# Автономная некоммерческая организация

# дополнительного профессионального образования

# «Компьютерная Академия ТОП»

**Онлайн филиал «Академии ТОП» г. Ярославль**

**ДИПЛОМНАЯ РАБОТА**

**Разработка приложения**

**«Менеджер производственных заказов»**

**Студент (разработчик) В.А. Главатских**

**Группа: ВПУ011**

**Направление: Разработка программного обеспечения**

**2024**

**Проектное задание**

**Цель:**

Разработка приложения **«Менеджер производственных заказов»** для ведения производственных заказов: учет поступающих заказов, учет состояния заказа, учет стоимости работ, хранение данных об изготавливаемых изделиях.

**Обоснование актуальности задания:**

Крупные производственные предприятия имеют множество постоянно поступающих производственных заказов от разных заказчиков. Заказы проходят основные этапы: рассмотрение, принятия решения об изготовлении, оценка стоимости работ, согласования договоров на изготовление, изготовление, продажа. Серийные заказы периодически повторяются.

Для автоматизации работ по сопровождению заказов актуальна разработка программного обеспечения, которая обеспечит учет заказов, отслеживание заказа на каждом этапе работ, хранение данных об изделиях входящих в данный заказ: номер изделия, чертеж, технологию изготовления, стоимость изготовления, применяемых основных и вспомогательных материалах, оборудовании, инструменте, а также исполнителях сопровождающих текущие заказы.

**СОДЕРЖАНИЕ**

Введение …………………………………………………….. 4

1. Теоретическая часть……………………………………….... 6
2. Разработка приложения……………………..…………….... 11
   1. Структура Web-приложения…………………………
   2. Разработка базы данных……………………………..
   3. Связи таблицы базы данных…………………………
   4. Выбор локального сервера……………………………
   5. Клиентская часть Web-приложения…………………
   6. Разработка функционала Web-приложения………….
3. Заключение
4. Список используемой литературы
5. Приложение

**ВВЕДЕНИЕ**

Интегрированные информационные системы (ИС) управления промышленными предприятиями присутствуют на Российском рынке относительно недавно.

Основным назначением ERP систем является автоматизация процессов планирования, учета и управления по основным направлениям деятельности предприятия.

Системы Планирования Ресурсов Предприятия в общих чертах можно рассматривать как интегрированную совокупность следующих основных подсистем:

* Управление финансами
* Управление материальными потоками
* Управление производством
* Управление проектами
* Управление сервисным обслуживанием
* Управление качеством
* Управление персоналом

Управление современным предприятием характеризуется увеличением объемов электронного обмена данными, что приводит к развитию информационных технологий. Развитие информационных систем предприятий происходит в направлении комплексной автоматизации обработки информации. Проблема выбора информационной технологии, которая соответствует требованиям предприятия, актуальна на современном этапе развития предпринимательской деятельности

Внедрение облачных технологий в процесс обработки экономических данных фирмы предоставляет управленческому персоналу практически неограниченные возможности доступа к информационным ресурсам, обеспечивает оперативность и своевременность принятия решений.

Из-за высокой конкуренции предприятия сталкиваются с необходимостью постоянного сокращения сроков выпуска изделий и оптимизации издержек. Неверно рассчитанные ресурсы могут привести к убыткам, потере клиентов и репутации, поэтому компании стремятся автоматизировать учет производственных заказов, чтобы повысить точность и обоснованность планирования, а также обеспечить производству максимальную эффективность и гибкость.

Заказы на производство – это заявки на изготовление готовой продукции, сборочных деталей или материалов, которые выдаются внутри предприятия, и предназначаются для производственных подразделений. Они формируются на основе плана выпуска продукции или заявок, поступающих от клиентов.

Производственный заказ является центральным объектом, содержащим данные о задачах, включая сроки выполнения, количество изделий, необходимое сырье, материалы и комплектующие, перечень операций, используемого оборудования и т.д.

Производственный учет - управленческая функция, необходимая для осуществления планирования, управления и контроля процесса производства. Он охватывает процессы, происходящие внутри предприятия, а также затрагивает его связи с внешней средой - в первую очередь это касается отношений с клиентами и организации внешних поставок. Его основная цель заключается в том, чтобы соотнести предстоящие расходы с ожидаемой прибылью, рационально распределить ресурсы, сократить издержки и обеспечить рентабельность.

Учет производственных заказов позволяет обеспечить:

* своевременный выпуск продукции согласно плану производства и договоренностям с клиентами;
* равномерную загрузку мощностей, ритмичную работу и минимизацию простоев;
* сокращение материальных запасов за счет синхронизации снабжения с производственными потребностями;
* контроль затрат и снижение издержек;
* точный расчет себестоимость готовой продукции.

1. Теоретическая часть
   1. Анализ технологии для построения серверных элементов веб-приложений.

На сегодняшний день у разработчиков веб-сайтов имеется богатый выбор относительно того, какой язык (или технологию) использовать для создания сайта. Вариантов много: Perl, PHP, ASP, ASP.Net, JSP, Coldfusion и т.д. Самыми распространенными сегодня являются PHP и ASP.Net.

**ASP.NET**

ASP.NET является основной технологией для построения серверных элементов веб-приложений от Microsoft. Заинтересовать разработчиков в ASP.NET совсем нетрудно. ASP.NET является наиболее полной платформой для веб-разработки.

Она оставила далеко позади свою предшественницу ASP, которая создавалась как минимальный набор средств для вставки динамического содержимого в обычные веб-страницы. В отличие от нее, ASP.NET представляет собой полнофункциональную платформу, позволяющую создавать сложные и чрезвычайно быстрые веб-приложения.

Ниже представлены основные разделы ASP.NET:

1.Основы ASP.NET

Основная инфраструктура ASP.NET: модель веб-страниц, конфигурирование приложений и управление состоянием. Основные средства работы с данными - доступ к данным с ADO.NET для взаимодействия с реляционными базами данных, элементы управления источником данных (SqlDataSource, ObjectDataSource) и многое другое.

2. Веб-сайты ASP.NET

Основные приемы и средства для управления группами веб-страниц: пользовательские элементы управления, позволяющие повторно использовать сегменты пользовательского интерфейса; темы (позволяющие автоматически настраивать стиль элементов управления); мастер-страницы (позволяющие повторно использовать один и тот же шаблон во множестве страниц); модель навигации ASP.NET и развертывание сайта на веб-сервере.

Расширение веб-страницы с помощью дополнительных технологий: специальные элементы управления, GDI+ для создания графики вручную, JavaScript и Ajax и средство Web Parts, которое позволяет легко создавать веб-порталы.

3.[Безопасность в ASP.NET](https://professorweb.ru/my/ASP_NET/security/level1/)

Средства безопасности, доступные в ASP.NET.

Обзор концепций безопасности, аутентификация с помощью форм и работающей вместе с ней системы членства. Аутентификация Windows и ограничение авторизованных пользователей с помощью сложных правил авторизации на основе ролей. Средство профилей - готовое решение для хранения информации о пользователях и показано, как с помощью шифрования защищать информацию, хранящуюся в базе данных, а также информацию, пересылаемую в URL.

**PHP**

PHP (Hypertext Preprocessor) – язык для написания серверных сценариев (скриптов). Интерпретатор языка бесплатен, с открытым исходным кодом, созданы версии для различных веб-серверов – прежде всего, для Apache и Internet Information Services.

PHP в чистом виде – это язык, который позволяет динамически выводить HTML-разметку и иную информацию (например, заголовки для Cookie и др.)

Laravel («Ларавел» или «Ларавель») — это бесплатный PHP-фреймворк с открытым исходным кодом, специально разработанный для создания сложных сайтов и веб-приложений. Позволяет упростить аутентификацию, маршрутизацию, сессии, кэширование, архитектуру приложения, работу с базой данных.

Laravel пользуются [backend-разработчики](https://skillfactory.ru/backend-razrabotchik-na-golang?utm_source=blog&utm_medium=referral&utm_campaign=Laravel&utm_content=laravel&utm_term=text), которые пишут код на [PHP](https://blog.skillfactory.ru/glossary/php/). Он помогает определить структуру веб-приложения и служит для нее каркасом. Фреймворк написан на PHP и расширяет его возможности.

Назначение Laravel — создание веб-приложений и сайтов на основе MVC (model-view-controller). Это вариант архитектуры, при котором компоненты программы делятся на три части:

* модель (model) предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность;
* представление (view) показывает пользователю эти данные и изменяется, если меняется модель;
* контроллер (controller) направляет данные от пользователя к системе и наоборот.

## Возможности фреймворка Laravel

### Консоль Artisan

Artisan — интерфейс командной строки, включенный в Lavarel. Он позволяет генерировать модели, новые тесты, контроллеры, уведомления из командной строки. Это намного удобнее, чем копировать откуда-либо шаблон класса или писать его вручную.

Консоль используют для работы с базами данных, управления контроллером и других действий. Разработчик может написать для нее собственные команды.

### Eloquent ORM

Система объектно-реляционного отображения (ORM) Eloquent - это технология программирования для связи базы данных и языка программирования. Использование ORM позволяет ускорить разработку. Для PHP существует множество реализаций ORM, но Laravel пользуется собственной. Она называется Eloquent и работает по схеме ActiveRecord, согласно которой каждой таблице в базе соответствует один класс. Eloquent любят за удобство и возможность писать понятный код, который легко поддерживать, защиту от SQL-инъекций — способа взлома сайтов и программ. ORM можно загрузить отдельно от остальных компонентов Laravel.

### Конструктор Fluent

Это конструктор для быстрого построения запросов к базе, который полностью совместим с ядром Eloquent ORM. Fluent — мощный инструмент, позволяющий с легкостью модифицировать записи в базе данных. Все запросы используют подготовленные выражения и защищены от SQL-инъекций.

