

## 4 РЕАЛИЗАЦИЯ

### 4.1 Выбор инструментальных средств, системного и дополнительного программного обеспечения

Разработка программного продукта осуществляется при помощи следующих инструментов:

1. NodeJS;
2. Фреймворк Prisma для создания структуры базы данных;
3. Фреймворк NextJS;
4. TypeScript;
5. PyCharm – IDE, в котором велась разработка web-приложения;
6. Git – для контроля версий написанного кода;
7. База данных MS SQL.

Рассмотрим подробно особенности используемых технологий:

1. Node.js: это среда выполнения JavaScript, основанная на движке Chrome V8. Она позволяет разрабатывать серверные приложения на JavaScript, а также обеспечивает доступ к множеству пакетов и модулей через свой менеджер пакетов npm.

2. Prisma: это инструмент для работы с базами данных, который позволяет создавать и мигрировать структуру базы данных с помощью декларативного языка моделирования. Prisma поддерживает различные базы данных, такие как PostgreSQL, MySQL и SQLite.

3. Next.js: это фреймворк для разработки React приложений. Он предоставляет множество функций, таких как серверный рендеринг, статическая генерация, предварительная загрузка и динамическая маршрутизация. Next.js позволяет создавать быстрые и масштабируемые

					УО «ВГТУ» ДП.009 1-40 05 01-01 РПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Казунка А.И.			Реализация	Лит.	Лист
Провер.		Соколова А.С.					
Реценз.						УО «ВГТУ» каф. ИСиТ гр.Итс-10	
Н. Контр.		Соколова А.С.					
Утверд.		Казаков В.Е.					

приложения.

4. TypeScript: это язык программирования, который является надмножеством JavaScript. Он добавляет статическую типизацию, что помогает выявлять ошибки на этапе разработки и облегчает сопровождение кода. TypeScript компилируется в JavaScript, поэтому его можно использовать вместе с Node.js и React.js.

5. PyCharm: это интегрированная среда разработки (IDE) для языка программирования Python. Она предоставляет удобные инструменты для разработки и отладки Python-приложений, а также интеграцию с системами управления версиями, такими как Git.

6. Git: это распределенная система контроля версий, которая позволяет отслеживать изменения в коде и сотрудничать с другими разработчиками. Git обеспечивает сохранность истории изменений, позволяет создавать ветки для параллельной разработки и упрощает слияние изменений.

7. MS SQL (Microsoft SQL Server) – это реляционная база данных, разработанная компанией Microsoft. Она предоставляет надежное и масштабируемое хранилище данных для различных приложений и систем. MS SQL поддерживает язык запросов SQL для работы с данными и предоставляет широкий набор функций и возможностей для управления и администрирования баз данных.

## 4.2 Описание реализации вариантов использования

Одним из способов реализации описание варианта использования программного продукта является построение диаграмм деятельности. С одной стороны, диаграмма деятельности – это полноценная диаграмма UML, с другой стороны, диаграмма деятельности немногим отличается от блок-схемы (а тем самым и от псевдокода). Таким образом, реализация варианта использования диаграммой деятельности является компромиссным способом ведения разработки – в сущности, это проектирование сверху вниз в терминах

					УО «ВГТУ» ДП.009 1-40 05 01-01 РПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		

и обозначениях UML.

Построим диаграммы деятельности по нескольким алгоритмам системы.

Алгоритмы авторизации, регистрации, управления организации не рассматриваются, так как данными алгоритмами управляет сторонний сервис – «Clerk».

При работе с приложением пользователь может создать доску (листинг 4.1). После создания доски, пользователь может создать лист для задач (листинг 4.2). Так же пользователь может создать задачу, внутри листа (листинг 4.3).

Для удаления задачи, пользователю необходимо открыть задачу, после чего нажать кнопку «Удалить» (листинг 4.4).

Диаграммы алгоритма создания доски, создания листа, создания задачи представлены на рисунках 4.1 – 4.3.

Диаграмма алгоритма удаления задачи представлена на рисунке 4.4.

