
              render={({ field: { value, onChange } }) => (
                <FormItem className="flex flex-col">
                  <FormLabel>Desire level</FormLabel>
                  <FormControl>
                    <DesireLevel desireLvl={value}
setDesireLvl={onChange} />
                  </FormControl>
                  <FormDescription>
                    The level of desire for the wish.
                  </FormDescription>
                  <FormMessage />
                </FormItem>
              )}
            />

            <div className="flex gap-4">
              <FormField
                control={form.control}
                name="price"
                render={({ field }) => (
                  <FormItem className="flex flex-col">
                    <FormLabel>Price</FormLabel>
                    <FormControl>
                      <Input
                        {...field}
                        type="number"
                        placeholder="The price of wish."
                      />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />

              <FormField
                control={form.control}
                name="currency"
                render={({ field }) => (
                  <FormItem className="flex flex-col">
                    <FormLabel>Currency</FormLabel>
                    <Select
                      onValueChange={field.onChange}
                      defaultValue={field.value}
                    >
                      <FormControl>
                        <SelectTrigger>
                          <SelectValue placeholder="Select
currency" />
                        </SelectTrigger>
                      </FormControl>
                      <SelectContent>
                        {currencies.map(({ code, name }) => (
                          <SelectItem key={code} value={code}>
```

```
                            {code}{" "}
                            <span className="text-muted-
foreground">
                              {name}
                            </span>
                          </SelectItem>
                        ))}
                      </SelectContent>
                    </Select>
                    <FormMessage />
                  </FormItem>
                )}
              />
            </div>

            <FormField
              control={form.control}
              name="url"
              render={({ field }) => (
                <FormItem className="flex flex-col">
                  <FormLabel>Url</FormLabel>
                  <FormControl>
                    <div className="relative">
                      <Input
                        {...field}
                        placeholder="Write the url of the wish
here."
                      />
                      <Button
                        type="button"
                        variant="secondary"
                        size="sm"
                        className="absolute top-1/2 -translate-
y-1/2 right-1"
                        onClick={async () => {
                          try {
                            const text = await
navigator.clipboard.readText();
                            field.onChange(text);
                          } catch (error) {
                            toast({
                              title: "Error during pasting",
                              description: `${error}`,
                              variant: "destructive",
                            });
                          }
                        }}
                      >
                        Paste
                      </Button>
                    </div>
                  </FormControl>
                  <FormMessage />
```

```
                </FormItem>
              )}
            />

            <FormField
              control={form.control}
              name="desiredGiftDate"
              render={({ field }) => (
                <FormItem className="flex flex-col relative">
                  <FormLabel className="w-fit">Desired gift
date</FormLabel>
                  <Popover>
                    <PopoverTrigger asChild>
                      <FormControl>
                        <Button
                          variant={"outline"}
                          className={cn(
                            "w-[240px] pl-3 text-left font-
normal",
                            !field.value && "text-muted-
foreground",
                          )}
                        >
                          {field.value ? (
                            format(field.value, "PPP")
                          ) : (
                            <span>Pick a date</span>
                          )}
                          <CalendarIcon className="ml-auto h-4
w-4 opacity-50" />
                        </Button>
                      </FormControl>
                    </PopoverTrigger>

                    <PopoverContent className="w-auto p-0"
align="start">
                      <Calendar
                        mode="single"
                        selected={
                          field.value === null ? undefined :
field.value
                        }
                        onSelect={field.onChange}
                        initialFocus
                      />
                    </PopoverContent>
                  </Popover>
                  <Button
                    type="button"
                    size="sm"
                    variant="secondary"
                    className="absolute top-1/2 -translate-y-1/2
right-1"
```

```
                    onClick={() =>
                      form.setValue("desiredGiftDate", null, {
                        shouldValidate: true,
                      })
                    }
                  >
                    Clear
                  </Button>
                  <FormMessage />
                </FormItem>
              )}
            />

            <FormField
              control={form.control}
              name="description"
              render={({ field }) => (
                <FormItem>
                  <FormLabel>Description</FormLabel>
                  <FormControl>
                    <Textarea
                      placeholder="Write your wish description
here."
                      className="resize-none"
                      rows={3}
                      {...field}
                    />
                  </FormControl>

                  <FormMessage />
                </FormItem>
              )}
            />

            <Button disabled={isCompressing} type="submit">
              {isEditing ? "Save" : "Create"}
            </Button>
          </form>
        </Form>
      </DialogContent>
    </Dialog>
  );
}
```

Б.3 React-компонент WishlistDialog

```
const visibilityOptions = [
  { value: "private", label: "Private" },
  { value: "public", label: "Public" },
];
```