### Шаблонизатор Blade

Шаблонизаторы используются для превращения HTML-шаблонов в готовые страницы. Шаблоны — это заготовки для будущих веб-страниц, которые включают HTML-верстку без контента и PHP-код. Задача программы-шаблонизатора — выполнить PHP-код и подставить в шаблон контент, чтобы превратить его в готовую страницу. PHP в итоговой веб-странице быть не должно.

Blade — это шаблонизатор фреймворка Laravel. Он не имеет ограничений на чистый PHP в шаблонах — это удобнее для backend-разработчика. Отсутствие ограничений не создает дополнительной нагрузки на приложение.

### Валидация

Валидация — проверка входящих данных. Она нужна, чтобы случайная ошибка была вовремя найдена и обработана. Логику валидации можно написать самостоятельно. Если пользователь введет неверные данные, он попадет на предыдущую страницу. Сайт не «упадет» и останется работоспособным.

### Система контроля версий баз данных

Это позволяет проводить миграции — модификации структуры базы данных. С помощью фреймворка можно гибко управлять миграцией прямо через Artisan: запускать, откатывать по отдельности или полностью, видоизменять.

### Юнит-тестирование

Его еще называют модульным тестированием. Это «базовый уровень» проведения тестов. На работоспособность тестируются отдельные модули кода. Это нужно, чтобы позже не столкнуться с ошибкой на более высоких уровнях.

### Аутентификация

С Laravel можно реализовать вход на сайт с помощью логина и пароля либо через социальные сети.

Например, пакеты Laravel Passport и Laravel Socialite позволяют входить через API других приложений с помощью стандарта OAuth. Это протокол авторизации, который дает возможность авторизоваться с помощью учетной записи другого сайта, чаще всего соцсети.

Пакет Laravel Sanctum отвечает за аутентификацию для простых приложений и одностраничных сайтов. Он более легковесный, чем Passport, и авторизует пользователя с помощью системы токенов — физических устройств.

* 1. Сравнение технологий

### Для новичка

Для разработчика с небольшим опытом лучше всего подойдет PHP. На этом языке написать простой сайт можно довольно быстро — так вы сразу увидите результат. В освоении этот язык тоже значительно проще.

К плюсам можно отнести и мощность этого языка. В нем доступны базы данных, объектно-ориентированное программирование, работа с файлами и изображениями и т.д. PHP может стать отличным плацдармом для дальнейшего развития и, возможно, перехода на более сложные технологии.

Из минусов этого языка — для него маловато учебных пособий и профессиональной литературы. В основном, знания придется получать из интернета, форумов и общения с опытными коллегами.

### Для опытных разработчиков

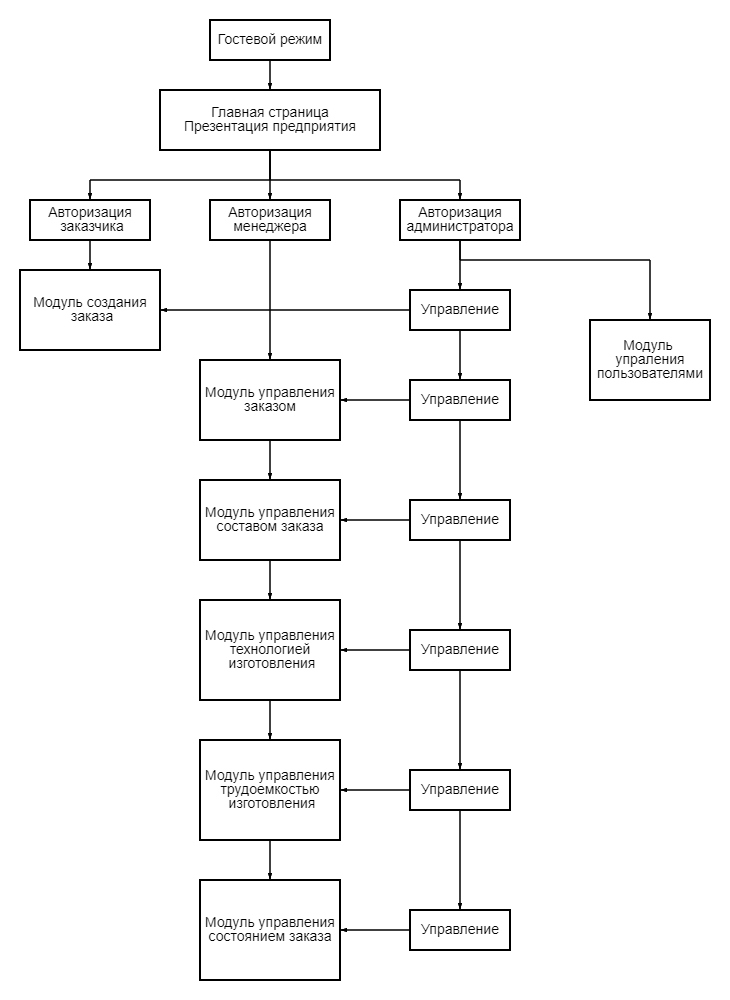
Для тех, кто хорошо знаком с программированием, лучше подойдет ASP.NET MVC. Эта технология — для тех, кто работает в долгую и не ожидает быстрых результатов.

Здесь есть подробная и понятная документация, которая поможет разобраться во всех деталях. Также по ASP.NET немало полезной литературы, которая поможет лучше понять эту технологию.

## Выбор технологии для проектирования

## Для удобства проектирования, применения готовых технических решений системы аутентификации, контроля доступа к ресурсам, обеспечения безопасности приложения и высокой скорости разработки воспользуемся фреймворком Laravel.

1. Разработка приложения.
   1. Структура Web-приложения



* 1. Разработка базы данных

Базы данных планирования заказов предприятия, является основой данного приложения и содержит важную информацию  о бизнес-операциях.

База данных представляет собой структурированный набор данных, специально разработанный для поддержки программного обеспечения для планирования ресурсов предприятия. Она служит основой для хранения и организации различных типов информации, необходимой для бесперебойного функционирования управлением заказами организации.

|  |  |  |
| --- | --- | --- |
| Список сущностей | |  |
| № | Название | Назначение |
| 1 | organization | Данные о предприятии-изготовителе. |
|  |  |  |
| Список атрибутов таблицы Organization | | |
| Ключевое поле | Название | Назначение |
| ПК | код | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
|  | Название | Название организации |
|  | Адрес | Юридический/почтовый адрес организации |
|  | password | Пароль для авторизации в приложении |
|  | Описание | Описание возможностей предприятия. Презентация. |
|  | тел приемной | Контакты предприятия |
|  | email |

|  |  |  |
| --- | --- | --- |
| № | Название | Назначение |
| 2 | customer | Данные о предприятии-заказчике. |
|  |  |  |
| Список атрибутов таблицы Customer | | |
| Ключевое поле | Название | Назначение |
| ПК | код | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
|  | Название | Название организации |
|  | Адрес | Юридический/почтовый адрес организации |
|  | password | Пароль для авторизации в приложении |
|  | Описание | Описание возможностей предприятия. Презентация. |
|  | тел приемной | Контакты предприятия |
|  | email |

|  |  |  |
| --- | --- | --- |
| № | Название | Назначение |
| 3 | user | Данные о менеджерах предприятия-изготовителя. |
| Список атрибутов таблицы User | | |
| Ключевое поле | Название | Назначение |
| ПК | код | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
|  | name | Login пользователя |
|  | email | Email пользователя |
|  | password | Пароль для авторизации в приложении |

|  |  |  |
| --- | --- | --- |
| № | Название | Назначение |
| 4 | Admin\_users | Данные об администраторах. |
| Список атрибутов таблицы Admin\_users | | |
| Ключевое поле | Название | Назначение |
| ПК | код | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
|  | name | Login пользователя |
|  | email | Email пользователя |
|  | password | Пароль для авторизации в приложении |

|  |  |  |
| --- | --- | --- |
| № | Название | Назначение |
| 5 | orders | Данные о заказе. |
| Список атрибутов таблицы Orders | | |
| Ключевое поле | Название | Назначение |
| ПК | код | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
| ВК | customer\_id | Внешний ключ для связи с организацией-заказчиком |
|  | organization\_id | Внешний ключ для связи с организацией-изготовителем |
|  | number | номер письма от заказчика |
|  | data | дата создания входящего письма |
|  | text | описание заказа |
|  | desiredDate | сроки выполнения заказа |
|  | letter | Файл-письмо заказчика |

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название | Назначение | |
| 6 | designdocument | Для каждого изделия создается документация, которая имеет обозначение, наименование  и подразделяется на чертеж , спецификацию, если это сборка, электронную модель и т.д\*/ | |
| Список атрибутов таблицы Designdocument | | | |
| Ключевое поле | Название | | Назначение |
| ПК | код | | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
|  | designation | | обозначение конструкторского документа |
|  | name | | наименование изделия |
|  | text | | описание заказа |
|  | type | | enum('type',['спецификация','чертеж','модель','нормаль','покупная','другое']);//тип документа |
|  | letter | | Файл-письмо заказчика |

|  |  |  |
| --- | --- | --- |
| № | Название | Назначение |
| 7 | user\_details | Подробности о пользователях |
| Список атрибутов таблицы User\_details | | |
| Ключевое поле | Название | Назначение |
| ПК | код | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
| ВК | user\_id | Внешний ключ для связи с user |
| ВК | organization\_id | Внешний ключ для связи с организацией-изготовителем |
|  | firstname | ФИО |
|  | fathername |
|  | lastname |
|  | jobtitle | должность |
|  | workphone | телефон |
|  | personalphone | сотовый телефон |

