# Сбор и анализ требований

## Назначение и область применения

Разрабатываемый генератор уроков предназначен для автоматизации процесса создания интерактивных занятий для изучения английского языка, повышение эффективности обучения за счет высокого уровня адаптивности и персонализации генерируемых заданий, а также гибкого управления личным словарем.

Областью применения являются организации, в которых требуется или желательно знание английского языка и может использоваться в программе корпоративного обучения персонала. Основными категориями пользователей будут являться сотрудники, желающие повысить свой уровень владения английским языком.

## Постановка задачи

Требуется разработать генератор уроков, предоставляющий доступ к следующей функциональности:

* регистрация и авторизация пользователей;
* просмотр и изменение информации о пользователях;
* фильтрация пользователей по уровню владения языком;
* сортировка пользователей по имени;
* прохождение обучающих занятий;
* просмотр, добавление и удаление записей в личном словаре;
* поиск слов в базе системы;
* определение уровня владения языком на основе тестирования;
* возможность экспорта выученных слов в формате pdf.

В целях обеспечения безопасности требуется предварительная авторизация для доступа к административным возможностям и основной функциональности системы. Интерфейс приложения должен быть интуитивно понятен для пользователя.

Неавторизованный пользователь должен иметь доступ к прохождению тестирования для определения текущего уровня владения языком и регистрации.

Авторизованный пользователь должен иметь доступ к просмотру, добавлению и удалению записей личного словаря, прохождению занятий и экспорту выученных слов в формате pdf.

Администратор должен иметь доступ к просмотру и изменению информации о пользователях, а также выполнению всех действий, доступных авторизованному пользователю.

На рисунке 1 представлена диаграмма вариантов использования приложения различными категориями пользователей.

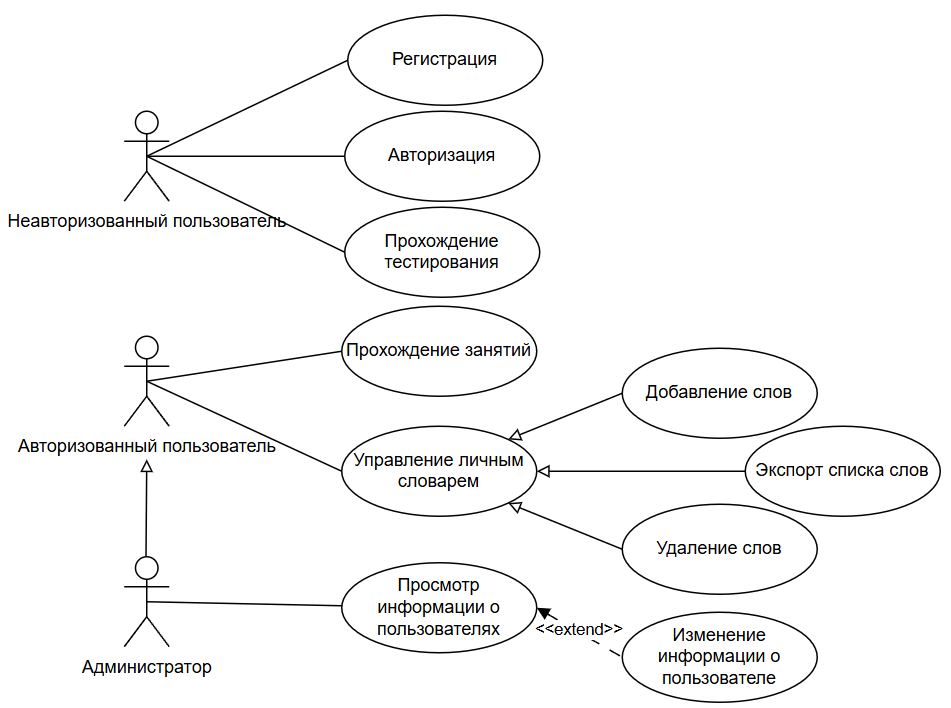


Рисунок 1 – Диаграмма вариантов использования

## Выбор состава программных и технических средств

Для достижения поставленной цели необходимо разработать БД, обеспечивающую хранение основной информации системы.

