**Постановка задачи**

* 1. **Организационно-экономическая сущность задачи**

Наименование задачи: создать сайт для отслеживания товаров.

Цель разработки: создание белорусского сайта, для комфортного отслеживания посылок, товаров.

Назначение: продукт разрабатывается для людей заказавших товары с торговых площадок.

Периодичность использования: при заказе посылки

Обзор существующих аналогичных ПП: рассмотрим сайт “Мои посылки”, я буду стараться сделать своё ПП похожи. При входе на сайт сразу можно будет выбрать страну в которую вы заказываете посылку и ввести код для отслеживания, можно будет зарегистрироваться, написать в поддержку, пользователи которые завершили авторизую смогут отслеживать сразу много посылок, просматривать историю отслеживаемых посылок возможно дальше появиться что-то ещё

* 1. **Функциональные требования**

Описание перечня функций и задач, которые должен выполнять

будущий ПП:

Гост: смогут просматривать сайт, зарегистрироваться, вводить код для отслеживания (только одной посылки), выбрать регион в который доставляется посылка

Пользователь: все те же функции, что может выполнять гость, но так

же появляется авторизация, выход из личного кабинета, просмотр каталога всех отслеживаемых посылок, видеть историю отслеживаемых посылок, обращаться в поддержку

Администратор: авторизация, отвечать на email и по телефону на вопросы или жалобы, иметь доступ к базе данных почт

**1.3** **Описание процессов с входной, выходной и условно-постоянной**

**информацией**

Таблица 1–Описание процессов с входной, выходной и условно-постоянной

информацией

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Категория  пользователей | Наименование  процесса | Краткое описание  алгоритма  выполнения  процесса | Входная  информация | Выходная  информация | Условно  -  постоян  ная  информа  ция |
| 1 | 2 | 3 | 4 | 5 | 6 |
| гость | Просмотр страницы | Любому  пользователю  разрешено заходить  и просматривать  информацию на  сайте | Отсутствует | Отсутствует | Данные  сайта |
| Гость | Выбор региона | Любой пользователь сможет выбрать страну, в которую отправляется посылка | отсутствует | отсутствует | Список стран |
| Гость | Ввод кода посылки | Любой пользователь сможет отслеживать местонахождение своей посылки(не авторизованные только одну) | Код посылки | Путь посылки | Отсутствует |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Продолжение таблицы 1 | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Гость | Регистрация | При нажатии будет  окно  регистрации(или  входа для  зарегистрированного  пользователя),  нужно заполнить  требуемые поля и  после этого гость  получит больше  функций на этом  сайте | Заполнение  полей  регистрации  (  электронная  почта/номер  телефона,  пароль) | Вход в  личных  кабинет | Отсутствует |
| Пользователь | Выбор региона | Авторизованный пользователь сможет выбрать страну, в которую отправляется посылки | Код посылки | Путь посылки | Отсутствует |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Продолжение таблицы 1 | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Пользователь | Просмотр каталога всех отслеживаемых посылок | При нажатии на строку «все посылки» в личном кабинете, будет открываться окно со всеми активными посылками | Отсутствует | Список активных посылок | Список активных посылок |
| Пользователь | Просмотр истории посылок | При нажатии на строку «история посылок» в личном кабинете, открывается окно истории доставок | Отсутствует | История заказов | История заказов |
| Пользователь | Обращение в поддержку | Если у пользователя возникнут проблемы или вопросы он сможет обратиться в поддержку | Вопрос пользователя | Ответ от поддержки | Отсутствует |
| Пользователь | Выход из  личного  кабинета | При входе в личный кабинет все данные пользователя остаются | Выход | Пользователь  ь становится  Гостем | Отсутствует |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Продолжение таблицы 1 | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Администратор | Ответы на  email и по  телефону | Ответы на вопросы  пользователей | Вопросы  пользователей | Ответы | Отсутствует |
| Администратор | Доступ к базе  данных  почт | Разрешен доступ ко  всем данным аптек | Пароль  админа | Базы данных | Базы  данных  всех  аптек |

**1.4 Эксплуатационные требования**

Требования к применению: помогает отследить товар Требования к реализации: Для реализации статических страниц и шаблонов должны использоваться языки HTML и CSS. Для реализации интерактивных элементов клиентской части должны использоваться языки JavaScript (всё это конечно в теории).

Требования к реализации: Для реализации статических страниц и шаблонов должны использоваться языки HTML и CSS. Для реализации интерактивных элементов клиентской части должны использоваться языки JavaScript.(в теории)

Требования к надежности: посторонние люди не смогу видеть и отслеживать чужие посылки, защита от взлома и спама.

Требования к интерфейсу: интерфейс будет максимально простым, без рекламы, только всё самое необходимое, цвет скорее всего белый+ оранжевый, Сайт должен адаптироваться под компьютер и телефон.

Требования к хостингу:

- Поддерживает 1С-Битрикс, WordPress, Joomla!, Drupal и любые

другие CMS

- Круглосуточный мониторинг серверов 24/7: сбои устраняются

раньше, чем их можно заметить

- Объем дискового пространство 8Гб,

- Эффективная защита от спама и взлома

- Базы данных хранятся на SSD-дисках.

- Максимальный размер базы данных - 2 Гб

- Все данные проходят через процедуру ежедневного резервного

копирования. Копии хранятся в течение 20 дней.

- Поддержка MySQL, Python, PHP, Ruby, Node.js

- Стоимость хостинга составит 13,77р в месяц (165,24 за год)

