ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

старший преподаватель		С. Ю. Гуков
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 1

АРХИТЕКТУРА СИСТЕМЫ. MV-ШАБЛОНЫ

по курсу:

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4326		Г. С. Томчук
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

1 Цель работы

Цель работы: вспомнить и применить на практике принципы объектно-ориентированного программирования (ООП), используя основные элементы и понятия ООП: классы, объекты, интерфейсы, наследование, инкапсуляция, полиморфизм, абстракция. Внедрить в проект шаблон проектирования MVP (рекомендуется, но можно MVVM, MVI и др.). Изучить и применить на практике принципы SOLID.

2 Задание

Необходимо проект своей курсовой работы, написанной в третьем семестре по дисциплине «Основы программирования» переделать в соответствии со стандартами шаблона MVP [Model-View-Presenter] (рекомендуется, но можно MVVM, MVI и др.). Если же проект курсовой был неудачный, некачественный, не соответствует требованиям ниже или просто не нравится, то можно и нужно дополнить старый либо создать новый проект на любую тематику.

Проект обязательно должен иметь графический пользовательский интерфейс (User Interface, UI), а также может быть написан на любом языке программирования.

3 Краткое описание хода разработки и назначение используемых технологий

В ходе выполнения лабораторной работы была выбрана тематика приложения для учёта складского инвентаря. Основная цель разработки заключалась в создании удобного интерфейса для добавления, редактирования, поиска и удаления товаров, а также в организации надёжного хранения данных с возможностью их дальнейшего расширения.

В качестве технологической основы был выбран Electron, позволяющий создавать кроссплатформенные настольные приложения на базе веб-технологий. Это дало возможность объединить преимущества веб-разработки с функционалом нативного приложения. Для клиентской части использовался React в связке с TypeScript, что обеспечило строгую типизацию, повысило

читаемость и поддерживаемость кода, а также позволило применять современные подходы к построению интерфейсов. Визуальная составляющая была реализована с помощью TailwindCSS и библиотеки компонентов shadcn/ui, что позволило быстро создавать адаптивный и современный дизайн.

Данные о товарах хранятся в JSON-файле, что упрощает работу с ними и делает структуру хранения прозрачной. Для взаимодействия с данными применён репозиторий – отдельный класс, отвечающий за чтение и запись в JSON, что отражает принцип инкапсуляции бизнес-логики.

Поскольку архитектура Electron разделяет процессы (основной процесс и процессы рендеринга), для связи между ними используются IPC-обработчики (Inter-Process Communication). Этот механизм позволяет безопасно вызывать методы работы с данными из рендер-процесса (где работает React-приложение), передавая команды в основной процесс, который имеет доступ к файловой системе.

В качестве архитектурного шаблона был выбран MVVM (Model–View–ViewModel). Модель описывает структуру данных и операции с ними, ViewModel инкапсулирует бизнес-логику и предоставляет удобный интерфейс для управления состоянием и действиями, а View отвечает за визуальное отображение и связывается с ViewModel. Такой подход обеспечивает разделение ответственности, повышает расширяемость и облегчает тестирование приложения.

В Приложении А представлена часть исходного кода приложения, поскольку полный исходный код слишком массивен. Полный исходный код находится по ссылке: https://github.com/grigorijtomczuk/warehouse-manager.

4 Результаты работы программы с примерами разных сценариев

На рисунках 1–4 представлены скриншоты финальной версии приложения при различных сценариях взаимодействия.

