

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Ижевский государственный технический университет имени М.Т. Калашникова»
(ФГБОУ ВО «ИжГТУ имени М.Т. Калашникова»)

Институт «Информатика и вычислительная техника»
Кафедра «Вычислительная техника»
Направление подготовки 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки «Вычислительные машины, комплексы, системы и сети»

ОТЧЕТ ПО ПРАКТИКЕ

Производственная практика (технологическая (проектно-технологическая))

наименование типа и вида практики

ООО «БИМИТ»

полное наименование места прохождения практики и структурного подразделения

Выполнил обучающийся _____ /Дерендяев Андрей Николаевич, 3 курс, Б19-781-1/
подпись *ФИО, курс, номер группы*

Дата сдачи отчета: « ____ » _____ 2022 г.

Дата аттестации: « ____ » _____ 2022 г.

Оценка _____

Руководитель практики от
ИжГТУ имени М.Т. Калашникова _____ /И.Г. Корнилов, доцент каф. ВТ, к.т.н.
подпись *И.О. Фамилия, должность, ученая степень*

Заведующий кафедрой _____ /К.Ю. Петухов, доцент каф. ВТ, к.т.н.
подпись *И.О. Фамилия, должность, ученая степень*

Содержание

ВВЕДЕНИЕ	3
1 АНАЛИТИЧЕСКИЙ ОБЗОР ТЕКУЩЕЙ СИСТЕМЫ.....	4
2 ПОСТАНОВКА ЗАДАЧИ	7
2.1 Определение требований к модулю	7
2.2 Аналитический обзор инструментов	7
3 РАЗРАБОТКА МОДУЛЯ.....	10
ЗАКЛЮЧЕНИЕ	15
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	16
ПРИЛОЖЕНИЕ 1: КОД ИНТЕРФЕЙСА.....	17

ВВЕДЕНИЕ

При планировании и строительстве объектов, инженеры-строители составляют различные акты, планы, сметы и прочую документацию.

В процессе проектирования и стройки условия могут измениться и тогда возникает необходимость вносить правки во все зависимые документы и отправлять их на дальнейшее согласование, что очень затягивает весь процесс в целом и увеличивает его стоимость.

Но также не исключено, что в время планировки и изменений может произойти ошибка и она будет проигнорирована и даст о себе знать только уже на этапе строительства, что может привести к увеличению затрат как для исполнителя, так и для заказчика. До недавнего времени была популярна технология 3D – моделирования, благодаря ей, на этапе планирования можно было отследить различные ошибки и исправлять их. Благодаря этому подходу, снизился риск получить критическую ошибку на этапе строительства.

Сейчас для автоматизации и ускорения различных процессов была введена концепция BIM (Building Information Model). Информационное моделирование здания — это подход к возведению, оснащению, эксплуатации и ремонту (а также сносу) здания (к управлению жизненным циклом объекта), который предполагает сбор и комплексную обработку в процессе проектирования всей архитектурно-конструкторской, технологической, экономической и иной информации о здании со всеми её взаимосвязями и зависимостями, когда здание и всё, что имеет к нему отношение, рассматриваются как единый объект, изменение какого-либо из его параметров влечёт за собой автоматическое изменение связанных с ним параметров и объектов, вплоть до чертежей, визуализации, спецификаций и календарного графика .

Целью данной работы является внести дополнительный функционал для одной из систем BIM – проектирования, который будет производить привязку «типа» к задаче того или иного этапа строительства.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить систему и весь функционал, который реализован в ней;
- выбрать необходимые методы реализации задачи и выбрать подходящий;
- разработать и внедрить прототип;
- провести отладку и настройку;
- внедрить готовый модуль в основную систему;

1 АНАЛИТИЧЕСКИЙ ОБЗОР ТЕКУЩЕЙ СИСТЕМЫ

В качестве исследуемой системы выступает отечественная разработка - система BIMIT. Система BIMIT представляет собой среду общих данных для всех участников процесса создания объектов — заказчики, проектировщики, строители и прочие. Процесс заключается в следующем: проектировщики создают BIM - модель: в формате IFC (Industry Foundation Classes) будущего здания вносят изменения, согласовывают с заказчиком. Затем в процессе строй контроля вносятся фактические данные со стройплощадки, в результате чего появляется исполнительная модель — точная копия физически построенного здания. После этого модель с актуальными данными и комплексной обратной связью от систем учёта и контроля передаётся в эксплуатацию управляющей компании. На рисунке 1.1 изображен пример рабочего окна системы с уже загруженной IFC моделью ФАПа (фельдшерско-акушерского пункта).