- Стоимость домена Леки.бел 25,20 за год

- https://domain.by/hosting-order/#tariffs

**Приложение А**

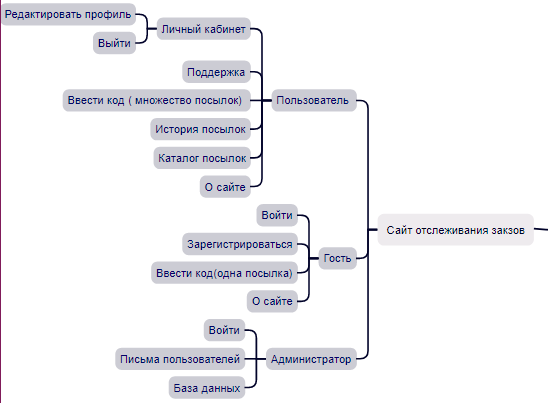


Рисунок А.1 – Структура сайта

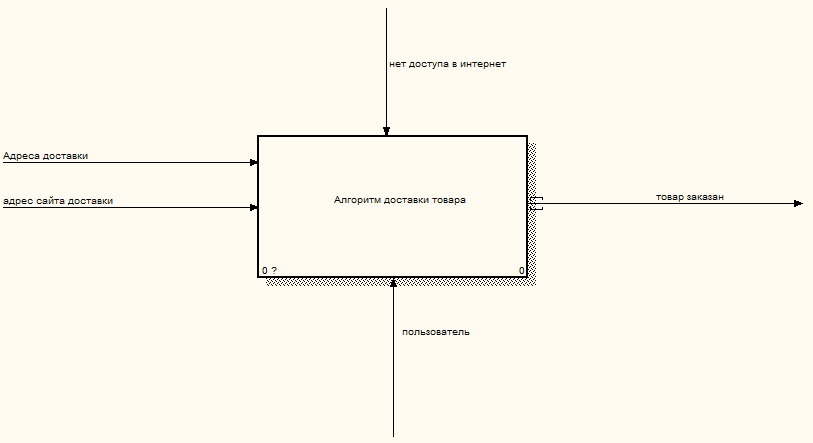
****

Рисунок А.2 – Обобщенная функциональная модель данных

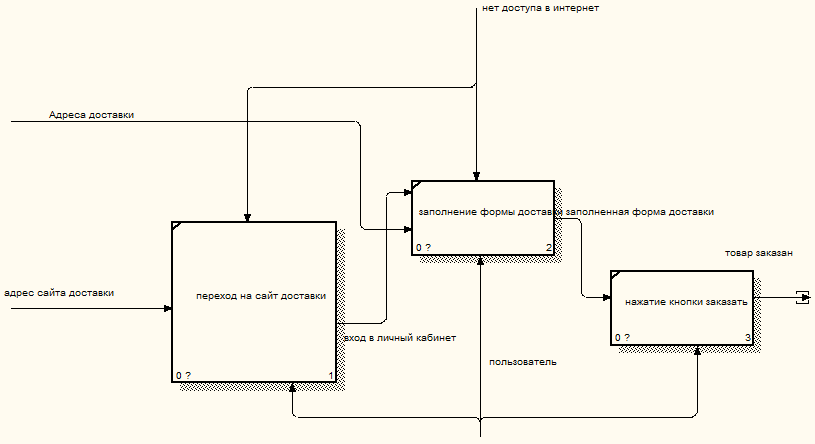


Рисунок А.3 – Подробная функциональная модель данных

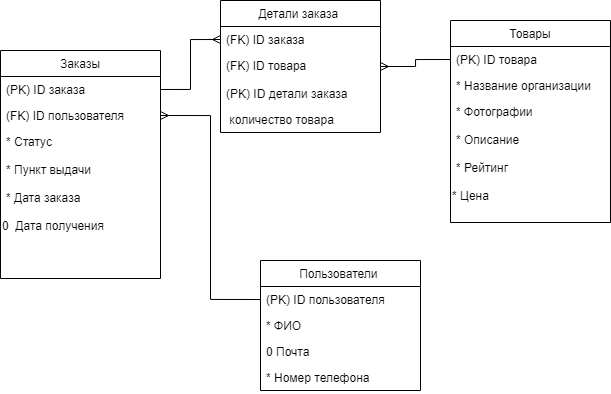


Рисунок А.4 – Модель данных

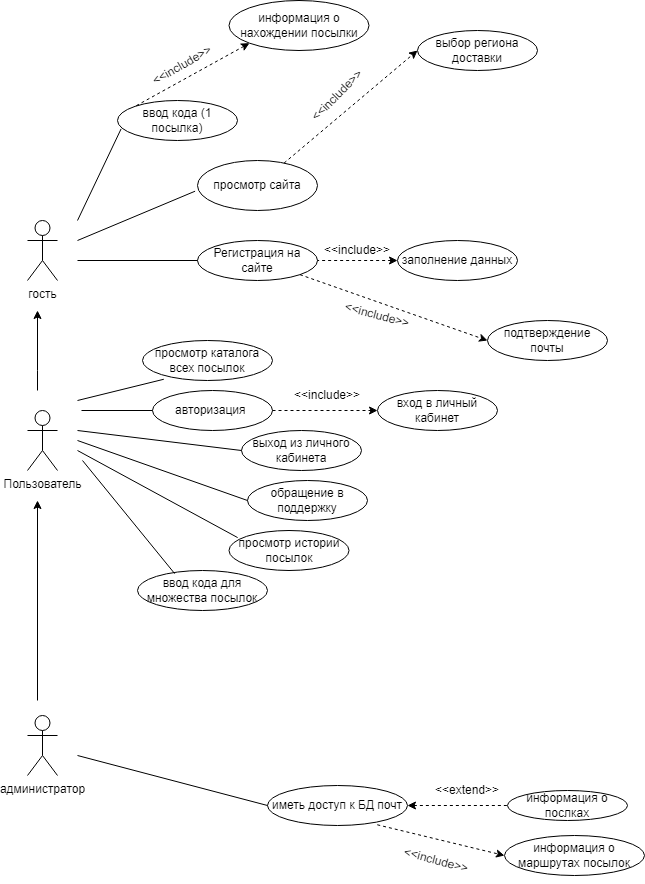


Рисунок А.5 – Диаграмма вариантов использования

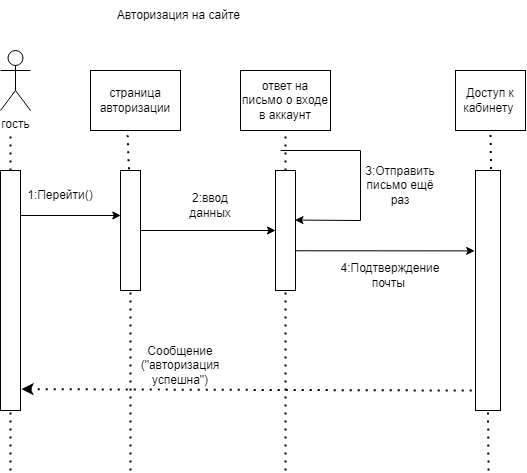


Рисунок А.6 – Диаграмма последовательности

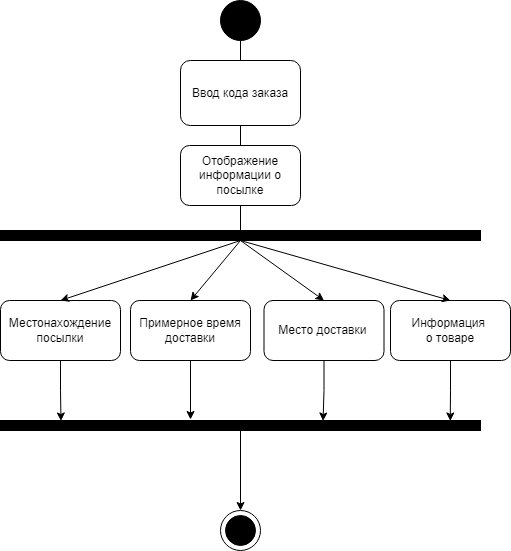


Рисунок А.7 – Диаграмма деятельности

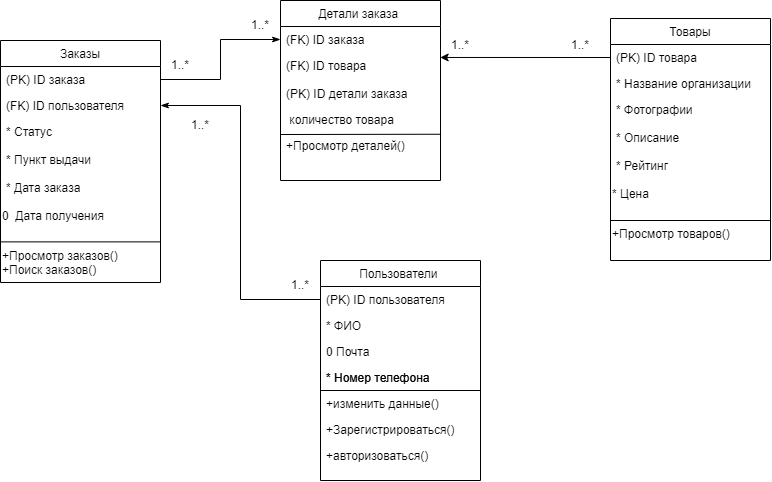


Рисунок А.8 – Диаграмма классов

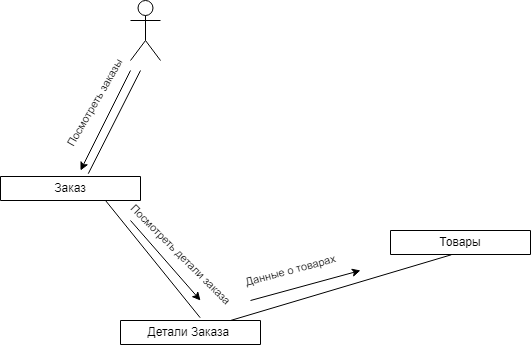


Рисунок А.9 – Диаграмма объектов

**Выбор и обоснование средств реализации проекта для разрабатываемой программной системы.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Название парадигмы | Сущность  парадигмы,  основные  идеи,  принципы,  объекты. | Языки  поддержив  ающие  данную  парадигму.  (2-3 языка) | Достоинства | Недостатки | Примеры  программ |
| 1 | Императивное  программирование | Императи́вное программи́рование — это [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) (стиль написания [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) [компьютерной программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)), для которой характерно следующее:  в исходном коде программы записываются [инструкции](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (команды);  инструкции должны выполняться последовательно;  данные, получаемые при выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями;  данные, полученные при выполнении инструкции, могут записываться в память.  Императивная программа похожа на приказы ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) imperative — приказ, [повелительное наклонение](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D0%B5%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BD%D0%B0%D0%BA%D0%BB%D0%BE%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5)), выражаемые повелительным наклонением в [естественных языках](https://ru.wikipedia.org/wiki/%D0%95%D1%81%D1%82%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA), то есть представляют собой последовательность команд, которые должен выполнить [процессор](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80).  При императивном подходе к написанию кода (в отличие от [функционального подхода](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), относящегося к [декларативной парадигме](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) широко используется [присваивание](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D1%81%D0%B2%D0%B0%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Наличие операторов присваивания увеличивает сложность [модели вычислений](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B9) и делает императивные программы подверженными специфическим ошибкам, не встречающимся при функциональном подходе[[1]](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#cite_note-1).  Основные черты императивных языков:  использование именованных [переменных](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5));  использование оператора присваивания;  использование составных выражений;  использование [подпрограмм](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0);  и др. | С, С++,  Java | Выходные данные зависят только от входных данных (как функции в математике), поэтому их легче понять. Чистые функции принимают аргументы один раз и выдают неизменяемый результат для каждого из них. Поэтому они не дают скрытых результатов. Они используют неизменяемые значения, что упрощает отладку и тестирование. В дополнение к созданию неизменяемых результатов функциональное программирование поддерживает концепцию ленивых вычислений, когда значение оценивается и сохраняется только тогда, когда это необходимо. Поскольку чистые функции тратят только входные значения, не только проще выполнять функции параллельно, но и легче создавать рекурсивные функции. Такие выражения, как map, reduce или filter, просты и лаконичны. | В функциональном программировании функции не имеют состояния, поэтому они всегда создают новые объекты для выполнения действий вместо изменения существующих объектов. По этой причине приложения FP занимают много памяти. | function double (arr) {  let results = [];  for (let i = 0; i < arr.length; i++){  results.push(arr[i] \* 2);  }  return results;  } |
| 2 | Декларативное  программирование | Декларати́вное программи́рование — [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в которой задаётся [спецификация](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) решения задачи, то есть описывается ожидаемый результат, а не способ его получения. Противоположностью декларативного является [императивное программирование](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), при котором на том или ином уровне детализации требуется описание последовательности шагов для решения задачи. В качестве примеров декларативных языков обычно приводят [HTML](https://ru.wikipedia.org/wiki/HTML) и [SQL](https://ru.wikipedia.org/wiki/SQL).  