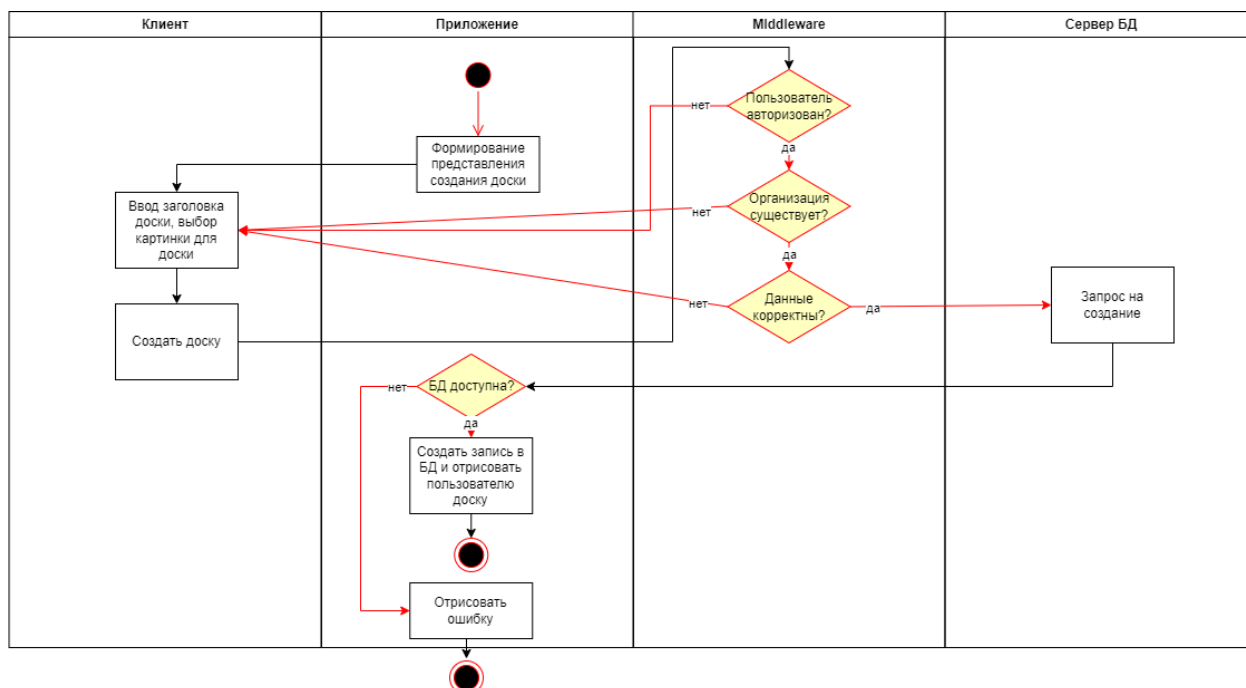


Рисунок 4.1 – Диаграмма активности выполнения кейса создания доски

#### Листинг 4.1 – Выполнение кейса создания доски

```
const handler = async (data: InputType): Promise<ReturnType> => {
  const { userId, orgId } = auth();

  if(!userId || !orgId){
    return {
      error: "Пользователь неавторизован.",
    }
  }
  const { title, image } = data;

  const [
    imageId,
    imageThumbUrl,
    imageLinkHTML,
    imageFullUrl,

imageUserName
  ] = image.split("|")

  if (!imageId || !imageThumbUrl || !imageFullUrl || !imageUserName || !imageLinkHTML)
  {
    return {
      error: "Переданы не все поля! Невозможно создать доску."
    };
  }

  let board;
  try{
    board = await db.board.create({
      data: {
        title,
        orgId,
        imageId,
        imageThumbUrl,
        imageFullUrl,
        imageUserName,
        imageLinkHTML
      }
    })
  })
}
```

					УО «ВГТУ» ДП.009 1-40 05 01-01 РПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		

```

    } catch (e) {
        return {
            error: "Невозможно создать доску."
        }
    }
}

await createAuditLog({
    entityType: board.title,
    entityId: board.id,
    entityType: ENTITY_TYPE.BOARD,
    action: ACTION.CREATE,
})

revalidatePath(`/board/${board.id}`)

return {data: board}
}