В качестве СУБД выбрана MySQL, так как она имеет высокую производительность, обладает кроссплатформенностью и легко масштабируется.

Для разработки системы выбран язык программирования C# с использованием фреймворка ASP.NET MVC, который обеспечивает высокую производительность, гибкую архитектуру удобство сопровождения.

В качестве средства разработки ПО выбрана IDE Visual Studio 2022, поскольку она обеспечивает быстрый и удобный процесс разработки, а также интеграцию со всеми необходимыми инструментами.

Для функционирования системы на стороне сервера необходимы следующие программные и технические средства:

* ОС Windows x86 64-бит или Linux x86 64-бит;
* MySQL Server не ниже 8.0;
* программное обеспечение для работы сайта: .NET SDK не ниже 8.0 и ASP.NET MVC 5;
* доступная оперативная память 3 ГБ;
* процессор с частотой не менее 1 ГГц и не менее 2 ядер;
* минимальный объем дискового пространства 10 ГБ.

Доступ к системе осуществляется с помощью браузеров, таких как Chrome (версии 90 и выше), FireFox (версии 88 и выше) и других, поддерживающих HTML5 и CSS3, необходимых для функционирования системы на стороне клиента. Также необходимы следующие технические средства:

* процессор с частотой 1 ГГц;
* доступная оперативная память 2 ГБ.

# Проектирование программного обеспечения

## Проектирование интерфейса пользователя

В рамках разработки ПО с использованием draw.io спроектирован пользовательский интерфейс в виде wireframe, который демонстрирует структуру сайта, его основные элементы и доступную функциональность. На рисунке 2 в виде wireframe представлены следующие страницы ПО: главная страница, страница профиля пользователя и страница со списком уроков.

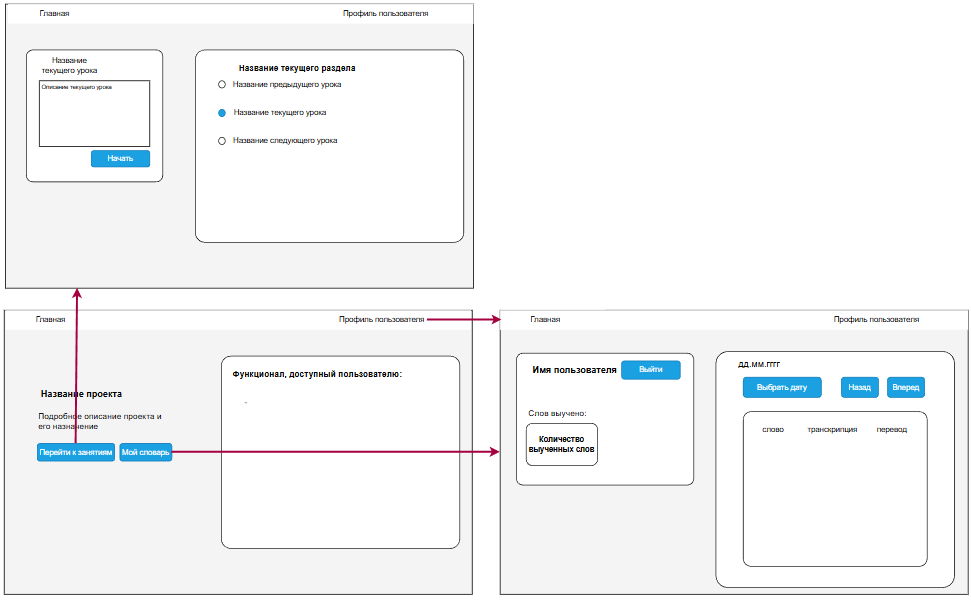


Рисунок 2 – Wireframe главной страницы, страниц профиля и списка уроков

Для интерфейса ПО выбрана следующая цветовая палитра:

* #FFFFFF – цвет фона и подписей кнопок;
* #D9D9D9 – вторичный цвет фона;
* #000000 – основной цвет текста;
* #666666 – вторичный цвет текста;
* #007BFF – цвет кнопок.