Декларативные программы не используют понятия состояния, в частности, не содержат [переменных](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и операторов [присваивания](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D1%81%D0%B2%D0%B0%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), обеспечивается [ссылочная прозрачность](https://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B7%D1%80%D0%B0%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C). Наиболее близким к «чисто декларативному» программированию является написание исполнимых спецификаций. | HTML, CSS,SQl | – Визуальную привлекательность и удобство использования: Декоративный подход может помочь создать интерфейсы, которые выглядят привлекательно и легко воспринимаются пользователями, что может повысить их удовлетворенность и лояльность к продукту. – Простоту и понятность: Декоративные интерфейсы могут быть более интуитивно понятными, что упрощает использование продукта и сокращает время обучения пользователей. – Быстрое время разработки: Декоративный подход позволяет создавать интерфейсы быстрее, поскольку не требуется разрабатывать сложную функциональность. | – Ограниченную гибкость и возможности: Декоративные интерфейсы не позволяют реализовывать все функции и возможности продукта, что может ограничить его потенциал. – Сложность в поддержке и обновлении: Декоративные интерфейсы требуют более тщательной работы по обновлению и поддержке, поскольку изменения в дизайне могут потребовать значительных изменений в коде. – Неэффективность в сложных задачах: Декоративный подход не всегда является лучшим выбором для решения сложных задач или создания высокопроизводительных приложений.  Таким образом, декоративное программирование может иметь свои преимущества, но также важно учитывать его недостатки и выбирать подходящий подход в зависимости от конкретной задачи и требований проекта. |  |
| 3 | Структурное  Программирование | Структурное программирование - это парадигма программирования, которая основывается на концепции структурирования программы на модули и подпрограммы. Основные идеи структурного программирования:  – Разделение программы на модули, которые выполняют определенную функцию. – Использование подпрограмм для повторного использования кода. – Обработка ошибок и исключений. – Четкое разделение интерфейса и реализации. – Применение принципов объектно-ориентированного программирования.  Объекты - это абстракции, которые представляют собой совокупность данных и методов для работы с ними. Объекты могут быть созданы и уничтожены, а также иметь свои свойства и методы. В структурном программировании объекты используются для структурирования кода и упрощения его понимания. | C, Ada, Pascal | – Более понятный и структурированный код; – Легкость тестирования и отладки; – Возможность повторного использования кода; – Улучшенная модульность и масштабируемость; – Повышение производительности за счет оптимизации кода. | – Необходимость более детального описания алгоритмов; – Увеличение времени разработки из-за необходимости более четкого разделения кода на модули; – Риск появления ошибок при изменении структуры программы; – Ограниченная гибкость при работе с динамическими структурами данных. |  |
| 4 | Процедурное  Программирование | Процедурное программирование - одна из парадигм, которая основывается на последовательном выполнении операций, каждая из которых имеет свой результат.  Основные идеи процедурного программирования:  – Программирование должно быть структурированным и последовательным. – Программа должна быть написана на понятном языке программирования. – Код должен быть легко читаемым и поддерживаемым.  Принципы процедурного программирования включают:  – Использование функций для определения подпрограмм. – Использование циклов для выполнения повторяющихся операций. – Применение условных операторов для принятия решений.  Объекты процедурного программирования могут быть различными, например, числа, строки, массивы, структуры данных и т.д. Они используются для хранения и обработки информации, а также для выполнения определенных операций. | C, Fortran, COBOL | Плюсы процедурного программирования:  – Простота и понятность кода; – Возможность использования готовых функций и библиотек; – Легкость отладки и сопровождения кода; – Универсальность, применимость к различным языкам программирования. | – Ограниченность в использовании объектно-ориентированных принципов; – Сложность в работе с динамическими данными; – Большая зависимость от контекста выполнения программы. | f (b \* b >= 4 \* a \* c)  {  double d = Sqrt(b \* b - 4 \* a \* c);  double x1 = (-b + d) / (2 \* a);  double x2 = (-b - d) / (2 \* a);  }  else //Решения нет! |
| 5 | Модульное  Программирование | Модульность: программа должна быть разбита на модули, которые можно легко модифицировать и заменять без изменения других частей программы.  Инкапсуляция: каждый модуль должен содержать только те данные и функции, которые необходимы для его работы.  Наследование: модули могут наследовать свойства и методы от других модулей, что позволяет создавать иерархию модулей.  Полиморфизм: модули могут предоставлять общий интерфейс для своих функций, что позволяет использовать их в различных контекстах.  Принципы модульного программирования включают в себя:  – модульность: разделение программы на независимые, легко модифицируемые и заменяемые модули; – инкапсуляцию: концентрация данных и функций в отдельных модулях; – наследование: возможность создания иерархии модулей; – полиморфизм: возможность использования общих интерфейсов для функций и методов в разных модулях.  