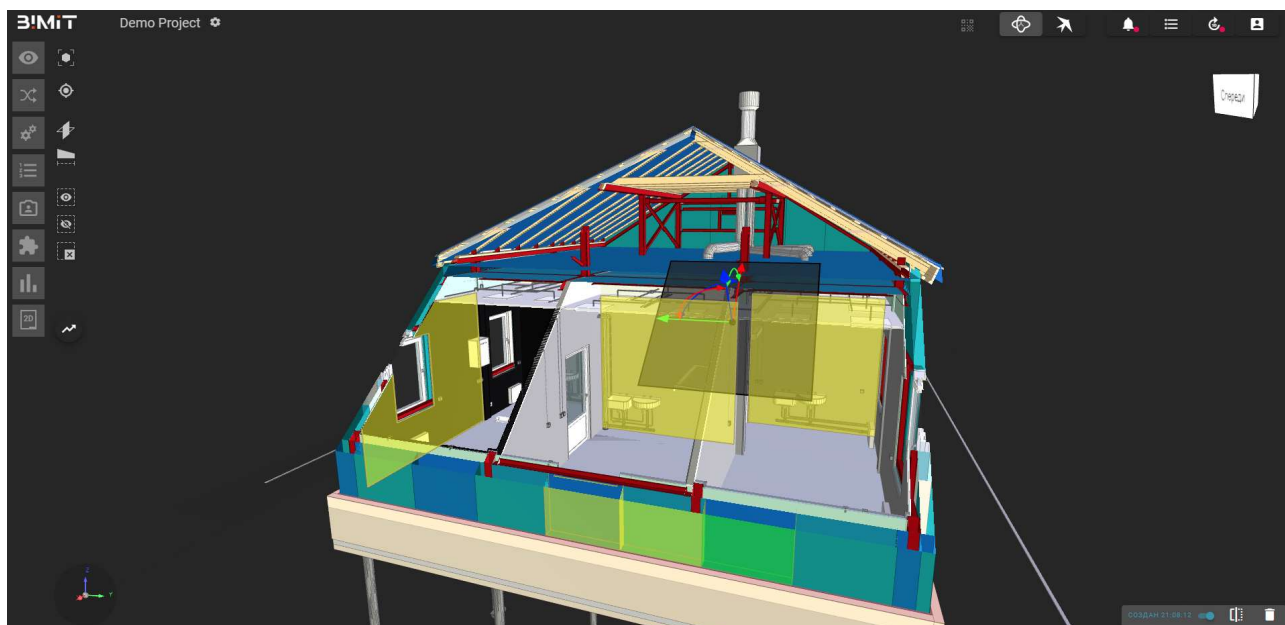


Рисунок 1.1 - Пример рабочей модели в системе BIMiT

Кроме того, система имеет различный функционал для работы с моделью, да и с каждой деталью в целом, благодаря чему можно производить необходимые расчеты в сметах или находить ошибки при проектировании. В качестве задания от компании, необходимо было изучить систему и в особенности модуль с формами. Используя модуль с формами, необходимо разработать функционал, привязывающий и заполняющий эти формы к какой-либо задаче. В системе BIMiT имеется модуль с задачами – рис.1.2.

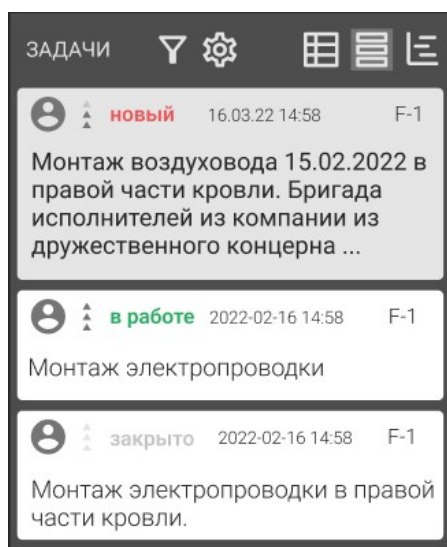


Рисунок 1.2 – Список задач в системе BIMiT

Задачи представляют собой описание некоторых действий, которые должен произвести исполнитель на этапе строительства. Сама задача включает в себя описание задачи, объем выполненной задачи, сроки выполнения задачи и её приоритет. Кроме того, в задаче имеются такие вкладки, как «Комментарии», «Вложения», «Модели», «Назначенные», «Подзадачи» и «Смета».