## Разработка архитектуры программного обеспечения

Архитектура ПО построена на основе клиент-серверной модели. Диаграмма развертывания компонентов представлена на рисунке 3.

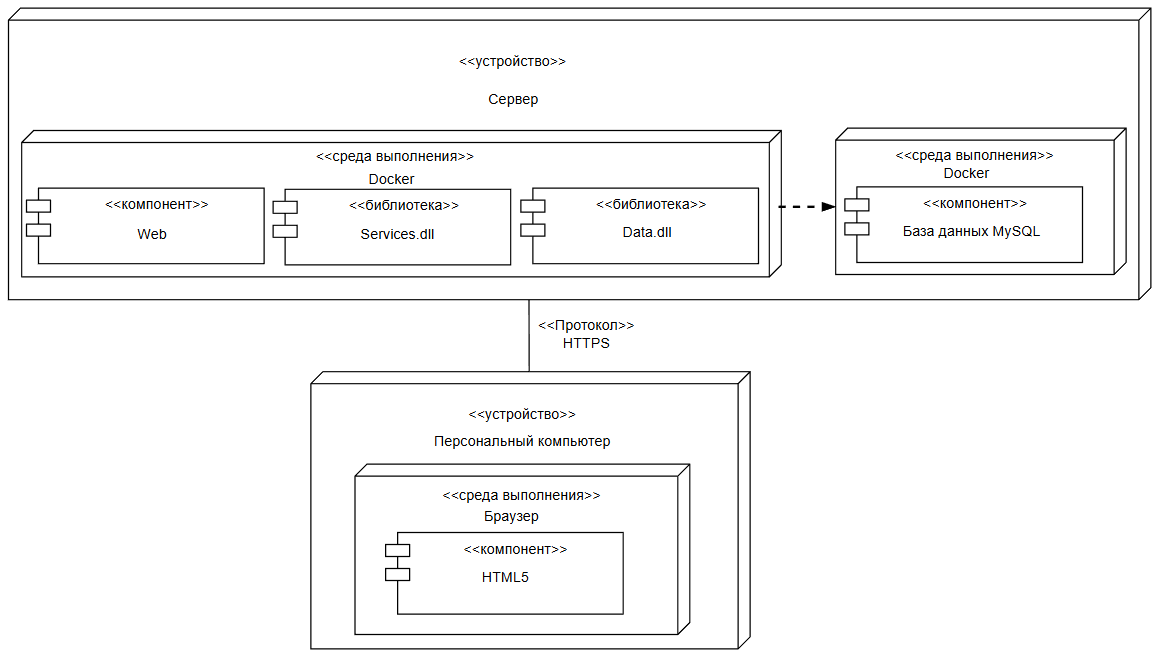


Рисунок 3 – Диаграмма развертывания компонентов

## Проектирование базы данных

В рамках курсового проектирования требуется разработать БД для системы генерации уроков. На рисунке 4 в виде ERD показана физическая модель предметной области, созданная при помощи MySQL Workbench.



Рисунок 4 – Физическая модель данных

# Разработка и интеграция модулей программного обеспечения

## Разработка программных модулей

В качестве серверного решения выбраны фреймворк ASP.NET Core MVC и язык программирования C#. Для обеспечения четкого разграничения ответственности и повышения масштабируемости решение разделено на следующие слои:

* библиотека классов Data, реализующая взаимодействие с БД, посредством ORM Entity Framework Core;
* библиотека классов Services, реализующая бизнес-логику системы;
* проект ASP.NET, реализующий обработку запросов пользователей, управление доступом и взаимодействие с БД, посредством сервисного слоя.