Объекты модульного программирования - это объекты, которые могут быть использованы в программе. Они могут быть созданы с помощью различных методов и функций, а также могут быть изменены или удалены в процессе работы программы. | Python, Java, C# | – Модульность: позволяет легко изменять и заменять отдельные модули без изменения остальной части программы. – Инкапсуляция: скрывает внутренние детали модулей от пользователя, что упрощает их использование и поддержку. – Наследование: позволяет создавать иерархии модулей, что облегчает создание сложных систем. – Полиморфизм: позволяет использовать один интерфейс для нескольких реализаций, что упрощает разработку и сопровождение кода. | – Более сложный код: модули часто содержат более сложный код, чем функции без модульности, что может затруднить понимание и отладку. – Большее время на разработку: модульность требует больше времени на разработку, так как каждый модуль должен быть разработан и протестирован отдельно. – Сложность поддержки: модули могут быть трудно поддерживать из-за сложности внутренней структуры и взаимодействия между ними. | //main.c  #include <stdlib.h>  #include "hello.h"  int main()  {  hello\_world();  return EXIT\_SUCCESS;  } |
| 6 | Объектно  ориентированное  программирование | Объектно-ориентированное программирование (ООП) - это парадигма программирования, основанная на представлении программ в виде объектов, взаимодействующих друг с другом через сообщения. Основные идеи и принципы ООП включают в себя следующие:  Классы и объекты: классы представляют абстрактные типы данных (атомы), а объекты являются экземплярами этих классов (молекулы). Классы определяют структуру объектов, а объекты содержат их состояние и поведение. Абстракция: ООП использует абстракцию для представления сложной структуры данных в виде более простых и понятных элементов. Абстракция позволяет скрывать детали реализации от пользователей, предоставляя им только необходимый интерфейс. Инкапсуляция и наследование: ООП обеспечивает инкапсуляцию данных и методов внутри объектов, а также наследование свойств и методов от родительских классов. Это позволяет создавать более сложные объекты путем комбинирования свойств и методов из разных источников. Полиморфизм и сокрытие информации: ООП поддерживает полиморфизм, то есть возможность использования одного имени метода для разных реализаций. Это позволяет скрыть информацию о реализации методов от пользователей и упростить их использование. Иерархия классов и наследование объектов: ООП позволяет создавать иерархические структуры классов, где объекты могут наследовать методы и свойства от родительских классов и переопределять их по необходимости. Это обеспечивает гибкость и модульность в проектировании систем. Интерфейсы и абстрактные классы: ООП также поддерживает интерфейсы, которые представляют собой абстрактные классы с набором методов, но без реализации. Объекты могут реализовывать несколько интерфейсов, что обеспечивает гибкость в выборе реализации. Наследование и полиморфизм позволяют создавать сложные системы из простых элементов, а инкапсуляция защищает от изменений в реализации, которые могут повлиять на работу других объектов. Это делает ООП мощным инструментом для разработки сложных программных систем. | Java, Python, C++ | Плюсы ООП:  Модульность. ООП позволяет разбить программу на множество мелких модулей, каждый из которых выполняет свою функцию. Это упрощает поддержку и развитие программы, а также делает ее более гибкой.  Наследование. Наследование позволяет создавать новые классы, которые наследуют свойства и методы от существующих классов. Это упрощает создание сложных объектов и уменьшает количество дублирующегося кода.  Полиморфизм. В ООП полиморфизм означает, что один и тот же метод может быть реализован по-разному в зависимости от типа объекта, на который он вызван. Это позволяет избежать ошибок при вызове методов с неправильным типом объекта.  Абстракция. Абстракция позволяет отделить реализацию от интерфейса объекта, что делает код более понятным и упрощает его сопровождение. | Сложность. ООП требует более высокого уровня абстракции и более сложного синтаксиса, чем другие парадигмы программирования. Это может привести к увеличению времени разработки и усложнению сопровождения кода.  Неэффективность. ООП может быть неэффективным при работе с большими объемами данных, так как объекты требуют больше памяти и времени для обработки.  Ограничения. ООП имеет ряд ограничений, например, невозможность работы с динамическими типами данных или невозможность использования процедур и функций.  Сложности в тестировании. ООП предполагает использование наследования и полиморфизма, что может усложнить тестирование кода. | class Calculator {  var lastOperation; // атрибут класса  // процедура из прошлого примера превратилась в метод  sum (a,b) {  this.lastOperation = ‘sum’;  return a + b;  }  // процедура из прошлого примера превратилась в метод  multiply (a,b) {  this.lastOperation = ‘multiply’;  return a \* b;  }  }  // Основная программа  number1 = 2;  number2 = 4;  calc = new Calculator (); // создаем объект класса Calculator  // вызываем методы для объекта calc  result = calc.sum (number1, number2);  result = calc.multiply (number1, number2); |
