

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Разработка системы визуальной навигации
на территории вуза

Студент _____ Давиденко Александр Александрович
(подпись, дата) (ФИО полностью)

Руководитель _____ Попова Н.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Давиденко Александр Александрович гр. 21ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Разработка системы визуальной навигации на территории вуза

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Попова Н.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.) _____

1) *Назначение: создание программной системы визуальной навигации по территории университета и интерактивного взаимодействия с инфраструктурой.*

4) *Среда выполнения: мобильное устройство*

Языковые средства разработки: Python, TypeScript, UML

Инструментальные средства: PyCharm, Visual Studio Code, Mermaid,

Технологии разработки: PWA, REST API, Vue.js.

5) *Основные функции приложения:*

а) интерактивная работа с картой: перемещение, масштабирование;

б) фильтрация объектов по функциональному назначению;

в) адаптивность интерфейса под различный размер экрана;

г) поиск зданий по аудитории, названию, назначению;

д) отображение информации о каждом объекте инфраструктуры.

5. Перечень подлежащих разработке вопросов:

1) *Анализ предметной области и требований к системе;*

2) *Проектирование программных средств;*

3) Реализация программных средств;
4) Тестирование программных средств;
5) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить

Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 94 листа, 26 рисунков, 11 таблиц и 20 использованных источников.

ИНТЕРАКТИВНАЯ КАРТА, НАВИГАЦИЯ, ПРИЛОЖЕНИЕ, СЛОИСТАЯ АРХИТЕКТУРА, БАЗА ДАННЫХ, Vue.js, SQLite, Python, JSON, REST, UML.

Объект исследования – цифровое представление инфраструктуры университета в формате визуальной системы навигации.

Цель выпускной квалификационной работы – разработка информационной системы «Интерактивная карта ПГУ», позволяющей пользователям ориентироваться на территории кампуса, осуществлять поиск зданий и помещений, а также получать информацию об их характеристиках.

Технологии разработки – клиентская часть реализована с использованием фреймворка Vue.js. Интерфейс адаптирован под различные устройства. Серверная часть написана на языке Python и включает модуль управления базой данных SQLite, хранящей сведения о зданиях, аудиториях, удобствах и структурных подразделениях. Передача данных осуществляется в формате JSON, взаимодействие организовано по принципам REST.

Среда исполнения – любой современный веб-браузер на настольных и мобильных устройствах. Сервер разворачивается локально или в контейнерной среде на базе Linux.

Результаты работы – разработан полнофункциональный прототип веб-приложения, позволяющий пользователю получать доступ к информации об объектах кампуса в наглядной и структурированной форме. Система обеспечивает фильтрацию, поиск, отображение характеристик зданий и помещений. Все заявленные функциональные и нефункциональные требования реализованы, архитектура оформлена в виде UML-диаграмм. Приложение демонстрирует стабильную работу, адаптивность интерфейса и возможность масштабирования.

					ПГУ 09.03.02– 08ВР211.10 ПЗ							
Изм.	Лист	№ докум.	Подпись	Дата								
Разраб.		Давиденко А.А.			Разработка системы визуальной навигации на территории вуза. Пояснительная записка			Лит.	Лист	Листов		
Пров.		Попова Н.А.								4	94	
								Группа 21ВИ1				
Н. Контр.		Такташкин Д.В										
Утв.		Козлов А.Ю.										

Содержание

Введение.....	7
1 Анализ программных средств интерактивных карт вузов.....	8
1.1 Исследование предметной области.....	8
1.2 Обзор и анализ аналогичных решений	10
1.2.1 Карта МГУ им. М. В. Ломоносова	10
1.2.2 Campus Map от MIT	11
1.2.3 Карта Университета Сиднея	12
1.2.4 Карта Высшей Школы Экономики	13
1.2.5 Карта Санкт–Петербургского государственного университета.....	14
1.2.6 Сравнение программных средств.....	15
1.3 Анализ функциональных требований	17
1.4 Анализ нефункциональных требований	20
1.5 Планирование разработки сервиса интерактивной карты ПГУ	22
2 Проектирование программных средств «Интерактивная карта ПГУ».....	25
2.1 Архитектура приложения.....	25
2.2 Проектирование логической структуры данных	26
2.3 Проектирование пользовательского интерфейса.....	31
3 Реализация программных средств «Интерактивная карта ПГУ».....	34
3.1 Реализация архитектуры серверной части приложения	34
3.2 Реализация базы данных	37
3.3 Реализация пользовательского интерфейса	39
4 Контроль качества приложения «Интерактивная карта ПГУ»	46
4.1 Функциональное тестирование.....	46

4.2 Оценка метрик кода	48
4.2.1 Метрика Чепина	48
4.2.2 Цикломатическая сложность функции	50
4.3 Нефункциональное тестирование	51
Заключение	53
Список использованных источников	54
Приложение А (обязательное) Листинг приложения.....	56
Приложение Б (рекомендуемое) Результаты тестирования	91

Введение

В условиях цифровой трансформации образовательной среды возрастает необходимость в инструментах, способствующих эффективной навигации и информационной доступности. Особенно это актуально для университетских кампусов, которые представляют собой сложные инфраструктурные объекты с множеством корпусов, подразделений и вспомогательных объектов.

Интерактивная карта Пензенского государственного университета (ПГУ) – это современное веб-приложение, призванное упростить ориентацию на территории вуза для студентов, преподавателей, сотрудников и гостей. Система предоставляет структурированную информацию о зданиях, институтах, аудиториях, местах общего пользования и объектах инфраструктуры. Актуальность разработки обусловлена отсутствием единого цифрового сервиса для визуализации пространственной структуры ПГУ, потребностью в удобном мобильном интерфейсе и важностью обеспечения инклюзивного доступа для пользователей с ограниченными возможностями.

Цель работы – разработка адаптивного и интерактивного веб-приложения, реализующего функциональность интерактивной карты, с поддержкой поиска, фильтрации, визуализации информации и работы в различных пользовательских сценариях.

1 Анализ программных средств интерактивных карт вузов

1.1 Исследование предметной области

Развитие современного высшего образования сопровождается необходимостью создания комфортной, безопасной и информационно насыщенной образовательной среды. В её рамках важную роль играют средства пространственной ориентации, особенно в условиях крупного учебного комплекса, включающего многочисленные здания, корпуса, общежития, библиотеки и иные объекты инфраструктуры.

На фоне увеличивающегося количества студентов, преподавателей, а также приглашённых участников конференций, олимпиад и иных мероприятий, возникает объективная необходимость в удобных и понятных инструментах, позволяющих эффективно ориентироваться в пространстве вуза. Это особенно важно для первокурсников, гостей университета и лиц, не знакомых с планировкой территории.

Как отмечается в исследованиях, информационная насыщенность физического пространства образовательной организации способствует снижению стрессогенных факторов и формирует у обучающихся чувство уверенности и принадлежности [1]. Отсутствие систем навигации, основанных на актуальной информации, приводит к дезориентации, потере времени и снижению вовлеченности студентов в образовательный процесс.

На протяжении последних лет отмечается активное внедрение различных цифровых решений в образовательную практику. Однако, несмотря на общий вектор цифровизации, значительная часть вузов по-прежнему использует устаревшие и малофункциональные схемы навигации: стендовые карты, бумажные указатели и таблицы с расписаниями. Подобные средства не обладают гибкостью, не обновляются своевременно и не адаптируются под индивидуальные запросы пользователя.

При этом возможность динамического отображения пространственной информации и её фильтрации по параметрам становится определяющим

условием в системах пространственной навигации [2]. Следовательно, разработка цифровой карты, способной учитывать особенности инфраструктуры вуза и потребности различных категорий пользователей, является актуальной и практически значимой задачей.

Кроме студентов и преподавателей, система пространственной навигации может быть полезна для: родителей, сопровождающих абитуриентов в приёмную кампанию, сотрудников административных подразделений, организующих мероприятия в нескольких локациях, лиц с ограничениями по здоровью, которым важно заранее оценить условия доступности.

Таким образом, потенциальная аудитория охватывает широкие слои участников образовательного процесса и выходящих за его пределы.

Необходимость разработки специализированного программного продукта обусловлена ещё и тем, что существующие в открытом доступе решения чаще всего не ориентированы на специфику образовательных учреждений. Они либо представлены в виде статичных изображений, либо требуют наличия внешних подключений и доступа к сторонним сервисам, что не всегда возможно или безопасно. Кроме того, такие решения не учитывают логическую структуру университета: деление на институты, факультеты, кафедры, распределение по этажам и функциональным зонам.

Вопросы архитектурной организации и адаптивности информационных систем навигации в кампусах вузов рассматриваются в научной и педагогической литературе. Создание пространственно-ориентированных цифровых сервисов позволяет формировать у студентов целостное восприятие университетской среды и способствует повышению мотивации к обучению [3]. Это подтверждает не только утилитарную, но и педагогическую значимость подобных разработок.

Подводя итог, следует отметить, что в условиях активного развития цифровой образовательной среды, создание системы пространственной навигации для вуза является актуальной задачей. Она отвечает не только требованиям времени, но и ожиданиям студентов, преподавателей и всех

участников образовательного процесса. Интерактивная карта университета может стать важной частью цифровой инфраструктуры, обеспечивая доступность, открытость и удобство взаимодействия с пространством вуза.

1.2 Обзор и анализ аналогичных решений

1.2.1 Карта МГУ им. М. В. Ломоносова

Карта Московского государственного университета имеет простой пользовательский интерфейс, отраженный на рисунке 1, в большей степени ориентированный на общее знакомство с географией кампуса. Пользователь может просматривать здания и их назначения, однако отсутствует интерактивность внутри объектов, детализация по этажам и возможность фильтрации по назначению. Также не предусмотрен поиск по аудиториям или структурным подразделениям.

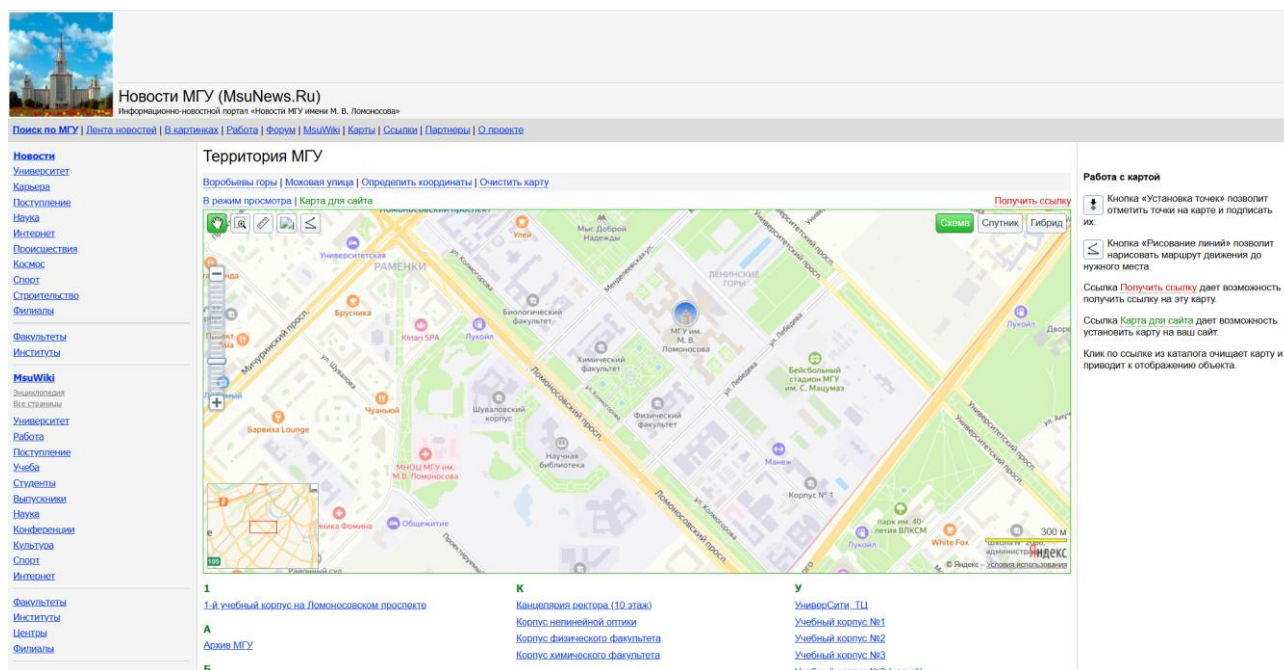


Рисунок 1 – Интерфейс сайта карты МГУ

Достоинства: выполнена в виде общей схемы с номерами и наименованиями корпусов, публичный доступ без регистрации.

Недостатки: отсутствует детализация внутри зданий, не реализована фильтрация или поиск по типу объектов, отсутствует адаптация под мобильные

устройства, устаревший интерфейс, карта МГУ не является самостоятельным сервисом, а интеграцией сервиса Яндекс Карты

1.2.2 Campus Map от MIT

Собственные решения университетов, такие как MIT Campus Map, представляют собой специализированные веб-сервисы, предназначенные для внутреннего пользования. В случае Массачусетского технологического института карта реализует функции поиска зданий, просмотра этажей, доступа к расписаниям и контактной информации. Интерфейс представлен на рисунке 2.

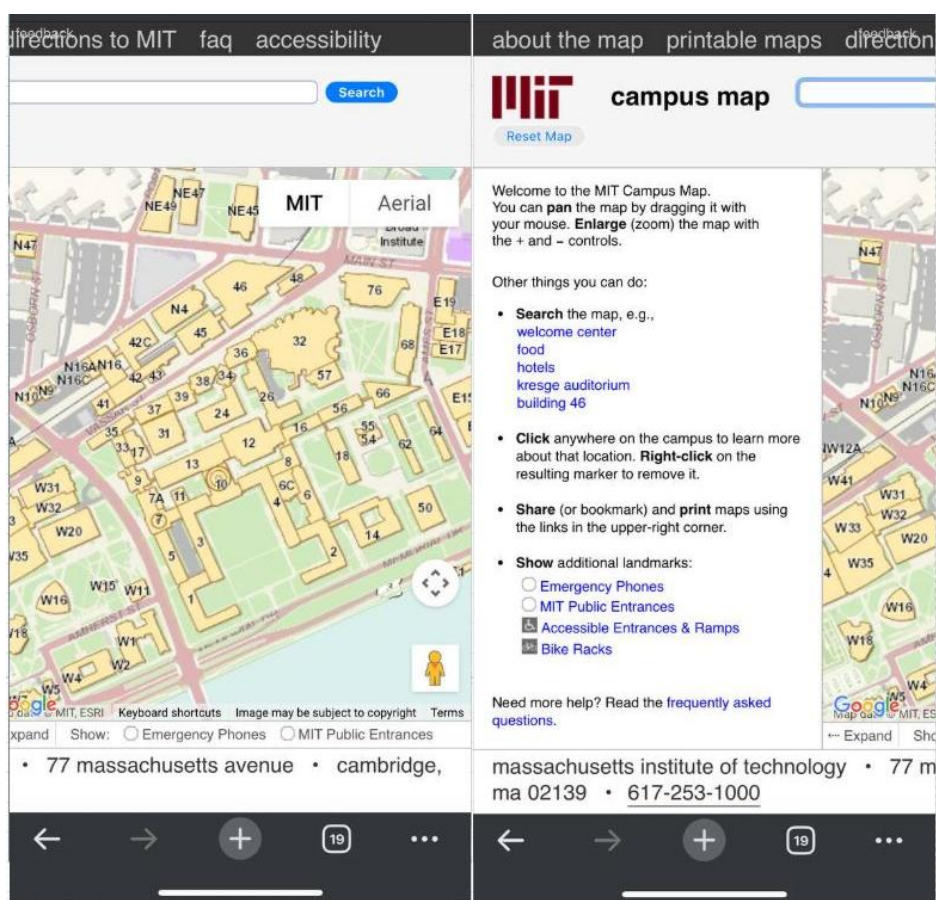


Рисунок 2 – Интерфейс сайта «Campus Map» от MIT

Преимущества: Глубокая интеграция с университетскими системами, Поэтажные планы, включая аудитории, лаборатории и офисы, Актуальные данные, соответствующие реальной инфраструктуре.

Недостатки: Отсутствие открытого исходного кода, Привязка к конкретной архитектуре и ИТ-инфраструктуре университета, Ограниченная возможность повторного использования другими организациями, не поддерживает работу без интернета.

1.2.3 Карта Университета Сиднея

Одной из наиболее развитых систем среди рассматриваемых является карта Университета Сиднея, интерфейс которой изображен на рисунке 3. Она поддерживает отображение нескольких этажей в каждом здании, предоставляет информацию о типах помещений, доступности, а также маршрут от одной точки до другой. Интерфейс переведён на несколько языков, включая английский и китайский. Однако для корректной работы карта требует высокой пропускной способности канала, что может затруднить использование в условиях нестабильного соединения.

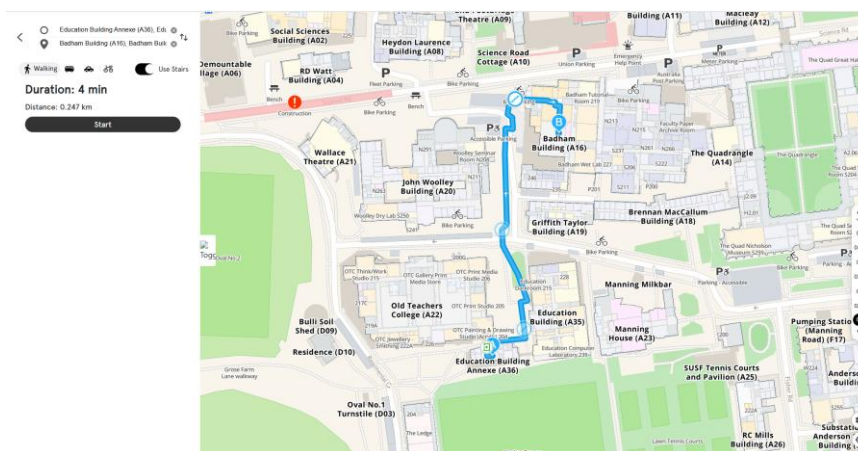


Рисунок 3 – Интерфейс сайта карты университета Сиднея

Достоинства: реализована поддержка поэтажной детализации внутри зданий, возможен выбор начальной и конечной точки маршрута с построением пути, имеется фильтрация по категориям, реализована мультиязычность интерфейса, отображаются доступные входы для маломобильных пользователей.

Недостатки: интерфейс перегружен визуальными элементами, что может затруднить восприятие, сложность первого взаимодействия: неочевидно, где

начать поиск, частичное отсутствие информации о внутренних помещениях некоторых объектов.

1.2.4 Карта Высшей Школы Экономики

Система навигации ВШЭ позволяет просматривать планы этажей, переходить между корпусами и выбирать конкретную аудиторию. Реализована фильтрация по функциональному назначению помещений и отображение краткой информации. Основной особенностью карты является ориентация на внутреннюю навигацию. Интерфейс изображен на рисунке 4.

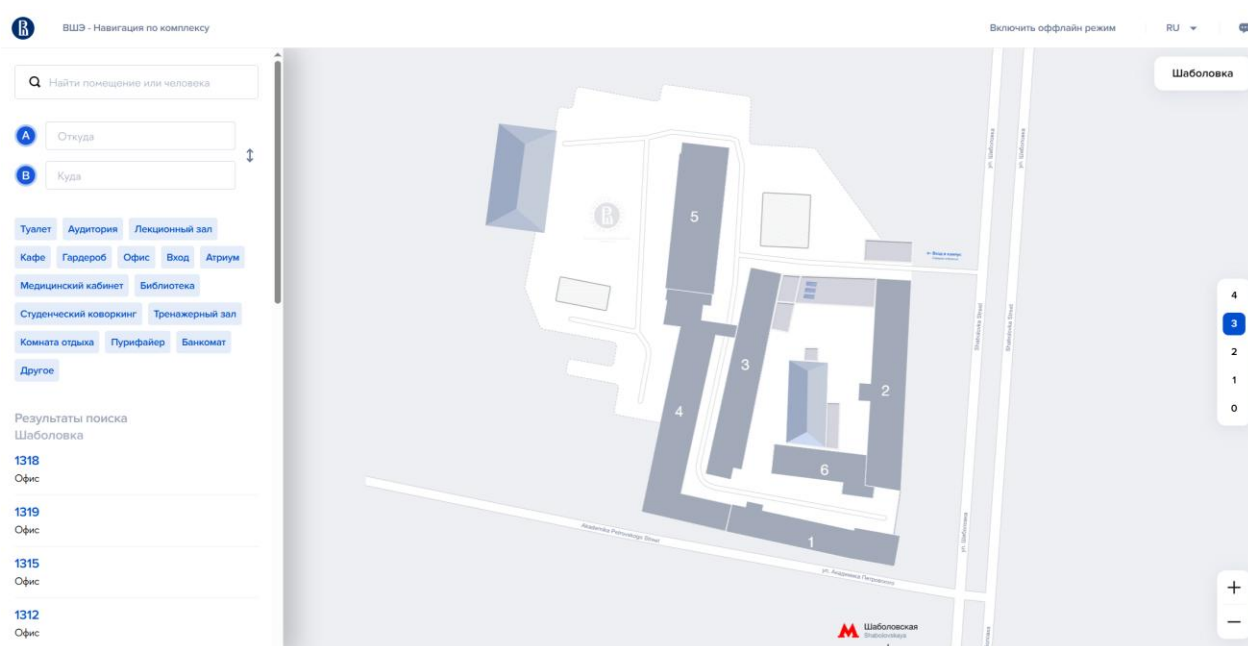


Рисунок 4 – Интерфейс сайта карты ВШЭ

Достоинства: реализована поэтажная навигация внутри учебных корпусов, отображаются аудитории, кабинеты и помещения по каждому этажу, имеется панель поиска с возможностью фильтрации по аудиториям; простой и понятный интерфейс.

Недостатки: интерфейс плохо адаптирован под мобильные устройства, отсутствует визуализация маршрутов между зданиями, не предоставляется расширенная информация по объектам, отсутствует поддержка смены языка интерфейса.

1.2.5 Карта Санкт–Петербургского государственного университета

В рамках анализа существующих решений, направленных на визуализацию университетской инфраструктуры, отдельное внимание заслуживает официальная карта Санкт-Петербургского государственного университета. Платформа реализует наглядное отображение зданий в виде масштабируемой интерактивной схемы. Карта охватывает несколько территориально разнесённых кампусов СПбГУ, что позволяет получить обобщённое представление о структуре университета. Интерфейс сервиса показан на рисунке 5.