```
interface WishlistDialogProps {
  wishlist: List | null;
  onClose: () => void;
  onSubmitAction: (
    data: WishlistSchemaType,
    id: string,
  ) => Promise<{ success: boolean; error?: string }>;
}

export function WishlistDialog({
  wishlist = null,
  onClose,
  onSubmitAction,
}: WishlistDialogProps) {
  const isEditing = !!wishlist;
  const updateOrCreateEntityId = wishlist?.id || "null";

  const form = useForm({
    resolver: zodResolver(wishlistSchema),
    defaultValues: {
      name: wishlist?.name || "",
      description: wishlist?.description || "",
      visibility: wishlist?.visibility || VISIBILITY.PUBLIC,
    },
    mode: "onTouched",
  });

  const onSubmit = async (values: WishlistSchemaType) => {
    const response = await onSubmitAction(values,
updateOrCreateEntityId);

    if (response.success) {
      toast({
        title: `Wishlist ${isEditing ? "edited" : "created"}
successfully!`,
      });
      onClose();
    } else {
      toast({
        title: "Error",
        description: response.error,
        variant: "destructive",
      });
    }
  };

  return (
    <Dialog open={true} onOpenChange={onClose}>
      <DialogContent>
        <DialogHeader>
          <DialogTitle>
            {isEditing ? "Edit Wishlist" : "Create Wishlist"}
          </DialogTitle>
```

```
          <DialogDescription>
            {isEditing
              ? "Make changes to your wishlist."
              : "Fill in the details for your new wishlist."}
          </DialogDescription>
        </DialogHeader>

        <Form {...form}>
          <form onSubmit={form.handleSubmit(onSubmit)}
classNam="space-y-4">
            <FormField
              control={form.control}
              name="name"
              render={({ field }) => (
                <FormItem>
                  <FormLabel>Name</FormLabel>
                  <FormControl>
                    <Input {...field} placeholder="Wishlist
name" />
                  </FormControl>
                  <FormMessage />
                </FormItem>
              )}
            />

            <FormField
              control={form.control}
              name="description"
              render={({ field }) => (
                <FormItem>
                  <FormLabel>Description</FormLabel>
                  <FormControl>
                    <Textarea
                      {...field}
                      placeholder="Write wishist description
here."
                      className="resize-none"
                      rows={3}
                    />
                  </FormControl>
                  <FormMessage />
                </FormItem>
              )}
            />

            <FormField
              control={form.control}
              name="visibility"
              render={({ field }) => (
                <FormItem>
                  <FormLabel>Visibility</FormLabel>
                  <Select
                    onValueChange={field.onChange}
```

```
                    defaultValue={field.value}
                  >
                    <FormControl>
                      <SelectTrigger>
                        <SelectValue placeholder="Select
visibility" />
                      </SelectTrigger>
                    </FormControl>
                    <SelectContent>
                      {visibilityOptions.map(({ value, label })
=> (
                        <SelectItem key={value} value={value}>
                          {label}
                        </SelectItem>
                      ))}
                    </SelectContent>
                  </Select>
                  <FormMessage />
                </FormItem>
              )}
            />

            <Button type="submit">{isEditing ? "Save" :
"Create"}</Button>
          </form>
        </Form>
      </DialogContent>
    </Dialog>
  );
}
```

Б.4 React-компонент FriendsList

```
interface FriendUser {
  id: string;
  name: string | null;
  email: string | null;
  image: string | null;
}

interface FriendRequest {
  id: string;
  sender: FriendUser;
}

interface FriendsListProps {
  friends: FriendUser[];
  pendingRequests: FriendRequest[];
}

export function FriendsList({ friends, pendingRequests }:
```

```
FriendsListProps) {
  const [localFriends, setLocalFriends] = useState(friends);
  const [localRequests, setLocalRequests] =
useState(pendingRequests);
  const { toast } = useToast();

  const searchParams = useSearchParams();
  const router = useRouter();
  const pathname = usePathname();

  const tabFromUrl = searchParams.get("tab") ??
FRIENDS_TABS.FRIENDS;
  const [activeTab, setActiveTab] = useState(tabFromUrl);

  const handleRemoveFriend = async (friendId: string) => {
    try {
      const result = await removeFriend(friendId);
      if (result.success) {
        setLocalFriends((prev) => prev.filter((f) => f.id !==
friendId));
        toast({ title: "Friend removed" });
      } else if (result.error) {
        toast({
          title: "Error",
          description: result.error,
          variant: "destructive",
        });
      }
    } catch (error) {
      console.error("Error removing friend:", error);
      toast({
        title: "Error",
        description: "Failed to remove friend",
        variant: "destructive",
      });
    }
  };

  const handleAcceptRequest = async (friendshipId: string) => {
    try {
      const result = await acceptFriendRequest(friendshipId);
      if (result.success) {
        const sender = localRequests.find((r) => r.id ===
friendshipId)?.sender;
        if (sender) {
          setLocalFriends((prev) => [...prev, sender]);
        }
        // Remove from pending requests
        setLocalRequests((prev) => prev.filter((r) => r.id !==
friendshipId));
        toast({ title: "Friend request accepted" });
      } else if (result.error) {
        toast({
```

```
          title: "Error",
          description: result.error,
          variant: "destructive",
        });
      }
    } catch (error) {
      console.error("Error accepting request:", error);
      toast({
        title: "Error",
        description: "Failed to accept friend request",
        variant: "destructive",
      });
    }
  };

  const handleDeclineRequest = async (friendshipId: string) => {
    try {
      const result = await declineFriendRequest(friendshipId);
      if (result.success) {
        setLocalRequests((prev) => prev.filter((r) => r.id !==
friendshipId));
        toast({ title: "Friend request declined" });
      } else if (result.error) {
        toast({
          title: "Error",
          description: result.error,
          variant: "destructive",
        });
      }
    } catch (error) {
      console.error("Error declining request:", error);
      toast({
        title: "Error",
        description: "Failed to decline friend request",
        variant: "destructive",
      });
    }
  };

  return (
    <Tabs
      defaultValue={FRIENDS_TABS.FRIENDS}
      value={activeTab}
      onValueChange={(tab) => {
        const params = new URLSearchParams(searchParams);
        params.set("tab", tab);
        router.replace(`${pathname}?${params.toString()}`);
        setActiveTab(tab);
      }}
    >
      <TabsList className="mb-4">
        <TabsTrigger value={FRIENDS_TABS.FRIENDS}>
          Friends ({localFriends.length})
```