При работе с личным словарём пользователя подсистема предоставляет возможность динамического поиска слов в БД для последующего добавления их в словарь. Для реализации этой функциональности в контроллере AccountController разработан метод SearchWords, представленный листингом 1.

Листинг 1 – Код метода поиска слов

[Authorize]

[HttpGet]

public async Task<IActionResult> SearchWords(string query, CancellationToken cancellationToken = default)

{

//Проверка наличия идентификатора пользователя.

if (!TryGetUserId(out var userId))

{

//Вызов механизма аутентификации

return Challenge();

}

//Валидация запроса

if (string.IsNullOrWhiteSpace(query)

|| query.Trim().Length < 2)

{

//Возврат пустого ответа при некорректном запросе

return Json(Array.Empty<WordSuggestionViewModel>());

}

//Получение списка релевантных запросу слов

var suggestions =

await \_memorizedWordsService.SearchAsync(userId,

query,

cancellationToken: cancellationToken);

//Преобразование DTO в формат модели представления

var viewModels = suggestions

.Select(WordSuggestionViewModel.FromDto)

.ToList();

//Формирование и возврат JSON-ответа для клиента

return Json(viewModels);

}

## Реализация интерфейса пользователя

Для оформления страниц применен фреймворк Bootstrap, а также использована технология Razor Pages, которая позволила использовать вставки C# кода на страницах.

Для отображения списка слов на странице с профилем пользователя разработан компонент, код верстки которого представлен листингом 2. Этот компонент использует стиль, который описан в site.css и представлен листингом 3.

Листинг 2 – Код верстки компонента отображения списка слов

@\* Определение контейнера списка слов и его артибутов \*@

<div class="profile-words-list@(Model.HasWords ?

string.Empty : " d-none")"

data-profile-words-list

data-selected-date="@Model.SelectedDateIso"

data-today-date="@Model.TodayIso"

data-remove-url="@Url.Action("RemoveMemorizedWord",

"Account")">

@\* Отображение списка слов пользователя \*@

@foreach (var word in Model.Words)

{

<div class="profile-word-row"

data-profile-word

data-word-id="@word.WordId"

data-concept-id="@word.ConceptId">

@\* Вывод начальной формы слова \*@

<span class="profile-word-lemma">@word.Lemma</span>

@\* Вывод транскрипции или placeholder \*@

<span class="profile-word-transcription">

@(string.IsNullOrWhiteSpace(word.Transcription)

? "—" : $"[{word.Transcription}]")

</span>

@\* Вывод перевода или placeholder \*@

<span class="profile-word-translation">

@(string.IsNullOrWhiteSpace(word.Translation) ?

"—" : word.Translation)

</span>

</div>

}

</div>

Листинг 3 – Код стиля для отображения списка слов

.profile-words-list {

/\* Настройка контейнера \*/

display: flex;

/\* Настройка направления размещения \*/

flex-direction: column;

/\* Настройка внешнего оформления \*/

border: 1px solid var(--card-border);

/\* Установка степени скругление углов \*/

border-radius: 1.15rem;

/\* Установка значения вертикальной прокрутки \*/

overflow-y: auto;

/\* Скрытие горизонтальной прокрутки \*/

overflow-x: hidden;

/\* Настройка поведения в родительском блоке \*/

flex: 1 1 auto;

/\* Установка минимальной высоты \*/

min-height: 0;

}

## Разграничение прав доступа пользователей

В веб-приложении реализовано разграничение прав доступа пользователей посредством механизма cookie-аутентификации. Код формирования cookie представлен листингом 4.

На панели навигации размещена кнопка перехода к панели администратора. Она отображается только для пользователей, имеющих права администратора. Код реализации этой логики в файле \_Layout.cshtml представлен листингом 5.