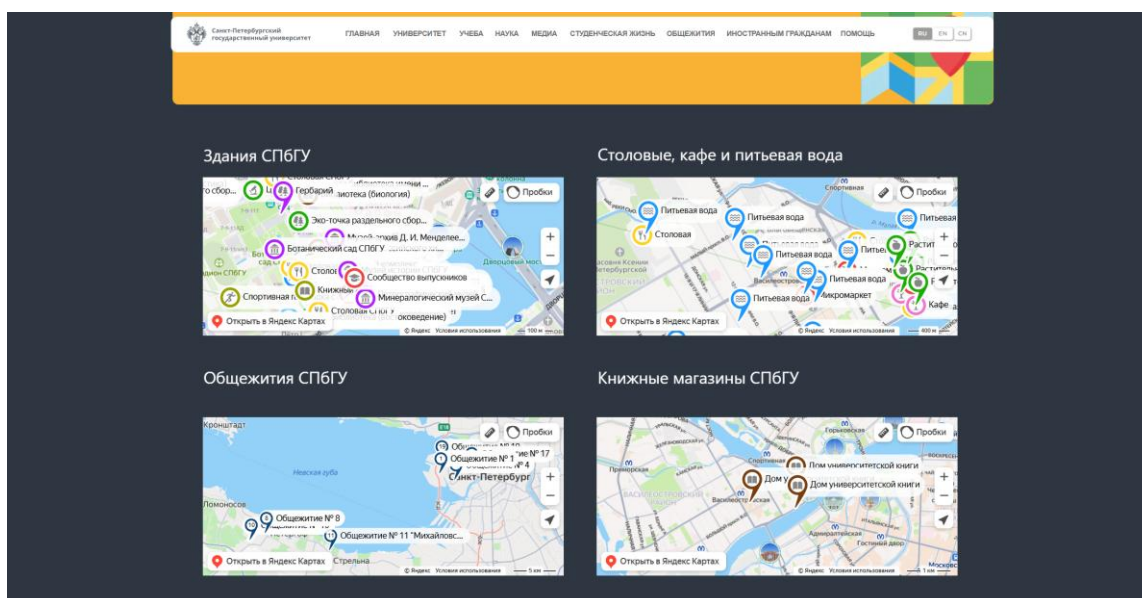


Рисунок 5 – Интерфейс сервиса карт СПбГУ

Достоинства:

- наличие масштабируемой схемы с подробной привязкой к районам города;
- возможность фильтрации зданий по типу и принадлежности к кампусу;
- структурированное представление информации в карточках объектов;
- реализованная поддержка нескольких кампусов на единой платформе.

Недостатки:

- ограниченность информации о помещениях внутри зданий;
- отсутствие дополнительных характеристик;
- частичное ограничение функционала при использовании на мобильных устройствах;
- отсутствие связи с внутренними подразделениями.

Таким образом, карта СПбГУ представляет собой пример функционального, но ориентированного на базовую навигацию решения, где основное внимание уделяется общей визуализации зданий. В отличие от проектируемого приложения «Интерактивная карта ПГУ», данный сервис не предусматривает глубокой детализации объектов и внутренней структуры помещений, а также не отображает параметры доступности и технического оснащения. Это позволяет сделать вывод о направленности решения на ориентирование в пространстве, а не на получение справочной информации о внутренней инфраструктуре.

1.2.6 Сравнение программных средств

Проведем сравнительный анализ рассмотренных выше программных средств, таких как онлайн карты следующих высших учебных заведений «МГУ», «MIT», «The University of Sydney», «ВШЭ», по ключевым функциям.

Для оценки приложений будет использоваться трехбалльная шкала соответствия. В таблице 1 представлены характеристики оценки.

Таблица 1 – Шкала баллов

Балл	Характеристика оценки
0	функция (качество) полностью отсутствует
1	существует частичная реализация функции (качества)
2	функция (качество) полностью реализована

Результаты сравнения программных средств интерактивных карт представлены в таблице 2.

Таблица 2 – Сравнение программных средств аналогичного назначения

Критерий \ Приложение	«МГУ»	«MIT»	«The University of Sydney»	«ВШЭ»	«СПбГУ»
1. Публичный доступ	2	2	2	2	2
2. Поэтажная навигация	0	0	2	2	0
3. Фильтрация по типам помещений	0	1	2	1	1
4. Поиск по объектам	0	1	2	1	2
5. Мобильная адаптация	0	2	1	0	0
6. Наличие дополнительных сведений	2	1	1	1	1
7. Наличие внешней навигации	2	2	2	0	2

Полученные результаты сравнения аналогов отображены в диаграмме, приведенной на рисунке 6.

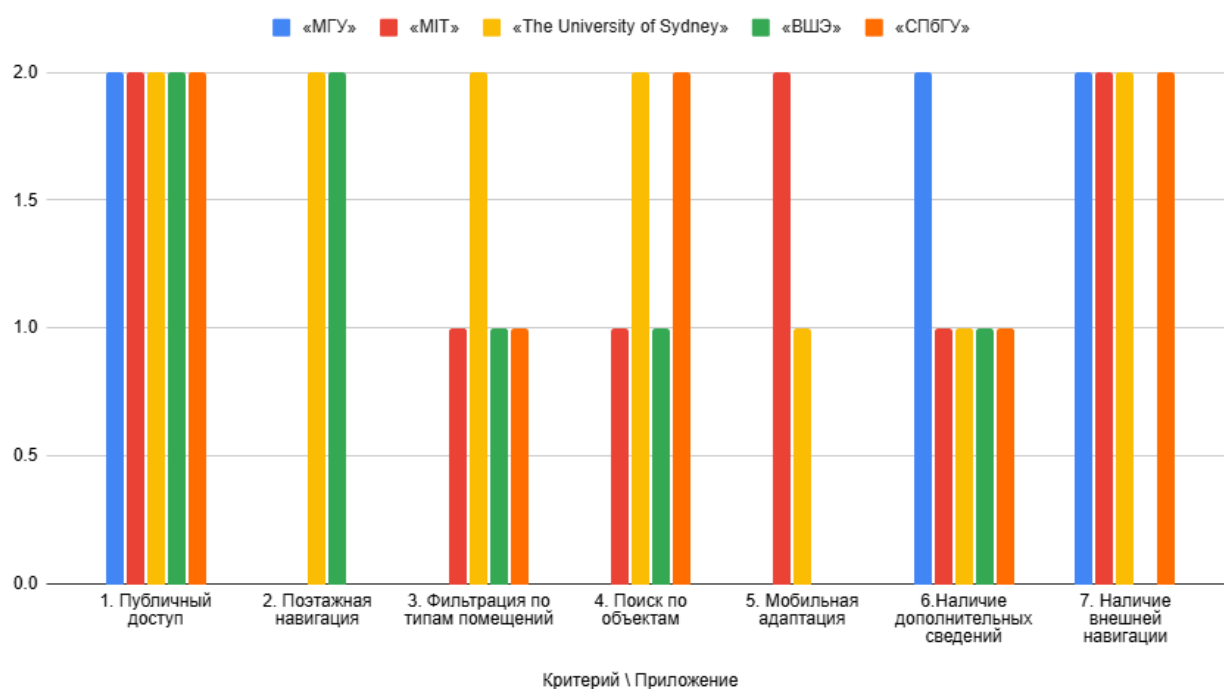


Рисунок 6 – Результаты сравнения аналогичных сервисов

Анализ показал, что ни одно из исследованных решений не обеспечивает комплексной реализации всех необходимых функций. Большинство систем акцентируют внимание либо на внешней схеме кампуса, либо на внутреннем размещении, но редко объединяют обе стороны навигации.

Кроме того, ограниченный функционал поиска и отсутствие адаптации под различные устройства существенно снижают удобство использования в реальных условиях. В отличие от указанных решений, разрабатываемое программное средство будет направлено на обеспечение:

- полноценной внутренней и внешней навигации;
- расширенной информации об объектах инфраструктуры;
- адаптации под мобильные устройства;
- фильтрации и контекстного поиска;
- учёта доступности объектов для маломобильных групп.

Таким образом, новое приложение нацелено на устранение функциональных и эргономических недостатков существующих систем и может быть использовано как основа для создания универсальной навигационной платформы в рамках образовательной среды ПГУ.

1.3 Анализ функциональных требований

В ходе работы были выявлены определенные функциональные требования:

- поиск зданий по названию;
- отображение карты с возможностью выбора объектов;
- фильтрация объектов по назначению;
- просмотр внутренней структуры объектов;
- получение информации об аудиториях;
- интерактивное взаимодействие;
- адаптация к различным устройствам.

Рассмотрим функциональные требования более подробно. Пользователь должен иметь возможность быстро найти нужный корпус по его названию. Интерфейс, способный поддерживать быстрый доступ к нужному элементу среды, минимизирует когнитивную нагрузку и упрощает навигацию [4].

Система должна предоставлять визуальное представление университетской территории с возможностью выбора зданий, просмотра их описания и координат. Это позволяет связать логическую структуру учебного заведения с её пространственной реализацией. Пространственная визуализация объектов обеспечивает пользователю более полное понимание структуры сложной среды [5]. Более того, доступ к плану этажей и списку помещений в выбранном здании поможет ориентироваться внутри корпусов и находить аудитории или службы. Доступ к структурированной информации о внутренних пространствах повышает эффективность перемещений и снижает неопределённость восприятия среды [6]. Отразим основные функциональные требования в диаграмме вариантов использования на рисунке 7.

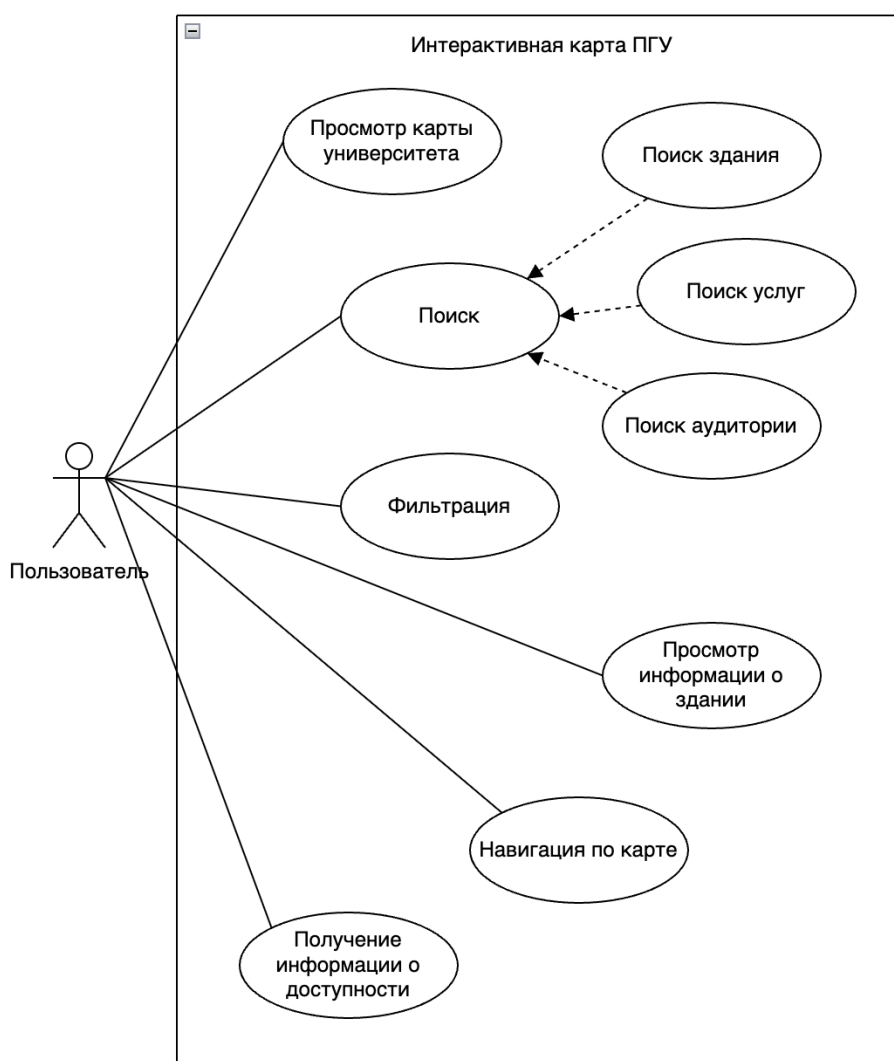


Рисунок 7 – Диаграмма вариантов использования

Ниже изложены ключевые характеристики основных бизнес-процессов системы. Описание сценария варианта использования «Найти здание по названию» приведено на рисунке 8.

Прецедент: Найти здание по названию
ID: 1
Краткое описание: Сервис отображает на карте здание, чье название указал пользователь
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Пользователь находится на главной странице приложения;
Основной поток событий: <ol style="list-style-type: none"> 1. Пользователь вводит часть названия корпуса в поле поиска; 2. Система предлагает список совпадений; 3. Пользователь выбирает нужное здание; 4. Карта автоматически переходит к выбранному объекту; 5. Отображается краткое описание здания.
Постусловие <ol style="list-style-type: none"> 1. Здание отображено на карте, пользователь получает доступ к его описанию.
Альтернативные потоки: При отсутствии совпадений выводится сообщение «Объект не найден»

Рисунок 8 – Сценарий варианта использования «Найти здание по названию»

Сценарий варианта использования «Отфильтровать здания по назначению» представлен на рисунке 9.

Прецедент: Отфильтровать здания по назначению
ID: 4
Краткое описание: Сервис отображает все здания соответствующие заданному пользователем типу
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: <ol style="list-style-type: none"> 1. Пользователь находится на главной странице приложения; 2. Открыта панель фильтрации на карте.
Основной поток событий: <ol style="list-style-type: none"> 1. Пользователь отмечает нужные категории (учебные, спортивные и др.); 2. Система выделяет соответствующие объекты; 3. Пользователь получает карту с выделением только отфильтрованных зданий.
Постусловие <ol style="list-style-type: none"> 1. Карта отображает только выбранные типы объектов.
Альтернативные потоки: Нет

Рисунок 9 – Сценарий варианта использования «Отфильтровать здания по назначению»

Сценарий варианта использования «Получить информацию об аудитории» представлен на рисунке 10.

Прецедент: Получить информацию об аудитории
ID: 7
Краткое описание: Сервис отображает информацию об искомом объекте
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Пользователь выбрал здание и перешел к его внутреннему содержанию;
Основной поток событий: <ol style="list-style-type: none"> 1. Пользователь выбирает аудиторию на карте; 2. Система отображает информации о помещении; 3. Пользователь изучает сведения (номер, назначение, вместимость и др.).
Постусловие Пользователь получил доступ к описанию интересующего помещения.
Альтернативные потоки: Нет

Рисунок 10 – Сценарий «Получить информацию об аудитории»

Таким образом, в процессе анализа функциональных требований были сформированы сценарии работы пользователя с системой. Некоторые из них были описаны с целью отображения логики взаимодействия между пользователем и системой.

1.4 Анализ нефункциональных требований

В процессе анализа были сформулированы следующие ключевые нефункциональные требования:

- доступность в любых условиях;
- время отклика;
- надёжность отображаемой информации;
- устойчивость к ошибкам;
- поддержка работы на различных устройствах;
- адаптивность интерфейса и корректное масштабирование интерфейсных элементов;

- интуитивность и простота использования;
- безопасность и устойчивость к внешнему вмешательству;
- расширяемость;
- визуальная целостность.

Рассмотрим нефункциональные требования более подробно. Система должна быть доступна всем категориям пользователей вне зависимости от используемого устройства и местоположения. Адаптивность интерфейса является условием универсальности цифровой среды [7]. При обращении к элементам интерфейса пользователь не должен испытывать длительных задержек. Быстрота реакции напрямую влияет на уровень восприятия и удовлетворённости работой системы. Все сведения, предоставляемые пользователю, должны быть актуальными, точными и непротиворечивыми. В особенности это касается информации о расположении зданий, назначении помещений и условиях доступности. При возникновении непредвиденных ситуаций система должна корректно обрабатывать ошибки, предоставляя пользователю понятные уведомления и не прерывая работу приложения.

Адаптивность интерфейса и корректное масштабирование интерфейсных элементов должны обеспечивать удобство взаимодействия в любых условиях. Кроссплатформенность является необходимым элементом современных информационных систем, ориентированных на массового пользователя [8].

Система должна иметь возможность будущего развития, включая добавление новых категорий объектов, расширение функционала поиска, локализацию интерфейса и интеграцию с другими цифровыми платформами. Гибкость архитектуры цифровых систем определяет их жизнеспособность в условиях изменяющихся требований [9].

Таким образом, нефункциональные требования представляют собой основу для оценки качества будущего программного продукта и формируют критерии, по которым будет определяться его эффективность и соответствие ожиданиям пользователей.

1.5 Планирование разработки сервиса интерактивной карты ПГУ

Для оценки временных затрат, а также упорядоченного планирования процесса разработки системы визуальной навигации на территории вуза была составлена диаграмма Ганта. Она показана на рисунке 11.



Рисунок 11 – Диаграмма Ганта

Данный график наглядно отражает продолжительность и последовательность выполнения работ, а также их структуру по этапам и задачам. Процесс разработки программного обеспечения включает четыре основных этапа, каждый из которых объединяет логически взаимосвязанные подзадачи. Структура проектной деятельности сформирована с учётом типовой последовательности этапов разработки программных систем и ориентирована на эффективную реализацию как технической, так и пользовательской составляющей приложения. В рамках разрабатываемого сервиса выделим следующие этапы:

- анализ предметной области и постановка задачи;
- проектирование программного обеспечения;
- реализация программных средств;
- контроль качества программного обеспечения.

Заключительный этап разработки направлен на проверку корректности функционирования системы. Проводятся функциональные и нефункциональные испытания, оцениваются пользовательские сценарии, вносятся корректировки на основе выявленных замечаний. Также осуществляется финальная сборка и подготовка проекта к представлению. Краткое описание сроков на выполнение задач, входящих в каждый из этапов, представлено на рисунке 12.

	Начало проекта	1 day	Tue 08.04.25	Tue 08.04.25
	Анализ предметной области и постановка задачи на разработку программных средств для визуальной навигации на территории ВУЗа	8 days	Wed 09.04.25	Fri 18.04.25
	Анализ предметной области визуальной навигации на территории ВУЗа	1 day	Wed 09.04.25	Wed 09.04.25
 	Обзор и анализ программных средств визуальной навигации ВУЗов	1 day	Fri 11.04.25	Fri 11.04.25
 	Сравнение программных средств визуальной навигации ВУЗов	1 day	Fri 11.04.25	Fri 11.04.25
	Постановка задачи на разработку программных средств	1 day	Mon 14.04.25	Mon 14.04.25
	Анализ функциональных требований	2 days	Tue 15.04.25	Wed 16.04.25
 	Анализ нефункциональных требований	1 day	Tue 15.04.25	Tue 15.04.25
 	Планирование разработки и оценка бюджета	3 days	Wed 16.04.25	Fri 18.04.25
	Проектирование программных средств визуальной навигации ВУЗа	7 days	Mon 21.04.25	Tue 29.04.25
	Архитектура приложения	1 day	Mon 21.04.25	Mon 21.04.25
	Проектирование структур данных	1 day	Tue 22.04.25	Tue 22.04.25
	Разработка концептуальной модели базы данных	1 day	Wed 23.04.25	Wed 23.04.25
	Разработка логической модели базы данных	1 day	Thu 24.04.25	Thu 24.04.25
	Проектирование пользовательского интерфейса	3 days	Fri 25.04.25	Tue 29.04.25
	Реализация программных средств визуальной навигации ВУЗа	17 days	Wed 30.04.25	Thu 22.05.25
	Реализация базы данных	2 days	Wed 30.04.25	Mon 05.05.25
	Реализация серверной части приложения	3 days	Tue 06.05.25	Mon 12.05.25
	Реализация клиентской части приложения	5 days	Tue 13.05.25	Mon 19.05.25
	Реализация интерфейса приложения	3 days	Tue 20.05.25	Thu 22.05.25
	Контроль качества программного обеспечения	5 days	Fri 23.05.25	Thu 29.05.25
	Обоснование выбора режима тестирования и метрик кода	2 days	Fri 23.05.25	Mon 26.05.25
	Функциональное тестирование	1 day	Tue 27.05.25	Tue 27.05.25
	Нефункциональное тестирование	1 day	Wed 28.05.25	Wed 28.05.25
	Оценка качества программного кода	1 day	Thu 29.05.25	Thu 29.05.25
	Оформление отчёта	5 days	Fri 30.05.25	Thu 05.06.25
	Оформление пояснительной записки	5 days	Fri 30.05.25	Thu 05.06.25
	Конец проекта	1 day	Wed 04.06.25	Wed 04.06.25

Рисунок 12 – Сроки выполнения задач

На рисунке отображены начало и завершение каждой задачи, а также последовательные зависимости между этапами разработки. В процессе планирования также была произведена предварительная оценка затрат на реализацию проекта. Для этого каждой задаче был назначен соответствующий

ресурс, включающий в себя трудозатраты и затраты на программные средства. Расчёт стоимости задействованных ресурсов представлен на рисунке 13.

Аналитик	127,00 ₽/h	200,00 ₽/h
Программист	230,00 ₽/h	350,00 ₽/h
Дизайнер	175,00 ₽/h	250,00 ₽/h
Тестировщик	150,00 ₽/h	350,00 ₽/h
Интернет		
Компьютер	70 000,00 ₽	

Рисунок 13 – Стоимость задействованных ресурсов

Итоговая сумма затрат представлена на рисунке 14.

