МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ РАДИОФИЗИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
Кафедра интеллектуальных систем

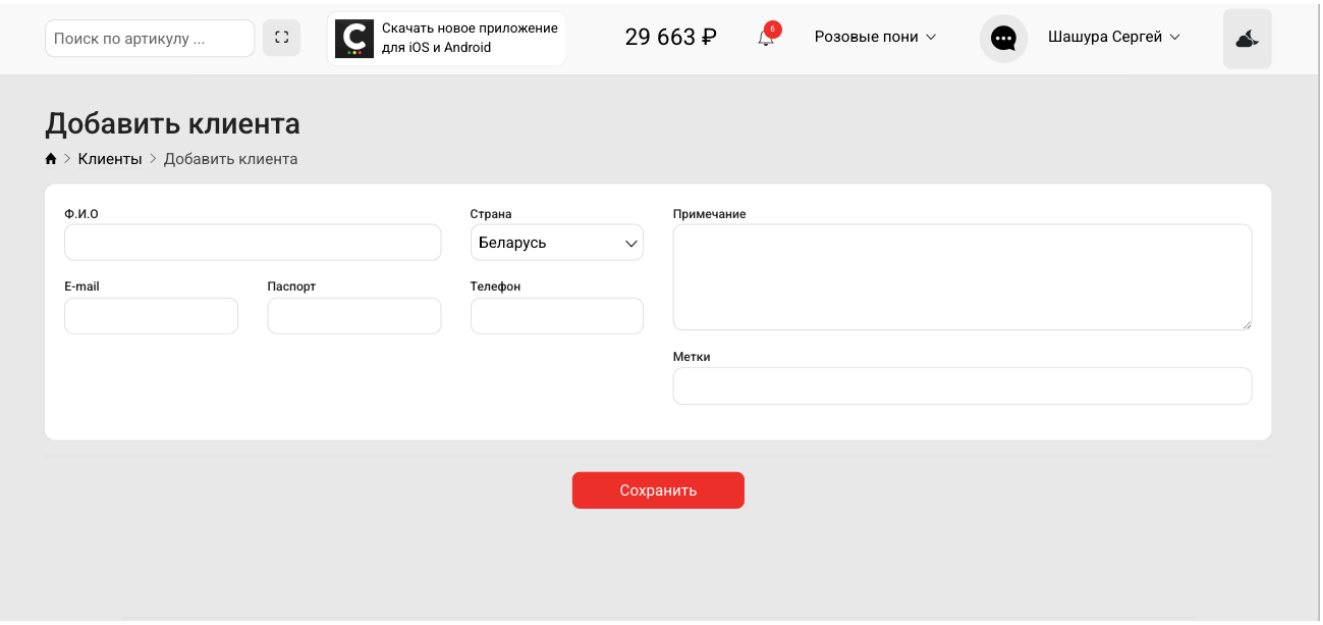
Создание прототипа интерфейса. Проектирование API и разработка архитектуры ИС

работу выполнили:
Шашура Сергей,
Гесть Анна,
студенты 3 курса, 5 группы

Минск, 2025

Цель работы:
Закрепить на практике принципы проектирования пользовательского интерфейса (UI) и взаимодействия (UX), освоить методы проектирования интерфейсов программирования (API) и применения архитектурных паттернов для разработки согласованной и масштабируемой архитектуры информационной системы.

1. Прототип пользовательского интерфейса, сценарий добавления нового клиента.



Основные шаги:

1. Пользователь открывает форму «Добавить клиента».
2. Вводит обязательные поля: *имя, номер телефона, e-mail, страна.*
3. При необходимости добавляет дополнительные поля: *описание, теги, паспортные данные.*
4. Нажимает кнопку «Сохранить».
5. Система проверяет корректность данных и уведомляет об успешном добавлении клиента.