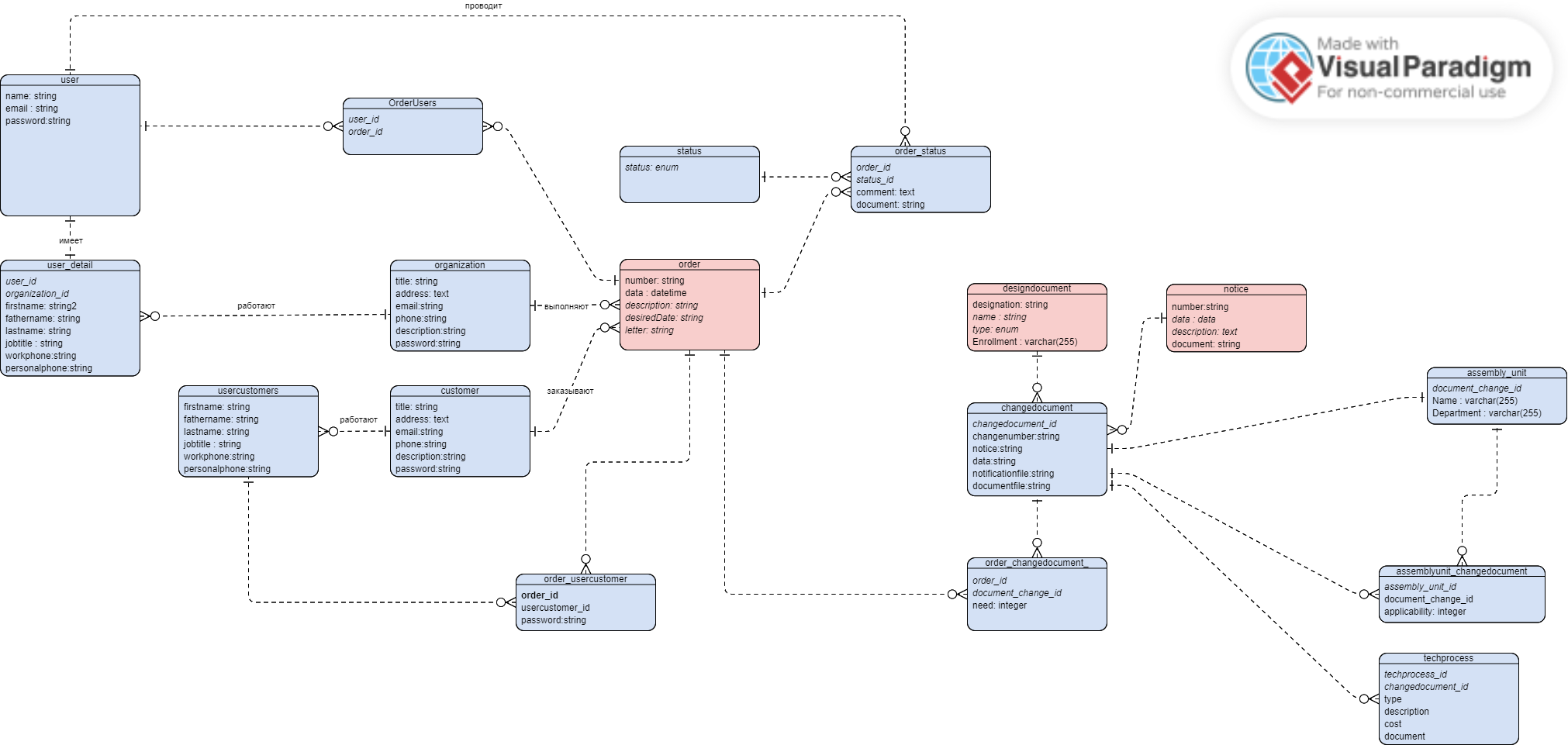
|  |  |  |
| --- | --- | --- |
| № | Название | Назначение |
| 8 | user\_customers | Подробности о менеджерах-заказчика. Они не зарегистрированы в системе и не приязаны таблице user |
| Список атрибутов таблицы User\_details | | |
| Ключевое поле | Название | Назначение |
| ПК | код | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
| ВК | user\_id | Внешний ключ для связи с user |
| ВК | customer\_id | Внешний ключ для связи с организацией-заказчика |
|  | firstname | ФИО |
|  | fathername |
|  | lastname |
|  | jobtitle | должность |
|  | workphone | телефон |
|  | personalphone | сотовый телефон |

|  |  |  |
| --- | --- | --- |
| № | Название | Назначение |
| 9 | notices | Извещения об изменении конструкторской документации |
| Список атрибутов таблицы notices | | |
| Ключевое поле | Название | Назначение |
| ПК | код | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
|  | designation | Обозначение извещения |
|  | data | дата выпуска извещения |
|  | description | Описание изменения |
|  | documentfile | Ссылка на документ - извещение |

|  |  |  |
| --- | --- | --- |
| № | Название | Назначение |
| 10 | techprocesses | Техпроцесс на изготовление изделия |
| Список атрибутов таблицы notices | | |
| Ключевое поле | Название | Назначение |
| ПК | код | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
| ВК | changedocument\_id | Внешний ключ для связи с конструкторским документов и актуальной ревизией |
|  | type | Тип технологии |
|  | description | Описание техпроцесса |
|  | document | Ссылка на документ - техпроцесс |

|  |  |  |
| --- | --- | --- |
| № | Название | Назначение |
| 11 | Assemblyunit | Изделие changedocument (чертеж) может относиться к сборочной единице (сборке) и состоять из нескольких деталей |
| Список атрибутов таблицы Assemblyunit | | |
| Ключевое поле | Название | Назначение |
| ПК | код | Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. |
| ВК | changedocument\_id | Внешний ключ для связи с конструкторским документов и актуальной ревизией |

<https://online.visual-paradigm.com/w/yxbpnnlb/drive/#infoart:proj=0&dashboard>



* 1. Связи таблицы базы данных.

Laravel содержит [ORM-библиотеку](https://ru.wikipedia.org/wiki/ORM) Eloquent, предоставляющую способ работы с базой данных, который часто удобнее обычного построителя запросов. При использовании Eloquent каждая таблица БД имеет соответствующую «Модель», которая используется для взаимодействия с этой таблицей. Помимо получения записей из таблицы БД, модели Eloquent также позволяют вставлять, обновлять и удалять записи из таблицы.

Отношение «один-ко-многим» используется для определения отношений, в которых одна модель является родительской для одной или нескольких дочерних моделей.

Order – Customer – один-ко-многим

|  |  |  |
| --- | --- | --- |
| Отношение 1 ко многим | Сущность/связь | Реализация в Laravel |
| 1. Нам нужно реализовать некую БД, которая ведет учет данных о заказах | Order | |  | | --- | |  |   Дочерняя модель Order  public function customer()      {         return $this->belongsTo(Customer::class); } |
| 2.У заказа есть заказчик | Customer | Родительская модель Customer  public function orders()  { return $this->hasMany(Order::class); } |
| 3. При этом у каждого заказчика может быть несколько заказов | много | Установка внешнего ключа в миграции Order  $table->foreignId('customer\_id') ->constrained()->cascadeOnDelete()->cascadeOnUpdate(); |
| 4. Но нельзя сказать, что заказ принадлежит нескольким заказчикам Заказ принадлежит одному заказчику | 1 |  |

Order – Organization – один-ко-многим

|  |  |  |
| --- | --- | --- |
| Отношение 1 ко многим | Сущность/связь | Реализация в Laravel |
| 1. Нам нужно реализовать некую БД, которая ведет учет данных о заказах | Order | |  | | --- | |  |   Дочерняя модель Order  public function customer()      {         return $this->belongsTo(  Organization  ::class); } |
| 2.У заказа есть изготовитель | Organization | Родительская модель Organization  public function orders()  { return $this->hasMany(Order::class); } |
| 3. При этом у каждого изготовителя может быть несколько заказов | много | Установка внешнего ключа в миграции Order  $table->foreignId('Organization\_id') ->constrained()->cascadeOnDelete()->cascadeOnUpdate(); |
| 4. Но нельзя сказать, что заказ принадлежит нескольким изготовителям  Заказ принадлежит одному изготовителю | 1 |  |

Customer - UserСustomer – один-ко-многим

|  |  |  |
| --- | --- | --- |
| Отношение 1 ко многим | Сущность/связь | Реализация в Laravel |
| Работник заказчика может быть не зарегистрирован в БД. Но нам надо хранить о нем данные о ФИО, должности и принадлежности к заказчику.  В дальнейшем для просмотра заказов исполнители -заказчика будут входить под паролем заказчика |  |  |
| Нужна новая сущность | UserСustomer | Дочерняя модель UserСustomer   public function customer()  { return $this->belongsTo(Customer::class); } |
| UserСustomer работает в организации Customer | Customer | Родительская модель Customer  public function usercustomers()  { return $this->hasMany(UserCustomer::class); } |
| Customer может иметь много UserСustomer | много | Установка внешнего ключа в миграции UserСustomer  $table->foreignId('customer\_id')->constrained()->cascadeOnDelete()->cascadeOnUpdate(); |
| UserСustomer принадлежит одному заказчику Customer | 1 |  |