```
        </TabsTrigger>
        <TabsTrigger value="requests">
          Requests ({localRequests.length})
        </TabsTrigger>
      </TabsList>

      <TabsContent value={FRIENDS_TABS.FRIENDS}>
        {localFriends.length === 0 ? (
          <div className="text-center p-8 bg-muted rounded-lg">
            <p className="text-muted-foreground">
              You don&apos;t have any friends yet.
            </p>
            <p className="text-muted-foreground">
              Search for users to add them as friends.
            </p>
          </div>
        ) : (
          <div className="grid grid-cols-1 md:grid-cols-2 gap-
4">
            {localFriends.map((friend) => (
              <Card key={friend.id}>
                <CardHeader className="flex flex-row items-
center gap-4 pb-2">
                  <div className="relative h-12 w-12 rounded-
full overflow-hidden">
                    {friend.image ? (
                      <Image
                        src={friend.image || ""}
                        alt={friend.name || "Friend"}
                        fill
                        className="object-cover"
                      />
                    ) : (
                      <div className="flex h-full w-full items-
center justify-center bg-muted">
                        <User className="h-6 w-6 text-muted-
foreground" />
                      </div>
                    )}
                  </div>
                  <div>
                    <CardTitle className="text-lg">
                      {friend.name || "User"}
                    </CardTitle>

<CardDescription>{friend.email}</CardDescription>
                  </div>
                </CardHeader>
                <CardFooter className="flex justify-between pt-
2">
                  <Link href={`/friends/${friend.id}`}>
                    <Button size="sm" variant="outline">
                      View Profile
```

```
                    </Button>
                  </Link>
                  <Button
                    size="sm"
                    variant="ghost"
                    className="text-destructive"
                    onClick={() =>
handleRemoveFriend(friend.id)}
                  >
                    <UserX className="h-4 w-4 mr-1" />
                    Remove
                  </Button>
                </CardFooter>
              </Card>
            ))}
          </div>
        )}
      </TabsContent>

      <TabsContent value={FRIENDS_TABS.REQUESTS}>
        {localRequests.length === 0 ? (
          <div className="text-center p-8 bg-muted rounded-lg">
            <p className="text-muted-foreground">
              You don&apos;t have any pending friend requests.
            </p>
          </div>
        ) : (
          <div className="grid grid-cols-1 md:grid-cols-2 gap-
4">
            {localRequests.map((request) => (
              <Card key={request.id}>
                <CardHeader className="flex flex-row items-
center gap-4 pb-2">
                  <div className="relative h-12 w-12 rounded-
full overflow-hidden">
                    {request.sender.image ? (
                      <Image
                        src={request.sender.image || ""}
                        alt={request.sender.name || "User"}
                        fill
                        className="object-cover"
                      />
                    ) : (
                      <div className="flex h-full w-full items-
center justify-center bg-muted">
                        <User className="h-6 w-6 text-muted-
foreground" />
                      </div>
                    )}
                  </div>
                  <div>
                    <CardTitle className="text-lg">
                      {request.sender.name || "User"}
```

```
                    </CardTitle>

<CardDescription>{request.sender.email}</CardDescription>
                  </div>
                </CardHeader>
                <CardFooter className="flex justify-between pt-
2">
                  <Button
                    size="sm"
                    onClick={() =>
handleAcceptRequest(request.id)}
                  >
                    <Check className="h-4 w-4 mr-1" />
                    Accept
                  </Button>
                  <Button
                    size="sm"
                    variant="outline"
                    onClick={() =>
handleDeclineRequest(request.id)}
                  >
                    <X className="h-4 w-4 mr-1" />
                    Decline
                  </Button>
                </CardFooter>
              </Card>
            ))}
          </div>
        )}
      </TabsContent>
    </Tabs>
  );
}
```

Б.5 React-компонент UserSearch

```
interface UserResult {
  id: string;
  name: string | null;
  email: string | null;
  image: string | null;
  status: string;
  friendshipId?: string | null;
}

export function UserSearch() {
  const [query, setQuery] = useState("");
  const [results, setResults] = useState<UserResult[]>([]);
  const [isSearching, setIsSearching] = useState(false);
  const { toast } = useToast();
```