```

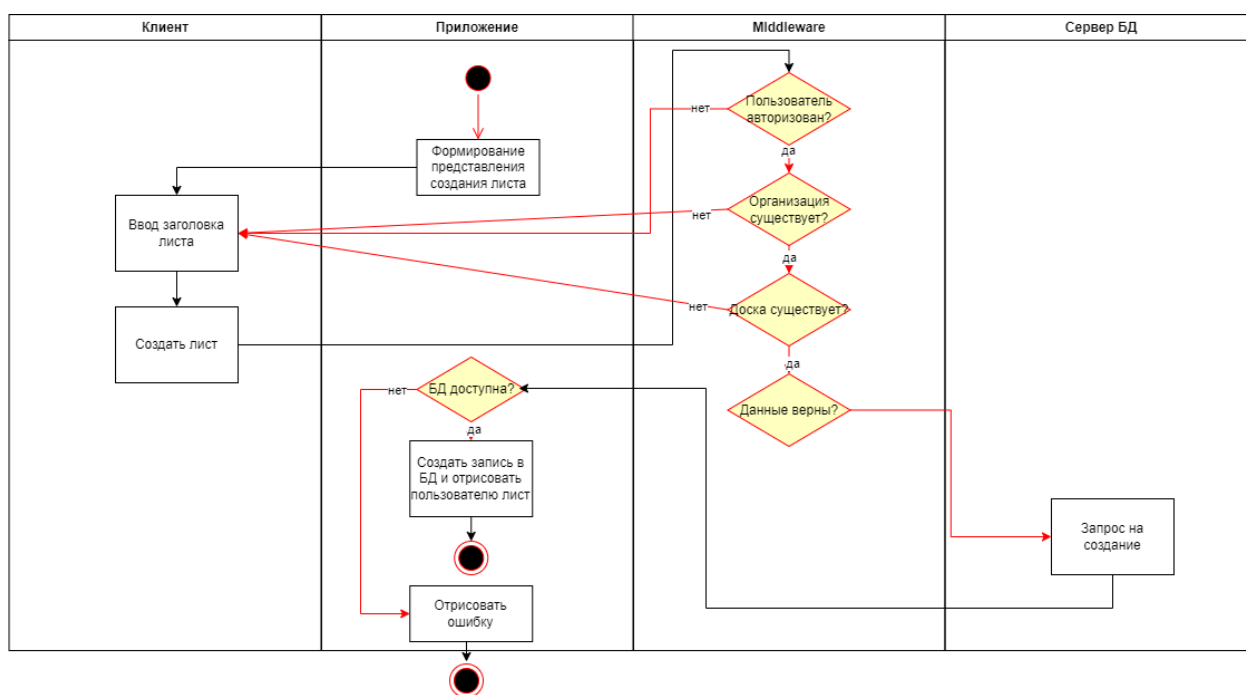


Рисунок 4.2 – Диаграмма активности выполнения кейса создания листа

#### Листинг 4.2 – Выполнение кейса создания листа

```

const handler = async (data: InputType): Promise<ReturnType> => {
    const { userId, orgId } = auth();
    if (!userId || !orgId) {
        return {
            error: "Пользователь не авторизован.",
        };
    }
};

```

```

}
const { title, boardId } = data;
let list;
try {
  const board = await db.board.findUnique({
    where: {
      id: boardId,
      orgId,
    }
  })
  if (!board) {
    return{
      error: "Доска не найдена.",
    }
  }

  const lastList = await db.list.findFirst({
    where: { boardId: boardId},
    orderBy: { order: "desc"},
    select: {order: true},
  })

  const newOrder = lastList ? lastList.order + 1 : 1;
  list = await db.list.create({
    data: {
      title,
      boardId,
      order: newOrder
    },
  });
  await createAuditLog({
    entityType: list.title,
    entityId: list.id,
    entityType: ENTITY_TYPE.LIST,
    action: ACTION.CREATE,
  })
} catch (error) {
  return {
    error: "Невозможно создать лист задач."
  }
}

```

					УО «ВГТУ» ДП.009 1-40 05 01-01 РПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		

```

revalidatePath(`/board/${boardId}`);
return { data: list };
};