User – UserDetail – один-ко-одному

|  |  |  |
| --- | --- | --- |
| Отношение 1 ко 1 | Сущность/связь | Реализация в Laravel |
| У нас есть user , который работает только на одном предприятии изготовителе  Необходимо добавить ему дополнительные данные ФИО и принадлежность к изготовителю  Создадим таблицу user\_detail,  с связью 1 к 1 с user |  | user :  public function userdetail()  { return $this->hasOne(UserDetail::class); }  UserDetail:  public function user()  { return $this->belongsTo(User::class); }  Установка внешнего ключа в миграции  $table->foreignId('user\_id')->unique()->constrained()->cascadeOnDelete()->cascadeOnUpdate();  $table->foreignId('organization\_id')->constrained()->cascadeOnDelete()->cascadeOnUpdate(); |
| Отношение 1 ко многим | | |
| Данные ФИО и принадлежность к Organization для user | UserDetail | Дочерняя модель   public function organization()  {  return $this->belongsTo(Organization::class);  } |
| В Organization имеет user (UserDetail) | Organization | Родительская модель   public function userdatails()  { return $this->hasMany(UserDetail::class); } |
| Изготовитель Organization может иметь много user (UserDetail) | много |  |
| User работает в одной Organization | 1 |  |

Order- User - OrderUsers - «многие-ко-многим»

|  |  |  |
| --- | --- | --- |
| Отношения «многие-ко-многим» | Сущность/связь | Реализация в Laravel |
| Заказ имеет исполнителей организации исполнителя, которые ведут данный заказ |  |  |
| Исполнитель может участвовать во многих заказа | User | Определить метод в связанной модели User:  public function orders()  { return $this->belongsToMany(Order::class);} |
| Заказ ведут многие исполнители технологи, экономисты, менеджеры и т.д | Order | Родительская модель  public function users()  { return $this->belongsToMany(Order::class);} |
| Нужна таблица посредник OrderUsers | OrderUsers | Для работы с отношениями «многие-ко-многим» требуется наличие промежуточной таблицы  Schema::create('order\_users', function (Blueprint $table) { $table->id();  $table->foreignId('user\_id')->constrained()->cascadeOnDelete()->cascadeOnUpdate();  $table->foreignId('order\_id')->constrained()->cascadeOnDelete()->cascadeOnUpdate(); |

Order- Status- OrderStatus - многие-ко-многим

|  |  |  |
| --- | --- | --- |
| Отношения «многие-ко-многим» | Сущность/связь | Реализация в Laravel |
| Заказ будет пререходить в разный статус: 'прием','резолюзия','отказ','номенклатура',  'нормирование','согласование','договор','  исполнение','контроль','отгрузка' |  |  |
| у заказа может быть много статусов | Order / много | Родительская модель  public function statuses()  {  return $this->belongsToMany(Status::class);  } |
| Один и тот же статус может быть у разных заказов | Status / много | Метод в связанной модели  public function orders()  {  return $this->belongsToMany(Order::class);  } |
| Для связи многие ко многим нужна таблица посредник OrderStatus | OrderStatus | Schema::create('order\_statuses', function (Blueprint $table) {  $table->id();  $table->foreignId("order\_id")->constrained()->cascadeOnDelete()->cascadeOnUpdate();  $table->foreignId("status\_id")->constrained()->cascadeOnDelete()->cascadeOnUpdate();  $table->foreignId("user\_id")->constrained()->cascadeOnDelete()->cascadeOnUpdate();  $table->text('comment')->nullable();; // комментарий исполнителя  $table->string('document')->nullable();; // письмо заказчика с визой директора, номенклатура, калькуляция.  $table->timestamps();  }); |

|  |  |  |
| --- | --- | --- |
| Отношения «многие-ко-многим» | Сущность/связь | Реализация в Laravel |
| 1.Каждый заказ будет состоять из одного или более изделий, имеющих свой чертеж.  2.Также чертеж может иметь изменения.  3. Возможно надо изготовить несколько деталей по старому изменению, а несколько по новому. |  |  |
| Начальные сведения о чертеже: номер, наименование, тип | designdocument |  |
| 1. У одного документа может быть от одного и более изменений  2.Изменения проводятся извещениями |  |  |
| Сведения о извещении: номер, дата, содержание, документ | notice |  |
| 1.Извещение будет изменять определенный документ или несколько документов |  |  |
| Сущность для хранения сведений об изменениях документа,  т.е о его версиях | changedocument |  |
| Заказ состоит из одного или более изделий, обозначенных через документ changedocumentID (чертеж) |  |  |
| changedocumentID (чертеж) может входить в разные заказы | order |  |
| Связь многие ко многим, таблица посредник ChangedocumentOrder | ChangedocumentOrder |  |

Структура хранения и учета конструкторской документации в базе данных.

Changedocument – Designdocument – один-ко-многим

|  |  |  |
| --- | --- | --- |
| Отношение 1 ко многим | Сущность/связь | Реализация в Laravel |
| 1. Нам нужно реализовать некую БД, которая ведет учет данных о конструкторских документах и их изменениях.  Сущность Changedocument будет иметь реквизиты указанные в Designdocumen и будет иметь порядковый номер изменения.  В сущности это новый конструкторский документ с новыми параметрами изделия. | Changedocument | |  | | --- | |  |   Дочерняя модель Changedocument  public function designdocument()  {  return $this->  belongsTo(Designdocument::class);  } |
| 2. Changedocument связана с сущностью designdocument, которая хранит реквизиты документа | Designdocument | Родительская модель Designdocument public function changedocuments()  {  return $this->  hasMany(Changedocument::class)->  orderBy("created\_at");  } |
| 3. При этом к исходному документу (designdocument) может быть проведено много изменений (Changedocument) | много | Установка внешнего ключа в миграции Changedocument  $table->foreignId('designdocument\_id')->constrained()->cascadeOnDelete()->cascadeOnUpdate(); |
| 4. Но нельзя сказать, что изменения документа (Changedocument) принадлежит нескольким исходным документам (Designdocument) Changedocument принадлежит одному Designdocument | 1 |  |

Changedocument – Notice – один-ко-многим

|  |  |  |
| --- | --- | --- |
| Отношение 1 ко многим | Сущность/связь | Реализация в Laravel |
| 1. Нам нужно реализовать некую БД, которая ведет учет данных о конструкторских документах и их изменениях.  Сущность Changedocument | Changedocument | |  | | --- | |  |   Дочерняя модель Changedocument  public function notice ()  {  return $this->belongsTo(Notice::class);  } |
| 2. Changedocument связана с сущностью Notice, которая хранит информацию о том что было изменено в конструкторском документе | Notice | Родительская модель Notice  public function changedocuments()  {  return $this->  hasMany(Changedocument::class);  } |
| 3. При этом к извещение о проведений изменений в конструкторском документе может распространяться на несколько конструкторских документов (Changedocument) | много | Установка внешнего ключа в миграции Changedocument  $table->foreignId('notice\_id')->constrained()->cascadeOnDelete()->cascadeOnUpdate(); |
| 4. Но нельзя сказать, что изменения документа (Changedocument) принадлежит нескольким извещениям об изменений. Изменения в Changedocument проводят только по одному извещению Notice. | 1 |  |

Changedocument – Assemblyunit – один-ко-одному

|  |  |  |
| --- | --- | --- |
| Отношение 1 ко 1 | Сущность/связь | Реализация в Laravel |
| У нас есть Changedocument, может относиться к сборочной единице  Необходимо добавить ему дополнительноу поле о принадлежности сборке  Создадим таблицу Assemblyunit,  с связью 1 к 1 с Changedocument |  | Changedocument :  public function assemblyunit()  {  return $this->  hasOne(Assemblyunit::class)->  withDefault();  }  Установка внешнего ключа в миграции  $table->  foreignId('changedocument\_id')->  unique()->constrained()->  cascadeOnDelete()->  cascadeOnUpdate(); |

Changedocument - Assemblyunit - AssemblyunitChangedocument - многие-ко-многим

|  |  |  |
| --- | --- | --- |
| Отношения «многие-ко-многим» | Сущность/связь | Реализация в Laravel |
| Changedocument может входить в разные сборочные единицы Assemblyunit |  |  |
| у Assemblyunit может быть много разных Changedocument | Assemblyunit / много | Родительская модель  public function changedocuments()  {  return $this->belongsToMany(Changedocument::class);  } |
| Один и тот же Changedocument может быть у разных Assemblyunit | Changedocumen / много | Метод в связанной модели  public function assemblyunits()  {  return $this->belongsToMany(Assemblyunit::class);  } |
| Для связи многие ко многим нужна таблица посредник assemblyunit\_changedocument  Поле 'applicability' устанавливает применяемость (количесво) входящих деталей (Changedocument) в сборочную единицу (assemblyunit) | Assemblyunit  \_changedocument | Schema::create('assemblyunit\_changedocument', function (Blueprint $table) {  $table->id();  $table->  foreignId('changedocument\_id')->  constrained()->cascadeOnDelete()->  cascadeOnUpdate();  $table->  foreignId('assemblyunit\_id')->  constrained()->cascadeOnDelete()->  cascadeOnUpdate();  $table->integer('applicability'); //количество деталей в сборке  $table->timestamps();  }); |

Структура учета изделий в заказе и их потребности

Order- Changedocument - ChangedocumentOrder - многие-ко-многим

|  |  |  |
| --- | --- | --- |
| Отношения «многие-ко-многим» | Сущность/связь | Реализация в Laravel |
| Заказ Order будет состоять из нескольких изделий, требования по изготовлению этих изделий указаны в Changedocument |  |  |
| У заказа Order может быть много разных Changedocumen.  Требуемое количество изделий для изготовления по заказу указано в поле 'need' | Order / много | Родительская модель  public function changedocuments()  {  return $this->  belongsToMany(Changedocument::class,'changedocument\_orders')->  withPivot('need','id');  } |
| Один и тот же Changedocumen входить в разные заказы Order | Changedocumen / много | Метод в связанной модели  public function orders()  {  return $this->belongsToMany(Order::class);  } |
| Для связи многие ко многим нужна таблица посредник ChangedocumentOrder | ChangedocumentOrder | Schema::create('changedocument\_order', function (Blueprint $table) {  $table->id();  $table->  foreignId('changedocument\_id')->  constrained()->cascadeOnDelete()->  cascadeOnUpdate();  $table->foreignId('order\_id')->  constrained()->cascadeOnDelete()->  cascadeOnUpdate();  $table->integer('need'); // требуемое количество изделий  $table->timestamps();  }); |

* 1. Выбор локального сервера

Локальный сервер – специальная программа, благодаря которой Web-разработчики имеют возможность разрабатывать Web-сайты на локальном

(домашнем) компьютере, без необходимости выхода в сеть Интернет.