| 7 | Функциональное  Программирование | В области функционального программирования парадигма определяет подход к написанию программ на языке, который использует функциональное программирование.  Основные идеи парадигмы функционального программирования:  Функциональный подход. Программы на языке функционального программирования определяются как функции, которые принимают аргументы и возвращают результаты. Функции не имеют побочных эффектов и не изменяют состояние программы.  Абстракция. Программы на функциональном языке абстрагируются от деталей реализации, что позволяет сосредоточиться на логике решения задачи.  Рекурсия. Рекурсивные функции могут быть написаны на функциональном языке, что делает его более мощным инструментом для решения сложных задач.  Оптимизация. Функциональные программы могут быть оптимизированы автоматически, что повышает производительность и эффективность работы программы.  Тип-безопасность. Тип-безопасность является одним из ключевых преимуществ функционального языка. Она гарантирует правильность типов данных и предотвращает ошибки, связанные с неправильным использованием типов.  Принципы функционального программирования включают:  – Императивное программирование: Это означает, что программа должна быть написана в виде последовательности инструкций, которые выполняются одна за другой. – Объектно-ориентированное программирование: Программа должна быть организована в виде объектов, которые имеют свойства и методы. – Функциональное программирование: Программа должна быть написана как функция, которая принимает аргументы и возвращает результат. – Абстрактное программирование: Программист должен абстрагироваться от деталей реализации программы и сосредоточиться на логике ее работы. – Рекурсивное программирование: Рекурсия используется для решения сложных задач путем повторного вызова функции.  Объекты функционального программирования - это функции, которые используются для решения задач в данной области. Они могут быть использованы для создания новых функций, обработки данных, управления состоянием программы и многого другого. | Haskell, Lisp, Scala | – Больше возможностей для оптимизации кода и повышения производительности. – Более безопасное и надежное программирование, так как ошибки типов данных можно обнаружить на этапе компиляции, а не во время выполнения программы. – Возможность использования функционального подхода для решения различных задач, таких как обработка данных, управление состоянием программы, создание алгоритмов и т.д. – Простота и понятность кода, что облегчает его поддержку и модификацию. | – Необходимость изучения нового языка программирования и его особенностей. – Ограниченность применения функционального программирования для некоторых типов задач, например, для разработки игр или приложений с графическим интерфейсом. – Сложность реализации некоторых алгоритмов, требующих использования рекурсии или итерации. – Требование более высокой квалификации программистов, так как функциональное программирование требует более глубокого понимания принципов работы языка и его основных конструкций.  Ответь иначе  Хороший ответ  Плохой ответ  Справка |  |
| 8 | Логическое  программирование | Логическое программирование — это парадигма программирования, основанная на использовании логических выражений и правил вывода для решения задач.  Основными идеями логического программирования являются:  Использование логических выражений для представления информации и выполнения вычислений.  Применение правил вывода, которые позволяют выводить новые утверждения из имеющихся утверждений и правил.  Разделение программы на модули, каждый из которых выполняет определенную задачу.  Использование метаданных для описания структуры программы и ее свойств.  Принципы логического программирования включают следующие:  Императивное программирование (imperative programming) — выполнение программы пошагово, следуя инструкциям. Объектно-ориентированное программирование (object-oriented programming) — разделение программы на объекты, которые взаимодействуют между собой через методы и свойства. Функциональное программирование (functional programming) — использование функций для обработки данных и решения задач. Абстрактное программирование (abstract programming) — абстрагирование от деталей реализации программы и сосредоточение на ее логике. Рекурсивное программирование — повторное использование функций при решении сложных задач | Phorolog, Mercury, Datalog | – Логическое программирование позволяет создавать программы, которые автоматически выводят новые утверждения на основе имеющихся правил и данных. – Оно может быть использовано для решения задач, связанных с обработкой естественного языка, логикой и другими областями, где требуется обработка сложных логических выражений. – Логическое программирование может быть легко интегрировано с другими языками программирования, что упрощает разработку сложных приложений. – Оно обеспечивает высокую степень модульности и гибкости, что позволяет легко изменять и расширять программы. | – Для создания программ на логическом языке необходимо знать логические правила и уметь работать с ними. – Программы на логическом языке могут быть сложными для понимания и отладки, особенно если они содержат много правил и условий. – Некоторые задачи, такие как разработка игр или графических приложений, могут быть лучше решены с помощью других парадигм программирования. | птица (голубь).  есть\_крылья (ворона).  умеет\_летать (ворона).  есть\_крылья (пингвин).  умеет\_плавать (пингвин).  птица (Объект):- умеет\_летать (Объект), есть\_крылья (Объект).. |