Листинг 4 – Код формирования cookie

services

.AddAuthentication(

CookieAuthenticationDefaults.AuthenticationScheme)

.AddCookie(options =>

{

// Настройка схемы аутентификации

options.LoginPath = "/Account/Login";

// Настройка пути при отсутствии доступа

options.AccessDeniedPath = "/Account/AccessDenied";

// Настройка параметров cookie

options.Cookie.Name = "SmartLang.Auth";

options.Cookie.HttpOnly = true;

options.Cookie.SecurePolicy = CookieSecurePolicy.Always;

options.Cookie.SameSite = SameSiteMode.Strict;

// Настройка срока действия cookie

options.SlidingExpiration = true;

options.ExpireTimeSpan = TimeSpan.FromHours(12);

});

Листинг 5 – Код отображения кнопки «Админ-панель»

//Выполнение проверки соответствия роли

if (isAdmin)

{

//Отображение кнопки перехода в административный раздел

//Определение атрибутов элемента

<**a** class="btn btn-sm btn-outline-primary btn-admin-panel"

**asp-area**=""

**asp-controller**="Admin"

**asp-action**="Index">Админ-панель</**a**>

}

При переходе к панели администратора в контроллере AdminController выполняется получение списка пользователей системы в методе Index, код метода представлен листингом 6.

Листинг 6 – Код метода Index

//Ограничение доступа по роли

[Authorize(Roles = "Admin")]

[HttpGet]

public async Task<IActionResult> Index(

[FromQuery] UserListFilterInputModel? request,

CancellationToken cancellationToken)

{

//Проверка наличия идентификатора текущего пользователя

if (!TryGetUserId(out var currentUserId))

{

return Challenge();

}

//Инициализация входной модели при отсутствии данных

request ??= new();

// Формирование параметров запроса для сервиса

var query = new AdminUserListQuery

{

LanguageLevelId = request.LanguageLevelId,

SortByNameDescending =

request.SortOrder == UserListSortOrder.UsernameDesc

};

//Получение списка пользователей по заданным параметрам

var result = await \_adminUserService.GetUsersAsync(

query,

cancellationToken);

//Формирование модели представления для отображения

var model = AdminUserListViewModel.FromDtos(

result,

request.LanguageLevelId,

request.SortOrder,

currentUserId);

//Возврат представления с моделью

return View(model);

}

## Экспорт данных

Для реализации возможности экспорта списка слов пользователя используется библиотека QuestPDF. Формирование структуры pdf-документа выполняется в классе WordListPdfDocument методом Compose (представлен листингом 7). Получение слов из БД и выполнение экспорта реализовано в классе UserDictionaryExportService в методе ExportLearnedWordsToPdfAsync (представлен листингом 8).

Листинг 7 – Код метода формирования pdf-файла

public void Compose(IDocumentContainer container)

{

container.Page(page =>

{

//Настройка отступов страницы

page.Margin(40);

//Формирование заголовка страницы

page.Header()

.Text("Список слов")

.FontSize(20)

.SemiBold()

.AlignCenter();

//Формирование основной области документа

page.Content().Column(col =>

{

//Настройка интервалов между элементами

col.Spacing(5);

//Обработка пустого набора данных

if (\_items.Count == 0)

{

col.Item().Text("Нет слов для экспорта.")

.FontSize(12);

return;

}

//Формирование элементов списка

foreach (var item in \_items)

{

col.Item().Text(text =>

{

var wordPart = item.Word;

var transPart =

string.IsNullOrWhiteSpace(

item.Transcription)

? null

: $" [{item.Transcription}]";

var translationPart =

$" {item.Translation}";

text.Span(wordPart).SemiBold();

if (transPart != null)

text.Span(transPart).Italic();

text.Span(translationPart);

});

}

});

//Формирование нижней части страницы

page.Footer()

.AlignCenter()

.Text(x =>

{

// Вывод номера страницы

x.CurrentPageNumber();

x.Span(" / ");

x.TotalPages();

});

});