Необходимость в локальном сервере возникает при создании динамических Web-сайтов или интерактивных Web-приложений, которые используют язык программирования PHP. Это связано с тем, что Web-браузеры понимают толькоHTML, CSS и JS, но не PHP.

Преимущества «Open Server»:

* приятный и удобный интерфейс;
* не требует установки;
* надежность и уверенность в работе;
* интуитивное управление доменами;
* Понятное руководство пользователя.
  1. Клиентская часть Web-приложения

Веб-приложение состоит из клиентской и серверной частей, тем самым

реализуя технологию «клиент-сервер». Клиентская часть реализует

пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него. Серверная часть получает запрос от клиента, выполняет вычисления, после этого формирует веб-страницу и отправляет её клиенту по сети с использованием протокола HTTP.

Основой клиентской части является процесс верстки. Этот процесс

считается следующим этапом после подготовки макета дизайна Web-приложения.

Сама верстка делится на три основных шага:

* нарезка макета дизайна на отдельные составляющие;
* создания всех необходимых файлов для проекта в форматах \*.html и\*.css;
* написание в них специальных конструкций на языках HTML и CSS.

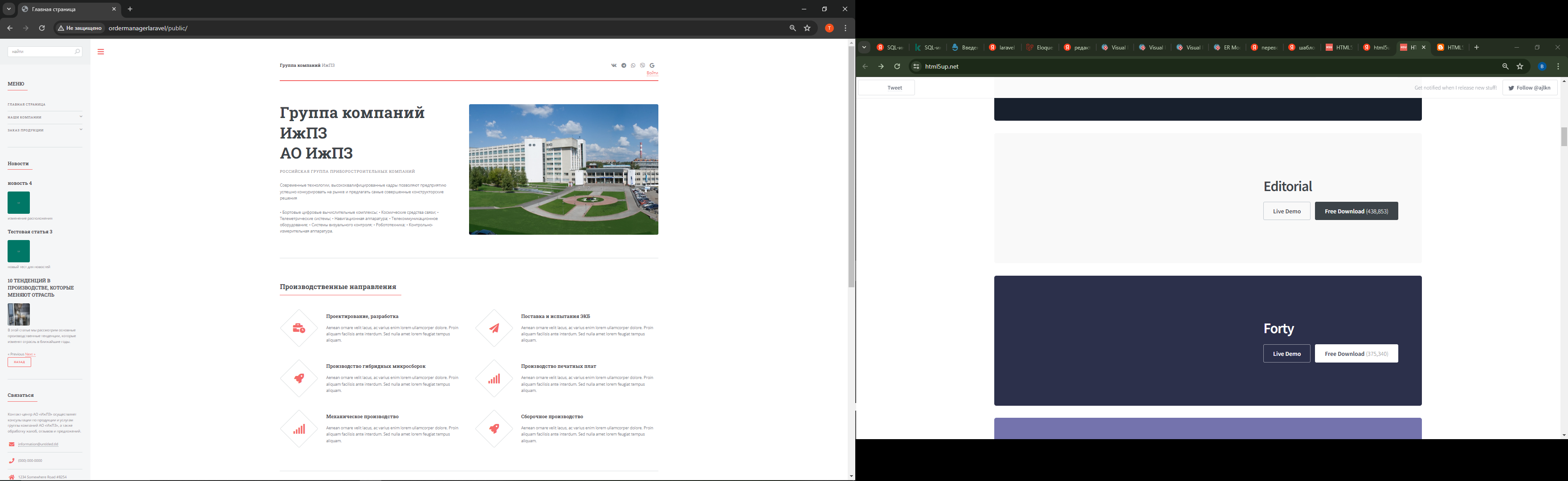
Выберем дизайн приложения из готовых предложений шаблонов

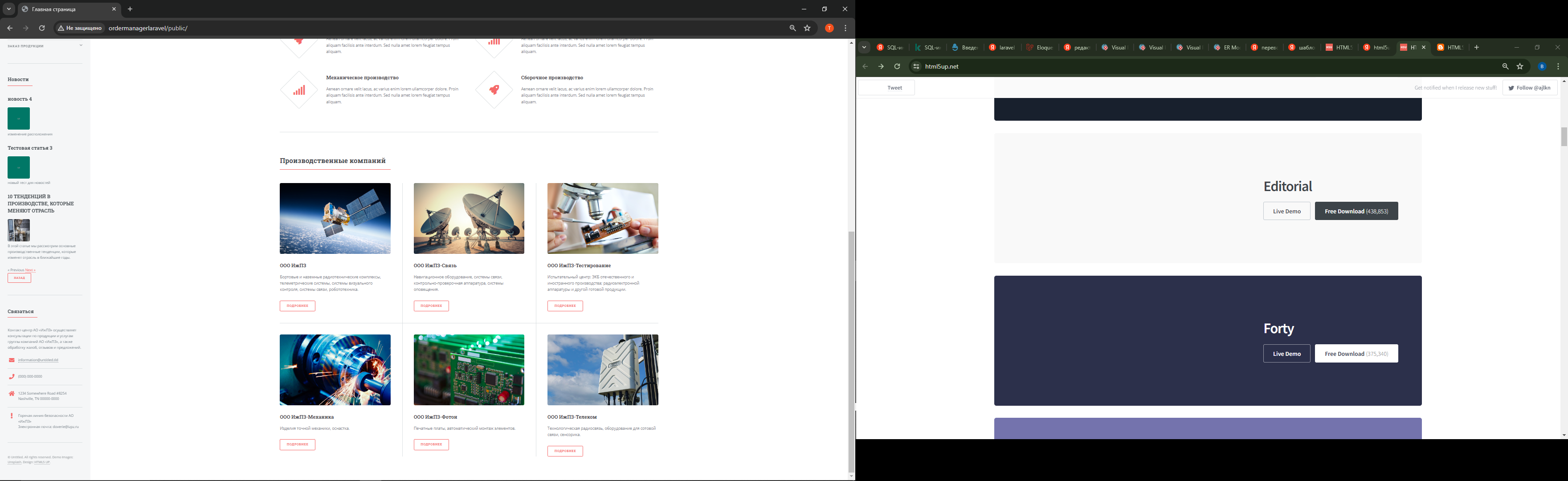
с сайта [HTML5UP](http://html5up.net/) . [**Editorial** by HTML5 UP](https://html5up.net/uploads/demos/editorial/index.html) - https://html5up.net/editorial

[HTML5UP](http://html5up.net/) – проект, в рамках которого для бесплатного использования создаются прекрасные, современные, респонсивные HTML5+CSS3 шаблоны.

* + 1. Дизайн главной страницы приложения

Приложение разработано для ведения заказов организации, в структуре которой несколько компаний. Каждая организация-изготовитель имеет свой юридический статус, свое направление по изготовлению продукции. Приложение обеспечивает разделение и ведение заказов для конкретного изготовителя.