Для разработки своего индивидуального задания выбран язык программирования, который наилучшим образом соответствует вашим потребностям и возможностям. Некоторые из наиболее популярных языков программирования для разработки сайтов включают PHP, Python, Ruby on Rails, JavaScript и Java.

**( UX и UI -интерфейсы)**

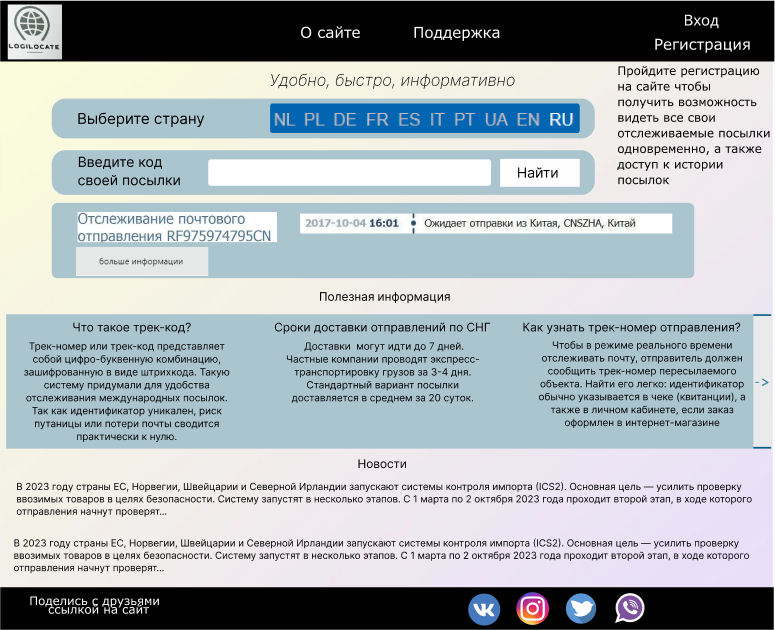


Рисунок 1 – UI главная страница на PC

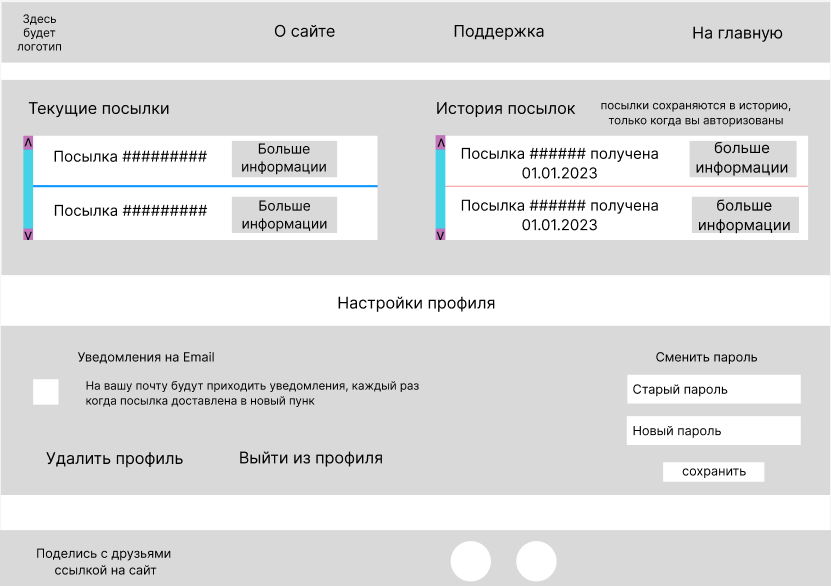


Рисунок 2 – UX личный кабинет на PC

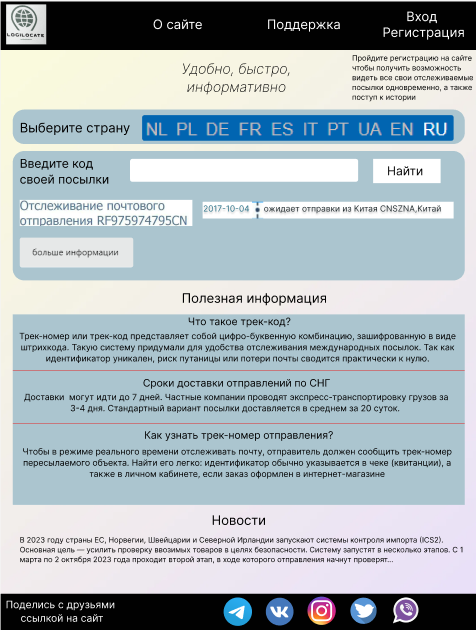


Рисунок 3 – UI главная страница на планшете

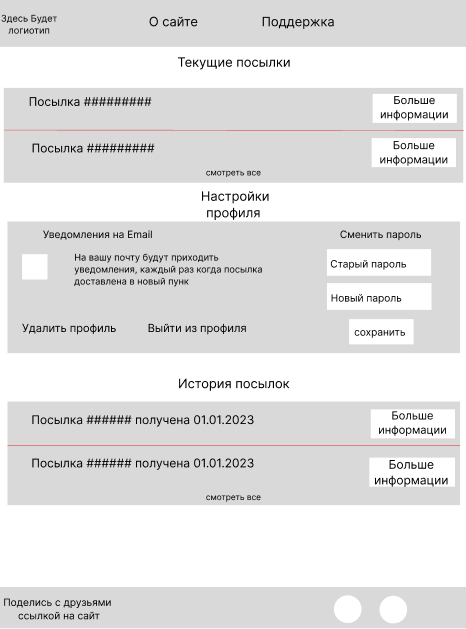


Рисунок 4 – UX личный кабинет на планшете

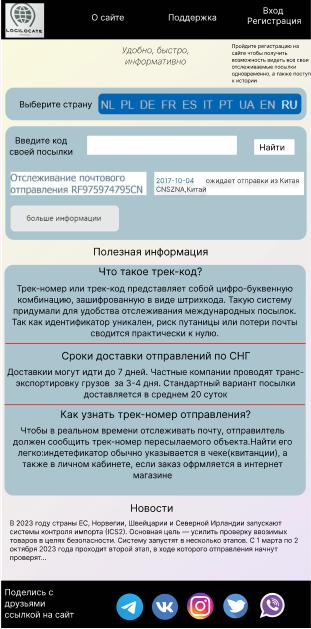


Рисунок 5 – UI главная страница на телефоне