Начало проекта	1 840,00 ₽
Анализ предметной области и постановка задачи на разработку программных средств для визуальной навигации на территории ВУЗа	10 160,00 ₽
Анализ предметной области визуальной навигации на территории ВУЗа	1 016,00 ₽
Обзор и анализ программных средств визуальной навигации ВУЗов	1 016,00 ₽
Сравнение программных средств средств визуальной навигации ВУЗов	1 016,00 ₽
Постановка задачи на разработку программных средств	1 016,00 ₽
Анализ функциональных требований	2 032,00 ₽
Анализ нефункциональных требований	2 032,00 ₽
Планирование разработки и оценка бюджета	2 032,00 ₽
Проектирование программных средств визуальной навигации ВУЗа	13 332,00 ₽
Архитектура приложения	1 840,00 ₽
Проектирование структур данных	1 840,00 ₽
Разработка концептуальной модели базы данных	1 016,00 ₽
Разработка логической модели базы данных	1 016,00 ₽
Проектирование пользовательского интерфейса	7 620,00 ₽
Реализация программных средств визуальной навигации ВУЗа	36 620,00 ₽
Реализация базы данных	5 180,00 ₽
Реализация серверной части приложения	5 520,00 ₽
Реализация клиентской части приложения	16 200,00 ₽
Реализация интерфейса приложения	9 720,00 ₽
Контроль качества программного обеспечения	6 000,00 ₽
Обоснование выбора режима тестирования и метрик кода	2 400,00 ₽
Функциональное тестирование	1 200,00 ₽
Нефункциональное тестирование	1 200,00 ₽
Оценка качества программного кода	1 200,00 ₽
Оформление отчёта	5 080,00 ₽
Оформление пояснительной записки	5 080,00 ₽
Конец проекта	0,00 ₽

Рисунок 14 – Итоговая сумма затрат

Согласно проведённой оценке, общая сумма предполагаемых финансовых затрат составила 73032 рубля, а суммарная продолжительность проекта – 61 календарный день. Представленная структура планирования позволяет обеспечить контролируемое и последовательное выполнение всех этапов проекта, а также оценить трудоёмкость и стоимость разработки с достаточной степенью достоверности.

2 Проектирование программных средств «Интерактивная карта ПГУ»

2.1 Архитектура приложения

Разрабатываемая система строится на принципах многоуровневого подхода, при котором логика приложения разделена на три взаимосвязанных слоя: Уровень представления, обеспечивающий взаимодействие с пользователем через визуальный интерфейс. Логический уровень, отвечающий за обработку запросов и управление данными. Уровень хранения, включающий структуры постоянного хранения информации об объектах кампуса. Уровни архитектуры разрабатываемого приложения представлены на рисунке 15.



Рисунок 15 – Уровни архитектуры приложения

Такая архитектура обеспечивает слабую связанность компонентов, масштабируемость и независимость функциональных блоков. Будет реализована «трёхуровневая архитектура», в которой каждый слой выполняет строго определённые задачи. Для взаимодействия между слоями используются согласованные интерфейсы. Визуальная часть приложения организует обработку действий пользователя, а серверная часть обеспечивает формирование ответов и доступ к данным. Физическая реализация архитектуры будет построена с учётом развёртывания в изолированной среде с разделением по контейнерам, что в будущем позволит гибко управлять средой исполнения, повысит устойчивость и облегчит техническое обслуживание. Многослойная

структура позволяет изменять отдельные компоненты системы без необходимости переработки всего программного комплекса [10].

2.2 Проектирование логической структуры данных

Информационная модель системы отражает реальную структуру университетской среды. Она описывает ключевые сущности: здания, аудитории, типы помещений, характеристики доступности, а также служебные элементы, такие как координаты, изображения и описание. Концептуальная модель базы данных изображена на рисунке 16.

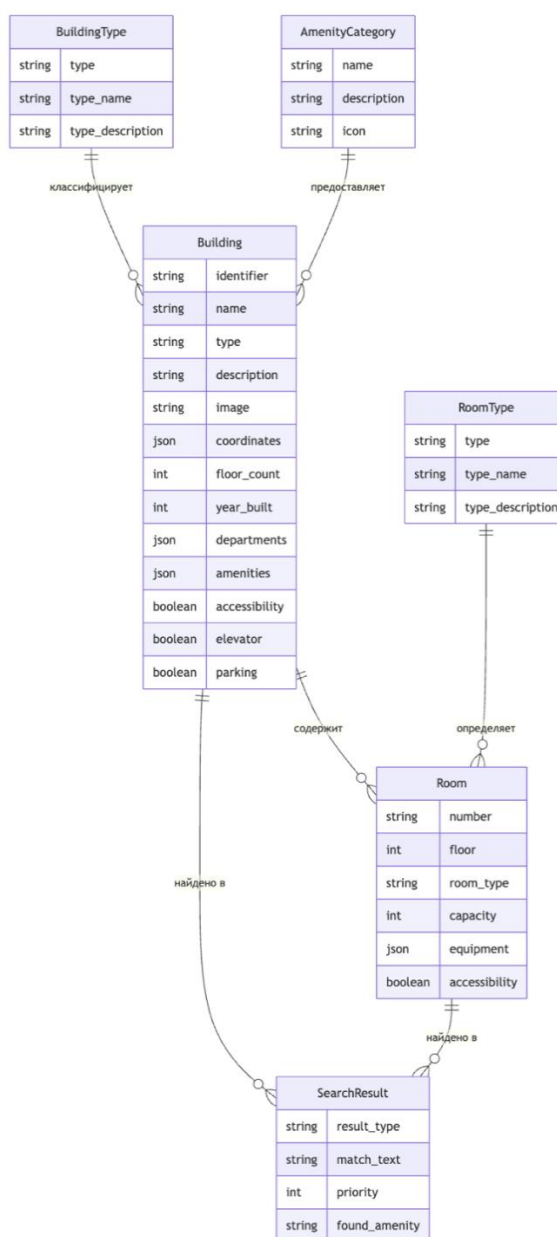


Рисунок 16 – Концептуальная модель базы данных

В рамках проектирования были выделены следующие основные сущности:

Здание – объект, имеющий уникальный идентификатор, координаты на карте, тип, описание, число этажей, год постройки, степень доступности, наличие лифта и парковки;

Аудитория – помещение в здании, характеризующееся номером, типом, вместимостью, принадлежностью к этажу и техническим оснащением; Тип помещения – классификатор, определяющий назначение;

Удобства – элементы инфраструктуры, обеспечивающие комфорт и доступность;

Оборудование – объекты материально-технического оснащения, закреплённые за конкретными помещениями;

Подразделение – кафедры, институты и иные единицы, размещённые в зданиях. Связи между сущностями реализуют типовые отношения: одно здание содержит множество помещений, может включать несколько подразделений и быть оснащено разными удобствами. Помещения, в свою очередь, могут быть связаны с несколькими единицами оборудования.

Также система предусматривает возможность расширения модели за счёт дополнительных атрибутов и связей. Это позволяет адаптировать систему под новые задачи, например, отображение расписания, зон Wi-Fi или камер видеонаблюдения. Назначение основных сущностей описано в таблице 3.

Таблица 3 – Основные сущности сервиса

№ п/п	Название сущности	Назначение сущности
1	Здание (Building)	Хранение информации об объектах инфраструктуры университета
2	Аудитория (Room)	Представление помещений внутри зданий
3	Тип помещения (RoomType)	Классификация назначений помещений
4	Удобство (Amenity)	Отражение особенностей зданий

Продолжение таблицы 3

5	Подразделение (Department)	Размещение структурных единиц: институтов, кафедр
6	Оборудование (Equipment)	Хранение сведений об оснащении помещений
7	Связь здание–удобство	Отражение состава удобств, закреплённых за зданиями
8	Связь аудитория–оборудование	Описание оборудования, установленного в конкретных помещениях

Теперь более подробно рассмотрим структурные части каждой из основных сущностей, используемых в рамках работы сервиса. Эти сущности необходимы для полноценного функционирования интерактивной карты Пензенского Государственного Университета. Подробная структура сущности, описывающей объект инфраструктуры университета, приведена в таблице 4. Поля таблицы охватывают характеристики, необходимые для отображения и идентификации здания на карте.

Таблица 4 – Структура сущности «Здание»

№ п/п	Наименование поля	Описание	Тип данных	Обязательное поле	Пример
1	id	Уникальный идентификатор здания	Строка (UUID)	Да	BLD-001
2	name	Название	Строка	Да	Главный корпус
3	type	Назначение	Строка	Да	учебное
4	description	Краткое описание	Текст	Нет	Здание инженерного факультета
5	x_coordinate	Координата по оси X	Целое число	Да	340
6	y_coordinate	Координата по оси Y	Целое число	Да	205
7	floor_count	Количество этажей	Целое число	Да	5
8	year_built	Год постройки	Целое число	Нет	1987

Продолжение таблицы 4

9	accessible	Доступность для МГН	Логическое	Да	true
10	has_elevator	Наличие лифта	Логическое	Нет	true
11	has_parking	Наличие парковки	Логическое	Нет	false

Таблица 5 отражает структуру сущности, описывающей помещение внутри здания. Атрибуты данной сущности обеспечивают хранение сведений, позволяющих различать аудитории по номеру, функциональному назначению и другим параметрам.

Таблица 5 – Структура сущности «Аудитория»

№ п/п	Наименование поля	Описание	Тип данных	Обязательное поле	Пример
1	id	Идентификатор помещения	Целое число	Да	102
2	building_id	Идентификатор здания	Строка	Да	BLD-001
3	number	Номер аудитории	Строка	Да	305
4	floor	Этаж	Целое число	Да	2
5	type	Назначение	Строка	Да	лекционная
6	capacity	Вместимость	Целое число	Нет	40
7	accessible	Доступность	Логическое	Нет	true

Классификация помещений по назначению описывается отдельной сущностью, структура которой представлена в таблице 6. Это обеспечивает корректное разграничение по видам использования пространства.

Таблица 6 – Структура сущности «Тип помещения»

№ п/п	Наименование поля	Описание	Тип данных	Пример
1	value	Код назначения	Строка	lab
2	label	Наименование типа	Строка	Лаборатория
3	description	Подробное описание	Текст	Учебная лаборатория для занятий

В таблице 7 представлена структура сущности, отражающей доступные в здании элементы, повышающие комфорт и функциональность. Такие элементы учитываются при формировании сведений о доступности объектов.

Таблица 7 – Структура сущности «Удобство»

№ п/п	Наименование поля	Описание	Тип данных	Пример
1	id	Идентификатор	Целое число	1
2	name	Наименование подразделения	Строка	Пандус
3	description	Описание	Текст	Для доступа МГН
4	icon	Иконка	Строка	/img/ramp.svg

Сущность, описывающая внутреннюю организационную структуру университета, представлена в таблице 8. Её атрибуты позволяют связать подразделения с конкретными зданиями и уточнить контактную информацию.

Таблица 8 – Структура сущности «Подразделение»

№ п/п	Наименование поля	Описание	Тип данных	Пример
1	id	Идентификатор	Целое число	5
2	name	Наименование подразделения	Строка	Институт права
3	description	Описание	Текст	Юридическое образование
4	building_id	Привязка к зданию	Строка	BLD-002
5	contact_info	Контактная информация	Строка	law@pgu.ru

Сведения о техническом оснащении помещений хранятся в виде отдельной сущности, структура которой отражена в таблице 9. Поля данной таблицы охватывают ключевые характеристики, необходимые для описания оборудования.

Таблица 9 – Структура сущности «Оборудование»

№ п/п	Наименование поля	Описание	Тип данных	Пример
1	id	Идентификатор	Целое число	3
2	name	Название	Строка	Проектор
3	category	Категория	Строка	Аудиовизуальное
4	description	Описание назначения	Текст	Для презентаций

При построении структуры данных важно обеспечить как логическую непротиворечивость, так и гибкость описания объектов. Что логическая модель данных служит концептуальной основой для проектирования структуры хранения, обеспечивая независимость предметной области от особенностей реализации [11].

2.3 Проектирование пользовательского интерфейса

Пользовательский интерфейс будет разработан с учётом требований адаптивности, визуальной целостности и удобства восприятия. Все основные элементы управления сгруппированы и интуитивно понятны при первом использовании. Основными элементами интерфейса являются:

Интерактивная карта – центральный элемент, отображающий территорию университета, координаты зданий и их визуальное представление. Пользователь может перемещать карту, изменять масштаб, выбирать объекты.

Поисковая строка – обеспечивает быстрый доступ к нужному зданию по названию или ключевому слову. Отображается список совпадений.

Панель фильтрации – позволяет выбирать категории зданий, скрывая остальные объекты с карты.

Карточка здания – всплывающий блок с описанием: назначение, адрес, оборудование, этажность, наличие удобств, сведения об аудиториях и подразделениях.

Диаграмма, показывающая части и слои пользовательского интерфейса, используемая в процессе проектирования приложения представлена на рисунке 17.

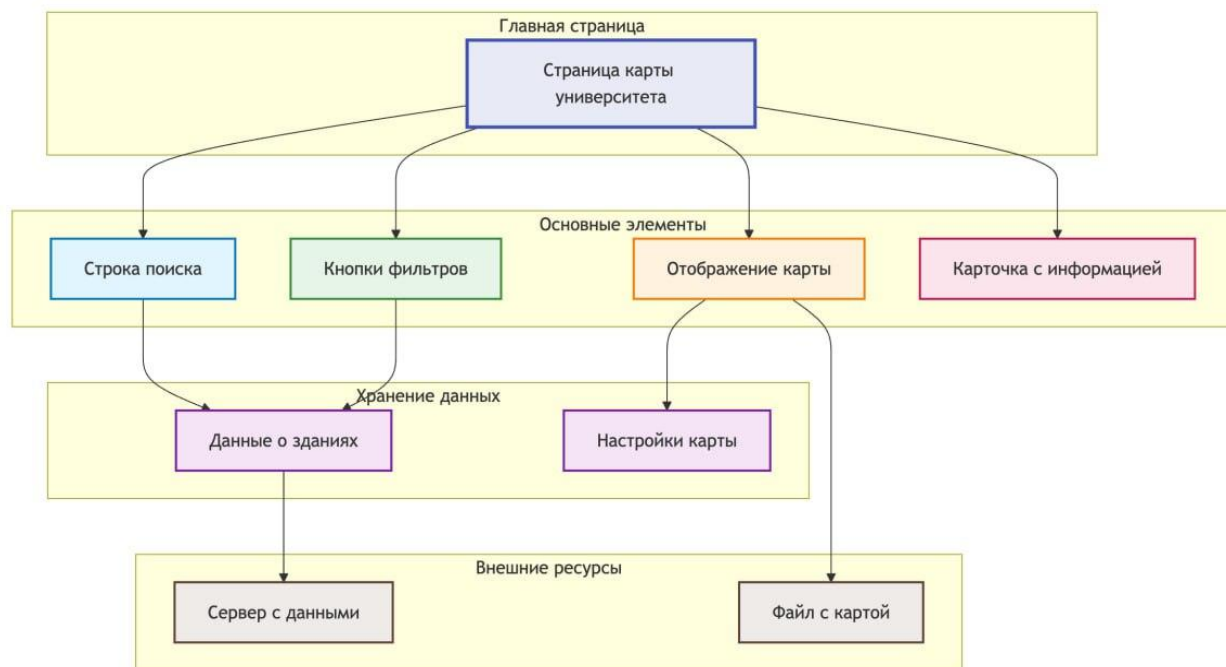


Рисунок 17 – Части пользовательского интерфейса

Интерфейс должен формировать у пользователя ощущение прозрачности взаимодействия: система воспринимается как продолжение действия, а не как препятствие [12]. Прототип пользовательского интерфейса изображен на рисунке 18.

Функциональные элементы:

- 1 – Поисковая строка;
- 2 – Панель фильтрации;
- 3 – Интерактивная карта;
- 4 – Кнопки изменения масштаба карты;
- 5 – Кнопки навигации интерфейса;
- 6 – Карточка здания;
- 7 – Кнопка рассмотрения внутреннего строения выбранного здания.

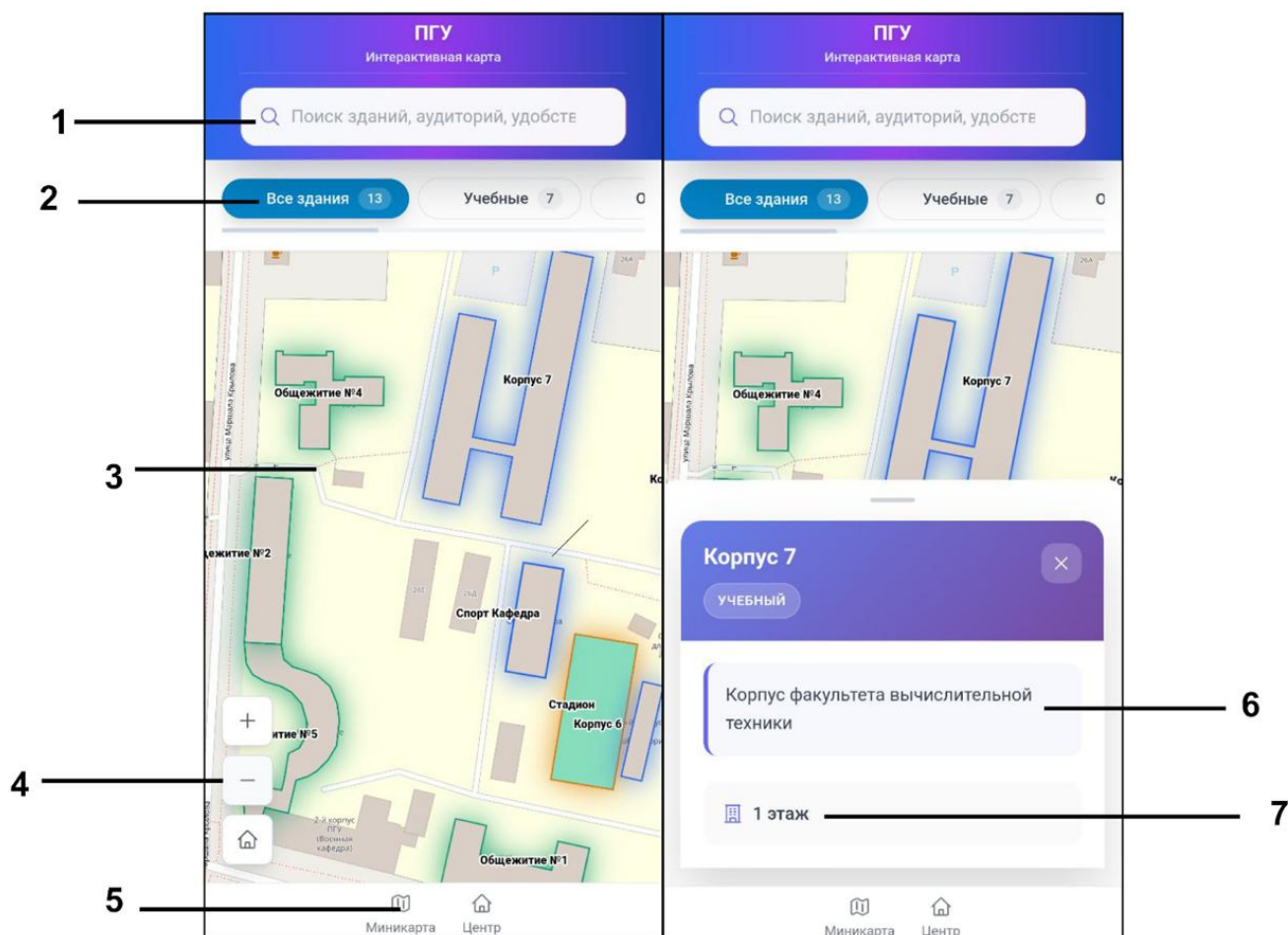


Рисунок 18 – Графический интерфейс приложения

В результате анализа и проектирования системы «Интерактивная карта ПГУ» были сформированы теоретические и практические основания для реализации функционального, устойчивого и визуально доступного программного продукта.

Основное внимание на этапе проектирования было сосредоточено на формировании архитектуры, логической модели данных и требований к интерфейсному взаимодействию.

Таким образом, проектирование системы завершено формированием полной логической и визуальной модели, на основе которой реализуется устойчивая, функциональная и удобная в использовании система пространственной навигации. Полученные решения обеспечивают основу для последующей реализации всех компонентов программного средства.

3 Реализация программных средств «Интерактивная карта ПГУ»

Разработка информационной системы представляет собой комплексную деятельность, включающую реализацию проектных решений, формализованных в ходе этапов анализа и моделирования. Основной целью данного этапа является переход от абстрактного описания архитектуры и логической структуры к конкретному исполнению функций, направленных на решение задач пользователей. В процессе реализации программных средств «Интерактивная карта ПГУ» особое внимание уделено качественным характеристикам: надёжности, наглядности, доступности и адаптивности. Система предназначена для широкого круга пользователей, включая студентов, преподавателей, административный персонал и посетителей, и потому её реализация ориентирована не только на функциональность, но и на восприятие, интуитивность и устойчивость в работе.

3.1 Реализация архитектуры серверной части приложения

Реализация системы опирается на трёхуровневую структуру, данный подход предполагает разграничение компонентов по функциям: взаимодействие с пользователем, обработка запросов и хранение информации. Такая организация обусловлена необходимостью повышения гибкости, масштабируемости и упрощения сопровождения. Архитектура с разделением на слои позволяет изолировать функциональные изменения, снизить взаимозависимость и обеспечить управляемость процессов разработки [13].

На рисунке 19 отображена фактическая структура приложения, данная схема помогает более наглядно представить каждый слой логической модели сервиса. Данная архитектурная диаграмма наглядно демонстрирует, как пользовательский запрос проходит путь от интерфейса до базы данных и обратно. Она подчёркивает разделение логики между клиентской частью и серверной, а также показывает, как обеспечивается масштабируемость и модульность системы.