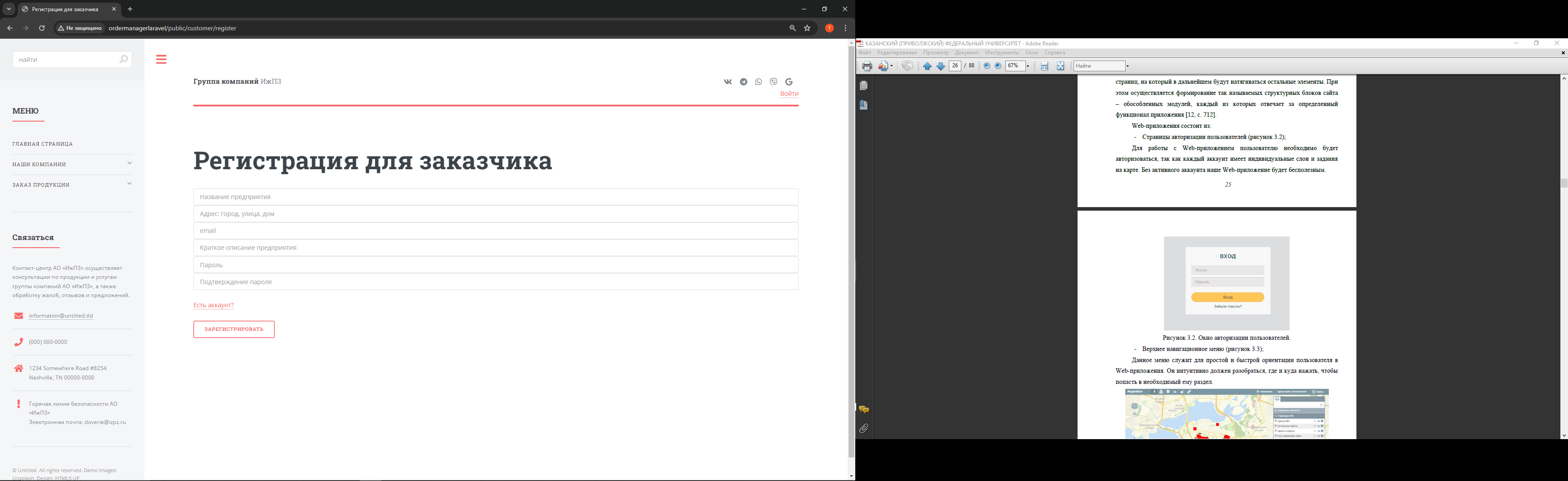


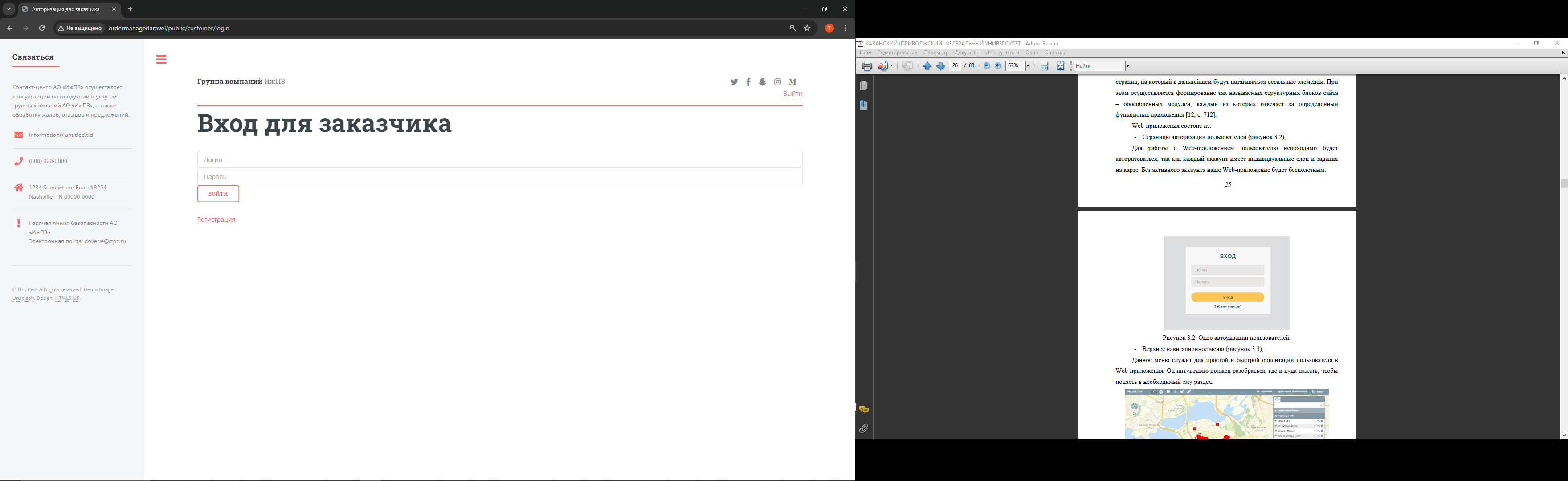


* + 1. Авторизация

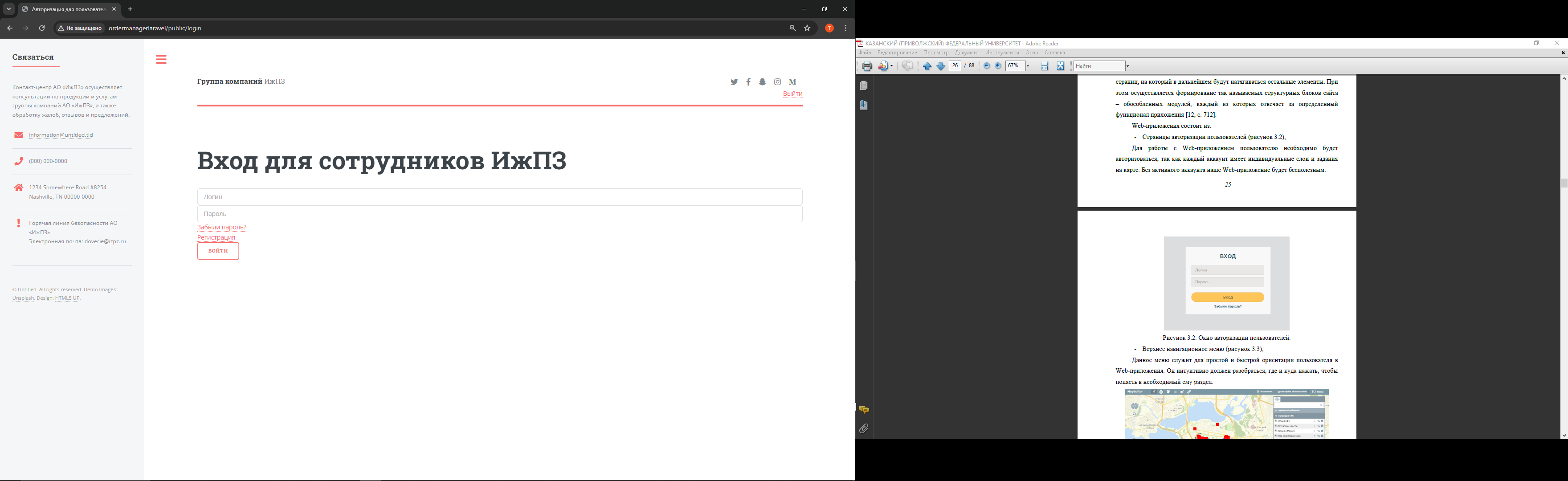
Приложение разработана для четырех групп пользователей: незарегистрированный пользователь, заказчик, работник(менеджер) предприятия –изготовителя и администратор

* Регистрация и авторизация заказчика.

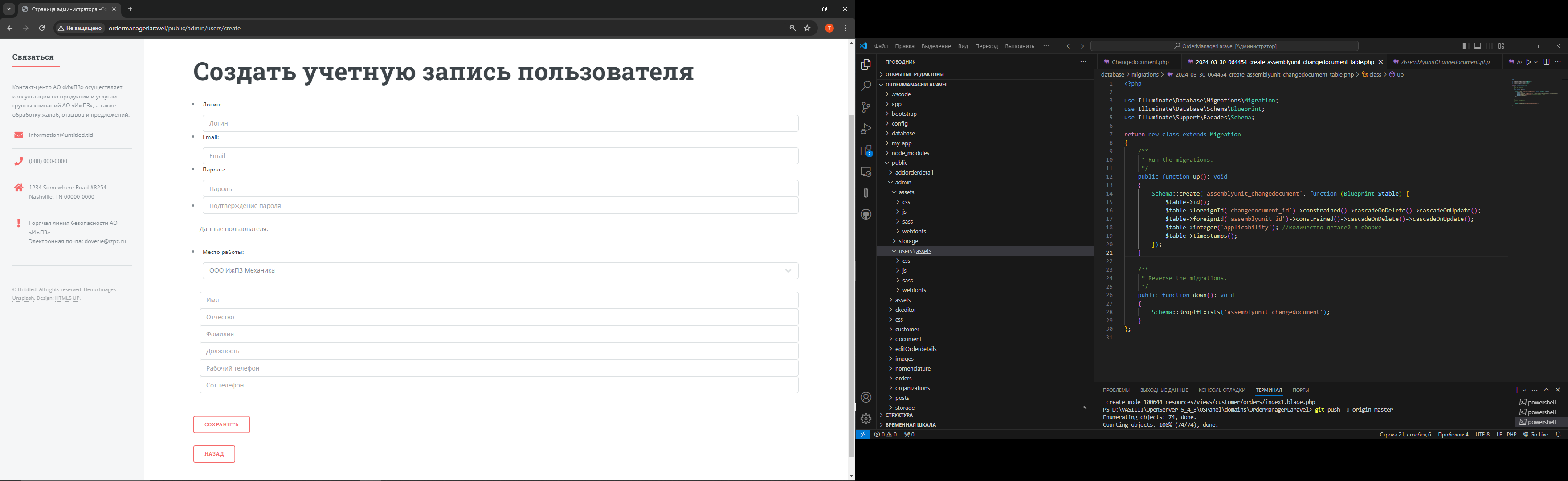




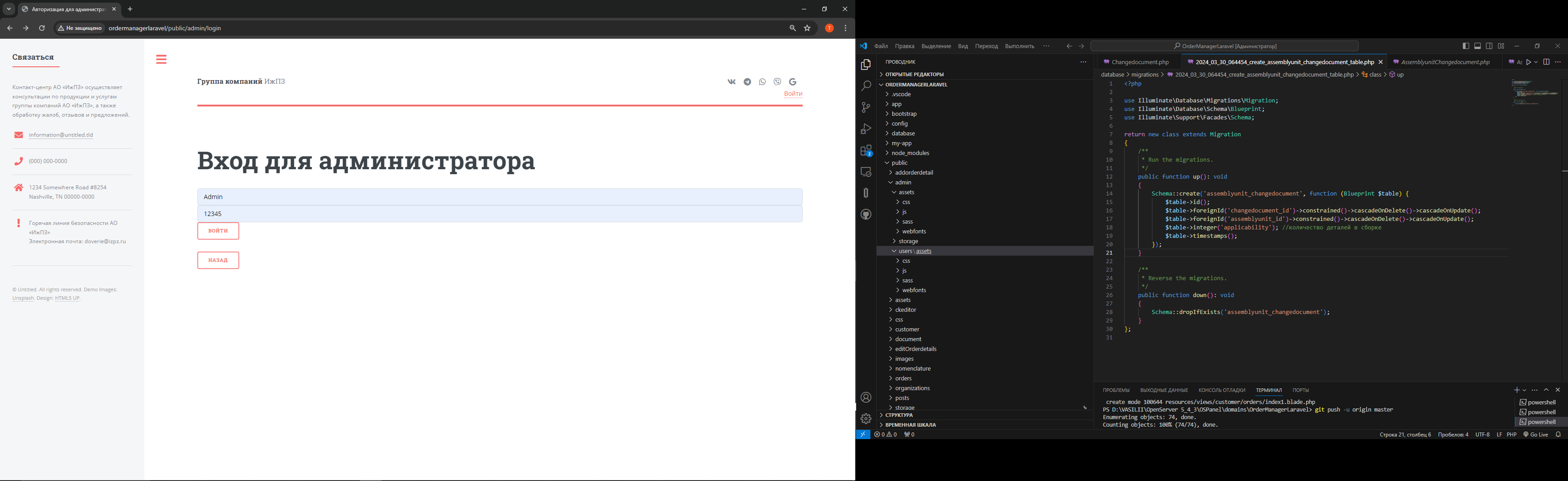
* Авторизация и регистрация менеджеров предприятия



За регистрацию менеджеров предприятия отвечает администратор сайта (приложения). Пароль и логин формируются при приеме работника на предприятие.