```
  const handleSearch = async () => {
    if (query.length < 2) {
      toast({ title: "Please enter at least 2 characters to
search" });
      return;
    }

    setIsSearching(true);
    try {
      const result = await searchUsers(query);
      if (result.success) {
        setResults(result.users);
      } else if (result.error) {
        toast({
          title: "Error",
          description: result.error,
          variant: "destructive",
        });
      }
    } catch (error) {
      console.error("Error searching user:", error);
      toast({
        title: "Error",
        description: "Failed to search users",
        variant: "destructive",
      });
    } finally {
      setIsSearching(false);
    }
  };

  const handleSendRequest = async (userId: string) => {
    try {
      const result = await sendFriendRequest(userId);
      if (result.success) {
        // Update the local state to reflect the sent request
        setResults((prev) =>
          prev.map((user) =>
            user.id === userId
              ? {
                  ...user,
                  status: "pending",
                  friendshipId: result.friendship.id,
                }
              : user,
          ),
        );
        toast({ title: "Friend request sent" });
      } else if (result.error) {
        toast({
          title: "Error",
          description: result.error,
          variant: "destructive",
```

```
        });
      }
    } catch (error) {
      console.error("Error sending friend request:", error);
      toast({
        title: "Error",
        description: "Failed to send friend request",
        variant: "destructive",
      });
    }
  };

  const handleAcceptRequest = async (friendshipId: string,
userId: string) => {
    try {
      const result = await acceptFriendRequest(friendshipId);
      if (result.success) {
        // Update the local state to reflect the accepted
request
        setResults((prev) =>
          prev.map((user) =>
            user.id === userId ? { ...user, status: "accepted" }
: user,
          ),
        );
        toast({ title: "Friend request accepted" });
      } else if (result.error) {
        toast({
          title: "Error",
          description: result.error,
          variant: "destructive",
        });
      }
    } catch (error) {
      console.error("Error accepting friend request:", error);
      toast({
        title: "Error",
        description: "Failed to accept friend request",
        variant: "destructive",
      });
    }
  };

  const handleDeclineRequest = async (friendshipId: string,
userId: string) => {
    try {
      const result = await declineFriendRequest(friendshipId);
      if (result.success) {
        // Update the local state to reflect the declined
request
        setResults((prev) =>
          prev.map((user) =>
            user.id === userId ? { ...user, status: "rejected" }
```

```
: user,
          ),
        );
        toast({ title: "Friend request declined" });
      } else if (result.error) {
        toast({
          title: "Error",
          description: result.error,
          variant: "destructive",
        });
      }
    } catch (error) {
      console.error("Error declining friend request:", error);
      toast({
        title: "Error",
        description: "Failed to decline friend request",
        variant: "destructive",
      });
    }
  };

  const getActionButton = (user: UserResult) => {
    switch (user.status) {
      case "accepted":
        return (
          <Button size="sm" variant="ghost" disabled>
            <UserCheck className="h-4 w-4 mr-1" />
            Friends
          </Button>
        );
      case "pending":
        return (
          <Button size="sm" variant="ghost" disabled>
            <Clock className="h-4 w-4 mr-1" />
            Pending
          </Button>
        );
      case "received_pending":
        return (
          <div className="flex gap-1">
            <Button
              size="sm"
              variant="outline"
              className="px-2"
              onClick={() =>
                user.friendshipId &&
                handleAcceptRequest(user.friendshipId, user.id)
              }
            >
              <Check className="h-4 w-4" />
            </Button>
            <Button
              size="sm"
```

```
                variant="outline"
                className="px-2"
                onClick={() =>
                  user.friendshipId &&
                  handleDeclineRequest(user.friendshipId, user.id)
                }
              >
                <X className="h-4 w-4" />
              </Button>
            </div>
          );
        default:
          return (
            <Button size="sm" onClick={() =>
handleSendRequest(user.id)}>
              <UserPlus className="h-4 w-4 mr-1" />
              Add
            </Button>
          );
      }
    };

    return (
      <Card>
        <CardHeader>
          <CardTitle>Find Friends</CardTitle>
        </CardHeader>
        <CardContent>
          <div className="flex gap-2 mb-4">
            <Input
              placeholder="Search by name or email"
              value={query}
              onChange={(e) => setQuery(e.target.value)}
              onKeyDown={(e) => e.key === "Enter" &&
handleSearch()}
            />
            <Button onClick={handleSearch} disabled={isSearching}>
              {isSearching ? "Searching..." : "Search"}
            </Button>
          </div>

          {results.length > 0 ? (
            <div className="space-y-4">
              {results.map((user) => (
                <div key={user.id} className="flex items-center
justify-between">
                  <div className="flex items-center gap-2">
                    <div className="relative h-10 w-10 rounded-
full overflow-hidden">
                      {user.image ? (
                        <Image
                          src={user.image || ""}
                          alt={user.name || "User"}
```

```
                        fill
                        className="object-cover"
                      />
                    ) : (
                      <div className="flex h-full w-full items-
center justify-center bg-muted">
                        <User className="h-5 w-5 text-muted-
foreground" />
                      </div>
                    )}
                  </div>
                  <div>
                    <div className="font-medium">{user.name ||
"User"}</div>
                    <div className="text-sm text-muted-
foreground">
                      {user.email}
                    </div>
                  </div>
                </div>

                {getActionButton(user)}
              </div>
            ))}
          </div>
        ) : (
          <div className="text-center p-4 bg-muted rounded-lg">
            <p className="text-muted-foreground">
              {query.length > 0
                ? "No users found. Try a different search term."
                : "Search for users by name or email."}
            </p>
          </div>
        )}
      </CardContent>
    </Card>
  );
}
```

Б.6 Головні Next.js події застосунка «app/actions.ts»