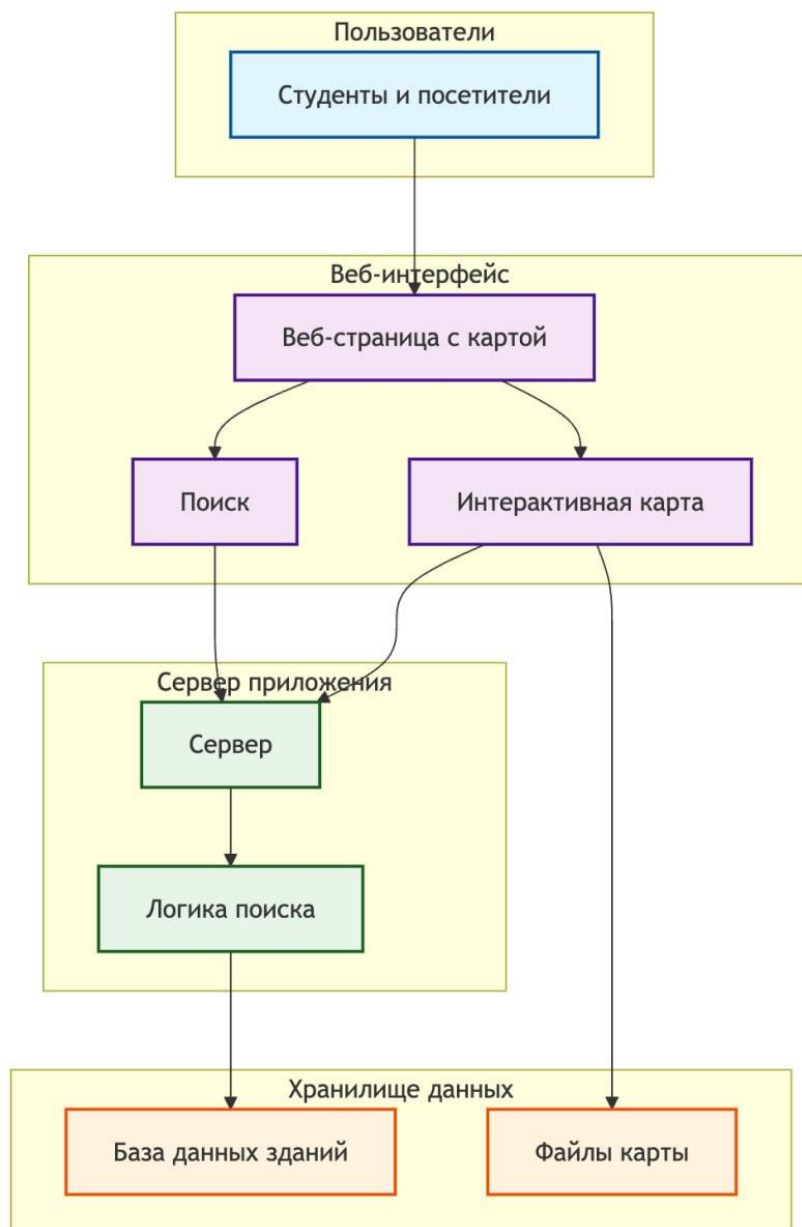


Рисунок 19 – Архитектура приложения

Благодаря такому подходу каждый компонент может дорабатываться независимо от других, что существенно упрощает модернизацию системы и внедрение новых функций. Следует отметить, что информационные системы в условиях образовательной среды должны учитывать специфику предметной области: разнообразие зданий, структурных подразделений, назначений помещений, а также необходимость учёта доступности для различных групп пользователей.

Важным аспектом этапа реализации является обоснование структуры программной части проекта. Система «Интерактивная карта ПГУ» разрабатывалась как многокомпонентное программное средство, реализующее принципы логического разделения функций.

Программная структура включает следующие компоненты:

Модуль представления данных отвечает за визуальное отображение карты, зданий, помещений и сопутствующей информации. Он реализует пользовательский интерфейс, обрабатывает действия пользователя, а также отображает результат в наглядной форме.

Модуль взаимодействия с данными обеспечивает получение сведений из хранилища, фильтрацию, выборку по запросу пользователя. Этот компонент принимает входные параметры, обрабатывает их и возвращает результат в форматированной форме.

Модуль конфигурации и инициализации определяет начальные параметры системы, устанавливает связи между сущностями, формирует наборы данных для первоначальной загрузки, обеспечивает предварительное размещение объектов на карте.

На рисунке 20 представлена диаграмма классов, отражающая структуру и взаимосвязи между основными компонентами объектной модели системы «Интерактивная карта ПГУ». Она иллюстрирует архитектуру приложения, ориентированную на обработку информации о зданиях, аудиториях и осуществление поисковых операций.

Диаграмма классов предназначена для формализации и визуализации логики взаимодействия между сущностями системы. Она позволяет описать внутреннюю структуру объектов, хранящих данные, показать, какие компоненты используют другие, определить, какие классы создают экземпляры других объектов, наглядно представить методы, которые доступны в каждом классе и реализуют прикладную логику. Вся система реализована с опорой на концепцию модульности, что обеспечивает независимость компонентов.

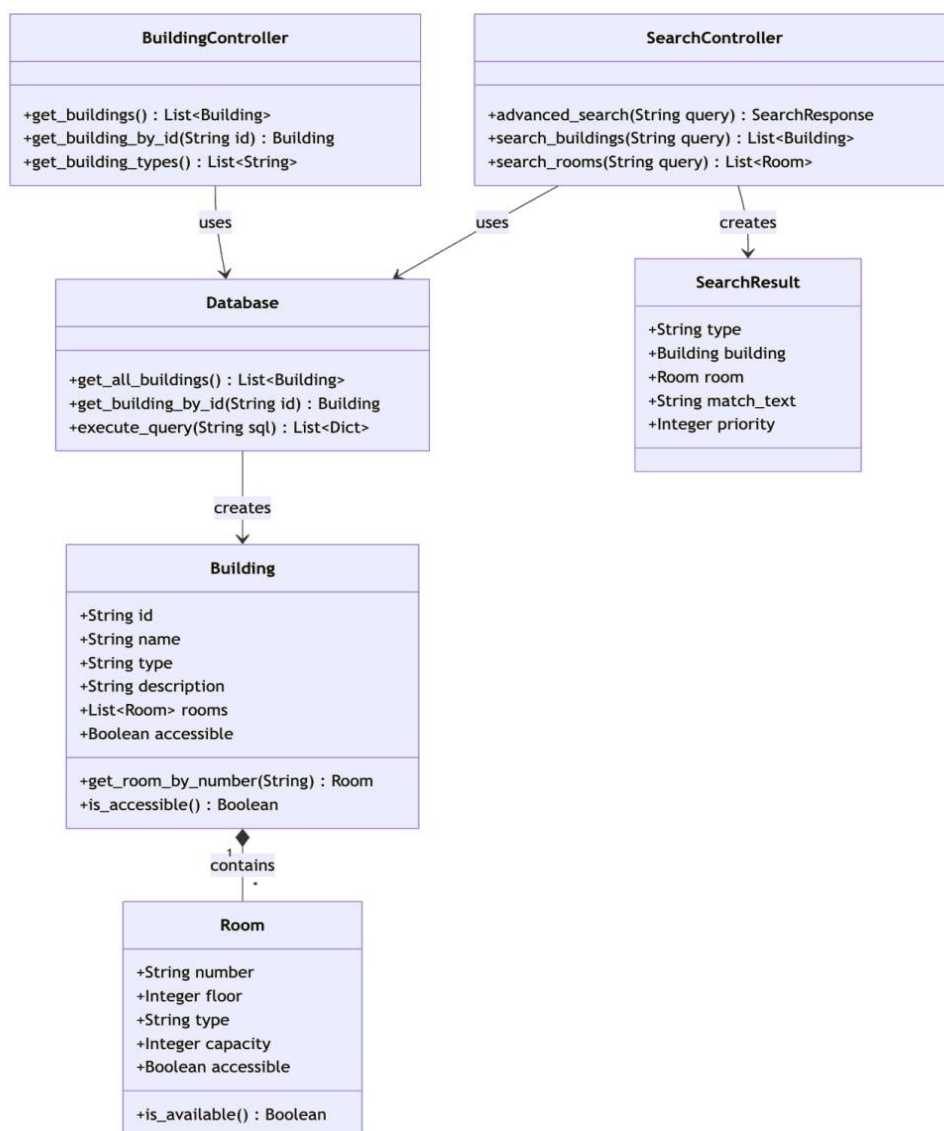


Рисунок 20 – Диаграмма классов реализованного приложения

Взаимодействие пользователя начинается с загрузки карты, отображающей территорию кампуса. На карту наносятся маркеры зданий, данные о которых берутся из заранее подготовленных информационных структур.

3.2 Реализация базы данных

База данных проекта реализована с использованием встроенной файловой системы хранения. Её основное назначение – обеспечить централизованное и структурированное хранение информации о зданиях, их характеристиках и связанных сущностях.

Вся информация сохраняется в единственной таблице, структура которой была определена на этапе проектирования. При первом запуске системы происходит автоматическая инициализация базы.

Все поля в таблице соответствуют логической модели: идентификатор, название, тип, описание, координаты, вложенные данные о подразделениях, удобствах и помещениях.

Структуры departments, amenities и rooms сохраняются в сериализованном формате, что позволяет гибко расширять структуру без модификации схемы таблицы. После создания таблицы в неё загружается начальный демонстрационный набор данных, взятый из файла с описанием объектов Пензенского государственного университета. Загруженные данные предварительно проверяются на предмет повторов, чтобы избежать дублирования.

Доступ к данным реализован через отдельные методы. Для получения всех зданий с фильтрацией по ключевому слову или типу применяется динамически формируемый SQL-запрос.

В рамках проекта также реализована вспомогательная функция для получения статистики по типам зданий, которая используется для визуального представления диаграммы распределения. Таким образом, база данных представляет собой устойчивый, расширяемый и внутренне согласованный компонент. Её структура и методы реализации позволяют гибко управлять информацией, обеспечивая полную поддержку всех функций пользовательского интерфейса. Каждая запись в базе представляет собой объект с фиксированным набором атрибутов: идентификатор, наименование, координаты, назначение, этажность, год постройки, доступность, наличие оборудования и связи с подразделениями. Для хранения вложенных структур используются текстовые.

Система поддерживает получение списка всех зданий с возможностью фильтрации и страничного отображения. Пользователь может указать ключевое слово для поиска и тип интересующих зданий. Запрос формируется динамически, а выборка ограничивается заданным числом объектов на странице.

3.3 Реализация пользовательского интерфейса

Пользовательский интерфейс системы «Интерактивная карта ПГУ» разработан с применением современных средств построения одностраничных приложений.

Его основная задача – обеспечить удобный, понятный и отзывчивый доступ к информации об объектах университетской инфраструктуры.

Интерфейс реализован в виде набора визуальных компонентов, каждый из которых отвечает за отдельную область функциональности. Основные компоненты пользовательского интерфейса приложения изображены на рисунке 21.

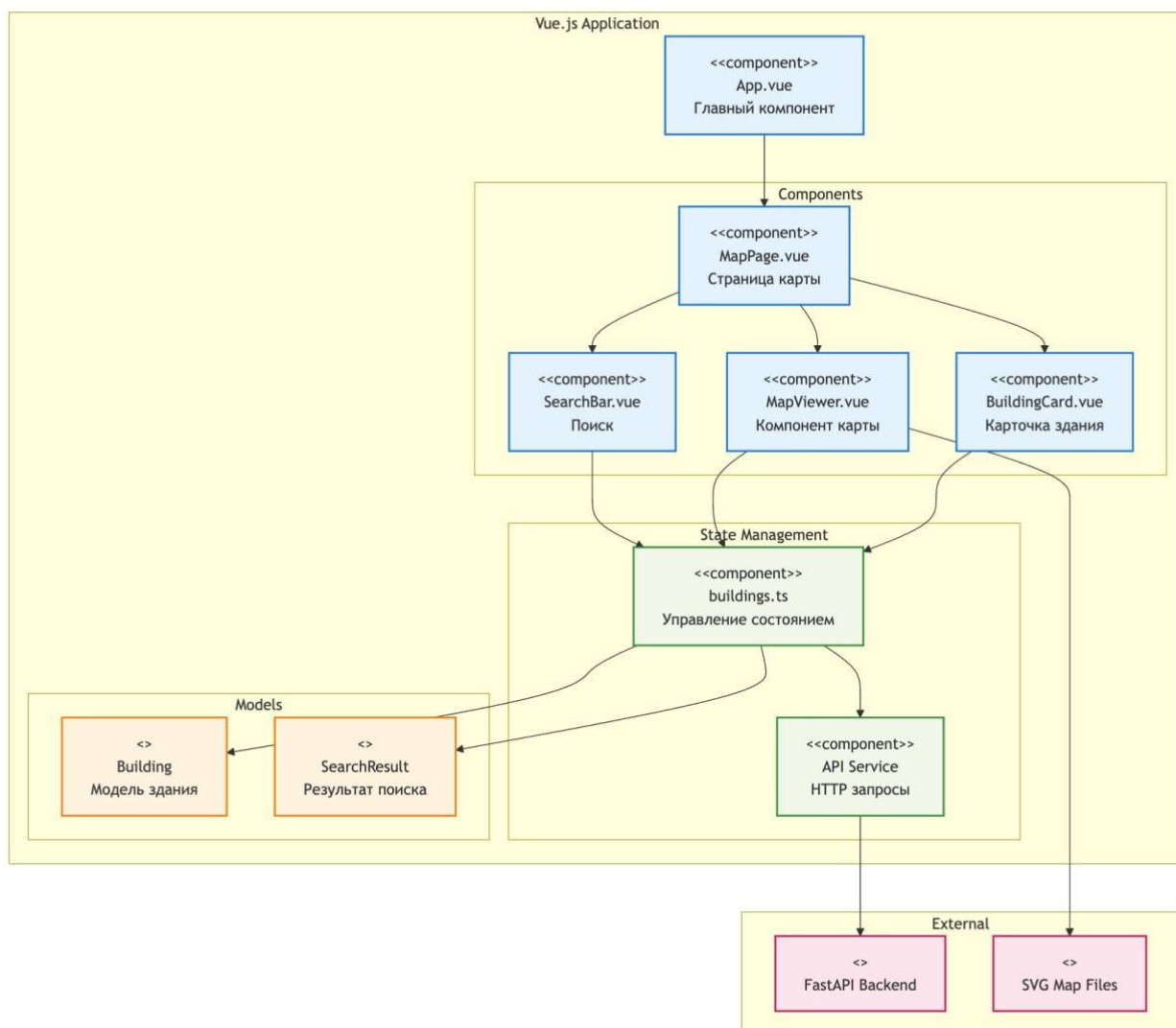


Рисунок 21 – Диаграмма компонентов разрабатываемого приложения

В частности, основным элементом взаимодействия является карточка здания, отображающаяся при выборе маркера на карте. Она содержит структурированную информацию: название объекта, его назначение, параметры доступности и список аудиторий. Карточка здания формируется динамически на основе полученных данных. Пример реализации функционала карточки отображен на листинге 1.

Листинг 1 – Реализация функционала карточки здания

```
const showAllRooms = ref(false)
<button
  v-if="building.rooms.length > 6"
  @click="showAllRooms = !showAllRooms"
>
  {{ showAllRooms ? 'Скрыть' : `Показать ещё
  ${building.rooms.length - 6}` }}
</button>
```

Заголовок визуально выделяется с помощью цветового градиента, содержит наименование и обозначения функционального типа. Также отображаются визуальные индикаторы, свидетельствующие о наличии доступа для маломобильных групп, лифта и парковки. Данные помещаются в адаптивную сетку, автоматически перестраивающуюся при изменении ширины экрана. Используется градиентная цветовая гамма и единые отступы, обеспечивающие визуальную целостность.

В рамках адаптивности компоненты интерфейса автоматически масштабируются и изменяют расположение. Например, на мобильных устройствах карточка здания занимает всю ширину экрана и упрощает структуру отображаемых данных. Поведение компонентов регулируется через внутреннюю логику: переключение между состояниями, закрытие карточки, отображение дополнительных деталей. Это обеспечивает интерактивность и динамичность без перезагрузки страницы. Таким образом, реализованный пользовательский интерфейс соответствует требованиям наглядности, доступности и отзывчивости. Реализованный интерфейс изображен на рисунке 22.

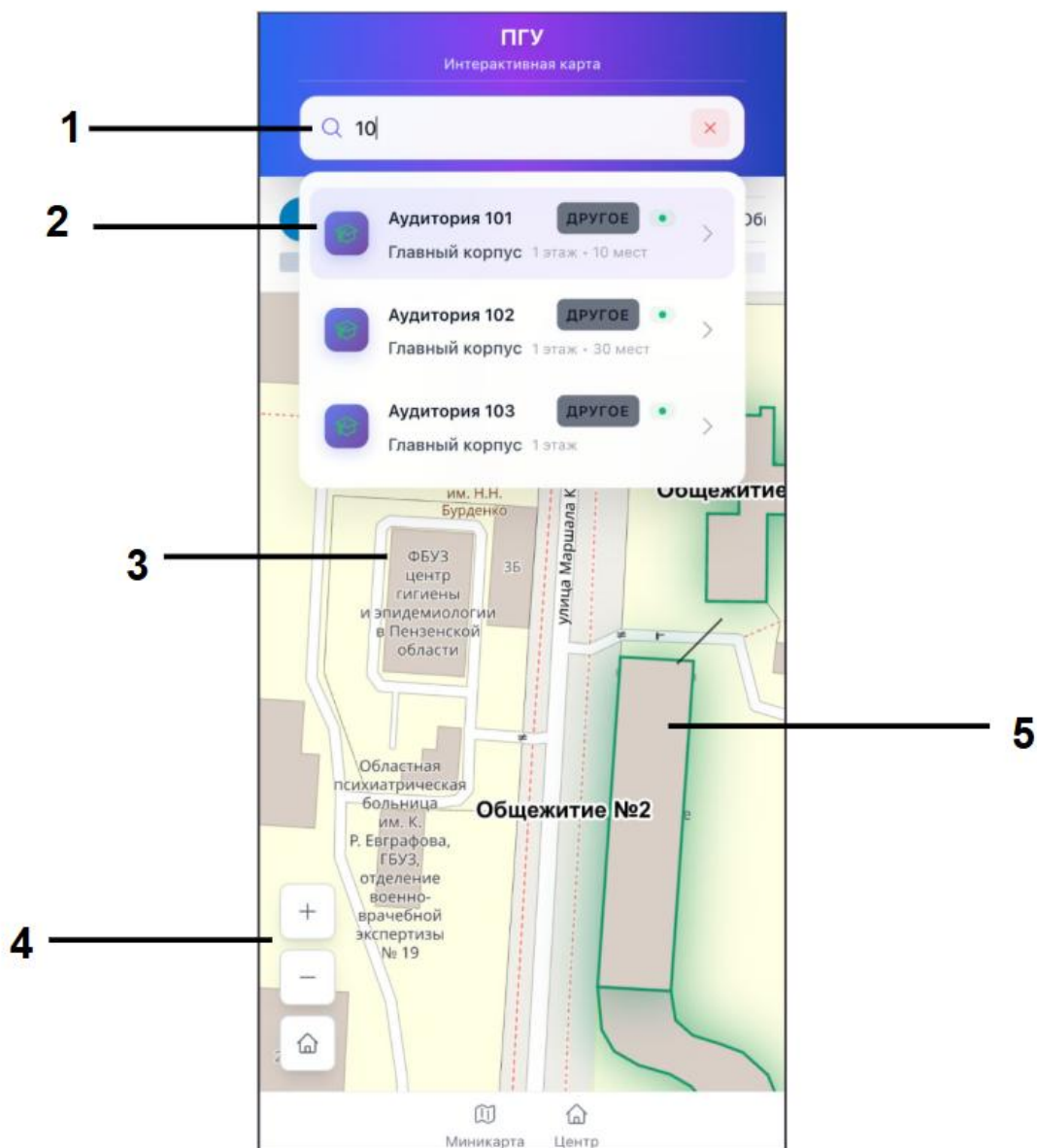


Рисунок 22 – Интерфейс мобильной версии приложения

Основными функциональными элементами интерфейса являются:

- 1 – поисковая строка;
- 2 – результаты поиска, выбранная аудитория;
- 3 – отображение не выбранного здания;
- 4 – кнопки изменения масштаба карты;
- 5 – выбранное пользователем здание.

Также существует версия приложения доступная на компьютерном экранном разрешении. Она изображена на рисунке 23.

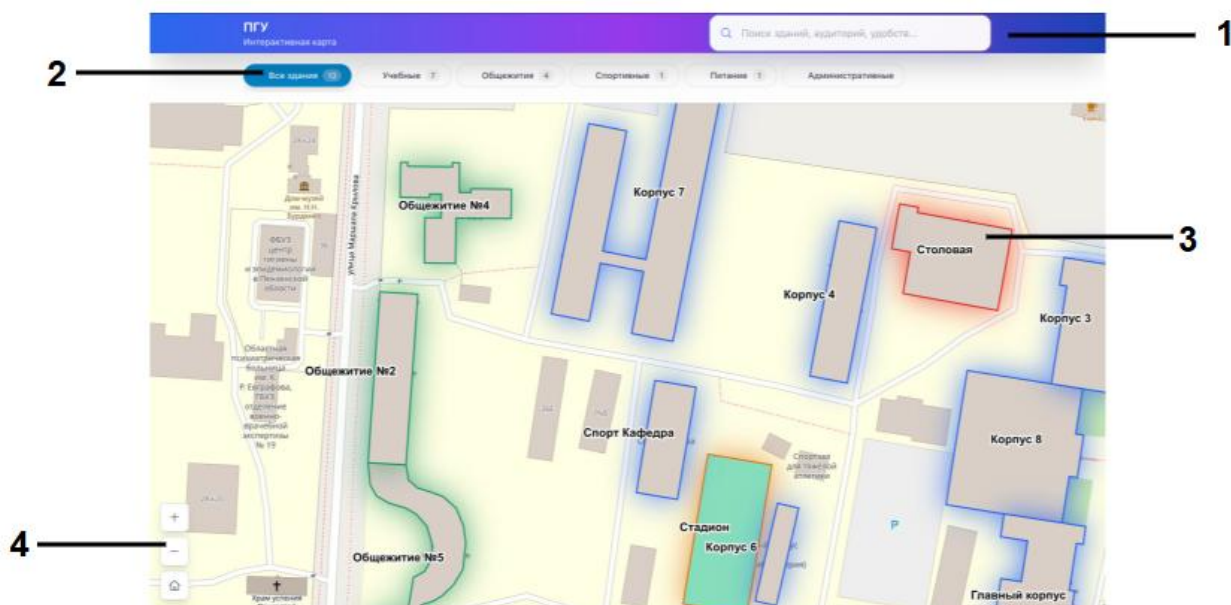


Рисунок 23 – Интерфейс компьютерной версии приложения

Основными функциональными элементами интерфейса являются:

- 1 – поисковая строка;
- 2 – кнопки сортировки зданий по категориям;
- 3 – здание выделенное цветом, в соответствии со своей категорией;
- 4 – кнопки изменения масштаба карты.

Центральным элементом пользовательского взаимодействия выступает цифровая карта кампуса, которая выполняет не только функцию визуализации, но и функцию ориентации, фильтрации и поиска. Графическое представление пространства способствует формированию устойчивого образа и сокращает временные затраты на поиск информации [14].

Для её реализации была применена структура на основе векторной графики, предоставляющая широкие возможности для визуального управления элементами и взаимодействия с ними без существенной нагрузки на производительность. В качестве основы была использована векторная карта, экспортированная из сервиса OpenStreetMap. Благодаря открытости платформы, возможна загрузка географической разметки в формате SVG, пригодном для последующего редактирования и интеграции в веб-приложение. Изображение

содержало масштабируемую схему с точным расположением зданий и инфраструктурных объектов Пензенского государственного университета. Полученный SVG-файл был обработан в редакторе Inkscape, где каждому зданию или иному значимому элементу вручную был присвоен уникальный идентификатор. Данный процесс изображен на рисунке 24.