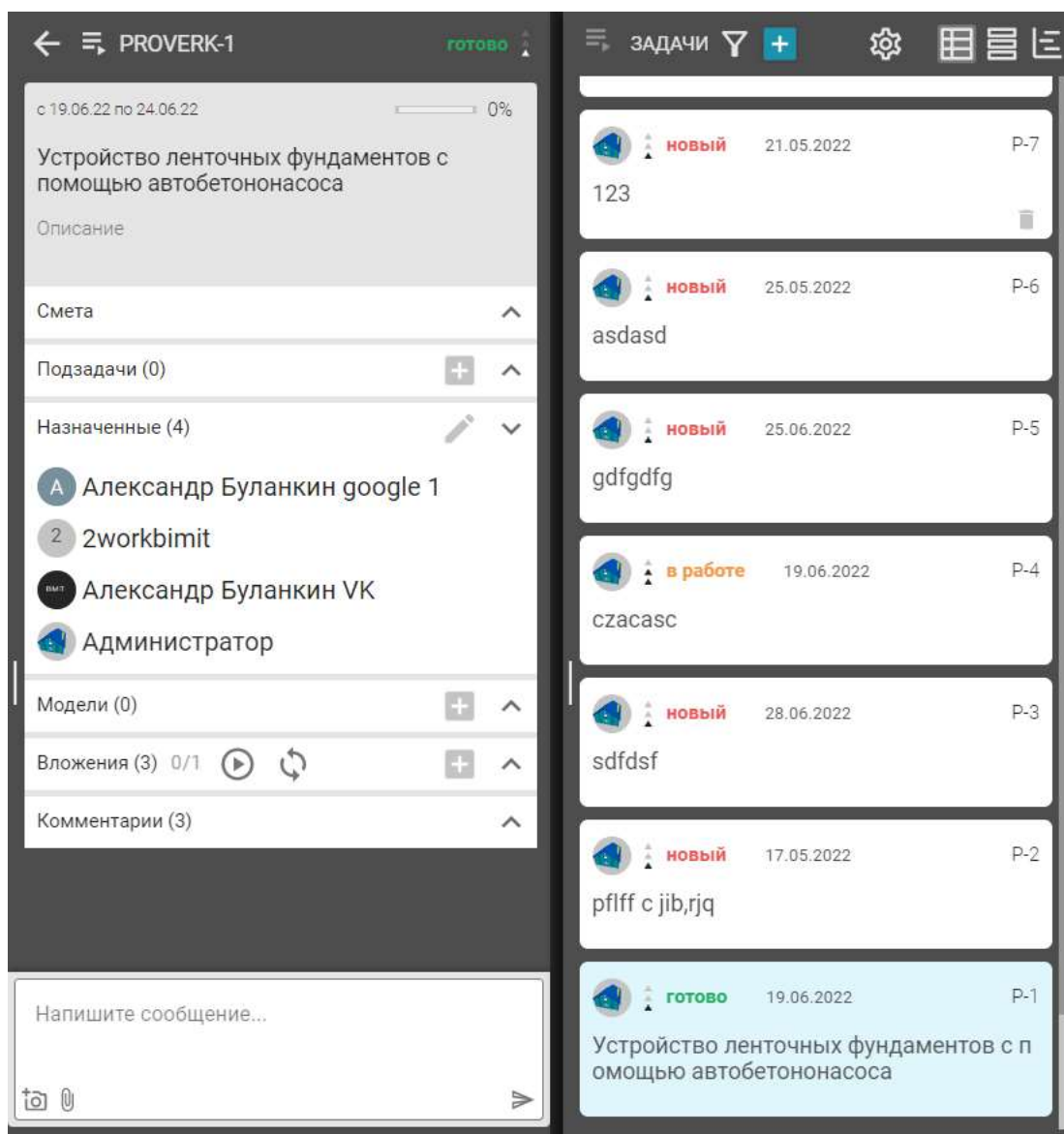


Рисунок 1.3 – Окно с описанием и параметрами задачи

2 ПОСТАНОВКА ЗАДАЧИ

В данной работе необходимо разработать интерфейса для привязки формы к задаче, доработать класс Task – добавить поле TaskType и разработать необходимые методы для реализации хранения состояния заполненной формы в хранилище, реализовать методы API на сервере, реализовать сетевые запросы.

2.1 Определение требований к модулю

Выделенные основные функциональные и нефункциональные требования к разрабатываемому модулю.

Функциональные требования:

- модуль внедряется в систему с использованием языка JavaScript и фреймворка Vue.js 2.6;
- модуль должен производить привязку формы к задаче;
- модуль должен позволять редактировать привязанную форму;
- модуль должен позволять изменять форму задачи;
- в модуль создания задачи интегрировать функционал выбора формы (типа задачи)

Нефункциональные требования:

- модуль должен быть прост во внедрении и в дальнейшем улучшении;

2.2 Аналитический обзор инструментов

В системе BIMIT используется следующий набор инструментов: frontend – javascript (стандарт es6) + vue.js, backend – java + spring boot. В самом приложении используется компонентный подход

Vue.js представляет современный прогрессивный фреймворк, написанный на языке JavaScript и предназначенный для создания веб-

приложений клиентского уровня. Основная сфера применения данного фреймворка - это создание и организация пользовательского интерфейса.

Vue.js имеет довольно небольшой размер - не более 20 кБ, и при этом обладает хорошей производительностью по сравнению с такими фреймворками как Angular или React. Поэтому неудивительно, что данный фреймворк в последнее время набирает обороты и становится все более популярным. На практике, высокая производительность Vue очень заметна. Компиляция происходит в разы быстрее, по сравнению с тем же Angular, который является самым тяжелым из выше упомянутых фреймворков.

Одним из ключевых моментов в работе Vue.js является **виртуальный DOM**. Структура веб-страницы, как правило, описывается с помощью DOM (Document Object Model), которая представляет организацию элементов html на странице. Для взаимодействия с DOM (добавления, изменения, удаления html-элементов) применяется JavaScript. Но когда мы пытаемся манипулировать html-элементами с помощью JavaScript, то мы можем столкнуться со снижением производительности, особенно при изменении большого количества элементов. А операции над элементами могут занять некоторое время, что неизбежно скажется на пользовательском опыте. Однако если бы мы работали из кода js с объектами JavaScript, то операции производились бы быстрее.

Для этого Vue.js использует виртуальный DOM. Виртуальный DOM представляет легковесную копию обычного DOM. Если приложению нужно узнать информацию о состоянии элементов, то происходит обращение к виртуальному DOM. Если данные, которые используются в приложении Vue.js, изменяются, то изменения вначале вносятся в виртуальный DOM. Потом Vue выбирает минимальный набор компонентов, для которых надо выполнить изменения на веб-странице, чтобы реальный DOM соответствовал виртуальному. Благодаря виртуальному DOM повышается производительность приложения.

Spring Boot является проектом на уровне IO Execution (уровень выполнения) IO Spring Framework.