}

Листинг 8 – Код метода создания pdf-файла со списком слов пользователя

public async Task<byte[]> ExportLearnedWordsToPdfAsync(

ulong userId,

CancellationToken cancellationToken = default)

{

// Получение списка выученных слов

var items = await GetUserWordItemsAsync(

userId,

cancellationToken);

// Подготовка документа для экспорта

var document = new WordListPdfDocument(items);

// Формирование PDF-файла

return document.GeneratePdf();

}

# Тестирование и отладка программного обеспечения

## Структурное тестирование

Для проверки корректности работы модуля администрирования реализовано модульное тестирование с использованием фреймворка xUnit.

Для выборки и фильтрации данных о пользователях в AdminUserService реализован метод GetUsersAsync. Для его проверки разработан тестовый сценарий GetUsersAsync\_FiltersAndSortsByName (представлен листингом 9).

Подготовка окружения для теста выполняется вспомогательным методом CreateService. Код метода представлен листингом 10.

Листинг 9 – Код метода GetUsersAsync\_FiltersAndSortsByName

[Fact]

public async Task GetUsersAsync\_FiltersAndSortsByName()

{

// Инициализация сервиса и контекста БД

var (service, dbContext) = CreateService();

// Создание справочных данных (роли и уровни языка)

var role = new Role { Id = 1, Name = "User" };

var beginner = new LanguageLevel {

Id = 1, Code = "A1", Name = "Beginner" };

var intermediate = new LanguageLevel {

Id = 2, Code = "B1", Name = "Intermediate" };

dbContext.Roles.Add(role);

dbContext.LanguageLevels.AddRange(beginner, intermediate);

// Добавление пользователей для тестирования сценариев

dbContext.Users.AddRange(

new User

{

Id = 1,

Username = "alice",

RoleId = role.Id,

CurrentLevelId = beginner.Id,

PasswordHash = "hash"

},

new User

{

Id = 2,

Username = "bob",

RoleId = role.Id,

CurrentLevelId = beginner.Id,

PasswordHash = "hash",

UserConceptWords = { new UserConceptWord { UserId = 2, WordId = 1, ConceptId = 1 } }

},

new User

{

Id = 3,

Username = "charlie",

RoleId = role.Id,

CurrentLevelId = intermediate.Id,

PasswordHash = "hash"

});

// Сохранение подготовленных данных в БД

await dbContext.SaveChangesAsync();

// Формирование запроса с фильтрацией и сортировкой

var query = new AdminUserListQuery

{

LanguageLevelId = beginner.Id,

SortByNameDescending = true

};

// Получение отфильтрованного списка пользователей

var result = await service.GetUsersAsync(query);

// Проверка количества: ожидаем 2 пользователей

Assert.Equal(2, result.Users.Count);

// Проверка сортировки

Assert.Equal(["bob", "alice"], result.Users.Select(

u => u.Username));

// Проверка корректности фильтрации

Assert.All(result.Users,

u => Assert.Equal(beginner.Id, u.LanguageLevelId));

// Проверка корректности маппинга

Assert.Equal(1, result.Users.First().MemorizedWordsCount);

Assert.Equal(beginner.Code,

result.Users.First().LanguageLevelCode);

Assert.Equal(beginner.Name,

result.Users.First().LanguageLevelName);

// Проверка списка всех доступных уровней (для UI-фильтров)

Assert.Equal(2, result.Levels.Count);

}

Листинг 10 – Код метода CreateService

private static (IAdminUserService Service, SmartLangDbContext DbContext) CreateService()

{

// Инициализация контейнера внедрения зависимостей

var services = new ServiceCollection();

// Конфигурация InMemory базы данных с уникальным ID для изоляции тестов

services.AddDbContext<SmartLangDbContext>(options => options.UseInMemoryDatabase(Guid.NewGuid().ToString()));

// Регистрация сервисов бизнес-логики

services.AddAdminServices();

// Построение провайдера и получение экземпляров сервиса и контекста

var provider = services.BuildServiceProvider();

return (provider.GetRequiredService<IAdminUserService>(), provider.GetRequiredService<SmartLangDbContext>());

}

Результат тестирования сервиса AdminUserService представлен на рисунке 5.