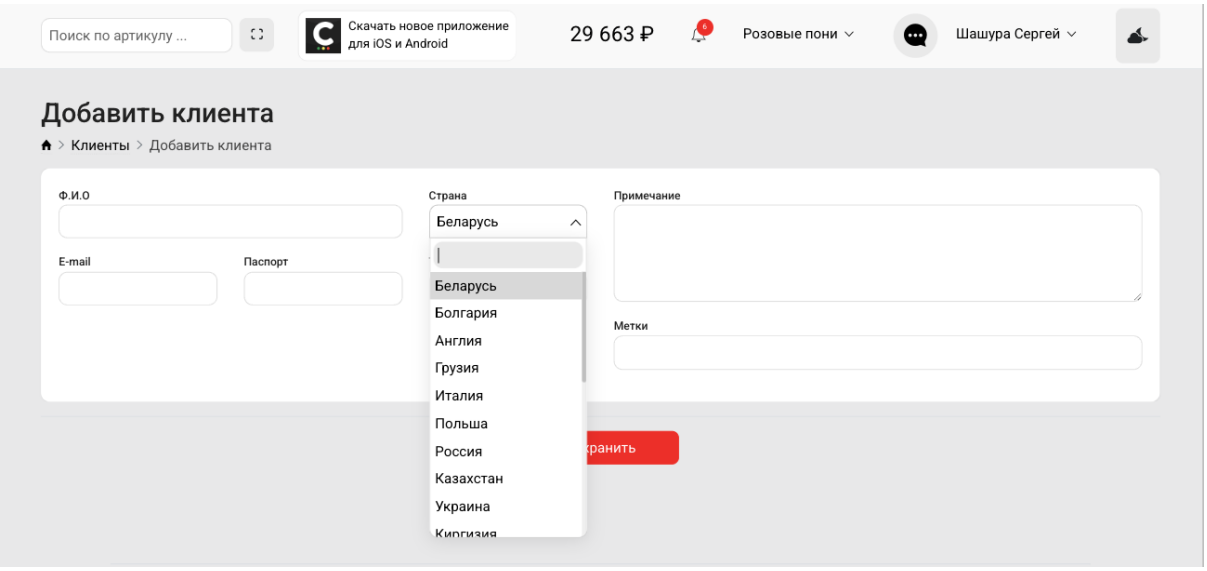
Принцип Нильсена	Как реализовано
1. Видимость состояния системы	После нажатия «Сохранить» отображается индикатор загрузки и сообщение «Клиент успешно добавлен».
2. Соответствие реальному миру	Поля названы простыми словами («Ф. И. О», «Телефон», «Email»), понятными пользователю, а не техническими терминами.
3. Пользовательский контроль и свобода	Кнопка домика или возврата позволяет выйти без сохранения;
4. Согласованность и стандарты	Все поля имеют одинаковый стиль, размеры, отступы и порядок: от наиболее обязательных к дополнительным.
5. Предотвращение ошибок	Проверка формата e-mail и номера телефона; подсветка обязательных полей и ошибок валидации.
6. Простота и эстетика	Минималистичная форма с минимумом визуальных отвлекающих элементов, единый шрифт и цветовая палитра.
7. Помощь пользователю распознать и исправить ошибку	Ошибки подсвечиваются под полем («Введите корректный e-mail»).

8. Гибкость и эффективность	Теги можно добавлять вручную или выбирать из выпадающего списка — ускоряет работу опытных пользователей.
9. Эстетика и минимализм	Используются только нужные поля; необязательные сгруппированы ниже, визуально отделены от обязательных.

Закон Хика:
Чем больше вариантов выбора, тем дольше принятие решения.

- Форма содержит только 6–7 полей, из которых обязательные 4 — это оптимальное количество для быстрого заполнения (2–3 секунды на поле).

- В выпадающих списках (например, «Страна» или «Теги») реализован поиск, чтобы не листать длинные списки вручную.



Закон Фиттса:
Время достижения цели зависит от расстояния и размера элемента.

- Кнопка «Сохранить» — крупная, выделена цветом и расположена внизу справа, где пользователь ожидает её найти.

- Все поля выровнены сеткой, чтобы курсор двигался минимально — это ускоряет ввод.

- Отступы и визуальные блоки позволяют легко сфокусировать внимание на одной зоне.

Категория	Требование
Валидация	Формат телефона и почты проверяется при вводе.- Страна выбирается только из списка.- Ошибки подсвечиваются и описываются текстом.
Адаптивность	Форма должна корректно отображаться на всех экранах.
Доступность (Accessibility)	- Контраст текста и фона не ниже 4.5:1.- Поддержка клавиатурной навигации (Tab, Enter).- Для всех полей указаны aria-label и подписи.- Ошибки должны быть доступны скринридерам (aria-live region).
Фокус и навигация	При открытии формы фокус ставится в первое поле. После сохранения, в случае ошибки — сообщение о результате с фокусом на нём.

2. Проектирование API добавления клиента в систему.