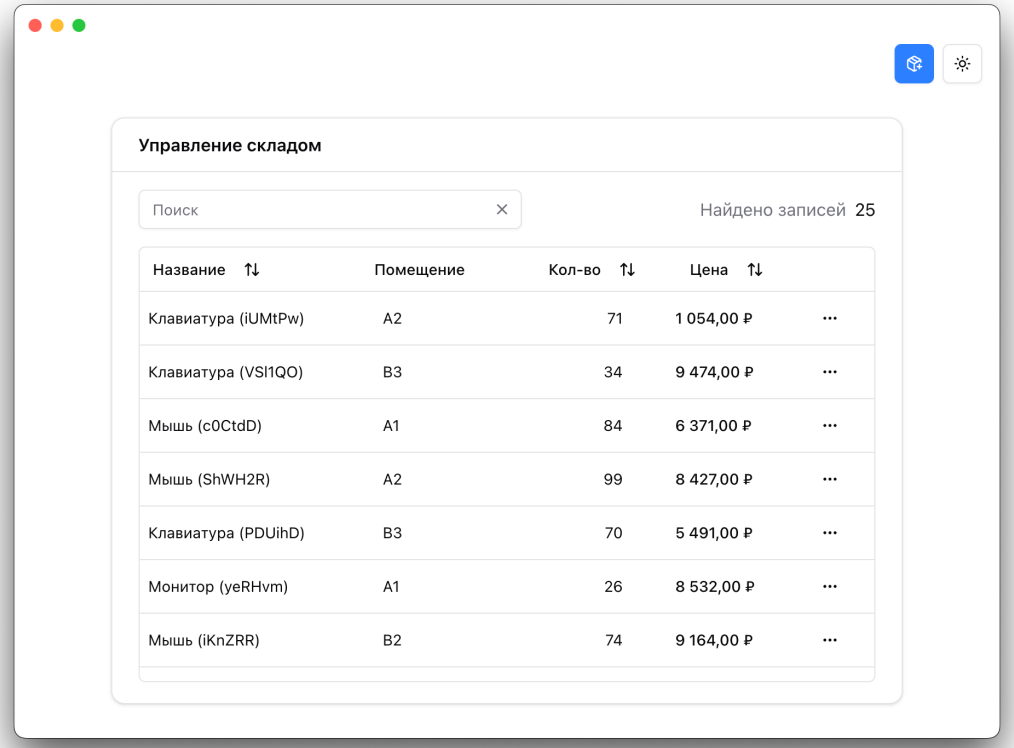

Рисунок 1 – Окно программы при открытии

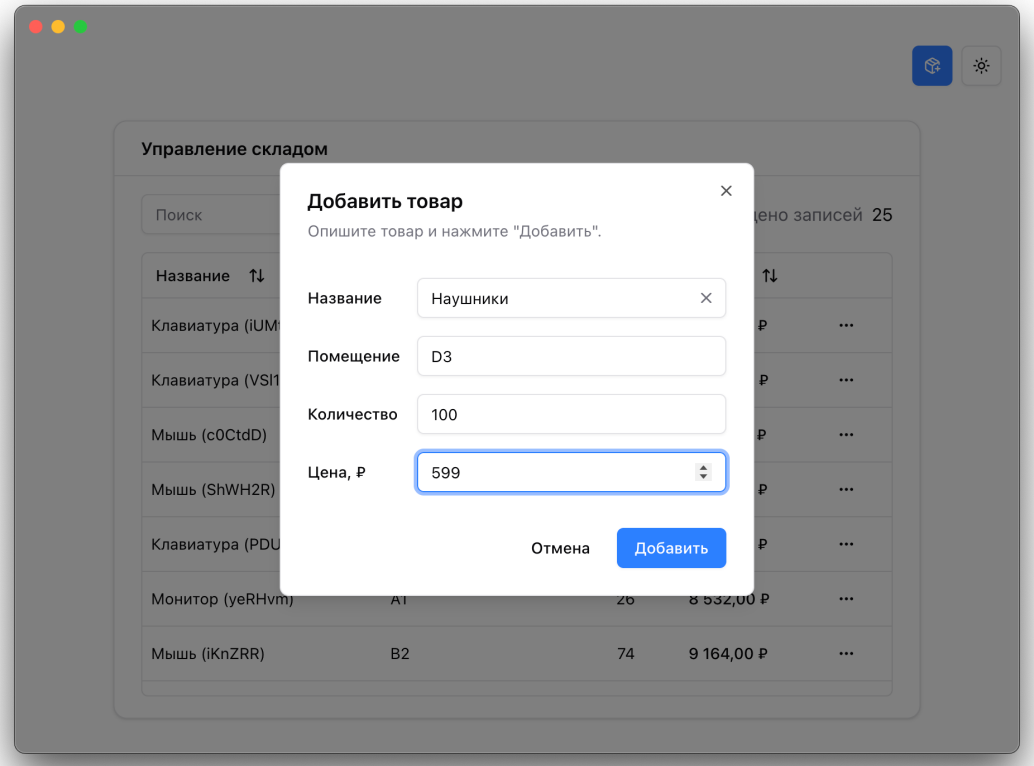

Рисунок 2 – Добавление нового товара на склад

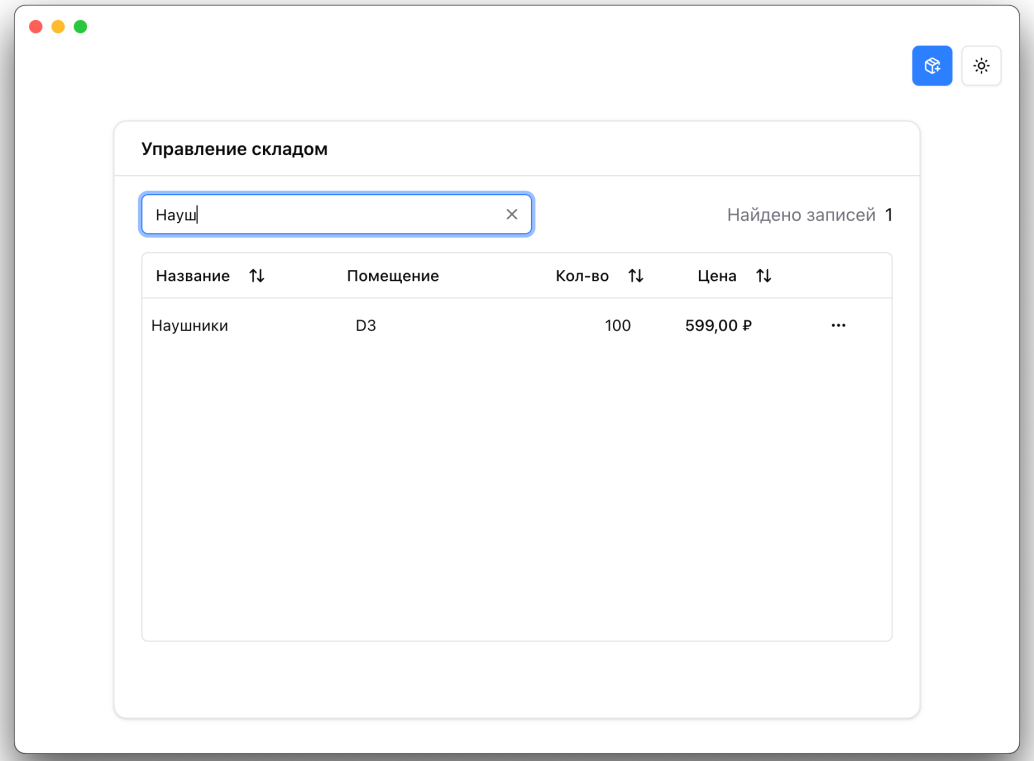

Рисунок 3 – Поиск товара по названию

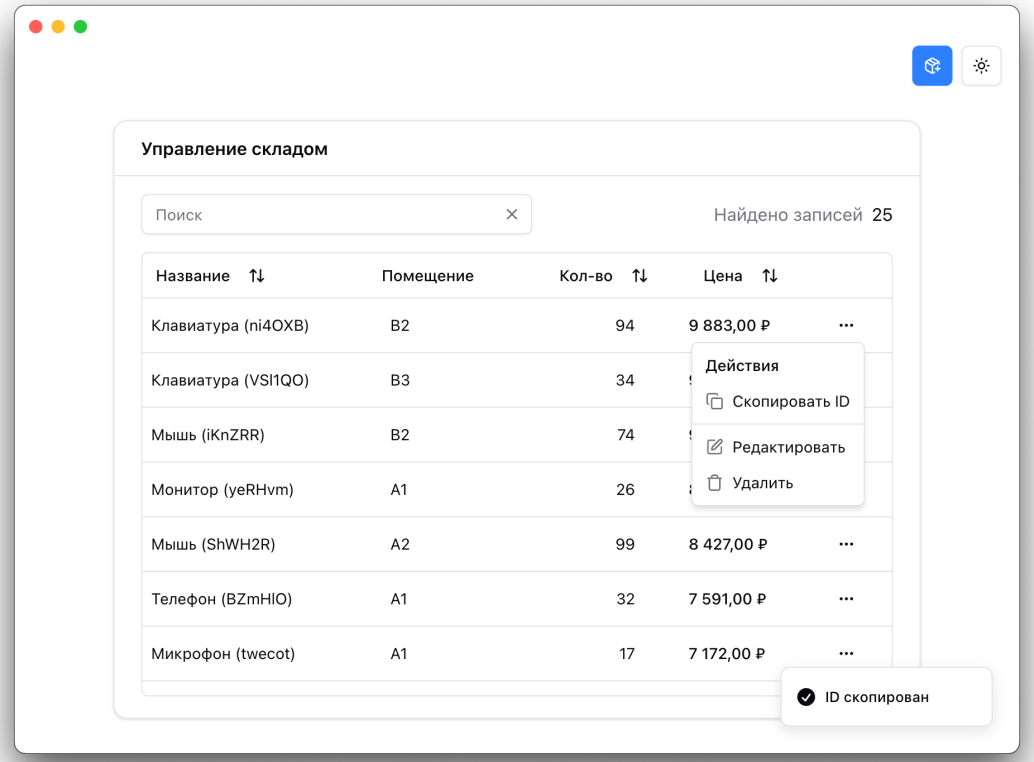

Рисунок 4 – Сортировка товаров по цене на убывание

5 Выводы

В ходе выполнения лабораторной работы была спроектирована и реализована настольная система для учёта складского инвентаря. В процессе разработки удалось закрепить навыки применения архитектурных шаблонов и принципов проектирования.

Была отработана работа с Electron, что позволило объединить веб-технологии и функционал настольных приложений. Использование React и TypeScript обеспечило удобное и надёжное построение пользовательского интерфейса с жёсткой типизацией. Для визуальной части были применены TailwindCSS и shadcn/ui, что позволило реализовать современный и адаптивный дизайн.

Хранение данных в JSON-файле и организация доступа к ним через репозиторий продемонстрировали применение принципов инкапсуляции и разделения ответственности. Использование IPC-обработчиков позволило безопасно связать основной процесс Electron и рендер-процесс, что является важным элементом при работе с файловой системой.

Реализация архитектурного шаблона MVVM обеспечила чёткое разделение между моделью, представлением и моделью представления, что повысило удобство сопровождения и масштабируемость приложения.

Таким образом, работа позволила не только освоить современные технологии разработки настольных приложений, но и закрепить применение принципов ООП и SOLID на практике.