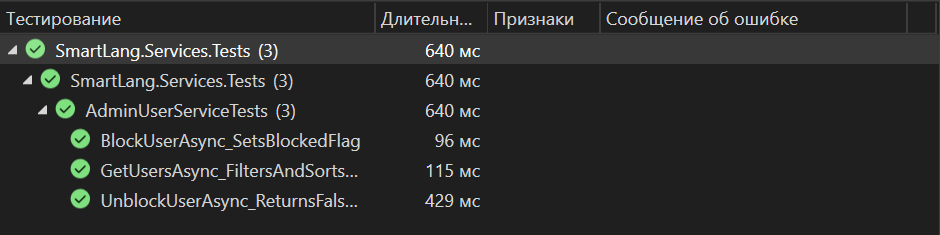


Рисунок 5 – Результат тестирования AdminUserService

## Функциональное тестирование

В ходе функционального тестирования веб-приложения была проверена корректность работы ключевых пользовательских сценариев. Тестирование страницы авторизации выполнено методом «чёрного ящика». Результаты тестирования представлены в таблице 1.

Таблица 1 – Набор тестов страницы авторизации

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Полученный результат** |
| Ввод существующего имени пользователя «Main» и корректного пароля «!Password1», нажатие на кнопку «Войти» | Открытие главной страницы. В панели навигации выведено имя пользователя «Main» | Совпадает с ожидаемым |
| Ввод существующего имени пользователя «Main» и некорректного пароля «123» | Вывод сообщения «Неверное имя пользователя или пароль» | Совпадает с ожидаемым |
| Ввод несуществующего имени пользователя «NoMain» и некорректного пароля «123» | Вывод сообщения «Неверное имя пользователя или пароль» | Совпадает с ожидаемым |
| Нажатие на кнопку «Войти», оставив поля ввода пустыми | Вывод подсказок «Введите имя пользователя» и «Введите пароль» | Совпадает с ожидаемым |
| Ввод существующего имени пользователя «Main» и нажатие на кнопку «Войти», оставив поле ввода пароля пустым | Вывод подсказки «Введите пароль» | Совпадает с ожидаемым |
| Ввод пароля «!Password1» и нажатие на кнопку «Войти», оставив поле ввода имени пользователя пустым | Вывод подсказки «Введите имя пользователя» | Совпадает с ожидаемым |
| Ввод существующего имени пользователя «Main», некорректного пароля «123» и нажатие на кнопку «Войти» 5 раз подряд | Вывод сообщения «Слишком много неудачных попыток входа. Попробуйте позже». Вход в систему невозможен до истечения 5 минут под этим именем пользователя | Совпадает с ожидаемым |
| Нажатие на ссылку «Создать профиль» | Открытие страницы регистрации | Совпадает с ожидаемым |

По результатам тестирования можно заключить, что разработанное приложение функционирует корректно и соответствует ожидаемому поведению.

# Инструкция по эксплуатации программного обеспечения

## Установка программного обеспечения

Для функционирования системы на стороне сервера необходимы следующие программные и технические средства:

* ОС Windows x86 64-бит или Linux x86 64-бит;
* MySQL Server не ниже 8.0;
* программное обеспечение для работы сайта: .NET SDK не ниже 8.0 и ASP.NET MVC 5;
* доступная оперативная память 3 ГБ;
* процессор с частотой не менее 1 ГГц и не менее 2 ядер;
* минимальный объем дискового пространства 10 ГБ.

Доступ к системе осуществляется с помощью браузеров, таких как Chrome (версии 90 и выше), FireFox (версии 88 и выше) и других, поддерживающих HTML5 и CSS3, необходимых для функционирования системы на стороне клиента. Также необходимы следующие технические средства:

* процессор с частотой 1 ГГц;
* доступная оперативная память 2 ГБ.