В системе можно выделить два ключевых компонента:

- Frontend (SPA) — форма «Добавление клиента».
- Backend (серверное API) — принимает данные, выполняет валидацию и сохраняет клиента в базе данных.

Подход	Преимущества	Недостатки
REST API (HTTP + JSON)	Простая реализация, высокая совместимость, кэширование, стандартные HTTP-методы.	Меньше гибкости при сложных связях между сущностями.
GraphQL	Гибкость запросов, экономия трафика.	Более сложная реализация и безопасность.
gRPC (Protocol Buffers)	Высокая производительность, строгая типизация.	Требует дополнительной настройки и бинарного протокола.

Выбор: REST API,
так как взаимодействие между frontend и backend простое (форма → сервер → база данных) в дальнейших кейсах все будет работать схожим образом.

Эндпоинт:
POST /api/v1/clients

Описание:

Создание нового клиента в системе. Принимает данные формы, валидирует их и сохраняет запись в базе данны

HTTP Метод:

POST

Тело запроса (Request Body):

Формат: JSON

Content-Type: `application/json`

```
{
  "name": "Иван Петров",
  "phone": "+375291234567",
  "email": "ivan.petrov@example.com",
  "country": 234,
  "description": "Постоянный клиент, интересуется оптовыми закупками",
  "tags": [1, 2],
  "passport": "MP1234567"
}
```

Пример корректного ответа (201 Created):

```
{
  "status": "success",
  "message": "Клиент успешно добавлен",
}
```

Пример ответа при ошибке валидации (400 Bad Request):

```
{
  "status": "error",
  "message": "Некорректные данные",
  "errors": {
    "email": "Введите корректный адрес электронной почты",
    "phone": "Неверный формат номера телефона"
  }
}
```

Пример ответа при ошибке сервера (500 Internal Server Error):

```
{
  "status": "error",
  "message": "Не удалось сохранить клиента. Повторите попытку позже."
}
```

Request Schema

```
{
  "type": "object",
  "required": ["name", "phone", "email", "country"],
  "properties": {
    "name": { "type": "string", "minLength": 2, "maxLength": 100 },
    "phone": { "type": "string", "pattern": "^\\+\\d{9,15}$" },
    "email": { "type": "string", "format": "email" },
    "country": { "type": "number" },
    "description": { "type": "string" },
    "tags": {
      "type": "array",
      "items": { "type": "number" }
    },
    "passport": { "type": "string", "maxLength": 15 }
  }
}
```

Response Schema (успешный ответ)

```
{
  "type": "object",
  "properties": {
    "status": { "type": "string", "enum": ["success"] },
    "message": { "type": "string" },
     }
  }
}
```

Response Schema (ошибка ответ)

```
{
  "type": "object",
  "properties": {
    "status": {
      "type": "string",
      "enum": ["error"]
```

```
    },
    "message": {
      "type": "string",
      "description": "Общее сообщение об ошибке"
    },
    "errors": {
      "type": "object",
      "description": "Детализированные ошибки по конкретным
полям",
       }
  },
  "required": ["status", "message"]
}
```

Код	Название
200 OK	Успешный запрос (например, при проверке клиента).
201 Created	Клиент успешно создан.
400 Bad Request	Ошибка валидации (неверные или неполные данные).
401 Unauthorized	Пользователь не авторизован для выполнения операции.
403 Forbidden	Недостаточно прав для добавления клиента.
404 Not Found	Ресурс не найден (например, API-эндпоинт указан неверно).
409 Conflict	Клиент с таким e-mail или телефоном уже существует.
422 Unprocessable Entity	Формат запроса корректен, но данные не проходят бизнес-валидацию.

500 Internal Server Error	Неожиданная ошибка сервера.

Категория	Требование
Безопасность	Доступ только для авторизованных пользователей (JWT, OAuth2).
Формат данных	Все ответы возвращаются в формате application/json.
Валидация	Проверка обязательных полей и уникальности e-mail / телефона.
Локализация	Сообщения об ошибках на языке интерфейса (например, ru, en).
Логирование	Логировать все запросы, ошибки и успешные операции.
Стандартизация	Использовать REST-названия и HTTP-коды по стандарту RFC 7231.