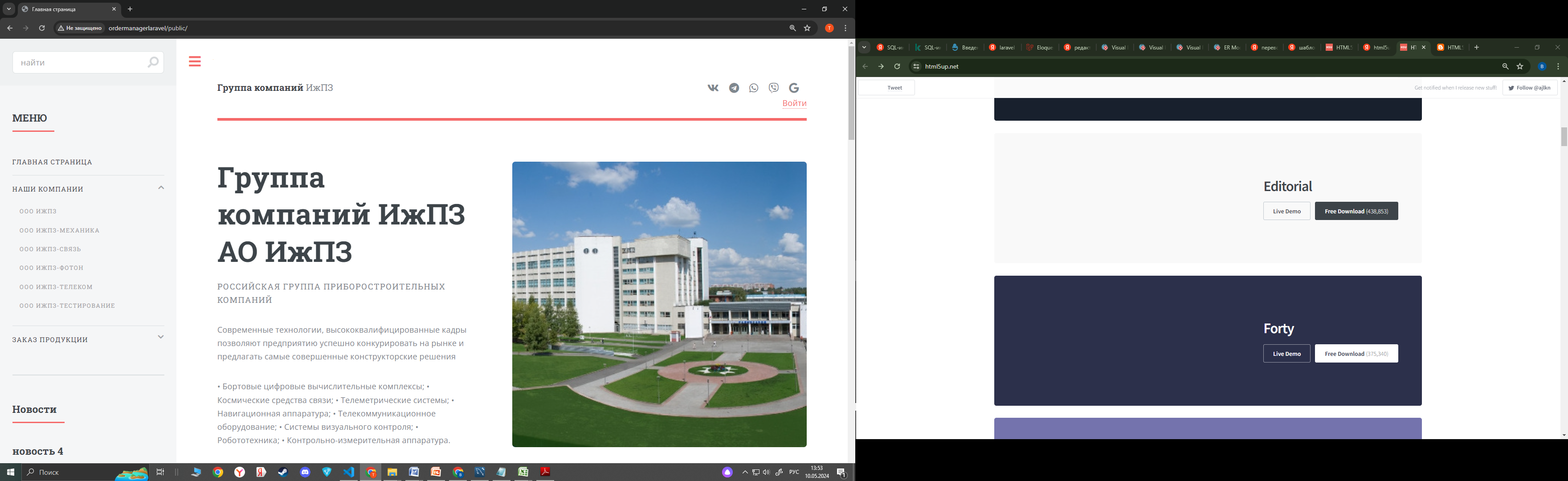
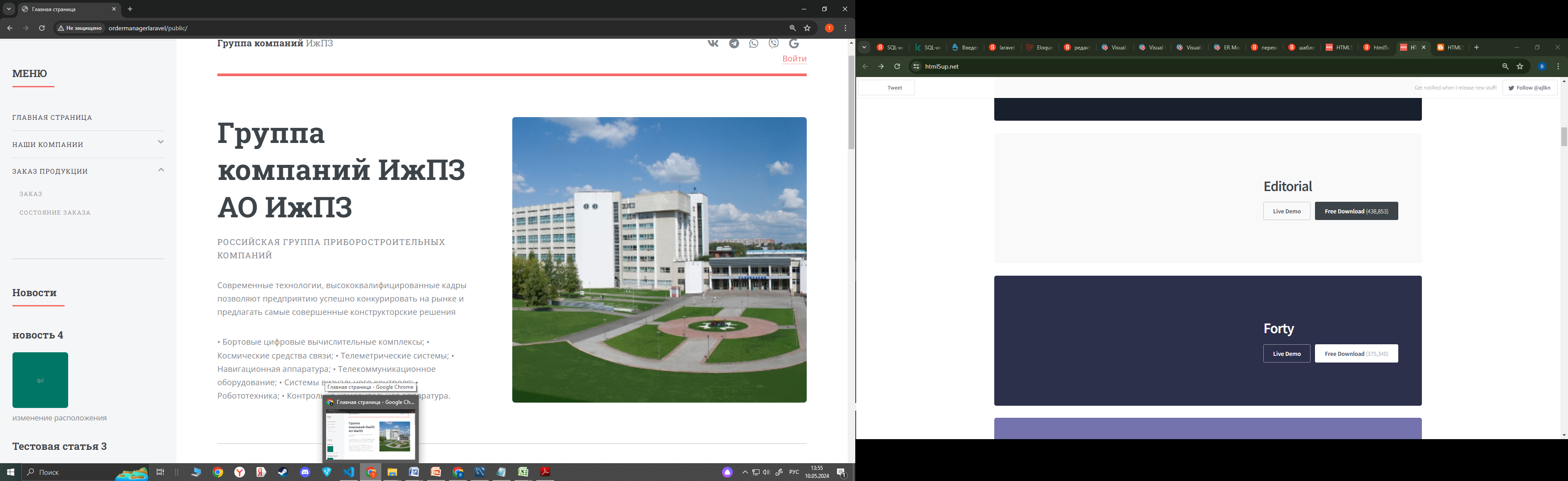


* Авторизация администратора



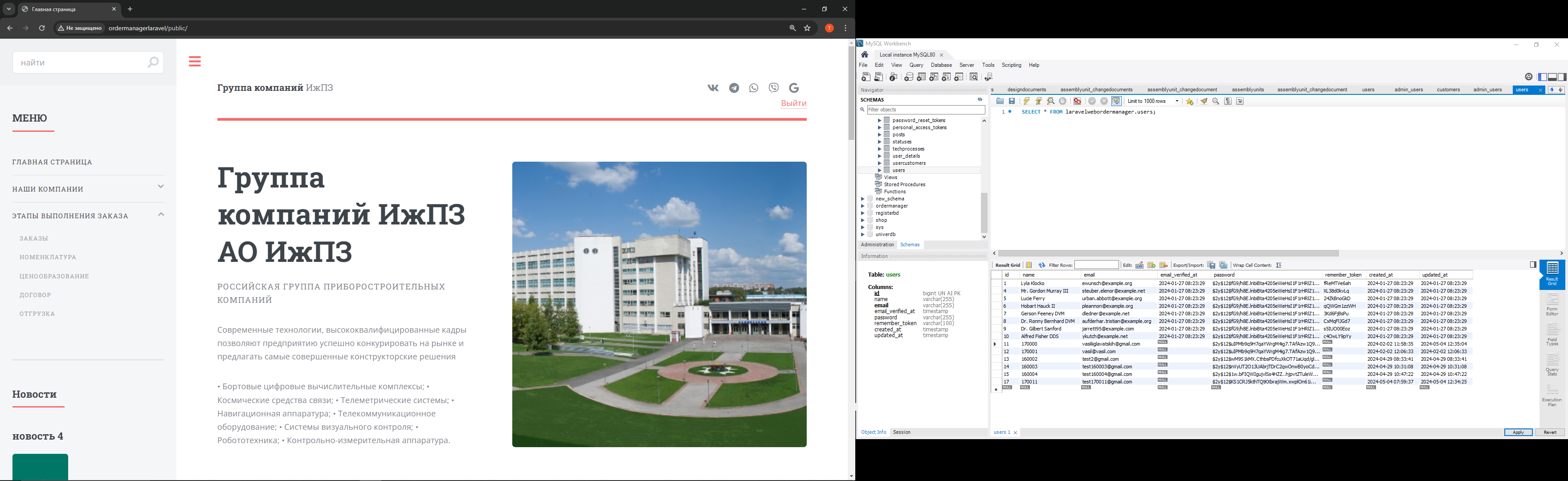
* + 1. Меню приложения
  + Клиент – Гость (незарегистрированный пользователь, заказчик)

Предоставляет доступ к информации о предприятиях-изготовителях продукции и имеет интерфейс для создания заказа на изготовление.