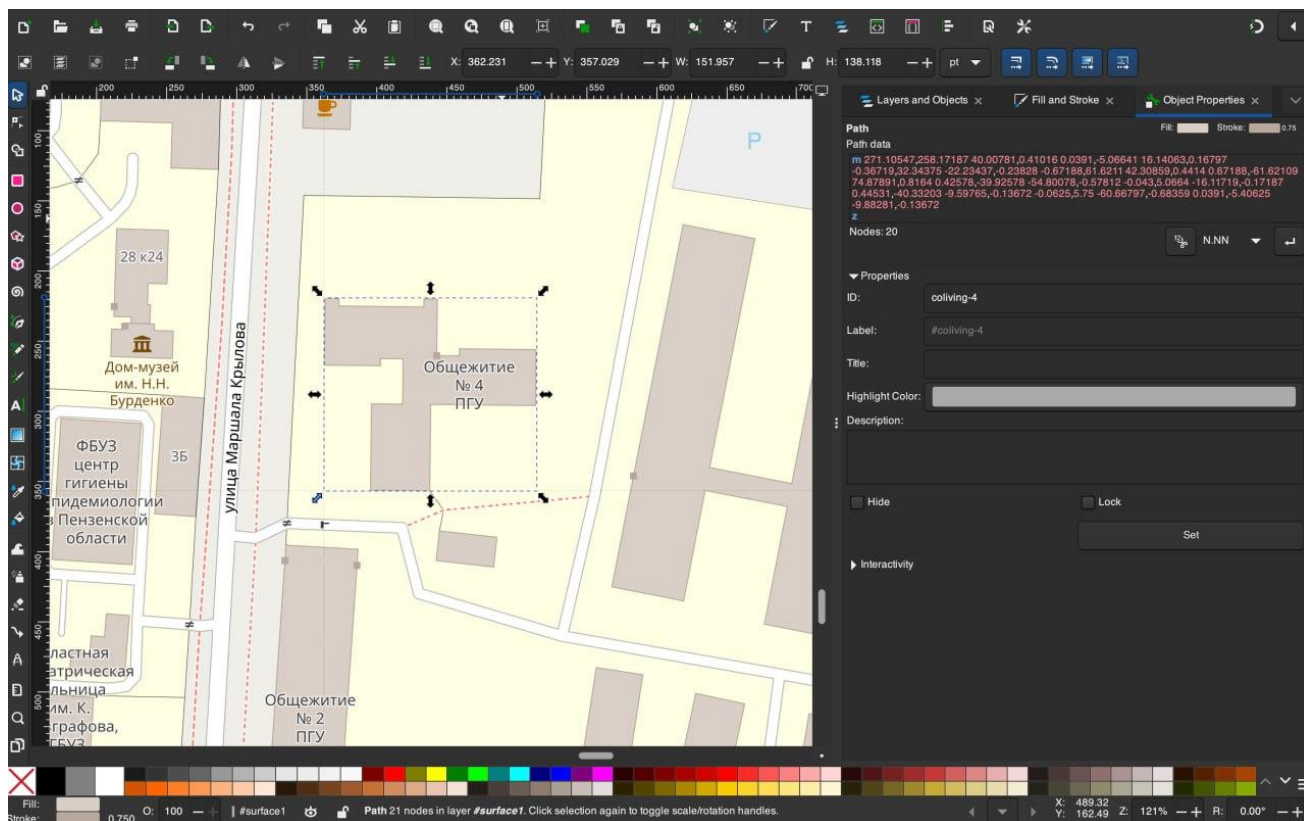


Рисунок 24 – Изменение карты в редакторе Inkscape

Это действие необходимо для того, чтобы в дальнейшем каждый объект на карте мог быть индивидуально распознан и обработан в логике клиентского интерфейса. Названия идентификаторов соответствуют ключам в базе данных, что позволило обеспечить прямую связь между данными и визуальным отображением.

Готовый SVG-файл был встроен в пользовательский интерфейс на клиентской стороне. Посредством JavaScript была реализована логика, позволяющая подсвечивать объекты карты в ответ на действия пользователя. Так, при выборе здания из списка, по его идентификатору производился поиск соответствующего элемента и к нему применялись стили подсветки:

- изменение заливки;
- отображение рамки;
- добавление анимации или смены цвета.

Эта реализация позволила обеспечить двустороннюю связь: пользователь мог выбрать объект на карте или из списка, и в обоих случаях визуально выделить. Для обеспечения интерактивности использовались события наведения и клика. При наведении курсора на объект карта временно изменяла стиль элемента, а при выборе открывалась информационная карточка. Таким образом, визуальный слой стал частью общей системы навигации и взаимодействия с данными. Использование векторной графики позволило сохранить чёткость изображения на различных разрешениях экранов, что особенно важно при работе с мобильными устройствами. Кроме того, SVG-графика обладает малым весом и не требует подключения дополнительных картографических библиотек.

Одним из приоритетов при разработке системы стала реализация механизма быстрого поиска объектов. Пользователь имеет возможность ввести часть названия или ключевое слово, после чего отображается список совпадений с возможностью выбора нужного результата. Особенность реализации заключается в контекстной реакции на действия пользователя: при выборе элемента срабатывает автоматическое перемещение к нужной точке на карте, а затем отображается вся сопутствующая информация. Это позволяет организовать процесс ориентации без промежуточных переходов и дополнительных действий. Каждая аудитория содержит атрибуты: номер, этаж, назначение, вместимость, наличие оборудования. Функция предварительного ознакомления с объектом до физического взаимодействия с ним расширяет адаптационные возможности системы и повышает её прикладную ценность [16].

На рисунке 25 представлена диаграмма последовательности, описывающая процесс фильтрации и отображения здания на карте в приложении «Интерактивная карта ПГУ». Диаграмма демонстрирует, как реализована логика фильтрации объектов и отображения результатов в виде визуального ответа.

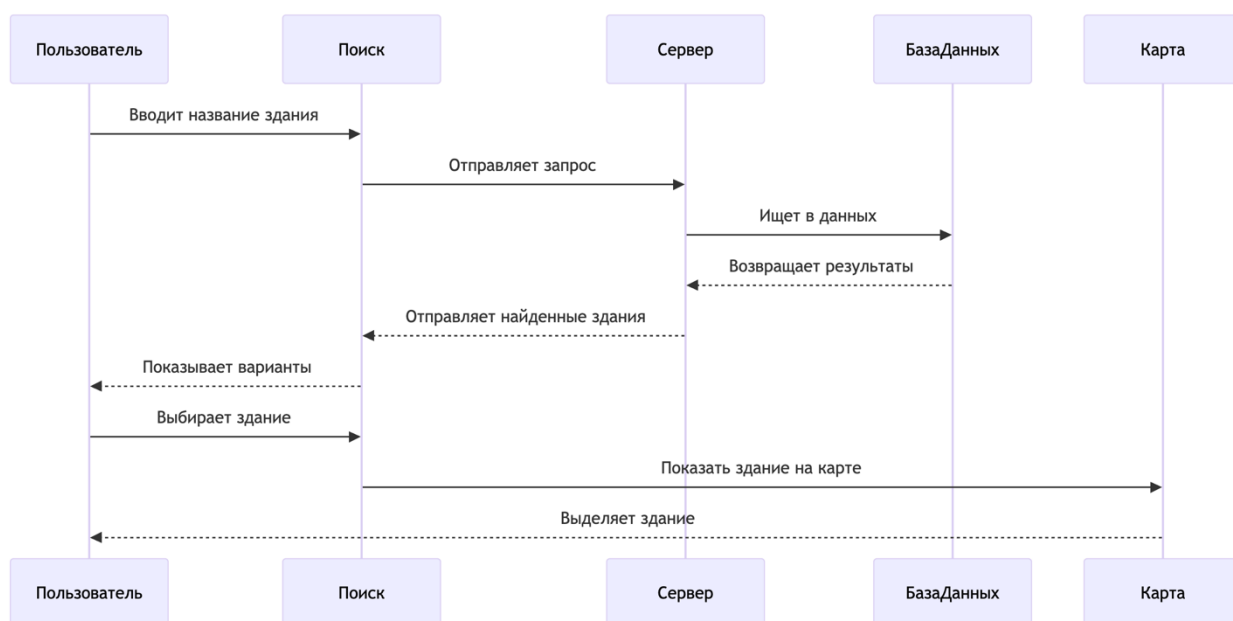


Рисунок 25 – Диаграмма последовательности

Важно отметить, что все действия пользователя приводят к последовательной цепочке запросов, система предоставляет быстрый отклик благодаря разделению ролей между модулями, итоговое взаимодействие завершается визуальным выделением здания на карте, обеспечивая понятную и предсказуемую реакцию интерфейса.

Таким образом, на этапе реализации была сформирована система, соответствующая не только функциональным ожиданиям, но и требованиям к восприятию, доступности и устойчивости. Она сочетает в себе современные принципы построения образовательной информационной среды, применимые как в учебном процессе, так и в управленческой деятельности вуза.

4 Контроль качества приложения «Интерактивная карта ПГУ»

После завершения этапа реализации была проведена проверка работоспособности созданной системы. Тестирование – это систематический процесс проверки и оценки соответствия фактического поведения программного средства заданным спецификациям [18].

Основной целью тестирования стало выявление ошибок в логике работы, проверка отклика интерфейса и оценка соответствия фактической реализации функциональным требованиям, зафиксированным в техническом задании.

Контроль качества программного продукта осуществлялся с использованием ручного тестирования. Проверке подвергались как отдельные функции пользовательского интерфейса, так и сценарии взаимодействия с системой в целом. Тестирование проводилось поэтапно: сначала оценивалась полнота реализации функциональных требований, затем соответствие интерфейса и поведения программы заявленным нефункциональным характеристикам.

4.1 Функциональное тестирование

Функциональные тесты должны охватывать все пути выполнения, соответствующие заявленным требованиям, включая предельные и исключительные случаи [19]. На этапе функционального тестирования проверялись базовые возможности системы, заявленные в рамках проектной документации. Все тесты проводились с участием реальных сценариев, описанных в предыдущих главах.

Функциональное тестирование включало проверку следующих операций:

- корректная загрузка интерфейса и карты кампуса при запуске;
- отображение зданий на карте в соответствии с координатами;
- выбор здания и открытие соответствующей карточки с информацией;
- поиск зданий по ключевому слову или названию;

- фильтрация объектов по типу;
- отображение аудиторий, относящихся к выбранному зданию;
- просмотр информации о характеристиках помещения;
- возврат корректного ответа при отсутствии данных по запросу.

Каждое действие выполнялось по отдельности, в изолированной среде. Были проведены тесты на основе реальных ситуаций, результаты данного тестирования отображены в таблице 10.

Таблица 10 – Тестирование на основе реальных сценариев

№ п/п	Наименование теста	Входные данные	Ожидаемый результат	Фактический результат
1	Загрузка карты кампуса	Запуск системы	Отображается карта с маркерами зданий	Весь пользовательский интерфейс корректно отображен (рисунок Б.1)
2	Выбор здания и отображение информации	Клик по зданию на карте	Открывается карточка здания с полной информацией	Описание здания корректно отображено (рисунок Б.2)
3	Поиск здания по названию	Ввод ключевого слова в поиске	Появляется список зданий, соответствующих запросу	Список найденных зданий корректно отображен (рисунок Б.3)
4	Фильтрация по типу здания	Выбор фильтра 'учебное'	На карте остаются только учебные здания	Фильтрация отображена корректно (рисунок Б.4)
5	Просмотр информации об аудитории	Клик по номеру аудитории в карточке здания	Открывается карточка с данными об аудитории	Данные об аудитории отображены корректно (рисунок Б.5)
6	Проверка адаптивности интерфейса	Изменение ширины окна браузера	Интерфейс перестраивается, элементы корректно отображаются	Адаптивность интерфейса работает корректно (рисунок Б.6)

Результаты выполнения операций соответствовали ожидаемым. Все основные функции системы были реализованы в полном объёме и работали без сбоев.

4.2 Оценка метрик кода

Одним из способов оценки качества реализованного программного обеспечения является анализ исходного кода с использованием формальных метрик. Такие показатели позволяют количественно выразить сложность, связанность и читаемость функций, а также определить потенциальные зоны риска в сопровождении и дальнейшем расширении системы.

4.2.1 Метрика Чепина

Для анализа была использована метрика Чепина – один из классических подходов к измерению сложности подпрограмм, основанный на классификации переменных по их назначению внутри функции. Код функции, выбранной для анализа представлен в листинге 5.

Листинг 5 – Функция получения здания по его ID

```
def get_building_by_id(self, building_id: str) ->
Optional[Building]:
    """Получение здания по ID"""
    with self.get_connection() as conn:
        cursor = conn.execute("SELECT * FROM buildings WHERE
id = ?", (building_id,))
        row = cursor.fetchone()

        if not row:
            return None
        building_data = dict(row)

        if building_data["coordinates"]:
            building_data["coordinates"] =
json.loads(building_data["coordinates"])
        if building_data["departments"]:
            building_data["departments"] =
json.loads(building_data["departments"])
        if building_data["amenities"]:
            building_data["amenities"] =
json.loads(building_data["amenities"])
        if building_data["rooms"]:
            rooms_data = json.loads(building_data["rooms"])
            building_data["rooms"] = [Room(**room) for room in
rooms_data]
        else:
            building_data["rooms"] = []
        return Building(**building_data)
```

В рамках этой метрики переменные делятся на четыре категории:

- переменные, выступающие в роли входных параметров (Р);
- переменные, модифицируемые внутри функции и сохраняющие значение после её завершения (М);
- переменные, используемые исключительно для внутренних вычислений (С);
- вспомогательные переменные, временно применяемые внутри алгоритма (Т).

Значение метрики определяется по следующей формуле:

$$МС = 1 * Р + 2 * М + 0.5 * С + 0.5 * Т. \quad (1)$$

В качестве объекта анализа была выбрана функция получения информации о здании по идентификатору, реализующая выборку данных из хранилища и их предварительную обработку. Результаты классификации переменных представлены в таблице 11.

Таблица 11 – Классификация переменных функции по метрике Чепина

Тип переменной	Количество	Назначение
Р (входные)	1	идентификатор здания
М (выходные)	0	отсутствуют
С (вычисления)	4	соединение, курсор, результат запроса, словарь данных
Т (временные)	1	промежуточная структура для обработки вложенных данных

Применяя формулу 1 и значения из таблицы 11 получаем:

$$МС = 1 * 1 + 2 * 0 + 0.5 * 4 + 0.5 * 1 = 1 + 0 + 2 + 0.5 = 3,5$$

Полученное значение указывает на умеренную структурную сложность функции. Она обладает ограниченным числом входных данных, отсутствием побочных эффектов и логически завершённым поведением. Однако наличие нескольких вычислительных переменных и необходимость обработки вложенных структур позволяют отнести данную функцию к уровню, требующему внимательности при сопровождении. Упрощение может быть достигнуто путём выноса логики обработки данных в отдельные подфункции.

Оценка метрик кода служит дополнительным инструментом для выявления потенциальных трудностей при расширении системы и может быть использована при оптимизации реализации.

4.2.2 Цикломатическая сложность функции

Для дополнительной количественной оценки сложности реализованной функции, наряду с метрикой Чепина, была использована метрика цикломатической сложности, предложенная Т. Маккейбом. Она применяется для анализа логической структуры функций и оценки необходимого числа тестов, обеспечивающих полное покрытие кода. Цикломатическая сложность определяется числом независимых путей в графе управления программой. Чем выше это значение, тем более разветвлённой является логика функции, и тем выше требования к её тестированию и сопровождению. Для тестирования этим способом была выбрана та же функция, которая предоставлена в листинге 5.

Упрощённая формула для оценки цикломатической сложности для функции с линейным телом:

$$V(G) = C + 1, \quad (2)$$

где C – количество точек ветвления;

Анализ исследуемой функции показал наличие следующих логических конструкций:

- одна проверка на отсутствие результата запроса;
- три условия, проверяющих наличие данных в отдельных полях;
- одна ветвь с альтернативной логикой обработки.

Таким образом, подставляя полученное значение в формулу 2, получаем:

$$V(G) = 5 + 1 = 6$$

Значение 6 указывает на умеренный уровень сложности, это означает, что для полного логического покрытия всех вариантов выполнения функции требуется не менее шести модульных тестов. Таким образом, метрика Маккейба подтверждает общую умеренную, но не чрезмерную сложность функции, и может использоваться в качестве ориентира при оптимизации реализации.

4.3 Нефункциональное тестирование

Целью нефункционального тестирования стало подтверждение соответствия реализованного интерфейса требованиям доступности, адаптивности, визуальной наглядности и устойчивости работы.

В ходе проверки учитывались следующие параметры:

- корректность отображения интерфейса на экранах различной ширины;
- расположение элементов интерфейса и их реакция на действия пользователя;
- стабильность поведения при вводе некорректных или отсутствующих запросов;
- визуальное соответствие карточек зданий и аудиторий общему стилю интерфейса;
- структурированность представления информации в карточках объектов;
- отсутствие визуальных сбоев при динамической подгрузке информации.

Удобство, адаптивность и предсказуемость поведения интерфейса определяют общее впечатление пользователя и влияют на эффективность восприятия информации [20]. По результатам нефункционального тестирования интерфейс системы был признан удобным и интуитивно понятным. Пользователь мог выполнять все действия без необходимости внешнего обучения или вспомогательных инструкций. Отображение объектов и структурированных данных соответствовало заявленным требованиям.

Осуществим нефункциональное тестирование с использованием фреймворка Playwright, отчет о работе которого изображен на рисунке 26.

Функциональное тестирование при помощи фреймворка Playwright позволяет автоматически воспроизводить действия пользователя в браузере и проверять корректность работы интерфейсных компонентов.

Данный подход обеспечивает надёжную имитацию реального поведения системы и позволяет выявить логические ошибки, неочевидные при ручной проверке.

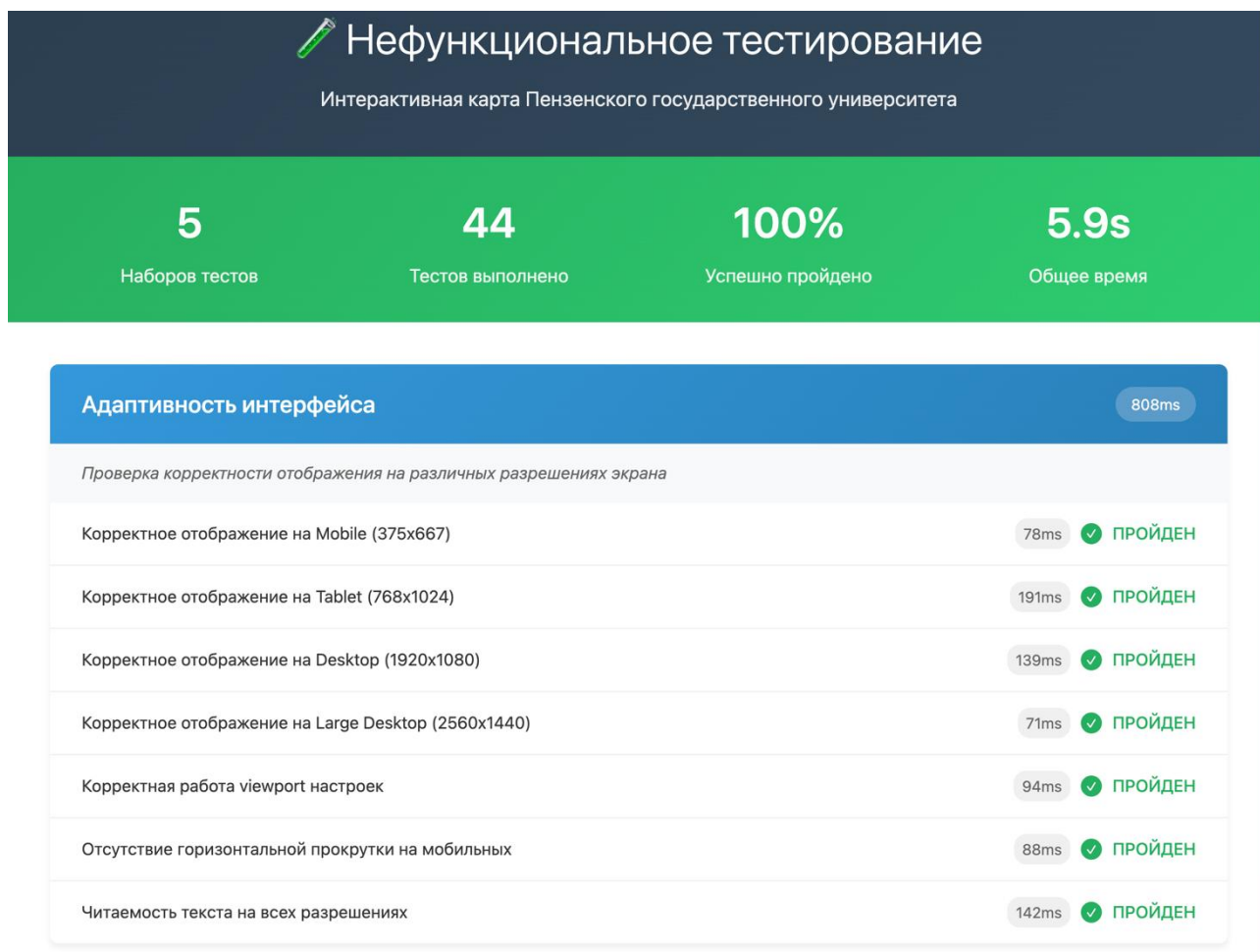


Рисунок 26 – Результат нефункционального тестирования

В результате тестирования можно сделать вывод о полной готовности программного продукта к использованию в образовательной среде. Все заявленные функции реализованы, интерфейс адаптирован под разные устройства, визуальные компоненты оформлены единообразно. Система работает устойчиво, интерфейс откликается на действия пользователя без задержек, структура информации логична и прозрачна. Ошибок в ходе тестирования зафиксировано не было. Поведение всех компонентов соответствует требованиям, изложенным в техническом и проектном описании.

Заключение

В ходе выполнения выпускной квалификационной работы была спроектирована и реализована информационная система пространственной навигации «Интерактивная карта Пензенского государственного университета».