ПРИЛОЖЕНИЕ А

JsonStorageRepo.ts

```
import { IInventoryItem } from "@shared/models/InventoryItem"
import fs from "fs"
import path from "path"

// Класс-репозиторий для работы с инвентарём, хранит данные в JSON-файле
export class JsonStorageRepo {
  // Путь к файлу с данными
  private readonly filePath: string

  constructor() {
    // Формируем путь к файлу с инвентарём
    this.filePath = path.join(__dirname, "../../data", "inventory.json")
    // Если файл не существует — создаём пустой массив
    if (!fs.existsSync(this.filePath)) {
      fs.writeFileSync(this.filePath, "[]", "utf-8")
    }
  }

  // Загружает все элементы инвентаря из файла
  load(): IInventoryItem[] {
    // Чтение файла как строки
    const raw = fs.readFileSync(this.filePath, "utf-8")
    // Преобразование строки в массив объектов
    return JSON.parse(raw)
  }

  // Сохраняет массив элементов инвентаря в файл
  save(items: IInventoryItem[]): void {
    // Сериализация массива в JSON с отступами для удобства
    fs.writeFileSync(this.filePath, JSON.stringify(items, null, 2), "utf-8")
  }
}
```

inventoryViewModel.ts

```
import { loadInventory, saveInventory } from "@renderer/lib/utils"

import { IInventoryItem } from "@shared/models/InventoryItem"

// ViewModel для управления инвентарём (MVVM)
export class InventoryViewModel {
  // Приватное поле с текущим списком элементов
  private _items: IInventoryItem[] = []

  // Геттер для получения элементов
  get items(): IInventoryItem[] {
    return this._items
  }

  // Асинхронная загрузка данных из хранилища
  async load() {
    this._items = await loadInventory()
  }

  // Добавление нового элемента (генерируется новый id)
  addItem(item: Omit<IInventoryItem, "id">): void {
    const newItem: IInventoryItem = { ...item, id:
crypto.randomUUID() }
    this._items.push(newItem)
    saveInventory(this._items)
  }

  // Обновление существующего элемента по id
  updateItem(updated: IInventoryItem): void {
    this._items = this._items.map((item) => (item.id === updated.id
? updated : item))
    saveInventory(this._items)
  }

  // Удаление элемента по id
  deleteItem(id: string): void {
    this._items = this._items.filter((item) => item.id !== id)
    saveInventory(this._items)
  }
}
```