```

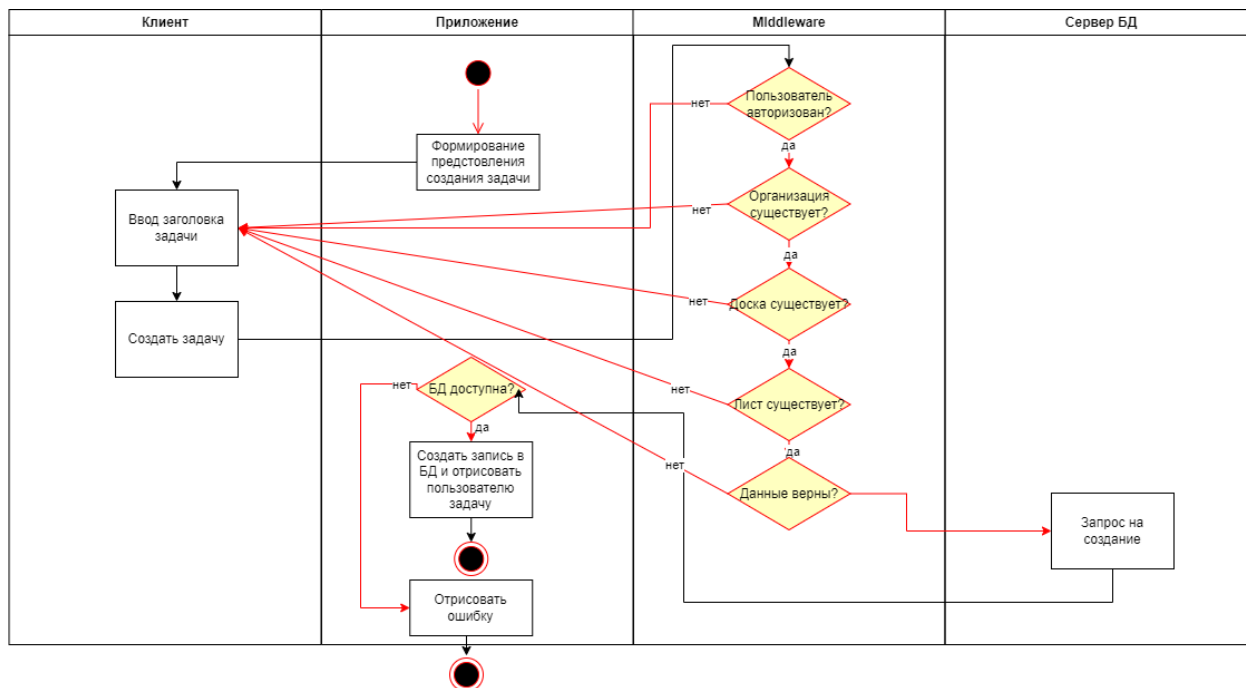


Рисунок 4.3 – Диаграмма активности выполнения кейса создания задачи

#### Листинг 4.3 – Выполнение кейса создания задачи

```

const handler = async (data: InputType): Promise<ReturnType> => {
  const { userId, orgId } = auth();

  if (!userId || !orgId) {
    return {
      error: "Пользователь не авторизован.",
    };
  }

  const { title, boardId, listId } = data;
  let card;

  try {
    const list = await db.list.findUnique({
      where: {
        id: listId,
        board: {
          orgId

```

```

    }
  }
})
if (!list) {
  return{
    error: "Список не найдена.",
  }
}
const lastCard = await db.card.findFirst({
  where:{listId},
  orderBy: {order: "desc"},
  select: {order: true}
})
const newOrder = lastCard ? lastCard.order + 1 : 1;
card = await db.card.create({
  data:{
    title,
    listId,
    order: newOrder
  }
})
await createAuditLog({
  entityId: card.id,
  entityType: card.title,
  entityType: ENTITY_TYPE.CARD,
  action: ACTION.CREATE
})

} catch (error) {
  return {
    error: "Невозможно создать задачу."
  }
}
revalidatePath(`/board/${boardId}`);
return { data: card };};

```

					УО «ВГТУ» ДП.009 1-40 05 01-01 РПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		



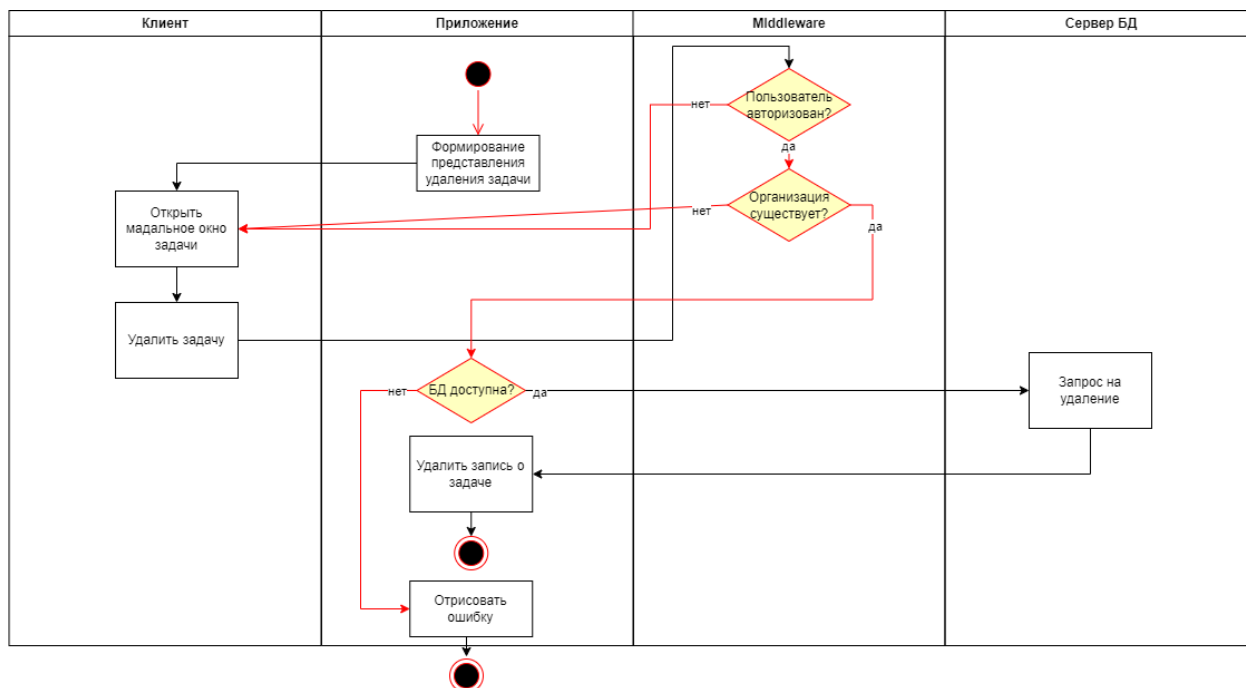


Рисунок 4.4 - Диаграмма активности выполнения кейса удаления задачи

#### Листинг 4.4 – Удаление задач

```

const handler = async (data: InputType): Promise<ReturnType> => {
  const { userId, orgId } = auth();

  if (!userId || !orgId) {
    return {
      error: "Пользователь не авторизован.",
    };
  }

  const { id } = data;
  let board;

  try {
    board = await db.board.delete({
      where: {
        id,
        orgId,
      },
    });

    await createAuditLog({
      entityType: board.title,
    });
  }
}
  