Разработка данного программного средства была направлена на решение актуальной задачи – обеспечения наглядного, доступного и структурированного представления информации об инфраструктуре вуза

Проведён анализ существующих решений, использующих картографический подход к визуализации кампусов образовательных учреждений.

Выделены функциональные и нефункциональные требования, сформированные с учётом особенностей предметной области; – разработана архитектура программной системы, обеспечивающая логическое разделение компонентов и удобство масштабирования.

Сформирована логическая модель данных, отражающая реальную структуру объектов университета: зданий, аудиторий, подразделений и элементов доступности.

Реализованы основные программные компоненты, обеспечивающие работу интерфейса, обработку пользовательских запросов, отображение карты и связанных с ней объектов

Созданная система демонстрирует стабильную и корректную работу всех ключевых функций. Пользователь может эффективно взаимодействовать с картой кампуса, осуществлять поиск зданий, просматривать информацию об аудиториях, использовать фильтры и адаптированный интерфейс как с настольных, так и с мобильных устройств. Таким образом, поставленные цели и задачи выпускной квалификационной работы были достигнуты, а результат разработки соответствует требованиям, предъявляемым к современным информационным системам образовательного назначения.

Список использованных источников

1. Астахова Е. А., Слепнева Ю. В. Пространственная навигация как фактор образовательной среды вуза // Вестник Томского государственного педагогического университета. – 2022. – № 5. – С. 33–38. [1, с. 36]
2. Коряковцев А. А., Шматко А. Д. Цифровизация университетской инфраструктуры: вызовы и решения // Университетское управление: практика и анализ. – 2023. – № 3. – С. 108–117. [2, с. 112]
3. Губанова Е. С. Информационная навигация в образовательной среде: подходы и практики // Инновации в образовании. – 2021. – № 2 (92). – С. 46–51. [3, с. 49]
4. Николаев В. Г. Основы проектирования интерфейсов: теория и практика. – СПб.: Питер, 2020. – 224 с. [4, с. 92]
5. Глухов А. В., Чернышев И. Ю. Пространственная визуализация в информационных системах // Вестник Санкт-Петербургского университета. Серия 9. 2021. № 1. С. 35–41. [5, с. 37].
6. Тихонов А. П. Эргономика образовательной среды. – М.: Просвещение, 2019. – 272 с. [6, с. 118]
7. Козлова И. В. Цифровая эргономика: проектирование удобных интерфейсов. – М.: Инфра-М, 2021. – 192 с [7, с. 67].
8. Погодин Д. С. Кроссплатформенные подходы в проектировании информационных систем // Информационные технологии и системы. – 2022. – № 3. – С. 89–94. [8, с. 91]
9. Филиппов А. Е. Адаптивная архитектура цифровых платформ. – СПб.: Питер, 2020. – 224 с [9, с. 118].
10. Бочкарёв В. В. Основы проектирования программного обеспечения: учебное пособие. – М.: Инфра-М, 2021. – 312 с. [10, с. 144]
11. Эльмасри Р., Наватхе Ш. Основы систем баз данных: пер. с англ. – 7-е изд. – М.: Вильямс, 2019. – 1272 с [11, с. 86].

12. Шнайдерман Б. Проектирование пользовательского интерфейса: стратегии успешного взаимодействия человека и компьютера. – СПб.: Питер, 2020. – 560 с. [12, с. 104].
13. Сомин В. В. Архитектура информационных систем: методология проектирования. – СПб.: Питер, 2022. – 368 с.
14. Глухов А. В. Визуализация данных в цифровых средах // Информационное общество. – 2021. – № 3. – С. 119–125.
15. Астахова Е. А., Слепнева Ю. В. Пространственная навигация как фактор образовательной среды вуза // Вестник ТГПУ. – 2022. – № 5. – С. 62–67.
16. Коршунов И. Ю. Адаптация информационных систем в образовательных учреждениях // Образовательные технологии. – 2020. – № 4. – С. 135–140.
17. Козлова И. В. Цифровая эргономика: проектирование удобных интерфейсов. – М.: Инфра-М, 2021. – 192 с.
18. Панкратов И. А. Тестирование программного обеспечения: теория и практика. – М.: КноРус, 2020. – 272 с.
19. Васильев А. Л. Технологии контроля качества программных продуктов. – СПб.: Питер, 2019. – 288 с.
20. Ларина О. И. Пользовательский интерфейс: проектирование и оценка удобства. – М.: Юрайт, 2021. – 304 с.

Приложение А
(обязательное)
Листинг приложения

Листинг А.1 – Файл «main.py»

```

from fastapi import FastAPI, HTTPException, Query, BackgroundTasks
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse
from typing import List, Optional, Dict, Any
import asyncio
from contextlib import asynccontextmanager
import logging

# Импорты новых модулей
from models import Building, BuildingResponse, SearchResponse
from controllers import BuildingController, SearchController
from database import db

# Настройка логирования
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

@asynccontextmanager
async def lifespan(app: FastAPI):
    # Startup
    logger.info("Запуск приложения...")
    logger.info(f"База данных инициализирована с {db.get_total_count()} зданиями")
    yield
    # Shutdown
    logger.info("Завершение приложения...")

# Инициализация FastAPI
app = FastAPI(
    title="University Map API",
    description="API для интерактивной карты Пермского государственного университета",
    version="2.0.0",
    lifespan=lifespan
)

# Настройка CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:5173",
                  "http://127.0.0.1:5173"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Middleware для логирования
@app.middleware("http")
async def log_requests(request, call_next):
    logger.info(f"{request.method} {request.url}")

```

```

    response = await call_next(request)
    logger.info(f"Ответ: {response.status_code}")
    return response

# Routes
@app.get("/", response_model=Dict[str, str])
async def root():
    """Главная страница API"""
    return {
        "message": "University Map API v2.0",
        "docs": "/docs",
        "health": "/health"
    }

@app.get("/health", response_model=Dict[str, str])
async def health_check():
    """Проверка здоровья API"""
    buildings_count = db.get_total_count()
    return {"status": "healthy", "buildings_count":
str(buildings_count)}

@app.get("/api/buildings", response_model=List[Building])
async def get_buildings(
    query: Optional[str] = Query(None, description="Поисковый
запрос по названию"),
    type: Optional[str] = Query(None, description="Фильтр по типу
здания"),
    page: int = Query(1, ge=1, description="Номер страницы"),
    limit: int = Query(50, ge=1, le=100, description="Количество
результатов на странице")
):
    """
    Получение списка зданий университета

    - **query**: Поиск по названию или описанию здания
    - **type**: Фильтр по типу здания (academic, living, sports,
dining, administrative)
    - **page**: Номер страницы для пагинации
    - **limit**: Количество результатов на странице (максимум 100)
    """
    return await BuildingController.get_buildings(query, type,
page, limit)

@app.get("/api/buildings/paginated",
response_model=BuildingResponse)
async def get_buildings_paginated(
    query: Optional[str] = Query(None, description="Поисковый
запрос по названию"),
    type: Optional[str] = Query(None, description="Фильтр по типу
здания"),
    page: int = Query(1, ge=1, description="Номер страницы"),

```

```

        limit: int = Query(50, ge=1, le=100, description="Количество
результатов на странице")
    ):
        """
        Получение списка зданий с метайнформацией о пагинации
        """
        return await
BuildingController.get_buildings_with_pagination(query, type,
page, limit)

@app.get("/api/buildings/{building_id}", response_model=Building)
async def get_building(building_id: str):
    """
    Получение детальной информации о здании по ID

    - **building_id**: Уникальный идентификатор здания
    """
    return await
BuildingController.get_building_by_id(building_id)

@app.get("/api/buildings/types", response_model=List[Dict[str,
Any]])
async def get_building_types():
    """
    Получение списка типов зданий со статистикой

    Возвращает список всех типов зданий с количеством зданий
каждого типа
    """
    return await BuildingController.get_building_types()

@app.get("/api/search", response_model=SearchResponse)
async def advanced_search(
    q: str = Query(..., description="Поисковый запрос"),
    limit: int = Query(10, ge=1, le=50, description="Максимальное
количество результатов")
):
    """
    Расширенный поиск по зданиям, аудиториям и услугам

    - **q**: Поисковый запрос (название здания, номер аудитории,
услуга)
    - **limit**: Максимальное количество результатов (по умолчанию
10, максимум 50)

    Поддерживаемые типы поиска:
    - Поиск по названию здания
    - Поиск по номеру аудитории (например: "101", "аудитория 202")
    - Поиск по услугам и удобствам (например: "кафе", "туалет",
"библиотека")
    """
    return await SearchController.advanced_search(q, limit)

```

```

@app.get("/api/suggestions", response_model=List[str])
async def get_search_suggestions():
    """
    Получение подсказок для поиска

    Возвращает список популярных поисковых запросов
    """
    suggestions = [
        "столовая", "библиотека", "аудитория 101", "туалет",
        "кафе", "спортзал", "главный корпус", "общежитие",
        "деканат", "ректорат", "лифт", "парковка"
    ]
    return suggestions

# Обработчики ошибок
@app.exception_handler(404)
async def not_found_handler(request, exc):
    return JSONResponse(
        status_code=404,
        content={"detail": "Ресурс не найден"}
    )

@app.exception_handler(500)
async def internal_error_handler(request, exc):
    logger.error(f"Внутренняя ошибка сервера: {exc}")
    return JSONResponse(
        status_code=500,
        content={"detail": "Внутренняя ошибка сервера"}
    )

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)

```

Листинг A.2 – Файл «database.py»

```

import sqlite3
import json
import logging
import os
from typing import List, Optional, Dict, Any
from models import Building, Room, RoomType
from pgu_real_data import get_pgu_real_data

logger = logging.getLogger(__name__)

class Database:
    def __init__(self, db_path: str = None):
        if db_path is None:
            # Получаем путь из переменной окружения или используем
            значение по умолчанию

```

```

        database_url = os.getenv('DATABASE_URL',
        'sqlite:///university_map.db')
        if database_url.startswith('sqlite:///'):
            self.db_path = database_url[10:] # Убираем
        'sqlite:/// ' префикс
        else:
            self.db_path = "university_map.db"
    else:
        self.db_path = db_path

    # Создаем директорию для базы данных если её нет
    db_dir = os.path.dirname(self.db_path)
    if db_dir and not os.path.exists(db_dir):
        os.makedirs(db_dir, exist_ok=True)

    self.init_database()
    self.populate_initial_data()

def get_connection(self):
    """Получение соединения с базой данных"""
    conn = sqlite3.connect(self.db_path)
    conn.row_factory = sqlite3.Row # Для получения данных как
dict
    return conn

def init_database(self):
    """Инициализация базы данных"""
    with self.get_connection() as conn:
        conn.execute("""
            CREATE TABLE IF NOT EXISTS buildings (
                id TEXT PRIMARY KEY,
                name TEXT NOT NULL,
                type TEXT NOT NULL,
                description TEXT,
                image_url TEXT,
                coordinates TEXT, -- JSON строка
                floor_count INTEGER,
                year_built INTEGER,
                departments TEXT, -- JSON строка
                amenities TEXT, -- JSON строка
                rooms TEXT, -- JSON строка для
аудиторий
                accessible BOOLEAN DEFAULT 0,
                has_elevator BOOLEAN DEFAULT 0,
                has_parking BOOLEAN DEFAULT 0
            )
        """)
        conn.commit()
        logger.info("База данных инициализирована")

def populate_initial_data(self):
    """Заполнение начальными данными"""

```

```

# Проверяем, есть ли уже данные
with self.get_connection() as conn:
    cursor = conn.execute("SELECT COUNT(*) FROM
buildings")
    count = cursor.fetchone()[0]

    if count > 0:
        logger.info(f"База уже содержит {count} зданий")
        return

# Реальные данные ПГУ (Пензенский государственный
университет)
initial_buildings = get_pgu_real_data()

# Добавляем данные в базу
with self.get_connection() as conn:
    for building_data in initial_buildings:
        conn.execute("""
            INSERT INTO buildings
            (id, name, type, description, image_url,
coordinates, floor_count, year_built, departments, amenities,
rooms, accessible, has_elevator, has_parking)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?
?)

            """, (
                building_data["id"],
                building_data["name"],
                building_data["type"],
                building_data.get("description"),
                building_data.get("image_url"),
                json.dumps(building_data.get("coordinates"))
if building_data.get("coordinates") else None,
                building_data.get("floor_count"),
                building_data.get("year_built"),
                json.dumps(building_data.get("departments",
[])),
                json.dumps(building_data.get("amenities",
[])),
                json.dumps(building_data.get("rooms", [])),
                building_data.get("accessible", False),
                building_data.get("has_elevator", False),
                building_data.get("has_parking", False)
            ))
        conn.commit()
        logger.info(f"Добавлено {len(initial_buildings)}
зданий в базу данных")

def get_all_buildings(self,
                        query: Optional[str] = None,
                        building_type: Optional[str] = None,
                        page: int = 1,
                        limit: int = 50) -> List[Building]:

```

```

        """Получение всех зданий с фильтрацией"""

        sql = "SELECT * FROM buildings WHERE 1=1"
        params = []

        if query:
            sql += " AND (name LIKE ? OR description LIKE ?)"
            like_query = f"%{query}%"
            params.extend([like_query, like_query])

        if building_type:
            sql += " AND type = ?"
            params.append(building_type)

        # Пагинация
        offset = (page - 1) * limit
        sql += " LIMIT ? OFFSET ?"
        params.extend([limit, offset])

        with self.get_connection() as conn:
            cursor = conn.execute(sql, params)
            rows = cursor.fetchall()

            buildings = []
            for row in rows:
                building_data = dict(row)

                # Парсим JSON поля
                if building_data["coordinates"]:
                    building_data["coordinates"] =
json.loads(building_data["coordinates"])
                if building_data["departments"]:
                    building_data["departments"] =
json.loads(building_data["departments"])
                if building_data["amenities"]:
                    building_data["amenities"] =
json.loads(building_data["amenities"])
                if building_data["rooms"]:
                    rooms_data =
json.loads(building_data["rooms"])
                    building_data["rooms"] = [Room(**room) for
room in rooms_data]
                else:
                    building_data["rooms"] = []

                buildings.append(Building(**building_data))

            return buildings

    def get_building_by_id(self, building_id: str) ->
Optional[Building]:
        """Получение здания по ID"""

```

```

        with self.get_connection() as conn:
            cursor = conn.execute("SELECT * FROM buildings WHERE
id = ?", (building_id,))
            row = cursor.fetchone()

            if not row:
                return None

            building_data = dict(row)

            # Парсим JSON поля
            if building_data["coordinates"]:
                building_data["coordinates"] =
json.loads(building_data["coordinates"])
            if building_data["departments"]:
                building_data["departments"] =
json.loads(building_data["departments"])
            if building_data["amenities"]:
                building_data["amenities"] =
json.loads(building_data["amenities"])
            if building_data["rooms"]:
                rooms_data = json.loads(building_data["rooms"])
                building_data["rooms"] = [Room(**room) for room in
rooms_data]
            else:
                building_data["rooms"] = []

            return Building(**building_data)

def get_building_types(self) -> List[Dict[str, Any]]:
    """Получение типов зданий со статистикой"""
    with self.get_connection() as conn:
        cursor = conn.execute("""
            SELECT type, COUNT(*) as count
            FROM buildings
            GROUP BY type
            ORDER BY count DESC
        """)

        type_labels = {
            "academic": "Учебные",
            "living": "Общежития",
            "sports": "Спортивные",
            "dining": "Питание",
            "administrative": "Административные",
            "other": "Другое"
        }

    return [
        {
            "type": row["type"],

```

```

        "label": type_labels.get(row["type"],
row["type"].title()),
        "count": row["count"]
    }
    for row in cursor.fetchall()
]

def get_total_count(self,
    query: Optional[str] = None,
    building_type: Optional[str] = None) ->
int:
    """Получение общего количества зданий с учетом фильтров"""
    sql = "SELECT COUNT(*) FROM buildings WHERE 1=1"
    params = []

    if query:
        sql += " AND (name LIKE ? OR description LIKE ?)"
        like_query = f"%{query}%"
        params.extend([like_query, like_query])

    if building_type:
        sql += " AND type = ?"
        params.append(building_type)

    with self.get_connection() as conn:
        cursor = conn.execute(sql, params)
        return cursor.fetchone()[0]

# Глобальный экземпляр базы данных
db = Database()

```

Листинг А.3 – Файл «models.py»

```

from pydantic import BaseModel, Field
from typing import List, Optional, Dict
from enum import Enum

class RoomType(str, Enum):
    CLASSROOM = "classroom"
    LAB = "lab"
    OFFICE = "office"
    TOILET = "toilet"
    CAFE = "cafe"
    LIBRARY = "library"
    AUDITORIUM = "auditorium"
    ROOM = "room" # Комнаты в общежитиях
    OTHER = "other"

class Room(BaseModel):
    number: str = Field(..., description="Номер аудитории")
    floor: int = Field(..., description="Этаж")
    type: RoomType = Field(..., description="Тип аудитории")

```

```

        capacity: Optional[int] = Field(None,
description="Вместимость")
        equipment: Optional[List[str]] = Field(default_factory=list,
description="Оборудование")
        accessible: bool = Field(False, description="Доступность для
маломобильных")

class Building(BaseModel):
    id: str = Field(..., description="Уникальный ID здания")
    name: str = Field(..., description="Название здания")
    type: str = Field(..., description="Тип здания")
    description: Optional[str] = Field(None, description="Описание
здания")
    image_url: Optional[str] = Field(None, description="URL
изображения")
    coordinates: Optional[Dict[str, float]] = Field(None,
description="Координаты на карте")
    floor_count: Optional[int] = Field(None,
description="Количество этажей")
    year_built: Optional[int] = Field(None, description="Год
постройки")
    departments: Optional[List[str]] = Field(default_factory=list,
description="Кафедры и отделы")
    amenities: Optional[List[str]] = Field(default_factory=list,
description="Удобства")
    rooms: Optional[List[Room]] = Field(default_factory=list,
description="Аудитории")
    accessible: bool = Field(False, description="Доступность для
маломобильных")
    has_elevator: bool = Field(False, description="Наличие лифта")
    has_parking: bool = Field(False, description="Наличие
парковки")

class SearchResultType(str, Enum):
    BUILDING = "building"
    ROOM = "room"
    AMENITY = "amenity"

class SearchResult(BaseModel):
    type: SearchResultType = Field(..., description="Тип
результата поиска")
    building: Building = Field(..., description="Здание")
    room: Optional[Room] = Field(None, description="Аудитория
(если поиск по аудитории)")
    amenity: Optional[str] = Field(None, description="Услуга (если
поиск по услуге)")
    match_text: str = Field(..., description="Текст совпадения")
    priority: int = Field(..., description="Приоритет для
сортировки")

class SearchResponse(BaseModel):
    results: List[SearchResult]

```

```

    total: int
    query: str

class BuildingResponse(BaseModel):
    buildings: List[Building]
    total: int
    page: int
    limit: int

```

Листинг А.4 – Файл «BuildingCard.vue»

```

<template>
  <Transition name="card-slide">
    <div v-if="building" class="building-card">
      <!-- Заголовок с градиентом -->
      <div class="card-header">
        <div class="header-content">
          <div class="building-info">
            <h3 class="building-title">{{ building.name }}</h3>
            <div class="building-meta">
              <span :class="getTypeBadgeClass(building.type)"
class="type-badge">
                {{ getTypeLabel(building.type) }}
              </span>
              <div class="accessibility-indicators">
                <div v-if="building.accessible" class="indicator
accessible" title="Доступно для людей с ограниченными
возможностями">
                  <UserIcon class="w-4 h-4" />
                </div>
                <div v-if="building.has_elevator" class="indicator
elevator" title="Есть лифт">
                  <ArrowUpIcon class="w-4 h-4" />
                </div>
                <div v-if="building.has_parking" class="indicator
parking" title="Есть парковка">
                  <TruckIcon class="w-4 h-4" />
                </div>
              </div>
            </div>
          </div>
          <button @click="closeCard" class="close-button">
            <XMarkIcon class="w-5 h-5" />
          </button>
        </div>

      <!-- Содержимое карточки -->
      <div class="card-content">
        <!-- Изображение здания -->
        <div v-if="building.image_url" class="building-image">
          

```

```

        <div class="image-overlay"></div>
    </div>

    <!-- Описание -->
    <div v-if="building.description" class="building-
description">
        <p>{{ building.description }}</p>
    </div>

    <!-- Информация о комнатах -->
    <div v-if="building.rooms && building.rooms.length > 0"
class="rooms-section">
        <h4 class="section-title">
            <HomeIcon class="w-5 h-5" />
            Помещения ({{ building.rooms.length }})
        </h4>
        <div class="rooms-grid">
            <div
                v-for="room in building.rooms.slice(0, 6)"
                :key="room.number"
                class="room-card"
            >
                <div class="room-header">
                    <span class="room-number">{{ room.number }}</span>
                    <span :class="getRoomTypeBadge(room.type)"
class="room-type">
                        {{ getRoomTypeLabel(room.type) }}
                    </span>
                </div>
                <div class="room-details">
                    <span v-if="room.floor" class="detail">{{
room.floor }} этаж</span>
                    <span v-if="room.capacity" class="detail">{{
room.capacity }} мест</span>
                    <div v-if="room.accessible" class="accessibility-
dot" title="Доступно"></div>
                </div>
            </div>
            <div>
                <button
                    v-if="building.rooms.length > 6"
                    class="show-more-button"
                    @click="showAllRooms = !showAllRooms"
                >
                    {{ showAllRooms ? 'Скрыть' : `Показать еще
${building.rooms.length - 6}` }}
                </button>
            </div>

    <!-- Дополнительная информация -->
    <div class="building-stats">
        <div class="stat-item">

```

```

        <BuildingOfficeIcon class="w-5 h-5" />
        <span>{{ building.floors || 1 }} {{
pluralize(building.floors || 1, 'этаж', 'этажа', 'этажей')
}}</span>
    </div>
    <div v-if="building.total_capacity" class="stat-item">
        <UsersIcon class="w-5 h-5" />
        <span>{{ building.total_capacity }} мест</span>
    </div>
</div>
</div>
</div>
</Transition>
</template>

<script setup lang="ts">
import { ref } from 'vue'
import {
    XMarkIcon,
    HomeIcon,
    BuildingOfficeIcon,
    UsersIcon,
    UserIcon,
    ArrowUpIcon,
    TruckIcon
} from '@heroicons/vue/24/outline'
import type { Building } from '@stores/buildings'

interface Props {
    building: Building | null
}

defineProps<Props>()

const emit = defineEmits<{
    close: []
}>()

const showAllRooms = ref(false)

const closeCard = () => {
    emit('close')
}

const getTypeBadgeClass = (type: string) => {
    const classes = {
        'academic': 'bg-blue-100 text-blue-800 border-blue-200',
        'living': 'bg-green-100 text-green-800 border-green-200',
        'sports': 'bg-orange-100 text-orange-800 border-orange-200',
        'dining': 'bg-red-100 text-red-800 border-red-200',
        'administrative': 'bg-purple-100 text-purple-800 border-
purple-200',
    }