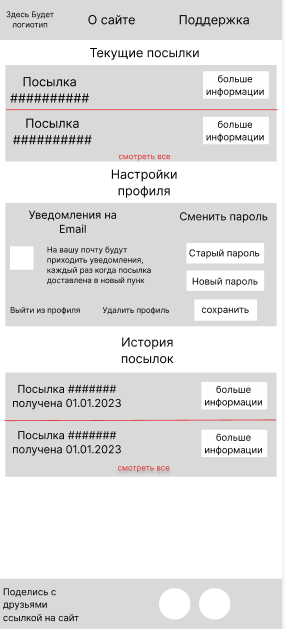


Рисунок 6 – UX личный кабинет на телефоне

**Прототип ПП**

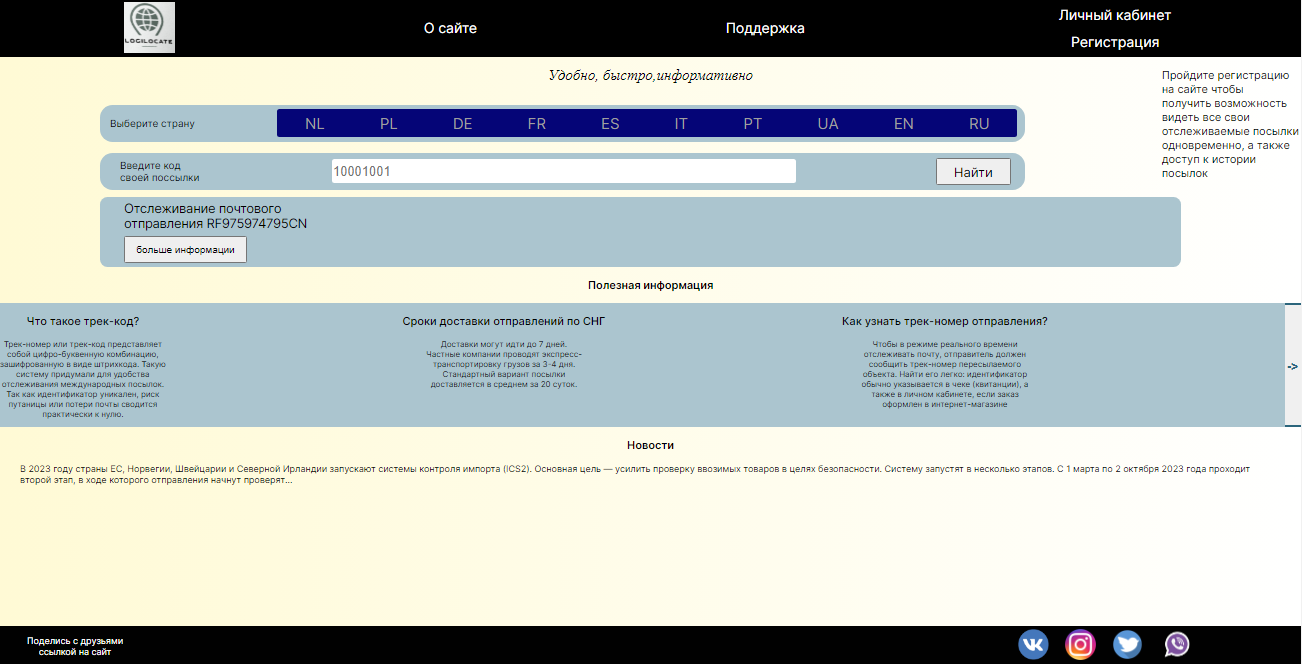


Рисунок 7 – Главная страница ПП

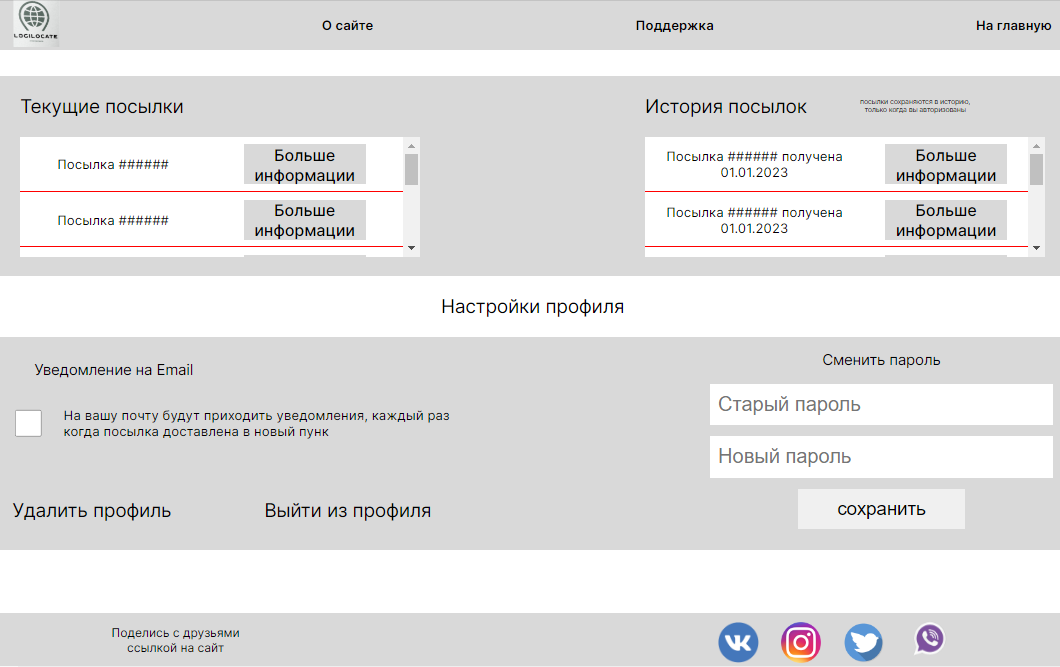


Рисунок 8 – Личный ПП

**Документация на ПП**

Данный Web-ресурс «Logilacate». Программный продукт разрабатывается для пользователей, желающих отследить заказанную посылку. Периодичность использования: по необходимости.

Требования:

* Стабильное интернет подключение;
* Наличие Web-браузера;

**Руководство программиста**

Программный продукт разработан с использованием таких языков: «Html» и «Css». Данные языки позволяют создавать полноценные веб-сайты, разберёмся с каждым компонентом по подробнее.