```

```

        entityId: board.id,
        entityType: ENTITY_TYPE.BOARD,
        action: ACTION.DELETE,
    })
} catch (error) {
    return {
        error: "Невозможно удалить данные."
    }
}

revalidatePath(`/organization/${orgId}`);
redirect(`/organization/${orgId}`)
};

```

### 4.3 Функциональное тестирование

Тестирование программного обеспечения – это метод проверки соответствия фактического программного продукта ожидаемым требованиям, который также необходим, чтобы убедиться, что продукт не содержит дефектов. Подразумевает выполнение предварительно определенных алгоритмов с использованием ручных или автоматизированных инструментов для оценки одного или нескольких интересующих свойств.

В качестве метода тестирования был использован метод «черного ящика», при котором не используется знание о внутреннем устройстве тестируемого объекта. Тестирование будет осуществляться только через пользовательский интерфейс в котором будет происходить сопоставление ожидаемых результатов с получаемыми.

Для тестирования разработанной информационной системы был разработан набор тест-кейсов, представленный в приложении А.

Тестирование было проведено на компьютере с установленной операционной системой Fedora Linux, средой разработки PyCharm 2023, используемый браузер Firefox. Результаты некоторых тест-кейсов показаны на рисунках 4.5 – 4.13.

					УО «ВГТУ» ДП.009 1-40 05 01-01 РПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		

Secured by clerk

### Create Organization

to continue to KMZ

Profile image  
[Upload image](#)

Organization name

Slug URL ?

CREATE ORGANIZATION

Рисунок 4.5 – Успешная авторизация пользователя

Secured by clerk

### Sign in

to continue to KMZ

Continue with Google

or

Email address

akazunka.rc@gmail.com

! Couldn't find your account.

CONTINUE

No account? [Sign up](#)

Рисунок 4.6 – Ввод некорректных данных пользователя

					УО «ВГТУ» ДП.009 1-40 05 01-01 РПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		