```
export async function signOutAction() {
  await signOut({ redirectTo: "/" });
}

export async function updateProfile(data: ProfileFormSchemaType)
{
  const session = await auth();

  if (!session?.user?.id) {
    return { error: "You must be logged in to update your
```

```
profile." };
  }

  const parsed = profileFormSchema.safeParse(data);

  if (!parsed.success) {
    return {
      success: false,
      error: parsed.error.toString(),
    };
  }

  try {
    if (!session?.user?.email)
      return { success: false, error: "User email not found." };

    const existingUser = await prisma.user.findUnique({
      where: { email: session?.user?.email },
    });

    if (existingUser && existingUser.id !== session.user.id) {
      return { success: false, error: "Email is already taken."
};
    }

    // Update the user profile
    const updatedUser = await prisma.user.update({
      where: { id: session.user.id },
      data: {
        name: parsed.data.name,
        email: parsed.data.email,
        ...(parsed.data.avatar && { image: parsed.data.avatar
}),
      },
    });

    return { success: true, data: updatedUser };
  } catch (error) {
    console.error("Error updating profile:", error);
    return {
      success: false,
      error: "Failed to update profile. Please try again.",
    };
  }
}

// Send a friend request
export async function sendFriendRequest(receiverId: string) {
  const session = await auth();

  if (!session?.user?.id) {
    return { error: "You must be logged in to send a friend
request." };
```

```
  }

  const senderId = session.user.id;

  // Don't allow sending requests to yourself
  if (senderId === receiverId) {
    return { error: "You cannot send a friend request to
yourself." };
  }

  // Check if a friendship already exists
  const existingFriendship = await prisma.friendship.findFirst({
    where: {
      OR: [
        { senderId, receiverId },
        { senderId: receiverId, receiverId: senderId },
      ],
    },
  });

  if (existingFriendship) {
    if (existingFriendship.status === FriendshipStatus.ACCEPTED)
{
      return { error: "You are already friends with this user."
};
    } else if (existingFriendship.status ===
FriendshipStatus.PENDING) {
      if (existingFriendship.senderId === senderId) {
        return {
          error: "You have already sent a friend request to this
user.",
        };
      } else {
        return {
          error:
            "This user has already sent you a friend request.
Check your notifications.",
        };
      }
    } else if (existingFriendship.status ===
FriendshipStatus.REJECTED) {
      // If previously rejected, allow to send again by updating
the status
      await prisma.friendship.update({
        where: { id: existingFriendship.id },
        data: {
          status: FriendshipStatus.PENDING,
          senderId: senderId,
          receiverId: receiverId,
          updatedAt: new Date(),
        },
      });
```

```
      // Create a notification for the receiver
      await prisma.notifications.create({
        data: {
          notifiedId: receiverId,
          notifierId: senderId,
          type: NotificationType.FRIEND_REQUEST,
        },
      });

      revalidatePath("/friends");
      revalidatePath("/notifications");
      return { success: true, friendship: existingFriendship };
    }
  }

  // Create the friendship with PENDING status
  try {
    const friendship = await prisma.friendship.create({
      data: {
        senderId,
        receiverId,
        status: FriendshipStatus.PENDING,
      },
    });

    // Create a notification for the receiver
    await prisma.notifications.create({
      data: {
        notifiedId: receiverId,
        notifierId: senderId,
        type: NotificationType.FRIEND_REQUEST,
      },
    });

    revalidatePath("/notifications");
    revalidatePath("/friends");
    return { success: true, friendship };
  } catch (error) {
    console.error("Error sending friend request:", error);
    return { error: "Failed to send friend request. Please try
again." };
  }
}

// Accept a friend request
export async function acceptFriendRequest(friendshipId: string)
{
  const session = await auth();

  if (!session?.user?.id) {
    return { error: "You must be logged in to accept a friend
request." };
  }
```

```
  const userId = session.user.id;

  try {
    // Get the friendship
    const friendship = await prisma.friendship.findUnique({
      where: { id: friendshipId },
      include: { sender: true },
    });

    if (!friendship) {
      return { error: "Friend request not found." };
    }

    if (friendship.receiverId !== userId) {
      return { error: "You are not authorized to accept this
friend request." };
    }

    if (friendship.status !== FriendshipStatus.PENDING) {
      return { error: "This friend request has already been
processed." };
    }

    // Update the friendship status
    const updatedFriendship = await prisma.friendship.update({
      where: { id: friendshipId },
      data: { status: FriendshipStatus.ACCEPTED },
    });

    // Create a notification for the notifier
    await prisma.notifications.create({
      data: {
        notifiedId: friendship.senderId,
        notifierId: friendship.receiverId,
        type: NotificationType.FRIEND_ACCEPTED,
      },
    });

    revalidatePath("/friends");
    revalidatePath("/notifications");
    return { success: true, friendship: updatedFriendship };
  } catch (error) {
    console.error("Error accepting friend request:", error);
    return { error: "Failed to accept friend request. Please try
again." };
  }
}

// Decline a friend request
export async function declineFriendRequest(friendshipId: string)
{
  const session = await auth();
```