Spring Boot это следующий шаг Spring, чтобы сделать его легче в настройке и развитии приложений. С Spring Boot конфигурации Spring минимизируются максимально. Spring Boot поддерживает встроенный контейнер (embedded containers), который позволяет веб-приложениям работать независимо и без необходимости применения на Web Server

Вы можете использовать spring Boot чтобы создать Java Web приложение, работающее через команду line "Java -jar" или экспортировать War файл для применения на Web Server как обычно. Spring Boot дает вам "CLI Tool" для запуска сценариев Spring (spring scripts).

Преимущества Spring Boot:

1. Легко используется для развития приложения на основе Spring с Java или Groovy Spring
2. Минимизирует время развития и поднимает производительность
3. Избегает написание многих кодов прототипа (boilerplate), Annotations и конфигурации XML
4. Легко позволяет вам взаимодействовать с приложениями Spring Boot с экологическими системами Spring как Spring JDBC, Spring ORM, Spring Data, Spring Security и т.д
5. Следует подходу "Принципы конфигурации по умолчанию" чтобы минимизировать время и старания, вложенные для развития приложений.
6. Обеспечивает встроенный Server (Embedded HTTP servers) как Tomcat, Jetty чтобы быстро и легко развивать и тестировать веб-приложения
7. Предоставляет инструменты CLI (Command Line Interface) для развития и тестирования приложений Spring Boot (Java или Groovy) из командных строк (command prompt) очень легко и быстро
8. Обеспечивает много плагинов для быстрого развития и тестирования приложения Spring Boot используя инструменты Build, как Maven и Gradle
9. Предлагает много плагинов для легкой работы с контейнерами встроенными базами данных (embedded database) и базами данных хранящиеся в памяти (in- memory Databases).

3 РАЗРАБОТКА МОДУЛЯ

На основе макета разработаем интерфейс привязки формы к задаче. В первую очередь добавим возможность добавлять форму к задаче и менять тип задачи с уже привязанной формой. Реализованные возможности показаны на рисунках 3.1 и 3.2.

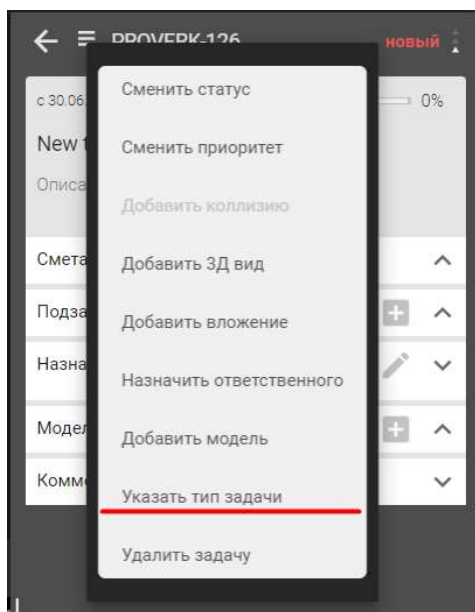


Рисунок 3.1 – Функция привязки формы к новой задаче

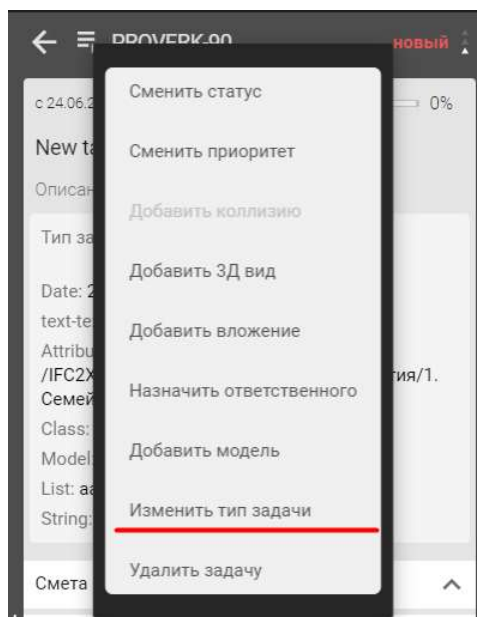


Рисунок 3.2 – Функция смены типа задачи

Смена типа задачи сопровождается диалоговым окном с подтверждением смены формы. Подтверждение необходимо, так как после смены типа задачи все данные формы будут удалены.

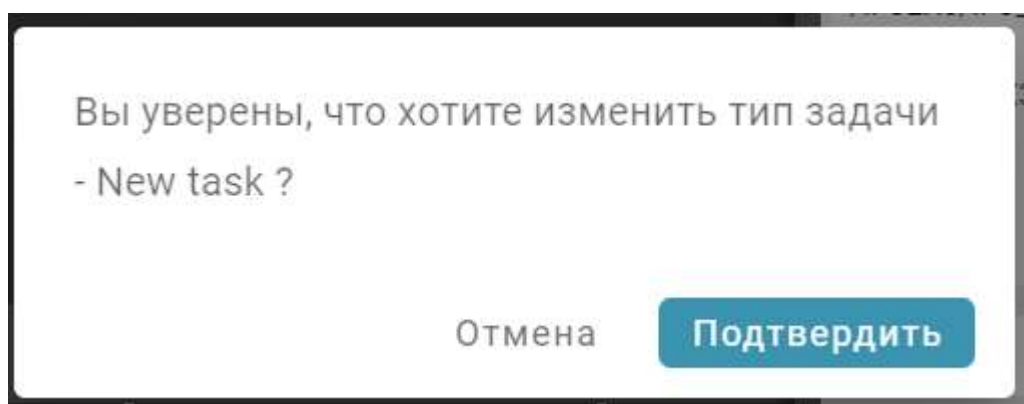


Рисунок 3.3 – Диалоговое окно с подтверждением об изменении типа задачи

Затем необходимо разработать диалоговое окно с выбором формы для привязки к задаче. Окно содержит интерактивные строки с названиями форм. При нажатии на одну из таких строк необходимо выводить форму диалоговое окно с заполнением полей формы. Само же создание формы с необходимыми полями уже реализовано в другом модуле системы VIMIT.