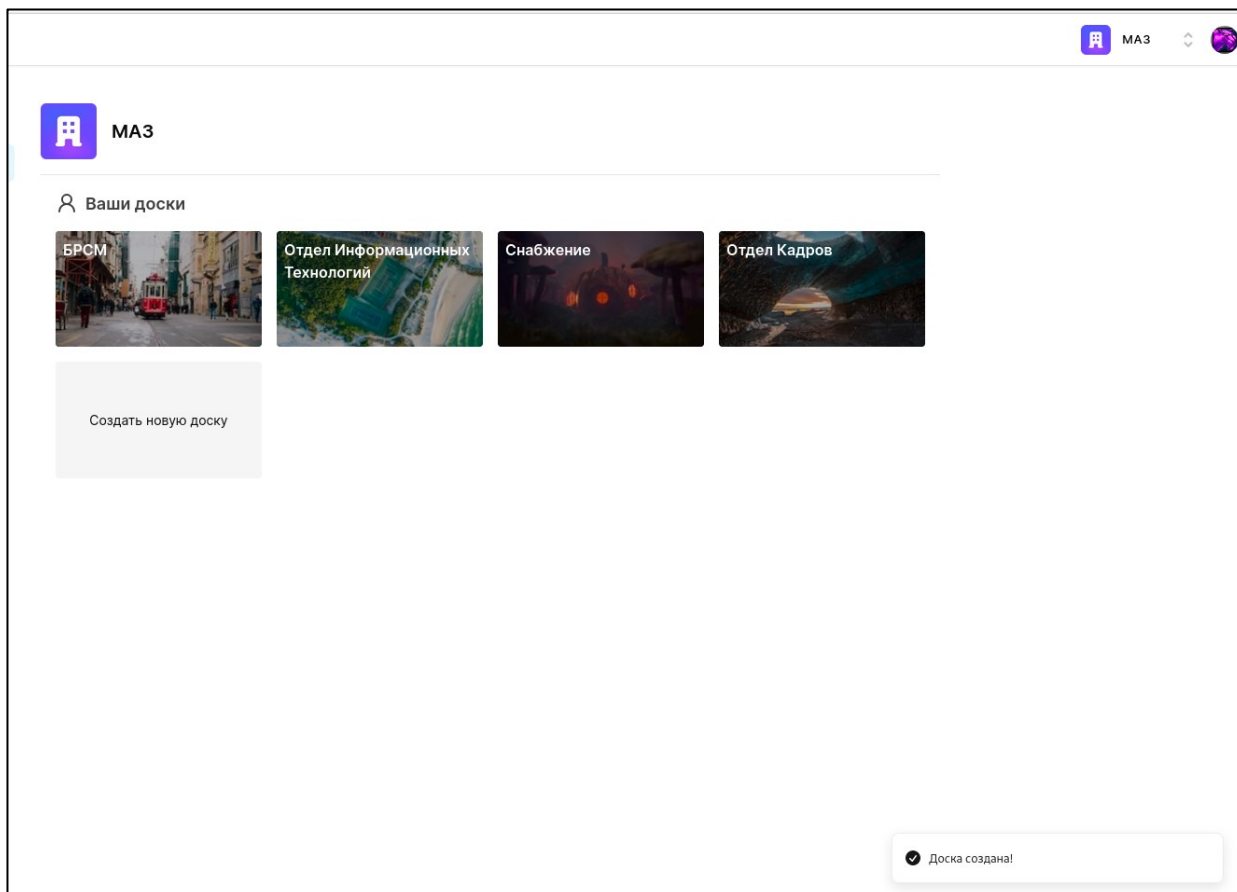


Рисунок 4.7 – Создание доски



Рисунок 4.8 – Ошибка заполнения обязательных полей

					УО «ВГТУ» ДП.009 1-40 05 01-01 РПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		