```

```

    'other': 'bg-gray-100 text-gray-800 border-gray-200'
  }
  return classes[type as keyof typeof classes] || classes.other
}

const getTypeLabel = (type: string) => {
  const labels = {
    'academic': 'Учебный',
    'living': 'Жилой',
    'sports': 'Спортивный',
    'dining': 'Столовая',
    'administrative': 'Административный',
    'other': 'Другое'
  }
  return labels[type as keyof typeof labels] || 'Неизвестно'
}

const getRoomTypeBadge = (type: string) => {
  const classes = {
    'classroom': 'bg-blue-50 text-blue-700',
    'lab': 'bg-purple-50 text-purple-700',
    'office': 'bg-gray-50 text-gray-700',
    'toilet': 'bg-cyan-50 text-cyan-700',
    'cafe': 'bg-orange-50 text-orange-700',
    'library': 'bg-green-50 text-green-700',
    'auditorium': 'bg-red-50 text-red-700',
    'other': 'bg-gray-50 text-gray-700'
  }
  return classes[type as keyof typeof classes] || classes.other
}

const getRoomTypeLabel = (type: string) => {
  const labels = {
    'classroom': 'Аудитория',
    'lab': 'Лаборатория',
    'office': 'Офис',
    'toilet': 'Туалет',
    'cafe': 'Кафе',
    'library': 'Библиотека',
    'auditorium': 'Актный зал',
    'other': 'Другое'
  }
  return labels[type as keyof typeof labels] || 'Неизвестно'
}

const pluralize = (count: number, one: string, few: string, many:
string) => {
  if (count % 10 === 1 && count % 100 !== 11) return one
  if (count % 10 >= 2 && count % 10 <= 4 && (count % 100 < 10 ||
count % 100 >= 20)) return few
  return many
}

```

```

</script>

<style scoped>
.building-card {
  background: rgba(255, 255, 255, 0.95);
  backdrop-filter: blur(20px);
  border: 1px solid rgba(255, 255, 255, 0.2);
  border-radius: 24px;
  box-shadow: 0 25px 50px rgba(0, 0, 0, 0.15);
  overflow: hidden;
  max-width: 400px;
  max-height: 80vh;
  overflow-y: auto;
}

.card-header {
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  padding: 24px;
  color: white;
  display: flex;
  justify-content: space-between;
  align-items: flex-start;
}

.header-content {
  flex: 1;
}

.building-title {
  font-size: 24px;
  font-weight: 700;
  margin-bottom: 12px;
  line-height: 1.2;
}

.building-meta {
  display: flex;
  align-items: center;
  gap: 12px;
  flex-wrap: wrap;
}

.type-badge {
  padding: 6px 12px;
  border-radius: 20px;
  font-size: 12px;
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.5px;
  background: rgba(255, 255, 255, 0.2);
  color: white;
  border: 1px solid rgba(255, 255, 255, 0.3);
}

```

```
}

.accessibility-indicators {
  display: flex;
  gap: 8px;
}

.indicator {
  padding: 6px;
  border-radius: 8px;
  background: rgba(255, 255, 255, 0.2);
  color: white;
  transition: all 0.2s ease;
}

.indicator:hover {
  background: rgba(255, 255, 255, 0.3);
  transform: scale(1.1);
}

.close-button {
  background: rgba(255, 255, 255, 0.2);
  border: none;
  border-radius: 12px;
  padding: 8px;
  color: white;
  cursor: pointer;
  transition: all 0.2s ease;
}

.close-button:hover {
  background: rgba(255, 255, 255, 0.3);
  transform: scale(1.1);
}

.card-content {
  padding: 24px;
}

.building-image {
  position: relative;
  margin-bottom: 20px;
  border-radius: 16px;
  overflow: hidden;
}

.building-image img {
  width: 100%;
  height: 200px;
  object-fit: cover;
}
```

```

.image-overlay {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: linear-gradient(to bottom, transparent 0%, rgba(0,
0, 0, 0.3) 100%);
}

.building-description {
  margin-bottom: 24px;
  padding: 16px;
  background: rgba(99, 102, 241, 0.05);
  border-radius: 12px;
  border-left: 4px solid #6366f1;
}

.building-description p {
  color: #374151;
  line-height: 1.6;
  margin: 0;
}

.rooms-section {
  margin-bottom: 24px;
}

.section-title {
  display: flex;
  align-items: center;
  gap: 8px;
  font-size: 18px;
  font-weight: 600;
  color: #111827;
  margin-bottom: 16px;
}

.rooms-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
  gap: 12px;
  margin-bottom: 16px;
}

.room-card {
  background: rgba(255, 255, 255, 0.8);
  border: 1px solid rgba(0, 0, 0, 0.1);
  border-radius: 12px;
  padding: 12px;
  transition: all 0.2s ease;
}

```

```
.room-card:hover {
  transform: translateY(-2px);
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);
}

.room-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 8px;
}

.room-number {
  font-weight: 600;
  color: #111827;
  font-size: 16px;
}

.room-type {
  padding: 2px 6px;
  border-radius: 6px;
  font-size: 10px;
  font-weight: 500;
  text-transform: uppercase;
}

.room-details {
  display: flex;
  align-items: center;
  gap: 8px;
  font-size: 12px;
  color: #6b7280;
}

.detail {
  background: rgba(0, 0, 0, 0.05);
  padding: 2px 6px;
  border-radius: 4px;
}

.accessibility-dot {
  width: 6px;
  height: 6px;
  border-radius: 50%;
  background: #10b981;
}

.show-more-button {
  width: 100%;
  padding: 12px;
```

```

    background: linear-gradient(135deg, rgba(99, 102, 241, 0.1) 0%,
    rgba(139, 92, 246, 0.1) 100%);
    border: 1px solid rgba(99, 102, 241, 0.2);
    border-radius: 12px;
    color: #6366f1;
    font-weight: 500;
    cursor: pointer;
    transition: all 0.2s ease;
}

.show-more-button:hover {
    background: linear-gradient(135deg, rgba(99, 102, 241, 0.2) 0%,
    rgba(139, 92, 246, 0.2) 100%);
    transform: translateY(-1px);
}

.building-stats {
    display: flex;
    gap: 16px;
    padding: 16px;
    background: rgba(0, 0, 0, 0.02);
    border-radius: 12px;
}

.stat-item {
    display: flex;
    align-items: center;
    gap: 8px;
    color: #374151;
    font-weight: 500;
}

.stat-item svg {
    color: #6366f1;
}

/* АНИМАЦИИ */
.card-slide-enter-active,
.card-slide-leave-active {
    transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
}

.card-slide-enter-from {
    opacity: 0;
    transform: translateX(100px) scale(0.9);
}

.card-slide-leave-to {
    opacity: 0;
    transform: translateX(100px) scale(0.9);
}

```

```

/* АДАПТИВНОСТЬ */
@media (max-width: 640px) {
  .building-card {
    max-width: 100%;
    border-radius: 20px 20px 0 0;
  }

  .card-header {
    padding: 20px;
  }

  .building-title {
    font-size: 20px;
  }

  .card-content {
    padding: 20px;
  }

  .rooms-grid {
    grid-template-columns: 1fr;
  }
}
</style>

```

Листинг А.5 – Файл «SearchBar.vue»

```

<template>
  <div class="search-container">
    <!-- Поисковая строка -->
    <div class="search-wrapper">
      <div class="search-input-group">
        <MagnifyingGlassIcon class="search-icon" />
        <input
          ref="searchInput"
          v-model="searchQuery"
          type="text"
          placeholder="Поиск зданий, аудиторий, удобств..."
          class="search-input"
          @input="handleInput"
          @focus="isInputFocused = true"
          @blur="handleBlur"
          @keydown="handleKeydown"
        />
        <button
          v-if="searchQuery"
          @click="clearSearch"
          class="clear-button"
          type="button"
        >
          <XMarkIcon class="w-4 h-4" />
        </button>
      </div>
    </div>
  </div>

```

```

    </div>
  </div>

  <!-- Результаты поиска -->
  <Transition name="search-dropdown">
    <div v-if="showResults" class="search-results">
      <div v-if="buildingsStore.isSearching" class="search-
loading">
        <div class="loading-spinner"></div>
        <span>Поиск...</span>
      </div>

      <div v-else-if="searchResults.length === 0 && searchQuery"
class="no-results">
        <ExclamationCircleIcon class="w-5 h-5 text-gray-400" />
        <span>Ничего не найдено для "<strong>{{ searchQuery
}}</strong>"</span>
      </div>

      <div v-else class="results-list">
        <div
          v-for="(result, index) in searchResults"
          :key="`${result.type}-${result.building?.id ||
result.amenity}-${index}`"
          :class="[
            'result-item',
            { 'highlighted': index === highlightedIndex }
          ]"
          @click="selectResult(result)"
          @mouseenter="highlightedIndex = index"
        >
          <!-- Иконка типа результата -->
          <div class="result-icon">
            <BuildingOfficeIcon
              v-if="result.type === 'building'"
              :class="getResultIconClass(result)"
            />
            <AcademicCapIcon
              v-else-if="result.type === 'room'"
              :class="getResultIconClass(result)"
            />
            <ArchiveBoxIcon
              v-else
              :class="getResultIconClass(result)"
            />
          </div>

          <!-- Информация о результате -->
          <div class="result-content">
            <div class="result-main">
              <span class="result-title">{{
getResultTitle(result) }}</span>

```

```

        <div class="result-badges">
            <span :class="getTypeBadgeClass(result.type)">
                {{ getTypeLabel(result.type) }}
            </span>
            <span v-if="result.room?.accessible"
class="accessibility-badge">
                <div class="accessibility-dot"></div>
            </span>
        </div>
    </div>
    <div class="result-details">
        <span class="building-name">{{
result.building?.name || 'Неизвестное здание' }}</span>
        <span v-if="result.room" class="room-details">
            {{ result.room.floor ? `${result.room.floor}
этаж` : '' }}
            {{ result.room.capacity ? `•
${result.room.capacity} мест` : '' }}
        </span>
    </div>
</div>
</div>

    <!-- Стрелка -->
    <ChevronRightIcon class="result-arrow" />
</div>
</div>
</div>
</Transition>
</div>
</template>

<script setup lang="ts">
import { ref, computed, watch, onMounted, onUnmounted } from 'vue'
import {
    MagnifyingGlassIcon,
    XMarkIcon,
    BuildingOfficeIcon,
    HomeIcon,
    AcademicCapIcon,
    ArchiveBoxIcon,
    ExclamationCircleIcon,
    ChevronRightIcon
} from '@heroicons/vue/24/outline'
import { useBuildingsStore } from '@stores/buildings'
import type { Building, SearchResult } from '@stores/buildings'

// Store
const buildingsStore = useBuildingsStore()

// Emits
const emit = defineEmits<{
    buildingSelect: [building: Building]

```

```

}>()

// Refs
const searchInput = ref<HTMLInputElement>()
const searchQuery = ref('')
const showResults = ref(false)
const searchTimeout = ref<number | null>(null)
const selectedResultIndex = ref(0)
const highlightedIndex = ref(0)
const isInputFocused = ref(false)

// Computed
const loading = computed(() => buildingsStore.loading)
const results = computed(() =>
buildingsStore.filteredBuildings.slice(0, 10)) // Ограничиваем до
10 результатов
const searchResults = computed(() =>
buildingsStore.filteredSearchResults)
const isSearching = computed(() => buildingsStore.isSearching)
const searchSuggestions = computed(() =>
buildingsStore.searchSuggestions)

// Methods
const handleInput = () => {
  selectedResultIndex.value = 0
  highlightedIndex.value = 0

  // Дебаунс для поиска
  if (searchTimeout.value) {
    clearTimeout(searchTimeout.value)
  }

  const query = searchQuery.value.trim()

  if (query.length === 0) {
    buildingsStore.clearSearchResults()
    showResults.value = false
    return
  }

  // Показываем результаты только если есть минимум 2 символа
  if (query.length < 2) {
    buildingsStore.clearSearchResults()
    showResults.value = false
    return
  }

  showResults.value = true

  searchTimeout.value = setTimeout(() => {
    buildingsStore.performSearchWithResults(query)
  })
}

```

```

    }, 200) as unknown as number // Уменьшили дебаунс для лучшей
    ОТЗЫВЧИВОСТИ
  }

  const clearSearch = () => {
    searchQuery.value = ''
    buildingsStore.setSearchQuery('')
    hideResults()
    searchInput.value?.focus()
  }

  const selectBuilding = (building: Building) => {
    emit('buildingSelect', building)
    hideResults()
  }

  const hideResults = () => {
    showResults.value = false
  }

  const getTypeColor = (type: string) => {
    const colors = {
      academic: 'bg-blue-500',
      living: 'bg-green-500',
      sports: 'bg-orange-500',
      dining: 'bg-red-500',
      administrative: 'bg-purple-500',
      other: 'bg-gray-500'
    }

    return colors[type as keyof typeof colors] || colors.other
  }

  const getTypeLabel = (type: string) => {
    const labels = {
      academic: 'Учебное здание',
      living: 'Общежитие',
      sports: 'Спортивное сооружение',
      dining: 'Столовая/Кафе',
      administrative: 'Административное здание',
      other: 'Другое'
    }

    return labels[type as keyof typeof labels] || labels.other
  }

  const getResultIcon = (result: SearchResult) => {
    switch (result.type) {
      case 'building':
        return BuildingOfficeIcon
      case 'room':
        return AcademicCapIcon
    }
  }

```

```

        case 'amenity':
            return ArchiveBoxIcon
        default:
            return MagnifyingGlassIcon
    }
}

const getResultIconColor = (result: SearchResult) => {
    const colors = {
        building: 'text-blue-500',
        room: 'text-green-500',
        amenity: 'text-orange-500'
    }
    return colors[result.type] || 'text-gray-400'
}

const getResultBadgeColor = (result: SearchResult) => {
    const colors = {
        building: 'bg-blue-100 text-blue-800',
        room: 'bg-green-100 text-green-800',
        amenity: 'bg-orange-100 text-orange-800'
    }
    return colors[result.type] || 'bg-gray-100 text-gray-800'
}

const getResultTypeLabel = (result: SearchResult) => {
    const labels = {
        building: 'Здание',
        room: 'Аудитория',
        amenity: 'Услуга'
    }
    return labels[result.type] || 'Неизвестно'
}

const getAccessibilityInfo = (result: SearchResult): string | null
=> {
    if (result.type === 'room' && result.room) {
        return result.room.accessible ? 'Доступно для маломобильных' :
        null
    }
    if (result.type === 'building') {
        return result.building.accessible ? 'Доступно для
        маломобильных' : null
    }
    return null
}

const getAccessibilityColor = (result: SearchResult) => {
    const accessible = result.type === 'room'
        ? result.room?.accessible
        : result.building.accessible
    return accessible ? 'bg-green-400' : 'bg-gray-300'
}

```

```

}

const selectSearchResult = (result: SearchResult) => {
  emit('buildingSelect', result.building)
  if (result.type === 'room') {
    // Дополнительная логика для выбора конкретной аудитории
    console.log('Selected room:', result.room?.number)
  }
  hideResults()
}

const applySuggestion = (suggestion: string) => {
  searchQuery.value = suggestion
  handleInput()
  searchInput.value?.focus()
}

const navigateResults = (direction: number) => {
  if (searchResults.value.length === 0) return

  selectedResultIndex.value += direction
  if (selectedResultIndex.value < 0) {
    selectedResultIndex.value = searchResults.value.length - 1
  } else if (selectedResultIndex.value >=
searchResults.value.length) {
    selectedResultIndex.value = 0
  }
}

const handleEnterSearch = () => {
  if (searchResults.value.length > 0 && selectedResultIndex.value
>= 0) {

selectSearchResult(searchResults.value[selectedResultIndex.value])
  }
}

// Обработка глобальных клавиш
const handleKeyDown = (event: KeyboardEvent) => {
  if (!showResults.value || searchResults.value.length === 0)
return

  switch (event.key) {
    case 'ArrowDown':
      event.preventDefault()
      highlightedIndex.value = Math.min(highlightedIndex.value +
1, searchResults.value.length - 1)
      break
    case 'ArrowUp':
      event.preventDefault()
      highlightedIndex.value = Math.max(highlightedIndex.value -
1, 0)

```

```

        break
      case 'Enter':
        event.preventDefault()
        if (highlightedIndex.value >= 0 && highlightedIndex.value <
searchResults.value.length) {
          selectResult(searchResults.value[highlightedIndex.value])
        }
        break
      case 'Escape':
        event.preventDefault()
        hideResults()
        searchInput.value?.blur()
        break
    }
  }

// Lifecycle
onMounted(() => {
  document.addEventListener('keydown', handleKeyDown)
  buildingsStore.fetchBuildings()
})

onUnmounted(() => {
  document.removeEventListener('keydown', handleKeyDown)
  if (searchTimeout.value) {
    clearTimeout(searchTimeout.value)
  }
})

// Watchers
watch(searchQuery, (newQuery) => {
  if (!newQuery) {
    buildingsStore.setSearchQuery('')
    hideResults()
  }
})

const getResultTitle = (result: SearchResult) => {
  return result.matchText || result.building?.name ||
result.amenity || 'Неизвестно'
}

const getResultIconClass = (result: SearchResult) => {
  const baseClasses = 'w-5 h-5'
  switch (result.type) {
    case 'building':
      return `${baseClasses} text-blue-500`
    case 'room':
      return `${baseClasses} text-green-500`
    case 'amenity':
      return `${baseClasses} text-orange-500`
    default:

```

```

        return `${baseClasses} text-gray-400`
    }
}

const getTypeBadgeClass = (type: string) => {
    return getTypeColor(type)
}

const handleBlur = () => {
    if (!searchQuery.value) {
        hideResults()
    }
}

const selectResult = (result: SearchResult) => {
    selectSearchResult(result)
}

const handleKeydown = (event: KeyboardEvent) => {
    if (!showResults.value || searchResults.value.length === 0)
    return

    switch (event.key) {
        case 'ArrowDown':
            event.preventDefault()
            highlightedIndex.value = Math.min(highlightedIndex.value +
1, searchResults.value.length - 1)
            break
        case 'ArrowUp':
            event.preventDefault()
            highlightedIndex.value = Math.max(highlightedIndex.value -
1, 0)
            break
        case 'Enter':
            event.preventDefault()
            if (highlightedIndex.value >= 0 && highlightedIndex.value <
searchResults.value.length) {
                selectResult(searchResults.value[highlightedIndex.value])
            }
            break
        case 'Escape':
            event.preventDefault()
            hideResults()
            searchInput.value?.blur()
            break
    }
}
}
</script>

<style scoped>
.search-container {
    position: relative;

```

```

    width: 100%;
    max-width: 600px;
}

/* Поисковая строка с современным дизайном */
.search-wrapper {
    position: relative;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    padding: 2px;
    border-radius: 16px;
    box-shadow: 0 10px 25px rgba(102, 126, 234, 0.15);
    transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
}

.search-wrapper:hover {
    box-shadow: 0 15px 35px rgba(102, 126, 234, 0.25);
    transform: translateY(-1px);
}

.search-input-group {
    position: relative;
    background: rgba(255, 255, 255, 0.95);
    backdrop-filter: blur(10px);
    border-radius: 14px;
    overflow: hidden;
}

.search-icon {
    position: absolute;
    left: 16px;
    top: 50%;
    transform: translateY(-50%);
    width: 20px;
    height: 20px;
    color: #6366f1;
    transition: color 0.2s ease;
}

.search-input {
    width: 100%;
    padding: 16px 50px 16px 50px;
    border: none;
    background: transparent;
    font-size: 16px;
    font-weight: 500;
    color: #1f2937;
    outline: none;
    transition: all 0.3s ease;
}

.search-input::placeholder {
    color: #9ca3af;
}

```

```

    font-weight: 400;
}

.search-input:focus {
    color: #111827;
}

.search-input:focus + .search-icon {
    color: #4f46e5;
}

.clear-button {
    position: absolute;
    right: 12px;
    top: 50%;
    transform: translateY(-50%);
    background: rgba(239, 68, 68, 0.1);
    border: none;
    padding: 8px;
    border-radius: 8px;
    cursor: pointer;
    transition: all 0.2s ease;
    color: #ef4444;
}

.clear-button:hover {
    background: rgba(239, 68, 68, 0.2);
    transform: translateY(-50%) scale(1.1);
}

/* Результаты поиска с современным дизайном */
.search-results {
    position: absolute;
    top: calc(100% + 8px);
    left: 0;
    right: 0;
    background: rgba(255, 255, 255, 0.95);
    backdrop-filter: blur(20px);
    border: 1px solid rgba(255, 255, 255, 0.2);
    border-radius: 16px;
    max-height: 400px;
    overflow: hidden;
    box-shadow: 0 20px 40px rgba(0, 0, 0, 0.1);
    z-index: 50;
}

.search-loading {
    padding: 24px;
    display: flex;
    align-items: center;
    justify-content: center;
    gap: 12px;

```

```

    color: #6366f1;
    font-weight: 500;
}

.no-results {
    padding: 24px;
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 8px;
    color: #6b7280;
}

.results-list {
    padding: 8px;
    overflow-y: auto;
    max-height: 350px;
}

.result-item {
    display: flex;
    align-items: center;
    padding: 16px;
    margin: 4px 0;
    border-radius: 12px;
    cursor: pointer;
    transition: all 0.2s cubic-bezier(0.4, 0, 0.2, 1);
    position: relative;
    overflow: hidden;
}

.result-item::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background: linear-gradient(135deg, rgba(99, 102, 241, 0.1) 0%,
    rgba(139, 92, 246, 0.1) 100%);
    opacity: 0;
    transition: opacity 0.2s ease;
}

.result-item:hover::before,
.result-item.highlighted::before {
    opacity: 1;
}

.result-item:hover {
    transform: translateX(4px);
}