Html является языком разметки, который определяет структуру веб-страницы. Он используется для создания элементов, таких как:

**Сематические теги:**

<header></header>;

<main></main>;

<footer></footer>;

**Параграфы:**

<p>Введите код своей посылки</p>;

<p>Отслеживание почтового отправления RF975974795CN</p>;

**Изображения:**

<img class="image\_\_header" src="../image/Logo.svg">**;**

<img class="icons" src="../image/Vk.svg" width="61" height="61">;

**Ссылки:**

<a href="../Html/private\_office.html">Личный кабинет</a>;

<a href="#">О сайте</a>;

CSS представляет собой язык стилей, который управляет внешним видом элементов, созданных с использованием «Html». С его помощью:

**Задаются цвета**:

background-color: black;

color: #FFFEFE;

background-color: rgb(5, 5, 119);

**Шрифты:**

font-size: 30px;

font-family: 'Inter', sans-serif;

**Расположение элементов:**

display: flex;

flex-direction: column;

justify-content: space-around;

**Анимации:**

a:hover{

    color: white;

}

Вместе эти две технологии формируют основу веб-разработки, предоставляя средства для создания структуры «Html», стилей «Css» на веб-страницах. Этот двойной подход обеспечивает возможность создания современных веб-приложений, способных к разнообразным и мощным функциональностям.

В разработке веб-приложений широко используются компоненты. Формы и вводные компоненты, включая **поля ввода** (рисунок 9).



Рисунок 9 – Поле ввода

Кнопки (рисунок 10)

Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Автоматически созданное описание

Рисунок 10 – Кнопка

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание Собирают данные от пользователей. Компоненты содержимого, такие как карточки (рисунок 11).

Рисунок 11 – Новостная карточка

Модальные окна (рисунок 12).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 12 – Модальное окно

Изображение выглядит как Графика, снимок экрана, Красочность, круг

Автоматически созданное описаниеПредоставляют информацию и взаимодействие. Графические элементы (рисунок 13).

Рисунок 13 – Графический элемент

Все они являются основными строительными блоками для разработки веб-приложений.

Этот компонентный подход упрощает разработку, делает код более читаемым и обеспечивает повторное использование, создавая тем самым современные и привлекательные веб-приложения.

**Руководство пользователя**

На данном этапе разработки на сайте пользователю доступны следующие функции:

Просмотр страниц сайта (рисунок 14).

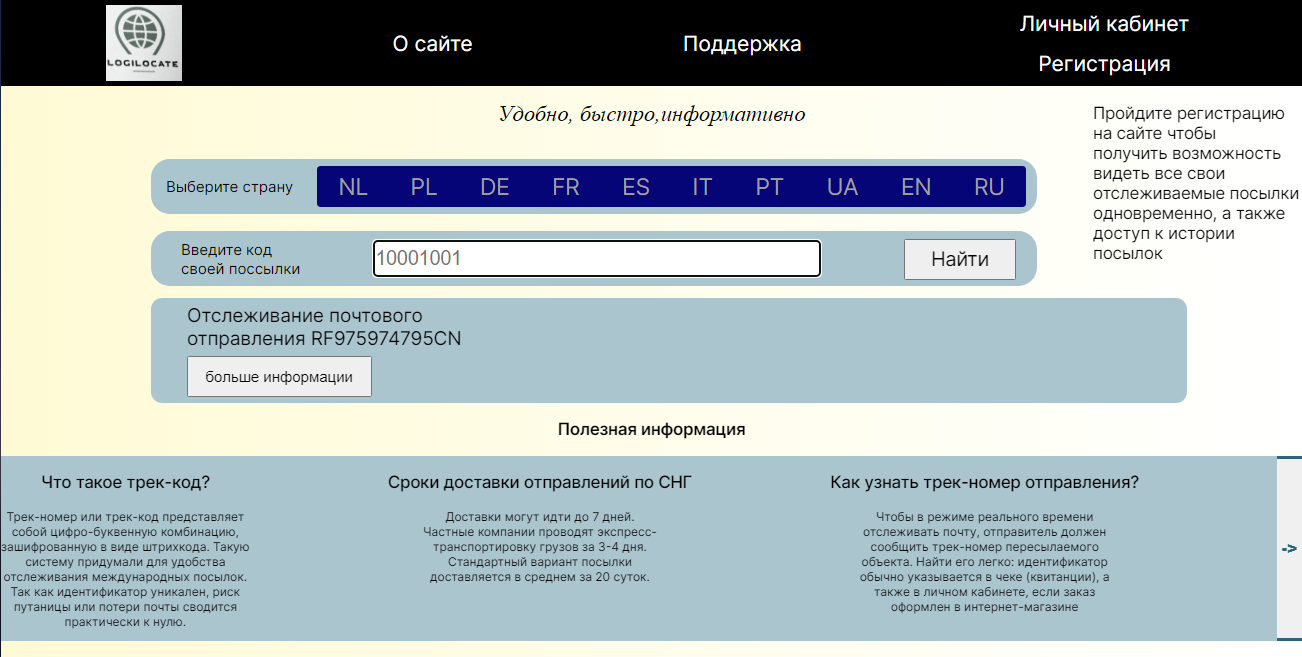


Рисунок 14 – Главная страница сайта

Вводить символов в строку на главной (рисунок 15).



Рисунок 15 – Строка ввода на главной

Переходить в личный кабинет нажатием на логотип (рисунок 16).



Рисунок 16 – Логотип на главной сайта

Переходить в личный кабинет нажатием на кнопку (рисунок 17).

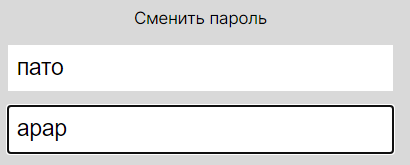
  
Рисунок 17 – Кнопка «Личный кабинет» на главной

Переходить на главную, нажатием на кнопку (рисунок 18).



Рисунок 18 – Кнопка «На главную» в личном кабинете

Вводить символов в поля на странице личного кабинета (рисунок 19).

  
Рисунок 19 – Строки ввода в личном кабинете

Взаимодействовать с чек-боксом включения уведомлений, на странице личного кабинета (рисунок 20).

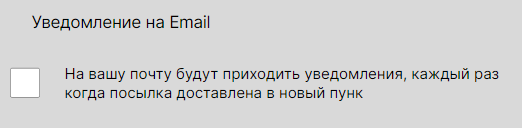


Рисунок 20 – Чек бокс включения уведомлений

Взаимодействовать с слеш-барами на странице личного кабинета (рисунок 21).

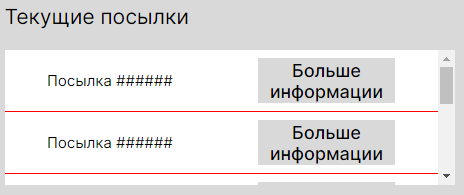


Рисунок 20 – Слеш бары в личном кабинете