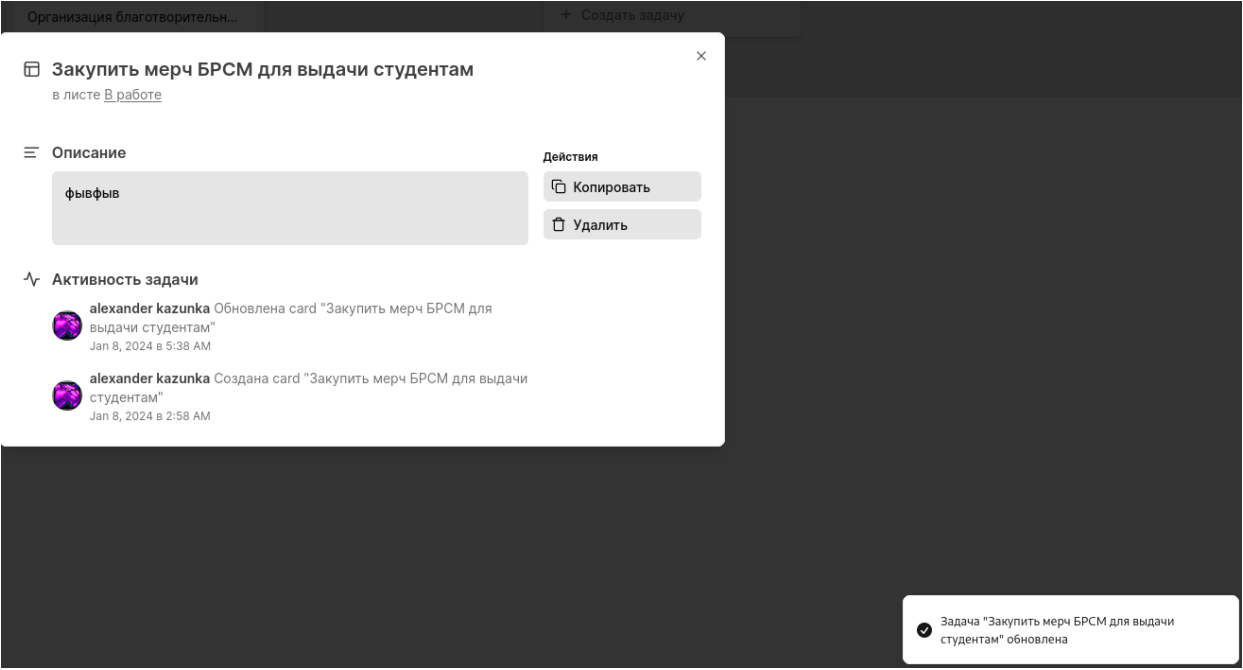


Рисунок 4.9 – Обновление задачи

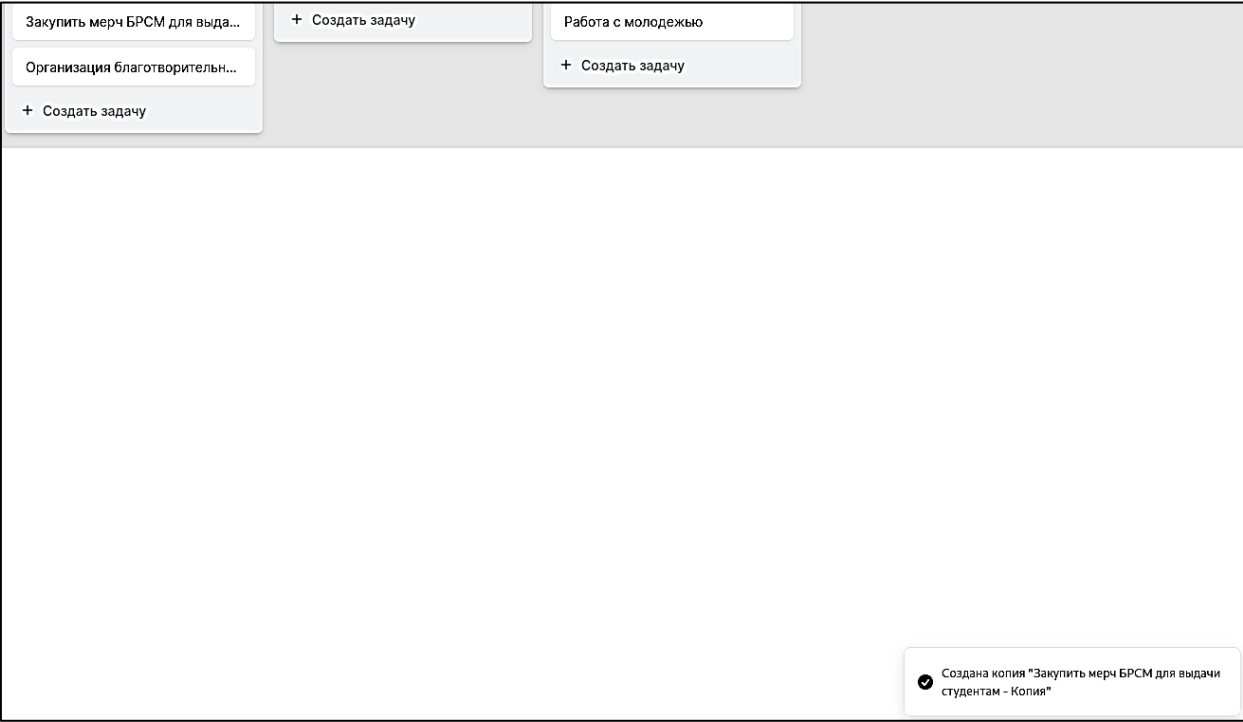





Рисунок 4.10 – Копирование задачи

 MA3

 Members

 Settings

Members / Invite members

×

# Invite members

Invite new members to this organization

Email addresses

Enter or paste one or more email addresses, separated by spaces or commas

akazunka.rc@gmail.com ✕


Role


Member ▾


CANCEL


SEND INVITATIONS


Рисунок 4.11 – Отправка приглашения пользователю

 MA3



 akazunka.rc@gmail.com

 Manage account

 Sign out

Secured by clerk

Ваши доски

test

БРСМ

Отдел Информационных Технологий

Снабжение

Отдел Кадров

Создать новую доску

Рисунок 4.12 – Проверка доступа пользователя