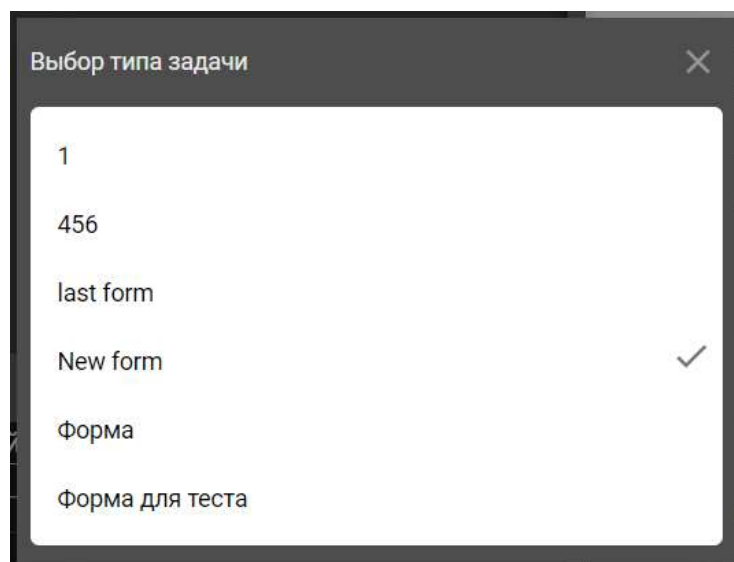


Рисунок 2.4 – Диалоговое окно с выбором формы

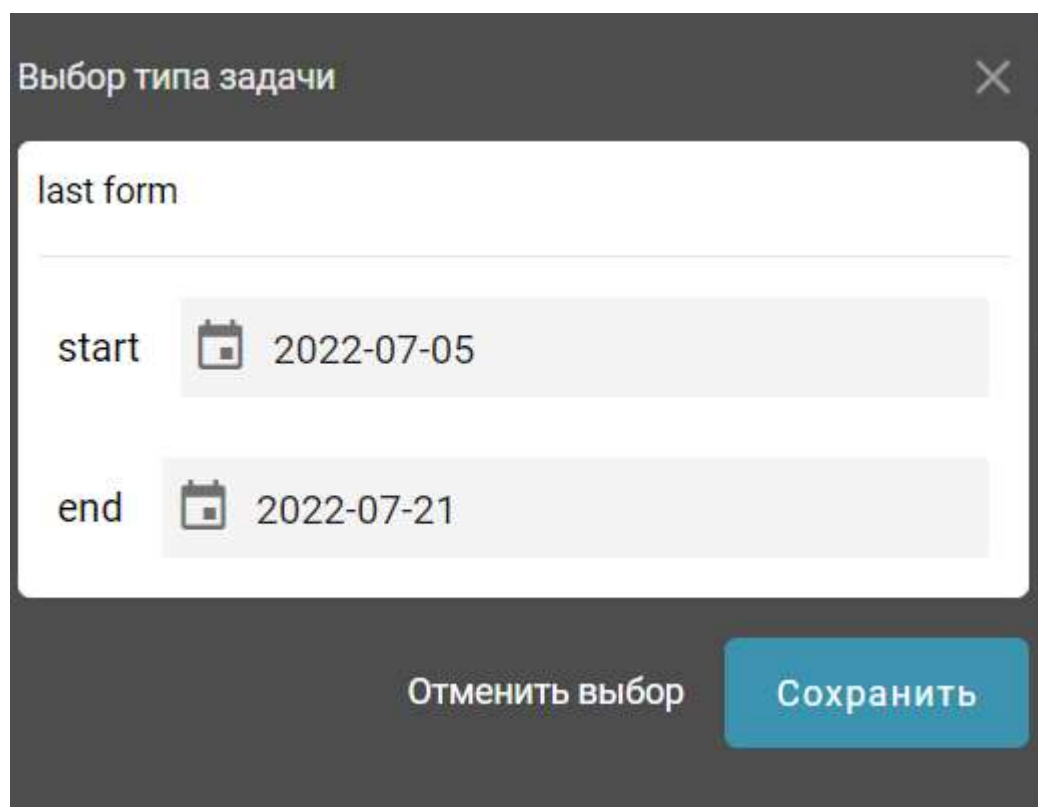


Рисунок 3.5 – Окно с заполнением формы

При сохранении формы данные необходимо выводить в панели с описанием и параметрами задачи. Данные заполненной формы должны храниться в задаче, к которой привязывается форма (рисунок 3.6). Для этого необходимо добавить в модель Task поле taskType. Поле taskType имеет следующий вид:

```
export class TaskType {
  constructor(obj) {
    this.authUser = obj?.authUser
    this.bind = obj?.bind
    this.createdDate = obj?.createdDate
    this.fields = obj?.fields
    this.form = obj?.form
    this.modifiedDate = obj?.modifiedDate
    this.uuid = obj?.uuid
  }
}
```

Для реализации хранения заполненной формы в задаче необходимо реализовать следующие методы:

Actions:

```
addTask({state, commit, dispatch }, obj) {
  obj.taskType = null
  return api.tasks.create(obj).then(task => {
    commit('setSelectedTask', task)
    if(state.creatingTaskType !== null)
      commit('setCreatingTaskTypeBind', task.uuid)
    dispatch('loadTasks')
    return task.uuid
  })
},

setTaskType({commit, state}, taskType){
  state.tasks.map((task, index) => {
    if(taskType && task.uuid == taskType.bind) {
      commit('setTaskType', {taskType, index})
    }
  })
  if(state.creatingTask !== null){
    commit('setCreatingTaskType', {taskType})
  }
},

setTaskTypeFields({dispatch, commit, state}, taskType) {
  let newTaskType = {}
  let indexTask = null
  state.tasks.map((task, index) => {
    if(task.uuid == taskType.bind) {
      newTaskType = task.taskType
      console.log(taskType)
    }
  })
}
```

```

        if(taskType.fields.length > 0) {
            taskType.fields.map((field, index) => {
                field.field.name ? newTaskType.fields[index].field.name =
field.field.name : null
                field.value ? newTaskType.fields[index].value = field.value
: null
            })
        }
        indexTask = index
    }
})

if(newTaskType != null && indexTask != null){
    commit('setTaskType', {taskType: newTaskType, index: indexTask})
    dispatch('loadTasks')
}
},

```

Mutations:

```

setCreatingTaskType: (state, { taskType }) => {
    state.creatingTask.taskType = new TaskType(taskType);
    state.creatingTaskType = new TaskType(taskType); },

setTaskType: (state, {taskType, index} ) => {
    state.tasks[index].taskType = new TaskType(taskType) },

```

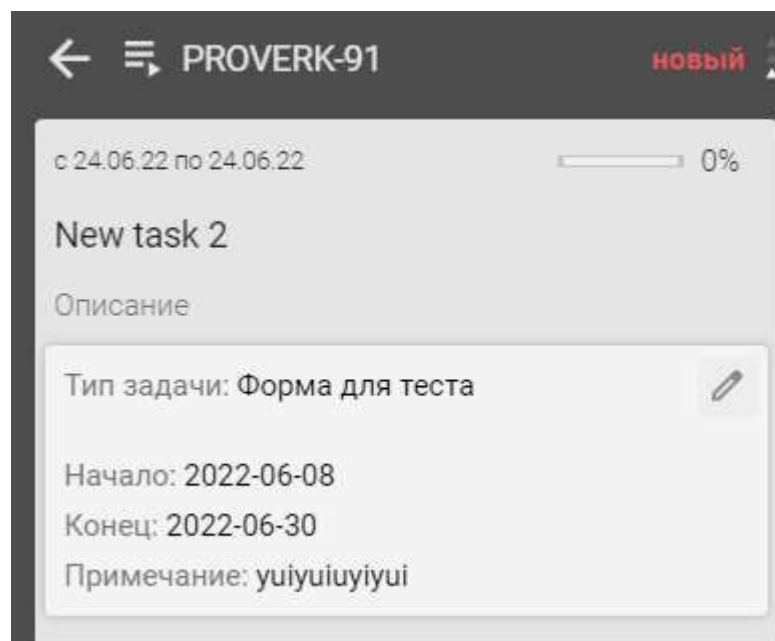


Рисунок 3.6 – Вывод данных в панель с описанием и параметрами задачи

Привязанную форму можно редактировать. Для этого в интерфейсе реализована специальная кнопка, позволяющая редактировать поля привязанной формы. При нажатии на эту кнопку всплывает то же окно, что

показано на рисунке 3.5. При этом все данные привязанной формы переносятся на окно с редактированием формы.

Так как данные формы, которые привязываются к задаче содержатся в хранилище, то после перезагрузки страницы эти данные пропадут. Для этого необходимо разработать методы для организации сетевых запросов.

```
fill: (data) => Factory.formdata.fill.post(data),
dataList: (bind) => Factory.formdata.list[bind].get(),
deleteData: (id) => Factory.formdata.data[id].delete(),
```

Выше перечисленные методы выполняют следующие действия: fill – отправляет на сервер форму с заполненными данными, dataList – выводит данные формы по ИД задачи, к которой привязывалась форма, deleteData – удаляются данные формы по ИД формы. Реализация этих методов на сервере приведена ниже.

```
package ru.mifors.forms.controller;
import java.util.List;
import java.util.UUID;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import ru.mifors.forms.model.FormData;
import ru.mifors.forms.service.FormDataService;
```

```
@RestController
@RequestMapping("/formdata")
public class FormDataController {
```

```
    @Autowired
    private FormDataService formDataService;
```

```

@PostMapping("/fill")
public FormData fillForm(@RequestBody FormData formData) {
    return formDataService.fillForm(formData);
}

@GetMapping("/list/{bind}")
public List<FormData> listByBind(@PathVariable("bind") UUID bind)
{
    return formDataService.listByBind(bind);
}

@DeleteMapping("/data/{id}")
public ResponseEntity<FormData> formDataDel(@PathVariable("id")
UUID id) {
    return formDataService.formDataDel(id);
}
}

```

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были выполнены все поставленные задачи, получены необходимые знания для дальнейшей разработки системы. Также были достигнуты предполагаемые результаты, а именно:

- Разработан интерфейс для привязки форм к задаче
- Реализованы методы для хранения данных формы в хранилище
- Реализованы методы для взаимодействия с сервером
- Модуль интегрирован в систему ВІМІТ

В дальнейшем данную работу необходимо продолжить, разрабатывая элементы и модули самой системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Руководство Spring Boot для начинающих / <https://betacode.net> [Сайт]
URL: <https://betacode.net/11267/spring-boot-tutorial-for-beginners> (дата обращения 30.06.22)
2. Руководство по Vue.js / <https://metanit.com> [Сайт]. [2012-2022]
URL: <https://metanit.com/web/vuejs/> (дата обращения 30.06.22)
3. BIMIT [Сайт]. URL: <https://www.bimit.ru/landing>. (дата обращения 30.06.2022).