```
  if (!session?.user?.id) {
    return { error: "You must be logged in to decline a friend
request." };
  }

  const userId = session.user.id;

  try {
    // Get the friendship
    const friendship = await prisma.friendship.findUnique({
      where: { id: friendshipId },
    });

    if (!friendship) {
      return { error: "Friend request not found." };
    }

    if (friendship.receiverId !== userId) {
      return {
        error: "You are not authorized to decline this friend
request.",
      };
    }

    if (friendship.status !== FriendshipStatus.PENDING) {
      return { error: "This friend request has already been
processed." };
    }

    // Update the friendship status
    const updatedFriendship = await prisma.friendship.update({
      where: { id: friendshipId },
      data: { status: FriendshipStatus.REJECTED },
    });

    // Create a notification for the notifier
    await prisma.notifications.create({
      data: {
        notifiedId: friendship.senderId,
        notifierId: friendship.receiverId,
        type: NotificationType.FRIEND_REJECTED,
      },
    });

    revalidatePath("/friends");
    revalidatePath("/notifications");
    return { success: true, friendship: updatedFriendship };
  } catch (error) {
    console.error("Error declining friend request:", error);
    return { error: "Failed to decline friend request. Please
try again." };
  }
```

```
}

// Remove a friend
export async function removeFriend(friendId: string) {
  const session = await auth();

  if (!session?.user?.id) {
    return { error: "You must be logged in to remove a friend."
};
  }

  const userId = session.user.id;

  try {
    await prisma.friendship.deleteMany({
      where: {
        OR: [
          {
            senderId: userId,
            receiverId: friendId,
            status: FriendshipStatus.ACCEPTED,
          },
          {
            senderId: friendId,
            receiverId: userId,
            status: FriendshipStatus.ACCEPTED,
          },
        ],
      },
    });

    revalidatePath("/friends");
    return { success: true };
  } catch (error) {
    console.error("Error removing friend:", error);
    return { error: "Failed to remove friend. Please try again."
};
  }
}

export async function getFriends() {
  const session = await auth();

  if (!session?.user?.id) {
    return { error: "You must be logged in to view your
friends." };
  }

  const userId = session.user.id;

  try {
    // Get friendships where the user is either sender or
receiver and status is ACCEPTED
```

```
    const friendships = await prisma.friendship.findMany({
      where: {
        OR: [
          { senderId: userId, status: FriendshipStatus.ACCEPTED
},
          { receiverId: userId, status:
FriendshipStatus.ACCEPTED },
        ],
      },
      select: {
        senderId: true,
        receiverId: true,
        sender: {
          select: {
            id: true,
            name: true,
            email: true,
            image: true,
          },
        },
        receiver: {
          select: {
            id: true,
            name: true,
            email: true,
            image: true,
          },
        },
      },
    });

    const friends = friendships.map((f) =>
      f.senderId === userId ? f.receiver : f.sender,
    );

    return { success: true, friends };
  } catch (error) {
    console.error("Error getting friends:", error);
    return { error: "Failed to get friends. Please try again."
};
  }
}

export async function searchUsers(query: string) {
  const session = await auth();

  if (!session?.user?.id) {
    return { error: "You must be logged in to search for users."
};
  }

  const userId = session.user.id;
```

```
  if (!query || query.length < 2) {
    return { users: [] };
  }

  try {
    const users = await prisma.user.findMany({
      where: {
        id: { not: userId },
        OR: [
          { name: { contains: query, mode: "insensitive" } },
          { email: { contains: query, mode: "insensitive" } },
        ],
      },
      select: {
        id: true,
        name: true,
        email: true,
        image: true,
      },
      take: 10,
    });

    // Get friendships to check status
    const friendships = await prisma.friendship.findMany({
      where: {
        OR: [{ senderId: userId }, { receiverId: userId }],
      },
    });

    const usersWithStatus = users.map((user) => {
      const sentFriendship = friendships.find(
        (f) => f.senderId === userId && f.receiverId ===
user.id,
      );
      const receivedFriendship = friendships.find(
        (f) => f.senderId === user.id && f.receiverId ===
userId,
      );

      let status = "none";
      let friendshipId = null;

      if (sentFriendship) {
        status = sentFriendship.status.toLowerCase();
        friendshipId = sentFriendship.id;
      } else if (receivedFriendship) {
        status =
`received_${receivedFriendship.status.toLowerCase()}`;
        friendshipId = receivedFriendship.id;
      }

      return {
        ...user,
```

```
        status,
        friendshipId,
      };
    });

    return { success: true, users: usersWithStatus };
  } catch (error) {
    console.error("Error searching users:", error);
    return { error: "Failed to search users. Please try again."
};
  }
}

export async function getUnreadNotificationCount(userId: string)
{
  try {
    const session = await auth();
    if (!session?.user?.id || session.user.id !== userId) {
      return { error: "You must be logged in to get
notifications." };
    }

    const count = await prisma.notifications.count({
      where: {
        notifiedId: userId,
        read: false,
      },
    });

    return { succes: true, data: count };
  } catch (error) {
    console.error("Error getting session:", error);
    return { error: "Failed to get session. Please try again."
};
  }
}

export async function markNotificationAsRead(notificationId:
string) {
  const session = await auth();
  if (!session?.user?.id) {
    throw { error: "Unauthorized" };
  }

  try {
    await prisma.notifications.update({
      where: { id: notificationId },
      data: { read: true },
    });

    revalidatePath("/notifications");
    return { succes: true };
  } catch (error) {
```