```

```

.result-icon {
  position: relative;
  z-index: 1;
  padding: 8px;
  border-radius: 10px;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  margin-right: 16px;
  color: white;
  box-shadow: 0 4px 12px rgba(102, 126, 234, 0.3);
}

.result-content {
  flex: 1;
  position: relative;
  z-index: 1;
}

.result-main {
  display: flex;
  align-items: center;
  justify-content: space-between;
  margin-bottom: 4px;
}

.result-title {
  font-weight: 600;
  color: #111827;
  font-size: 16px;
}

.result-badges {
  display: flex;
  align-items: center;
  gap: 8px;
}

.result-badges span {
  padding: 4px 8px;
  border-radius: 6px;
  font-size: 12px;
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.5px;
}

.result-details {
  display: flex;
  align-items: center;
  gap: 8px;
  margin-top: 4px;
  color: #6b7280;
}

```

```

    font-size: 14px;
}

.building-name {
    font-weight: 500;
    color: #374151;
}

.room-details {
    font-size: 12px;
    color: #9ca3af;
}

.result-arrow {
    position: relative;
    z-index: 1;
    width: 20px;
    height: 20px;
    color: #9ca3af;
    margin-left: 16px;
    transition: all 0.2s ease;
}

.result-item:hover .result-arrow {
    color: #6366f1;
    transform: translateX(4px);
}

.accessibility-badge {
    display: flex;
    align-items: center;
    gap: 4px;
    padding: 4px 8px;
    background: rgba(34, 197, 94, 0.1);
    border-radius: 6px;
}

.accessibility-dot {
    width: 6px;
    height: 6px;
    border-radius: 50%;
    background: #10b981;
}

/* АНИМАЦИИ */
.search-dropdown-enter-active,
.search-dropdown-leave-active {
    transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
}

.search-dropdown-enter-from,
.search-dropdown-leave-to {

```

```

    opacity: 0;
    transform: translateY(-16px) scale(0.95);
}

.loading-spinner {
    width: 20px;
    height: 20px;
    border: 2px solid rgba(99, 102, 241, 0.2);
    border-top: 2px solid #6366f1;
    border-radius: 50%;
    animation: spin 1s linear infinite;
}

@keyframes spin {
    to {
        transform: rotate(360deg);
    }
}

/* АДАПТИВНОСТЬ */
@media (max-width: 640px) {
    .search-wrapper {
        padding: 1px;
    }

    .search-input {
        padding: 14px 45px 14px 45px;
        font-size: 16px;
    }

    .search-results {
        max-height: 300px;
    }

    .result-item {
        padding: 12px;
    }

    .result-title {
        font-size: 14px;
    }
}
</style>

```

Приложение Б
(рекомендуемое)
Результаты тестирования

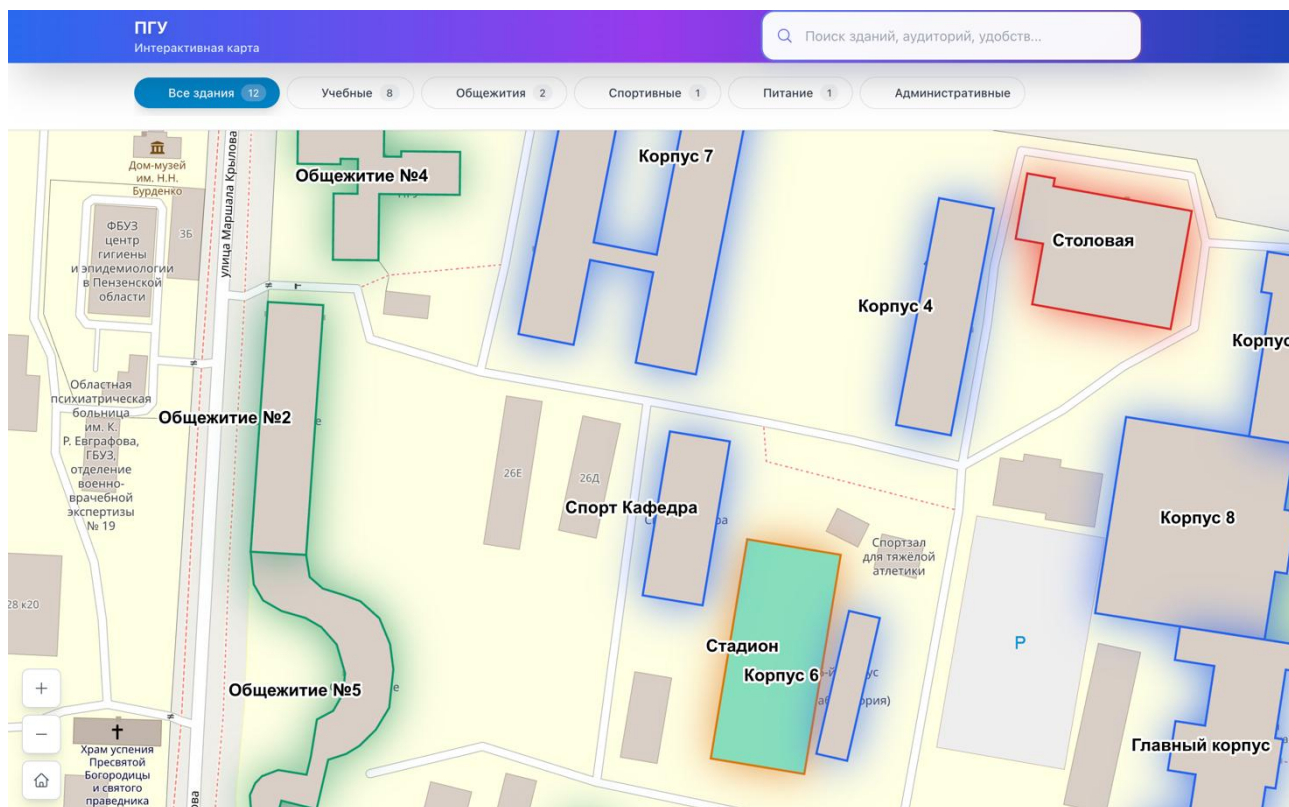


Рисунок Б.1 – Результат отображения пользовательского интерфейса

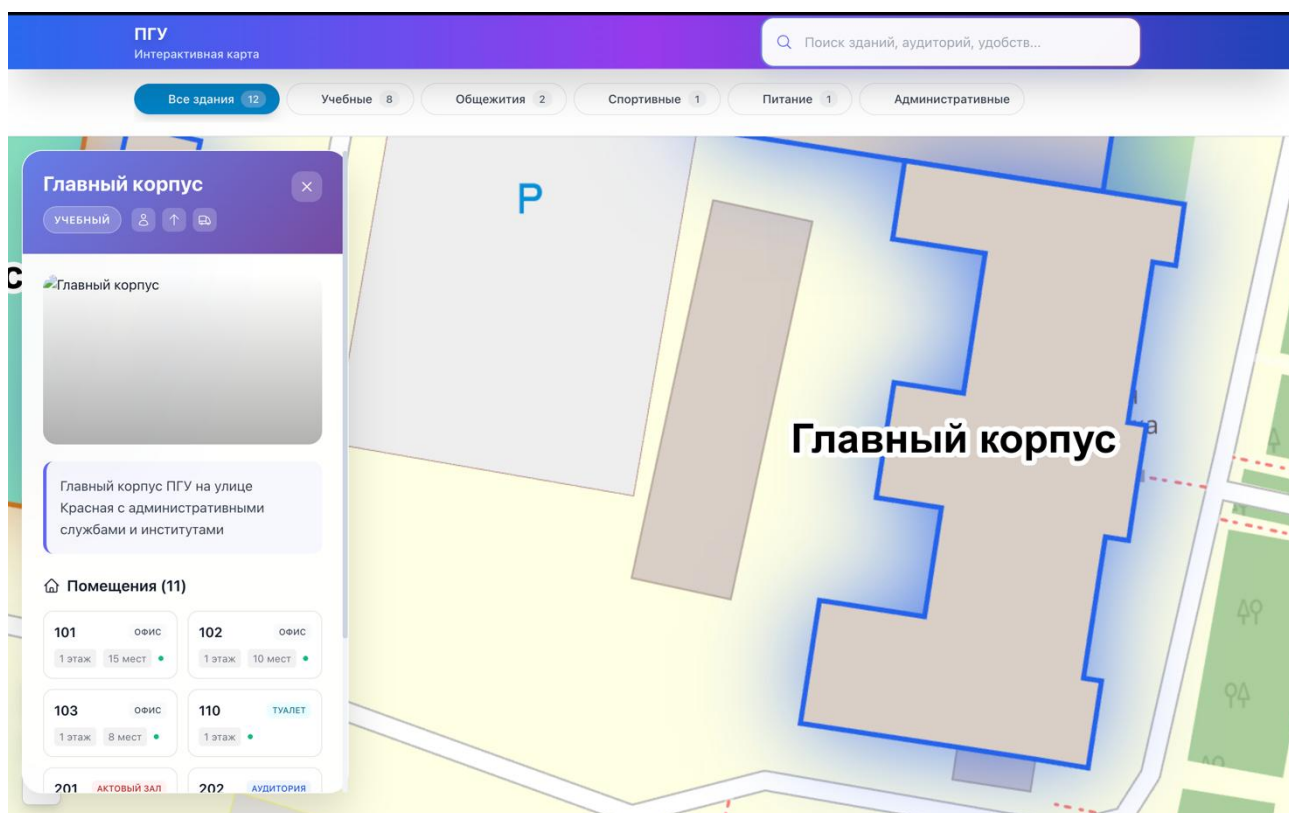


Рисунок Б.2 – Результат отображения информации о здании

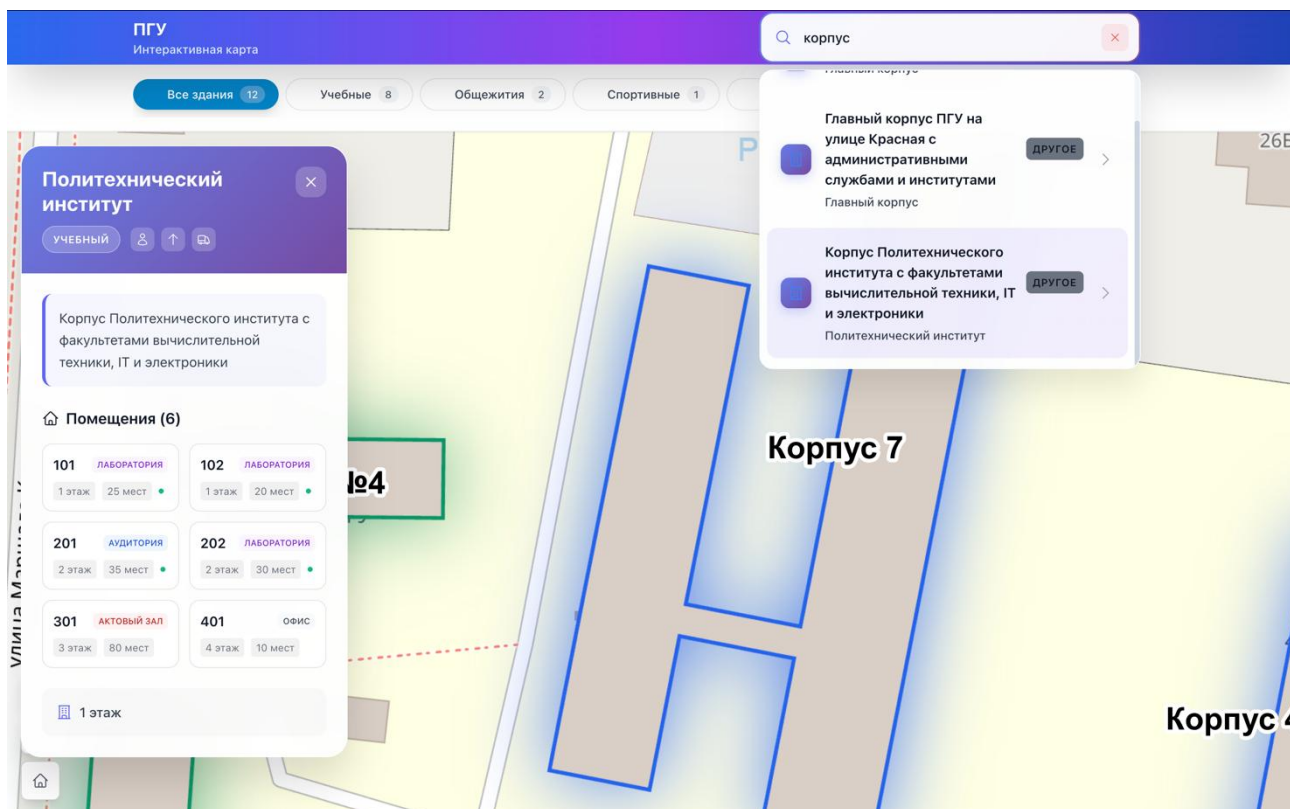


Рисунок Б.3 – Результат отображения списка зданий при поиске

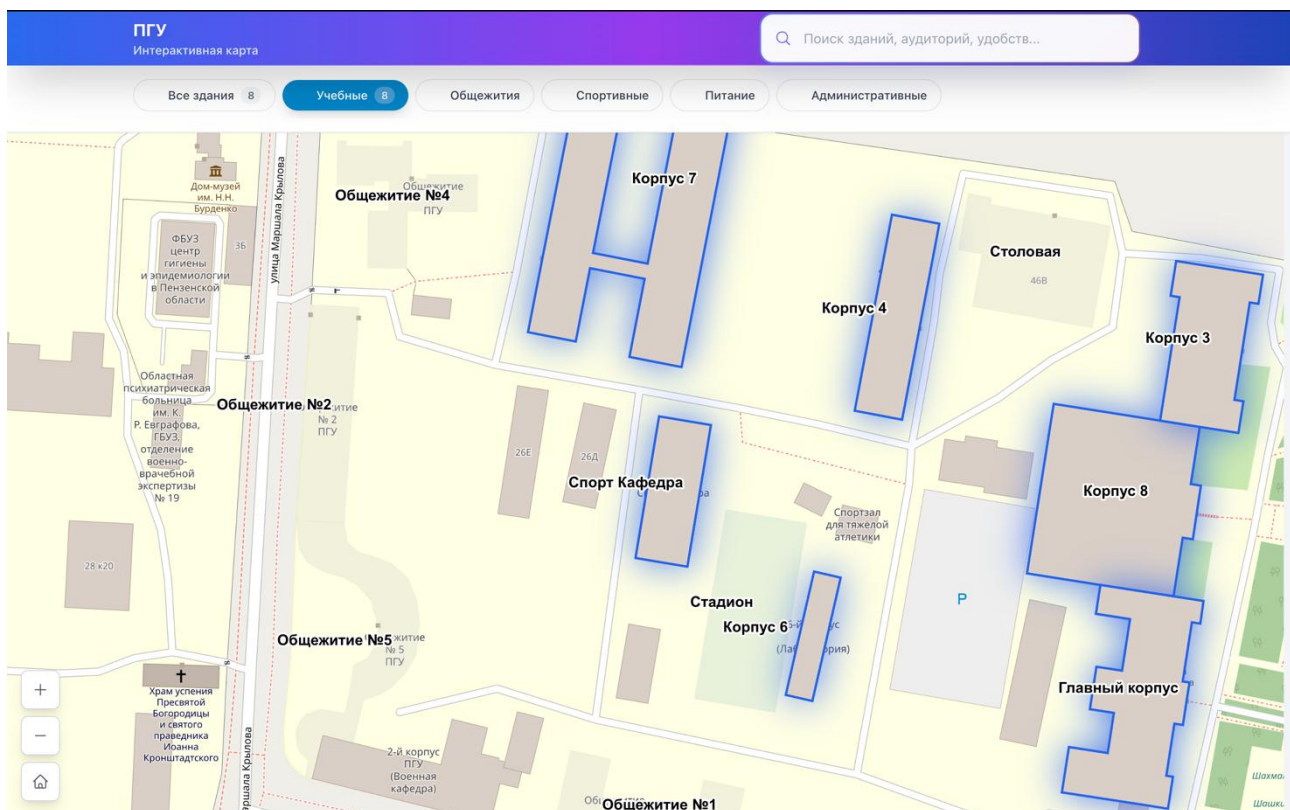


Рисунок Б.4 – Результат отображения фильтрации для только учебных зданий

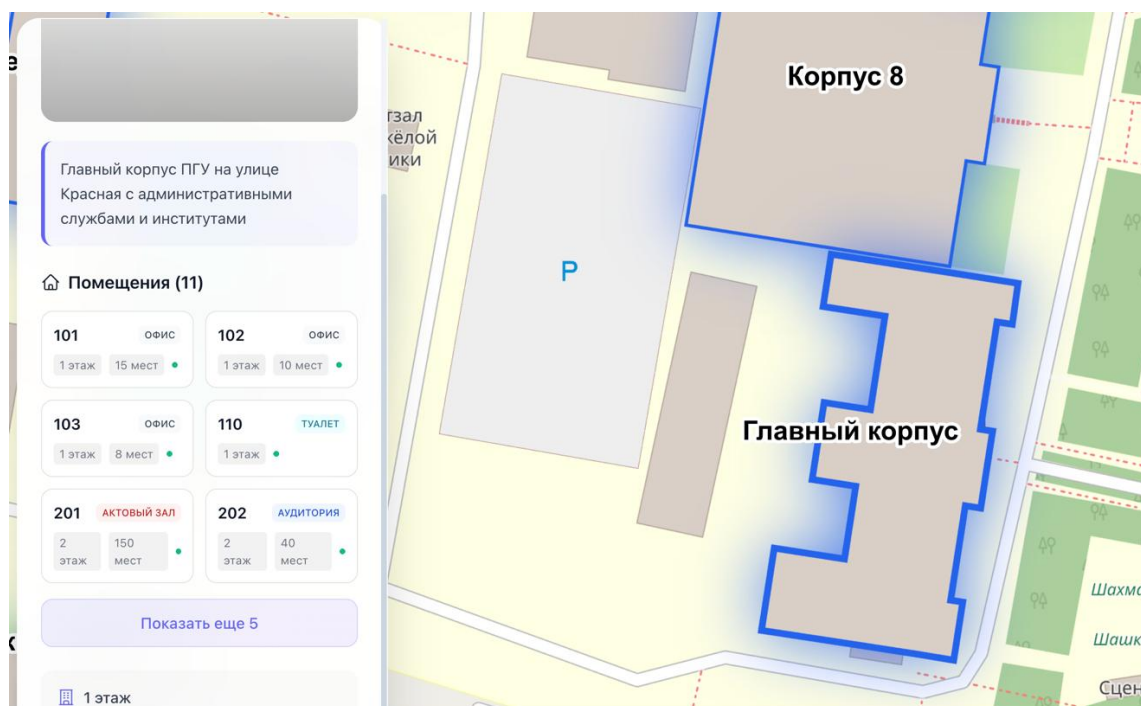


Рисунок Б.5 – Результат отображения информации об аудитории

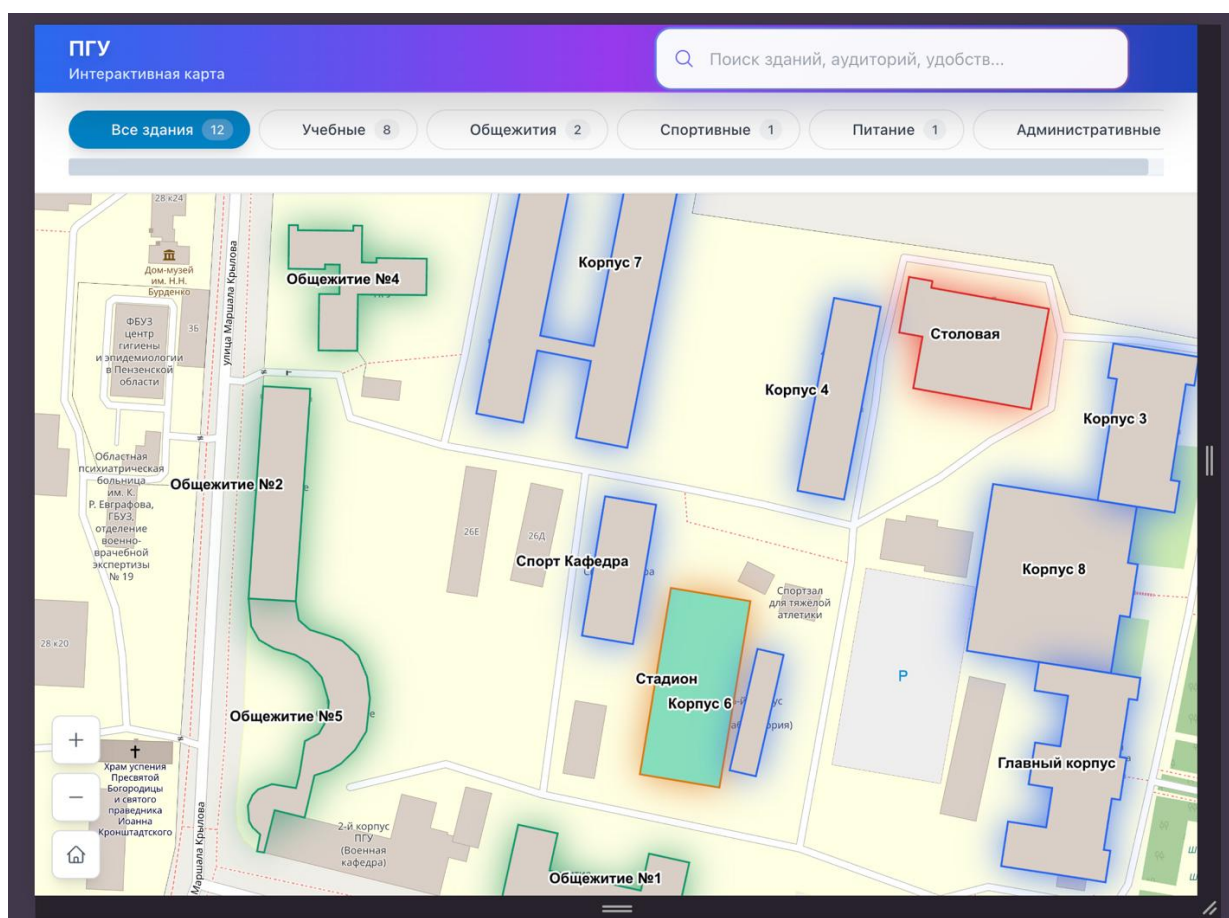


Рисунок Б.6 – Результат отображения пользовательского интерфейса при нестандартном разрешении экрана