ПРИЛОЖЕНИЕ 1: КОД ИНТЕРФЕЙСА

Компонент TaskFormTypeDialog.vue

```
<template lang="pug">
div
  app-dialog(v-model="$_show" :header="'Выбор типа задачи'" width="420")
    app-section(:title="'Выберите тип задачи'")
      div(v-for="type in formTypes")
        v-hover(v-if="type.value == 0" v-slot="{hover}" v-for="form in
formsByType(type.value)" :key="form.uuid" style="cursor: pointer;")
          div.pa-2(@click="onSelect(form)" :class="{ 'hovered-sec': hover }"
style="position: relative;")
            span.black--text.task-form-font {{ form.name }}
            v-icon.check-mark.hovered-mark(v-if="selTaskFormId !=null && form.uuid ==
selTaskFormId && !hover") mdi-check
            v-icon.check-mark(v-else :class="{ 'hovered-mark': hover }") mdi-check
            //-v-img.check-mark( :src="/img/project/tools/check_mark.svg" :width="20"
:height="20" :class="{ 'hovered-sec': hover }")
            //-app-header-action.check-mark(icon="project/tools/check_mark.svg"
:class="{ 'hovered-mark': hover }" size=20)
            fill-form(v-if="this.selForm" :title="'Выбор типа задачи'" v-model="dialog.showForm"
:cancelText="'Отменить выбор'"
:selectedTask="selectedTask" :formId="this.selForm"
@updateFormData="onUpdateFormData")
            //-fill-form(v-model="dialog.fillFormFields" :selectedTask="selectedTask"
:formId="this.selectedTask.taskType.form.uuid" @updateFormData="onUpdateFormData" )
</template>

<script>
import { mapGetters, mapActions, mapMutations } from 'vuex'
import { api } from '@api'
import Field from '@pages/admin/forms/Field'
import DialogToggable from '@components/app/DialogToggable'
import FillForm from '@pages/admin/forms/Form'

export default {
  name: "TaskFormTypeDialog",

  props: {
    selectedTask: null,
    returnData: {
      default:false,
      type:Boolean
    }
  },
  mixins: [DialogToggable],
  components: {
    Field,
    FillForm
  },

  data () {
    const defaultForm = Object.freeze({
      uuid: null,
      bind: null,
      fields: [],
      form: { "uuid" : this.formId, "name": null }
    })
    return {
      selectedForm: null,
      formId: null,
      dialog: {
        showForm: false,
        saveForm: false,
      },
      formData: Object.assign({}, defaultForm),
      defaultForm,
      type: null,
    }
  }
}
```

```

        fieldValues: []
    }
},

mounted () {
    this.$store.dispatch('project/loadAxisEnum')
},

watch: {
    $_show (value) {
        if(!value){
            this.loadTasks()
        }
    }
},

computed: {
    ...mapGetters('forms', ['form', 'forms', 'isFormsLoaded', 'formTypes',
    'isFormTypesLoaded', 'fieldTypes', 'isFieldTypesLoaded']),
    ...mapGetters('project', ['projectUuid']),
    ...mapGetters('task', ['getTaskType']),

    selTaskFormId(){
        if(this.selectedTask      &&      this.selectedTask.taskType      &&
this.selectedTask.taskType.form != undefined){
            return this.selectedTask.taskType.form.uuid
        } else {
            return null
        }
    },

    formTitle() {
        return this.isFormLoaded ? "Opma: " + this.form.name : "Opma "
    },

    fields() {
        return this.isFormLoaded ? this.form.fields : []
    },

    selForm(){
        return this.selectedForm != null ? this.selectedForm.uuid : null
    },
},

methods: {
    ...mapActions('forms', ['saveForm', 'deleteForm', 'gotoForm', 'saveField']),
    ...mapActions('task', ['setTaskType', 'setTaskTypeFields', 'loadTasks']),
    ...mapMutations('forms', ['setForm']),

    onSelect (item) {
        this.selectedForm = item
        this.dialog.showForm = true
        //this.removeFormData(item)
        this.prepareNewForm(item)
    },

    removeFormData(item) {
        if(this.selectedTask.taskType && this.selectedTask.taskType.uuid) {
            api.forms.deleteData(this.selectedTask.taskType.uuid).then(data => {
                console.log(data)
                this.prepareNewForm(item)
            })
        } else {
            this.prepareNewForm(item)
        }
    },

    prepareNewForm(item) {
        this.formId = item.uuid
        if(this.formId != null){