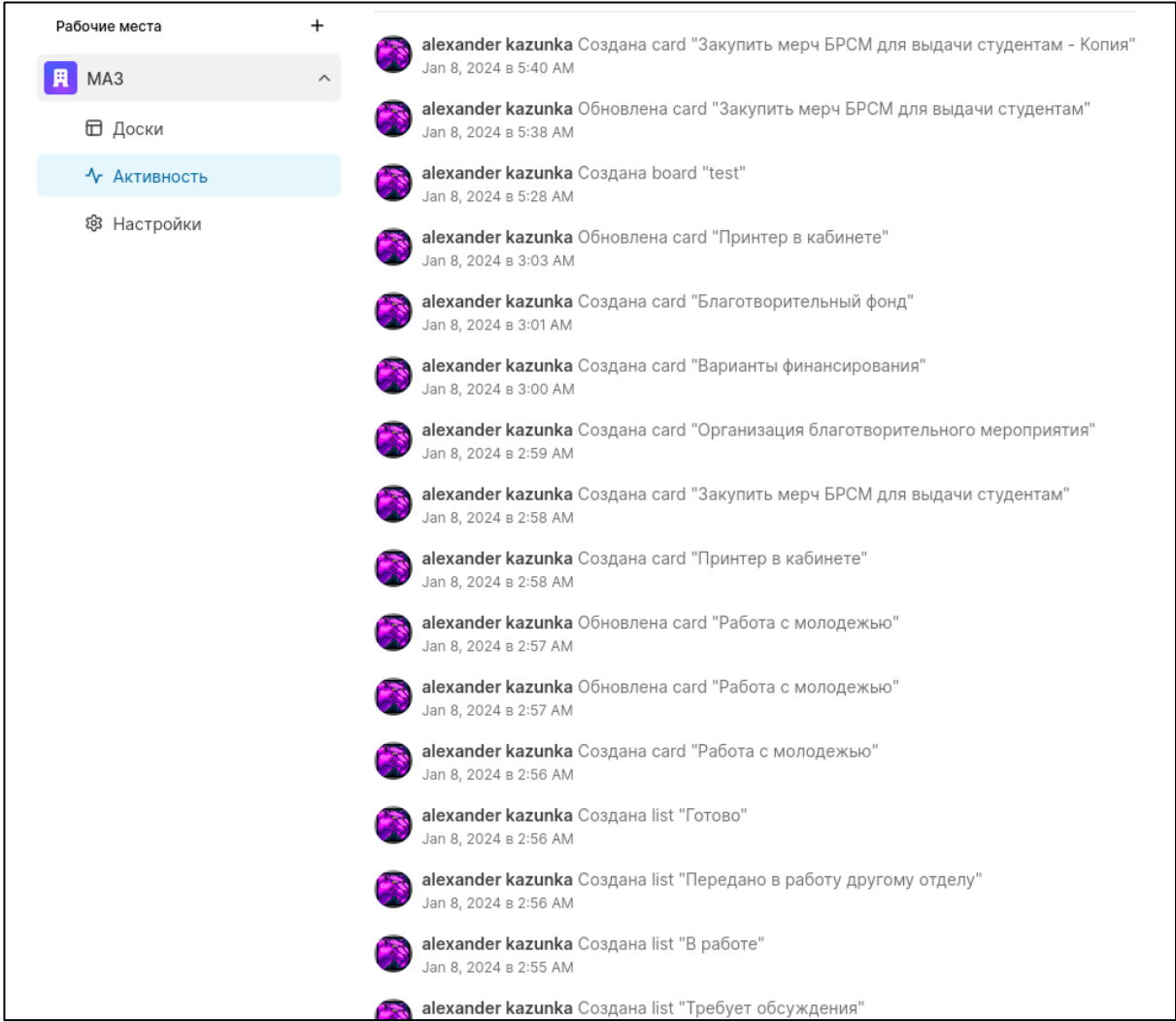


Рисунок 4.13 – Активность по доске

4.4 Прочие виды тестирования

Тестирование производительности в инженерии программного обеспечения – тестирование, которое проводится с целью определения, как быстро работает вычислительная система или её часть под определённой нагрузкой. Также может служить для проверки и подтверждения других атрибутов качества системы, таких как масштабируемость, надёжность и потребление ресурсов.

Тестирование программного обеспечения было проведено посредством инструмента «Системный монитор» операционной системы Fedora Linux. Показатели нагруженности можно увидеть на рисунке 4.14.

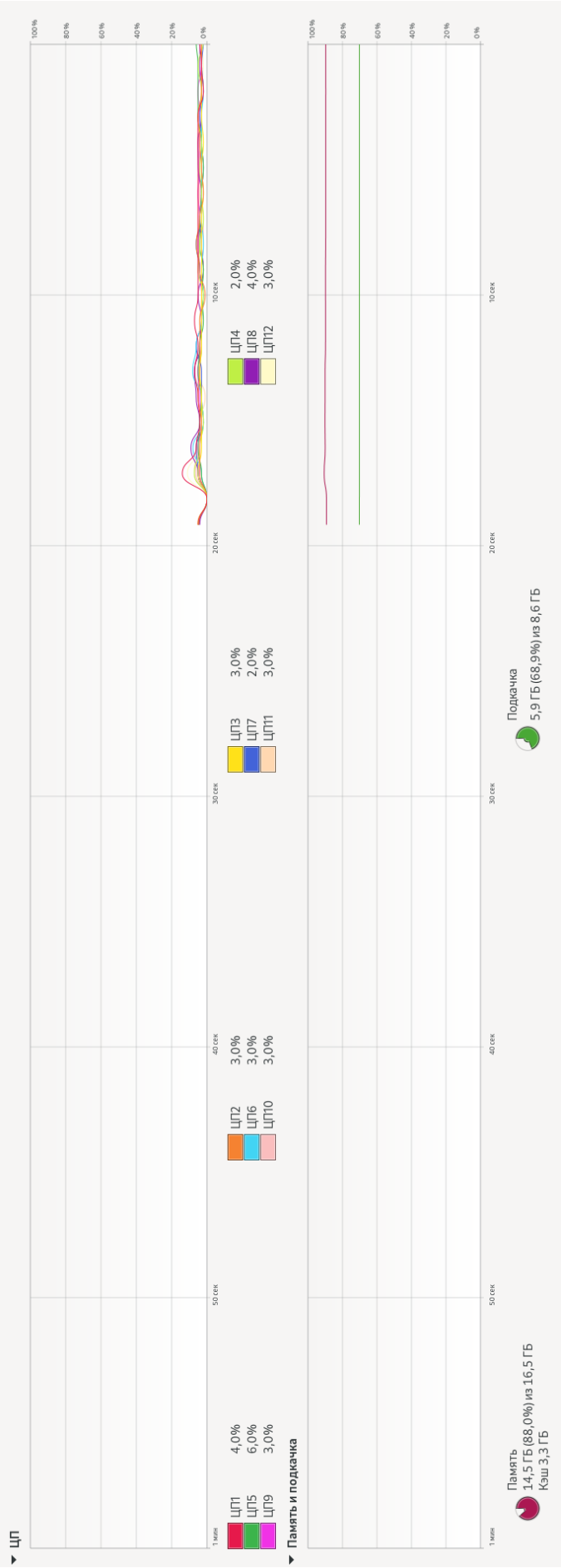


Рисунок 4.14 –Диспетчер задач