```
    console.error("Failed to mark as read:", error);
    return { error: "Failed to mark as read." };
  }
}

export async function markAllNotificationsAsRead() {
  const session = await auth();
  if (!session?.user?.id) return { error: "Unauthorized" };

  try {
    await prisma.notifications.updateMany({
      where: { notifiedId: session.user.id, read: false },
      data: { read: true },
    });
    revalidatePath("/notifications");
    return { succes: true };
  } catch (error) {
    console.error("Failed to mark all as read:", error);
    return { error: "Failed to mark all as read." };
  }
}

export async function deleteAllNotifications() {
  const session = await auth();
  if (!session?.user?.id) return { error: "Unauthorized" };

  try {
    await prisma.notifications.deleteMany({
      where: { notifiedId: session.user.id },
    });
    revalidatePath("/notifications");
    return { succes: true };
  } catch (error) {
    console.error("Failed to mark all as read:", error);
    return { error: "Failed to delete all notifications." };
  }
}
```

Б.7 React-компонент NotificationItem

```
export function NotificationItem({ nft }: { nft:
NotificationData }) {
  const router = useRouter();
  const { toast } = useToast();

  const handleClick = async () => {
    // Mark the notification as read
    const tab =
      nft.type === NotificationType.FRIEND_REQUEST
        ? FRIENDS_TABS.REQUESTS
        : FRIENDS_TABS.FRIENDS;
```

```
    if (nft.read) {
      router.push(`/friends?tab=${tab}`);
      return;
    }

    const { succes, error } = await
markNotificationAsRead(nft.id);
    if (!succes || error) {
      toast({ title: "Error", description: error, variant:
"destructive" });
      return;
    }

    router.push(`/friends?tab=${tab}`);
  };

  return (
    <Card
      className={cn(
        "transition cursor-pointer hover:bg-accent",
        nft.read ? "opacity-70" : "border-primary border",
      )}
      onClick={handleClick}
    >
      <CardContent className="p-4">
        <div className="flex items-center gap-4">
          <UserAvatar name={nft.notifier.name}
image={nft.notifier.image} />
          <p>{getNotificationMessage(nft)}</p>
        </div>
        <p className="text-xs text-muted-foreground mt-1">
          {new Date(nft.createdAt).toLocaleString("uk-UA")}
        </p>
      </CardContent>
    </Card>
  );
}
```

Б.8 React-компонент NavUser для відображення даних користувача

```
interface NavUserProps {
  user?: User;
  ntfBadge: string | null;
}

export function NavUser({ ntfBadge }: NavUserProps) {
  const { isMobile } = useSidebar();
  const [open, setOpen] = useState(false);
  const [user, setUser] = useState<User | null>(null);
  const { data: session } = useSession();
```

```
  useEffect(() => {
    if (session?.user) {
      setUser(session.user);
    }
  }, [session]);

  return (
    <>
      <SidebarMenu>
        <SidebarMenuItem>
          <DropdownMenu>
            <DropdownMenuTrigger asChild>
              <SidebarMenuButton
                size="lg"
                className="data-[state=open]:bg-sidebar-accent
data-[state=open]:text-sidebar-accent-foreground"
              >
                <UserAvatarItem user={user} />
              </SidebarMenuButton>
            </DropdownMenuTrigger>
            <DropdownMenuContent
              className="w-[--radix-dropdown-menu-trigger-width]
min-w-56 rounded-lg"
              side={isMobile ? "bottom" : "right"}
              align="end"
              sideOffset={4}
            >
              <DropdownMenuLabel className="p-0 font-normal">
                <div className="flex items-center gap-2 px-1 py-
1.5 text-left text-sm">
                  <UserAvatarItem user={user}
showChevron={false} />
                </div>
              </DropdownMenuLabel>

              <DropdownMenuSeparator />
              <DropdownMenuGroup>
                <DropdownMenuItem onClick={() => setOpen((prev)
=> !prev)}>
                  <UserIcon />
                  Account
                </DropdownMenuItem>
                <Link
                  href="/notifications"
                  className="flex items-center gap-2 w-full"
                >
                  <DropdownMenuItem className="w-full">
                    <div className="relative">
                      <Bell size={16} />
                      {ntfBadge && ntfBadge != "0" && (
                        <span className="absolute -top-1 -right-
1 flex h-3 min-w-2 items-center justify-center rounded-full bg-
red-500 p-1 text-xs font-medium">
```

```
                          {+ntfBadge < 100 ? ntfBadge : "99+"}
                        </span>
                      )}
                    </div>

                    <span>Notifications</span>
                  </DropdownMenuItem>
                </Link>
              </DropdownMenuGroup>
              <DropdownMenuSeparator />
              <DropdownMenuItem onClick={signOutAction}>
                <LogOut />
                Log out
              </DropdownMenuItem>
            </DropdownMenuContent>
          </DropdownMenu>
        </SidebarMenuItem>
      </SidebarMenu>
      <ProfileDialog
        user={user}
        open={open}
        onClose={() => setOpen((prev) => !prev)}
      />
    </>
  );
}
```

Б.9 React-компонент ProfileDialog