Main.tsx

```
import { Card, CardContent, CardHeader, CardTitle } from
"@renderer/views/components/ui/card"
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogFooter,
  DialogHeader,
  DialogTitle
} from "@renderer/views/components/ui/dialog"

import Button from "@renderer/views/components/Button"
import DataTable from "@renderer/views/components/DataTable"
import { IInventoryItem } from "@shared/models/InventoryItem"
import Input from "@renderer/views/components/Input"
import { Label } from "@renderer/views/components/ui/label"
import { PackagePlusIcon } from "lucide-react"
import { Separator } from "@renderer/views/components/ui/separator"
import ThemeToggle from "@renderer/views/components/ThemeToggle"
import useInventoryItemsColumns from "@renderer/views/data-table-
columns/inventoryItems"
import { useInventoryViewModel } from
"@renderer/hooks/useInventoryViewModel"
import { useState } from "react"

function Main(): React.JSX.Element {
  const [selectedItem, setSelectedItem] = useState<IInventoryItem |
null>(null)
  const [isDialogOpen, setDialogOpen] = useState(false)

  const emptyFormData = {
    name: "",
    room: "",
    quantity: 1,
    price: 0
  }

  const [formData, setFormData] = useState<Omit<IInventoryItem,
"id">>(emptyFormData)
  const [formError, setFormError] = useState<string | null>(null)
```

```
  const vm = useInventoryViewModel()

  const openForm = (item?: IInventoryItem): void => {
    if (item) {
      setSelectedItem(item)
      setFormData({
        name: item.name,
        room: item.room,
        quantity: item.quantity,
        price: item.price
      })
    } else {
      setSelectedItem(null)
      setFormData(emptyFormData)
    }
    setDialogOpen(true)
  }

  const handleSubmit = (): void => {
    // Проверка обязательных полей
    if (!formData.name.trim() || !formData.room.trim() ||
!formData.quantity || !formData.price) {
      setFormError("Пожалуйста, заполните все обязательные поля.")
      return
    }
    setFormError(null)
    if (selectedItem) {
      vm.updateItem({ ...selectedItem, ...formData })
    } else {
      vm.addItem({ ...formData })
    }
    setDialogOpen(false)
  }

  return (
    <div className="relative mr-4 ml-4 flex h-full items-center
justify-center">
      <div className="absolute top-0 right-0 flex gap-2">
        <Button size="icon" tooltip="Добавить товар" onClick={() =>
openForm()}>
          <PackagePlusIcon />
```

```
        </Button>
        <ThemeToggle />
      </div>

      <Card className="h-4/5 w-5/6 gap-0 py-0">
        <CardHeader className="gap-0 py-4">
          <CardTitle>Управление складом</CardTitle>
        </CardHeader>
        <Separator />
        <CardContent className="h-full overflow-y-clip pb-34">
          <DataTable
            className="h-full"
            columns={useInventoryItemsColumns(vm, openForm)}
            data={[...vm.items]} // Передаем копию массива, чтобы
таблица ререндерилась при обновлении данных
          />
        </CardContent>
      </Card>

      {/* Диалог для добавления/редактирования */}
      <Dialog open={isDialogOpen} onOpenChange={setDialogOpen}>
        <DialogContent className="sm:max-w-[425px]">
          <DialogHeader>
            <DialogTitle>{selectedItem ? "Редактировать товар" :
"Добавить товар"}</DialogTitle>
            <DialogDescription>
              Опишите товар и нажмите &amp;quot;{selectedItem ?
"Сохранить" : "Добавить"}&amp;quot;.
            </DialogDescription>
          </DialogHeader>

          {/* Show error message if validation fails */}
          {formError && <div className="text-destructive text-
sm">{formError}</div>}

          <div className="grid gap-4 py-4">
            <div className="grid grid-cols-4 items-center gap-4">
              <Label htmlFor="name" className="text-right">
                Название
              </Label>
              <Input
                id="name"
```

```
                  className="col-span-3"
                  clearable
                  value={formData.name}
                  required
                  onChange={(event) => setFormData({ ...formData,
name: event.target.value })}
                />
              </div>
              <div className="grid grid-cols-4 items-center gap-4">
                <Label htmlFor="name" className="text-right">
                  Помещение
                </Label>
                <Input
                  id="room"
                  className="col-span-3"
                  value={formData.room}
                  maxLength={2}
                  required
                  placeholder="A1"
                  onChange={(event) => setFormData({ ...formData,
room: event.target.value })}
                />
              </div>
              <div className="grid grid-cols-4 items-center gap-4">
                <Label htmlFor="quantity" className="text-right">
                  Количество
                </Label>
                <Input
                  id="quantity"
                  className="col-span-3"
                  type="number"
                  required
                  value={formData.quantity || 1}
                  onChange={(event) =>
                    setFormData({ ...formData, quantity:
Number(event.target.value) })
                  }
                />
              </div>
              <div className="grid grid-cols-4 items-center gap-4">
                <Label htmlFor="price" className="text-right">
                  Цена, ₽
```

```
              </Label>
              <Input
                id="price"
                className="col-span-3"
                required
                type="number"
                value={formData.price || 0}
                onChange={(event) =>
                  setFormData({ ...formData, price:
Number(event.target.value) })
                }
              />
            </div>
          </div>

          <DialogFooter>
            <Button variant="ghost" onClick={() =>
setDialogOpen(false)}>
              Отмена
            </Button>
            <Button onClick={handleSubmit}>{selectedItem ?
"Сохранить" : "Добавить"}</Button>
          </DialogFooter>
        </DialogContent>
      </Dialog>
    </div>
  )
}

export default Main
```