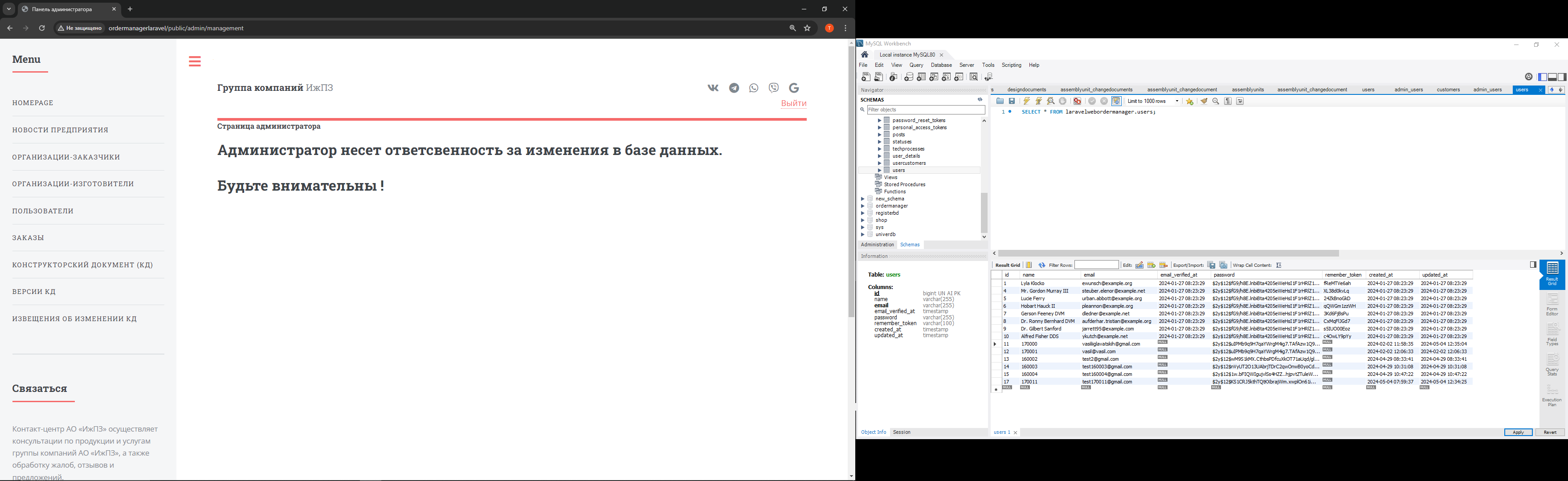
* + Пользователь – работник(менеджер) предприятия –изготовителя

Предоставляет доступ менеджеру предприятия –изготовителя для формирования структуры заказа (раздел номенклатура), формирование стоимости заказа (раздел ценообразование), формирование договора (раздел договор), учет данных по отгрузке изделий заказа (раздел отгрузка).

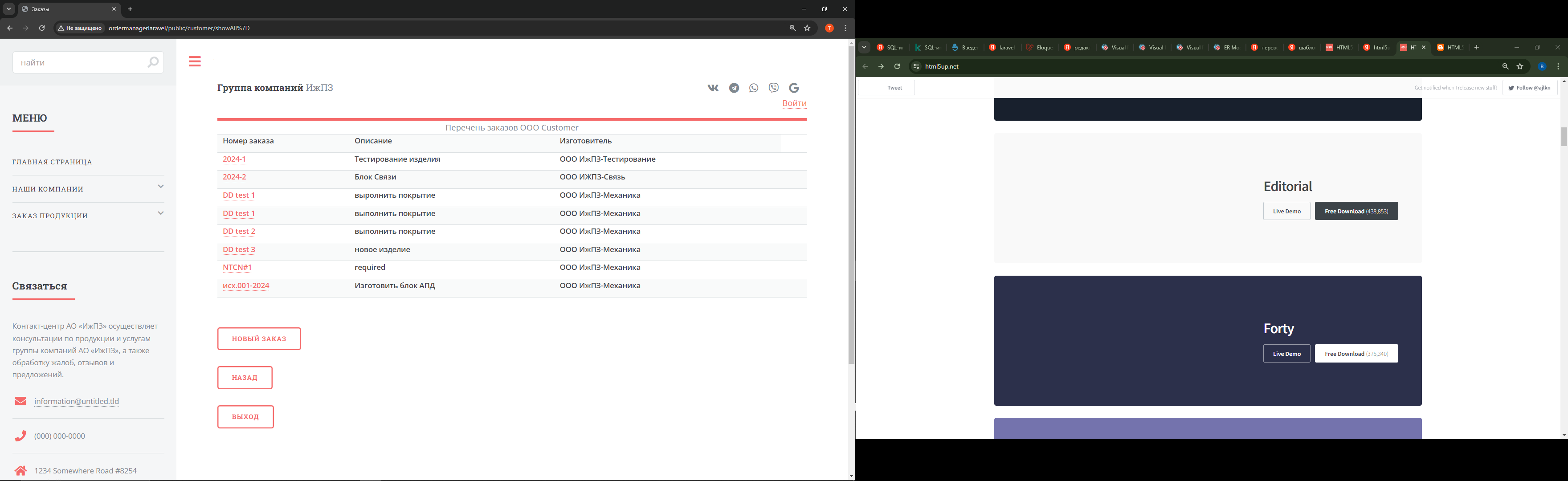


* + Администратор приложения

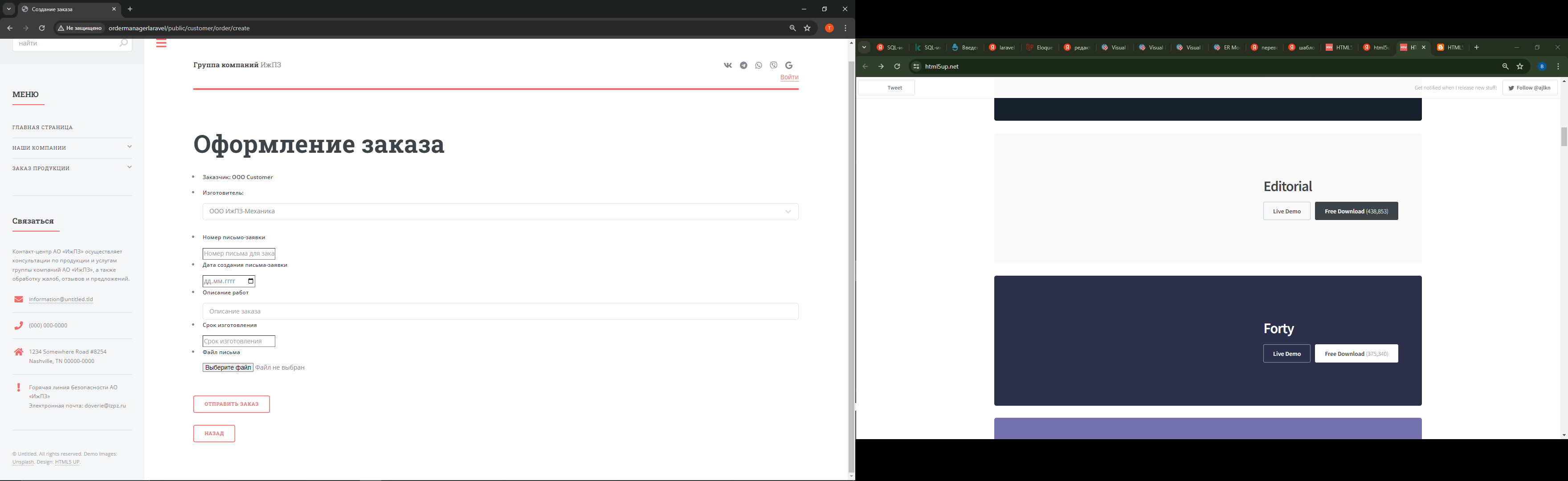
Предоставляет доступ к управлению разделами приложения



* 1. Интерфейс для заказчика
* Просмотр заказов



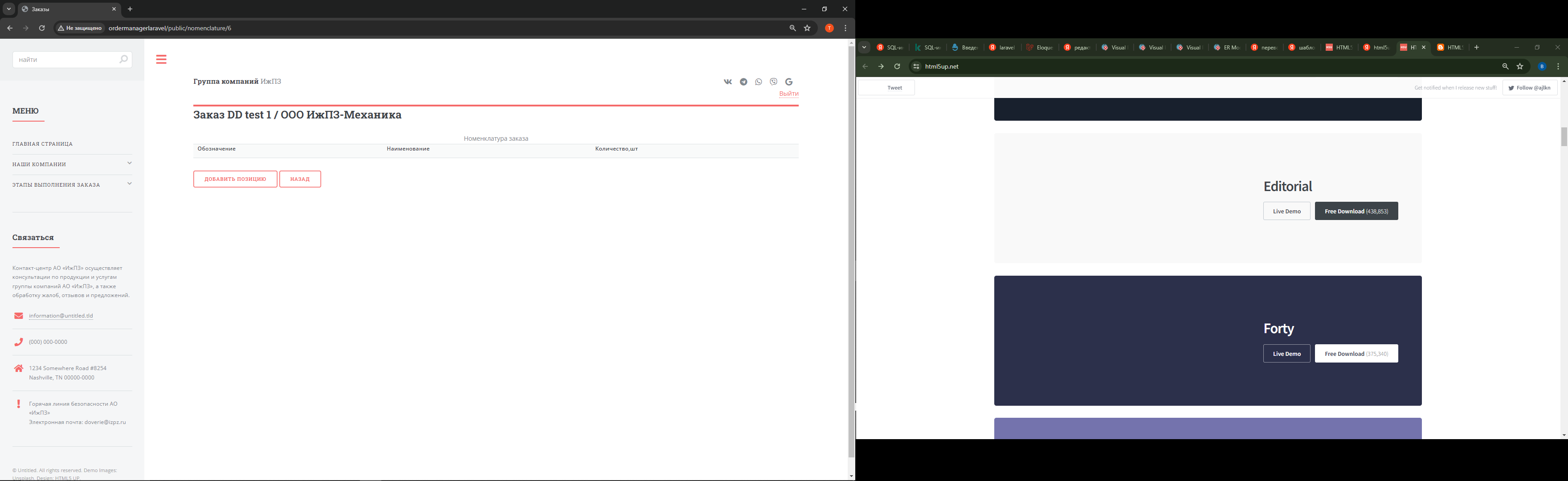
* Создание нового заказа



* 1. Интерфейс для работника(менеджера) предприятия –изготовителя
* Заказы



* Номенклатура



* Добавить позицию в заказ