```
export function ProfileDialog({
  onClose,
  open,
  user,
}: {
  onClose: () => void;
  open: boolean;
  user: UserType | null;
}) {
  const { update } = useSession();
  const [isSubmitting, setIsSubmitting] = useState(false);
  const [previewImage, setPreviewImage] = useState<string |
null>(null);
  const [compressedFile, setCompressedFile] = useState<File |
null>(null);
  const [isCompressing, setIsCompressing] = useState(false);
  const { toast } = useToast();

  const form = useForm<ProfileFormSchemaType>({
    resolver: zodResolver(profileFormSchema),
    defaultValues: {
      name: user?.name!,
```

```
      email: user?.email!,
    },
  });

  useEffect(() => {
    if (user) {
      form.reset({
        name: user.name ?? "",
        email: user.email ?? "",
      });
    }
  }, [user, form]);

  const handleImageChange = async (e:
React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];

    if (!file) return;

    setIsCompressing(true);
    try {
      const compressed = await imageCompression(file, {
        maxSizeMB: 1,
        maxWidthOrHeight: 512,
        useWebWorker: true,
      });
      setCompressedFile(compressed);

      const reader = new FileReader();
      reader.onloadend = () => {
        setPreviewImage(reader.result as string);
      };
      reader.readAsDataURL(compressed);
    } catch (err) {
      console.error("Compression failed", err);
      toast({
        title: "Image compression failed",
        description: "Please try a different image.",
        variant: "destructive",
      });
    } finally {
      setIsCompressing(false);
    }
  };

  const onSubmit = async (data: ProfileFormSchemaType) => {
    setIsSubmitting(true);

    try {
      const imgPath = await fileUploadService.uploadFile(
        compressedFile!,
        user?.id!,
        BUCKETS.AVATARS,
```

```
      );

      const result = await updateProfile({ ...data, avatar:
imgPath });

      if (result.error) {
        toast({
          title: "Error",
          description: result.error,
          variant: "destructive",
        });
      } else if (result.success) {
        // Update the session to reflect the changes
        update({ ...data, image: imgPath });

        toast({
          title: "Profile updated",
          description: "Your profile has been updated
successfully.",
        });
        onClose();
      }
    } catch (error) {
      console.error("Error updating profile:", error);
      toast({
        title: "Error",
        description: "An unexpected error occurred. Please try
again.",
        variant: "destructive",
      });
    } finally {
      setIsSubmitting(false);
    }
  };

  return (
    <Dialog open={open} onOpenChange={onClose}>
      <DialogContent className="sm:max-w-[425px]">
        <DialogHeader>
          <DialogTitle>Edit profile</DialogTitle>
          <DialogDescription>
            Update your profile information. Click save when
you&apos;re done.
          </DialogDescription>
        </DialogHeader>
        <Form {...form}>
          <form onSubmit={form.handleSubmit(onSubmit)}
className="space-y-6">
            <div className="flex flex-col items-center justify-
center mb-6">
              <div className="relative h-24 w-24 rounded-full
border border-border">
                {previewImage ? (
```

```
                  <Image
                    src={previewImage || ""}
                    alt="Profile preview"
                    fill
                    className="object-cover rounded-full"
                  />
                ) : user?.image ? (
                  <Image
                    src={user.image || ""}
                    alt="Profile picture"
                    fill
                    className="object-cover rounded-full"
                  />
                ) : (
                  <div className="flex h-full w-full items-
center justify-center rounded-full bg-muted">
                    <User className="h-12 w-12 text-muted-
foreground" />
                  </div>
                )}
                {isCompressing && (
                  <div className="absolute inset-0 flex items-
center justify-center bg-black/50 rounded-full z-10">
                    <Loader2 className="h-6 w-6 animate-spin
text-white" />
                  </div>
                )}
                <label
                  htmlFor="image-upload"
                  className="absolute bottom-0 right-0 p-1 bg-
primary rounded-full cursor-pointer"
                >
                  <Camera className="h-5 w-5 text-primary-
foreground" />
                </label>
                <input
                  id="image-upload"
                  type="file"

accept="image/jpeg,image/png,image/webp,image/svg+xml"
                  onChange={handleImageChange}
                  className="sr-only"
                />
              </div>
              <p className="text-sm text-muted-foreground mt-2">
                Click the camera icon to change your photo
              </p>
            </div>

            <FormField
              control={form.control}
              name="name"
              render={({ field }) => (
```

```
                <FormItem>
                  <FormLabel>Name</FormLabel>
                  <FormControl>
                    <Input {...field} />
                  </FormControl>
                  <FormMessage />
                </FormItem>
              )}
            />

            <FormField
              control={form.control}
              name="email"
              render={({ field }) => (
                <FormItem>
                  <FormLabel>Email</FormLabel>
                  <FormControl>
                    <Input {...field} type="email" />
                  </FormControl>
                  <FormDescription>
                    This is the email associated with your
account.
                  </FormDescription>
                  <FormMessage />
                </FormItem>
              )}
            />

            <DialogFooter>
              <Button type="submit" disabled={isSubmitting ||
isCompressing}>
                {isSubmitting || isCompressing ? (
                  <>
                    <Loader2 className="mr-2 h-4 w-4 animate-
spin" />
                    Loading...
                  </>
                ) : (
                  "Save changes"
                )}
              </Button>
            </DialogFooter>
          </form>
        </Form>
      </DialogContent>
    </Dialog>
  );
}
```