```

```

    this.$store.dispatch('forms/LOAD_FORM', { formId: this.formId })
    this.formData = Object.assign({}, this.defaultForm)

    this.formData.fields = []

    this.formData.bind = this.selectedTask.uuid
    this.formData.form.name = item.name
    this.formData.form.uuid = item.uuid

    item.fields.forEach(field => {
      this.fillField(field)
    })
    //this.formData.form.name = this.form.name
    if(this.selectedTask.uuid){
      this.fillForm()
    } else {
      this.setTaskType(this.formData)
    }
  }
},

fillField(field) {
  let newField = {
    uuid: null,
    field: { "uuid" : null, "name" : null},
    alias: null,
    type: null,
    value: null,
    valueId: null
  }

  if(field) {
    newField.field.uuid = field.uuid
    newField.field.name = field.name
    newField.field.type = field.type
    this.formData.fields.push(newField)
  }
},

formsByType (type) {
  return this.forms.filter(f => f.formType.value == type)
},

fillForm() {
  if (this.returnData) {
    this.$emit("filled", this.formData.fields)
  } else {
    console.log(this.selectedTask.taskType)
    console.log(this.formData)
    if(this.selectedTask.taskType.form.uuid != this.formData.form.uuid){
      console.log("HERE")
      this.setTaskType(this.formData)
    }
  }
},

onUpdateFormData(data) {
  if(data.bind) {
    let newTaskType = data
    newTaskType.bind = this.selectedTask.uuid
    this.setTaskTypeFields(newTaskType)
    this.$emit('updateForm', false)
  } else {
    this.$emit('updateForm', false)
  }
},

}
}
</script>

```

Компонент Form.vue

```
<template lang="pug">

  app-dialog(v-model="$_show" :header="title" @confirm="fillForm()"
:hideOverlay="hideOverlay" confirmText="Сохранить" :cancelText="cancelText"
width="420")

    app-section

      .form-title.pt-1

        .black--text.title-form-font {{ formTitle }}

        div.separator.d-block

        div(v-for="(f, index) in fields")

          field.mt-1(:field="f" :key="f.type + ':' + f.uuid"
:valueList="formData.fields")

</template>
```

```
<script>
import { mapActions, mapGetters, mapMutations } from 'vuex'
import { api } from "@api"
import Field from "../Field"
import DialogToggable from '@components/app/DialogToggable'

export default {
  name: "FillForm",

  props: {
    title: {
      default: '',
      type:String
    },
    cancelText: {
      default: '',
      type:String
    },
    formId: null,
    selectedTask: null,
    returnData: {
      default:false,
      type:Boolean
    },
    hideOverlay: {
      default:false,
      type:Boolean
    }
  },

  mixins: [DialogToggable],
  components: {
    Field
  },

  data () {
    const defaultForm = Object.freeze({
      uuid: null,
      bind: null,
      fields: [],
      form: { "uuid" : this.formId }
    })

    return {
      taskFields: [],

```

```

    dialog: {
      showForm: true,
    },
    formData: Object.assign({}, defaultForm),
    defaultForm,
    type: null,
    fieldValues: [],
  }
},

mounted () {
  this.$store.dispatch('project/loadAxisEnum')
},

watch: {
  $_show (value) {
    if (value) {
      this.formData = Object.assign({}, this.defaultForm)
      console.log(this.formData)
      this.taskFields = []
      if(this.selectedTask && this.selectedTask.uuid != null) {
        console.log(this.formData.fields)
        this.formData.fields = []
        this.selectedTask.taskType.fields.map((el, index) => {
this.taskFields.push(el.field); this.taskFields[index].defaultValue = el.value })
        console.log(this.taskFields)
      }

      if(this.selectedTask.uuid == null) {
        this.formData.fields = []
      }

      // this.formData.bind = UUID того, к чему привязываем данные заполняемой формы
      this.$store.dispatch('forms/LOAD_FORM', { formId: this.formId })
      console.log(this.formData)
    } else {
      this.formData = Object.assign({}, this.defaultForm)
      this.formData.fields = []
    }
  },
},

computed: {
  ...mapGetters('forms', ['form', 'isFormLoaded']),
  ...mapGetters('task', ['getTaskType']),

  formTitle() {
    return this.isFormLoaded ? this.form.name : " "
  },

  fields() {
    if(this.selectedTask && this.selectedTask.taskType && this.selectedTask.uuid &&
this.selectedTask.taskType.form){
      if (this.selectedTask.taskType.form.uuid == this.formId && this.formId != null
&& this.taskFields.length != 0){
        return this.taskFields
      } else {
        return this.isFormLoaded ? this.form.fields : []
      }
    } else {
      return this.isFormLoaded ? this.form.fields : []
    }
  },
},

methods: {
  ...mapMutations('task', ['setTaskType', 'setSelectedTaskField']),
  ...mapActions('task', ['loadTasks']),

  fillForm() {

```

```

let formObj = {}
if(this.selectedTask){
  formObj = this.selectedTask.taskType
  this.formData.uuid = formObj.uuid
  this.formData.bind = formObj.bind
  this.formData.form = this.form
}

if(this.selectedTask.uuid == null){
  console.log(this.formData.fields)
  this.formData.fields.map((field, index) => {
    formObj.fields[index].value = field.value
    //this.selectedTask.taskType.fields[index] = field.value
  })
}

if (this.returnData){
  this.$emit("filled", this.formData.fields)
} else {
  if(this.formData.bind){
    api.forms.fill(this.formData).then(data => {
      this.$emit('updateFormData', data);
    })
  } else {
    this.$emit('updateFormData', this.formData);
  }
}
},
}
}
</script>

```