3. Проектирование архитектуры

1. Выбранный паттерн: MVC (Model – View – Controller)

Паттерн MVC разделяет приложение на три взаимосвязанных компонента:

- Model — бизнес-логика и данные (например, объект "Клиент" и операции сохранения в БД);
- View — пользовательский интерфейс (форма “Добавление клиента”);
- Controller — принимает запросы, управляет потоком данных между Model и View.

Критерий	Обоснование
Упрощение разработки	Компоненты изолированы: UI-разработчики работают с View, backend-разработчики — с Model и Controller.
Повышение масштабируемости	Новые функции (например, "Редактирование клиента", "Поиск клиента") легко добавляются, не затрагивая остальной код.
Улучшение тестируемости	Можно тестировать контроллеры и модели независимо от интерфейса.
Гибкость интеграции	Один и тот же backend можно использовать для разных интерфейсов — веб, мобильного приложения, API.

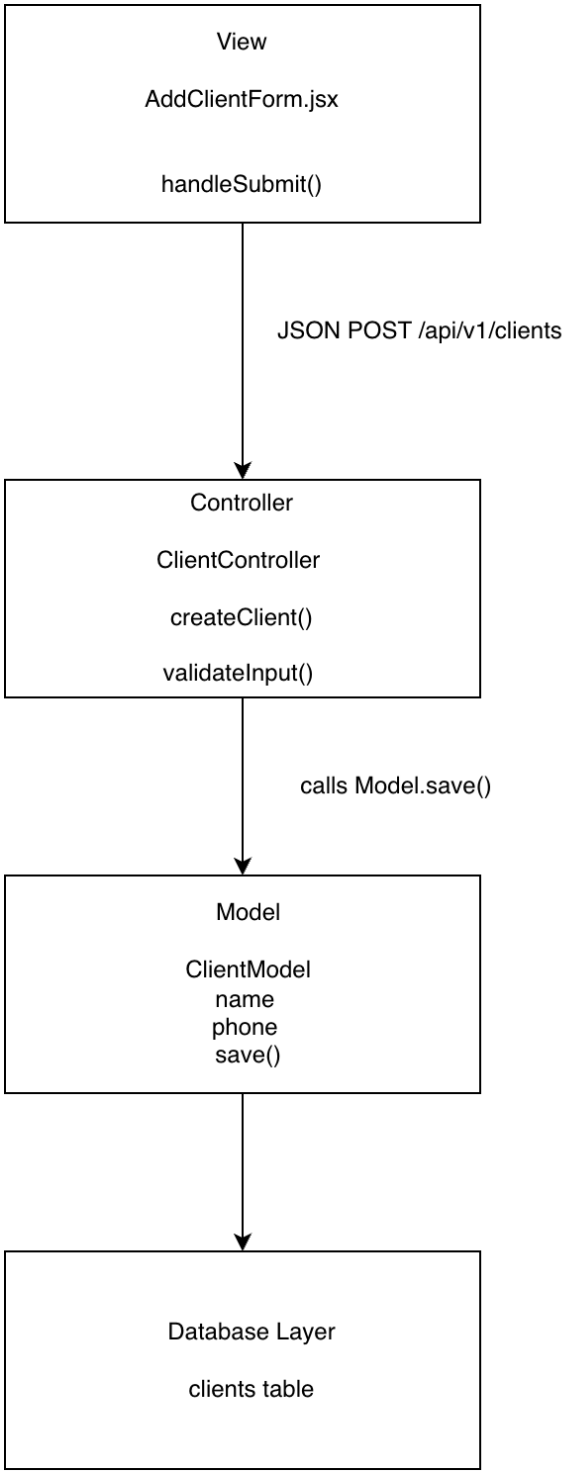
3. Применение паттерна в разрабатываемой системе

Информационная система CRM, где менеджеры работают с клиентами.
Модуль “Добавление клиента” реализуется как цепочка:
Форма (View) → Контроллер API (Controller) → Модель клиента (Model) → База данных

Пример взаимодействия

1. Пользователь открывает форму “Добавление клиента” (View).
2. Вводит данные и нажимает кнопку “Сохранить”.
3. Controller принимает запрос (POST /api/v1/clients), валидирует и вызывает Model.
4. Model создаёт запись клиента в базе данных.
5. Controller возвращает ответ (успех или ошибка) обратно во View.

4. Диаграмма


View
AddClientForm.jsx
handleSubmit()
JSON POST /api/v1/clients
Controller
ClientController
createClient()
validateInput()
calls Model.save()
Model
ClientModel
name
phone
save()
Database Layer
clients table