В качестве учётных данных используются следующие данные:

* Имя пользователя: [Main](mailto:profile@mail.com)
* Пароль: !Password1

## Инструкция по работе

При открытии веб-приложения пользователя встречает начальная страница с описанием возможностей платформы (представлена на рисунке 6). Для авторизации требуется нажать кнопку «Войти» и на открывшейся странице заполнить форму авторизации учетными данными. Вид формы авторизации представлена на рисунке 7.

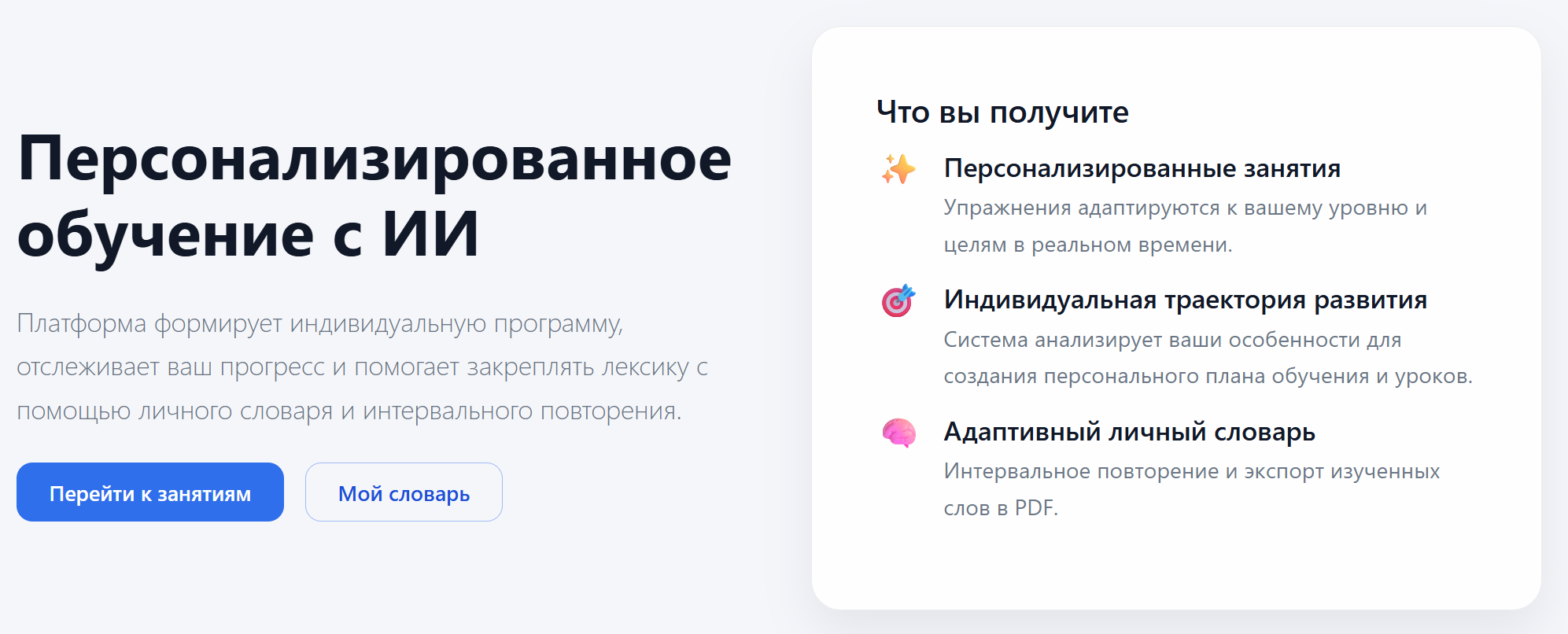


Рисунок 6 – Вид начальной страницы

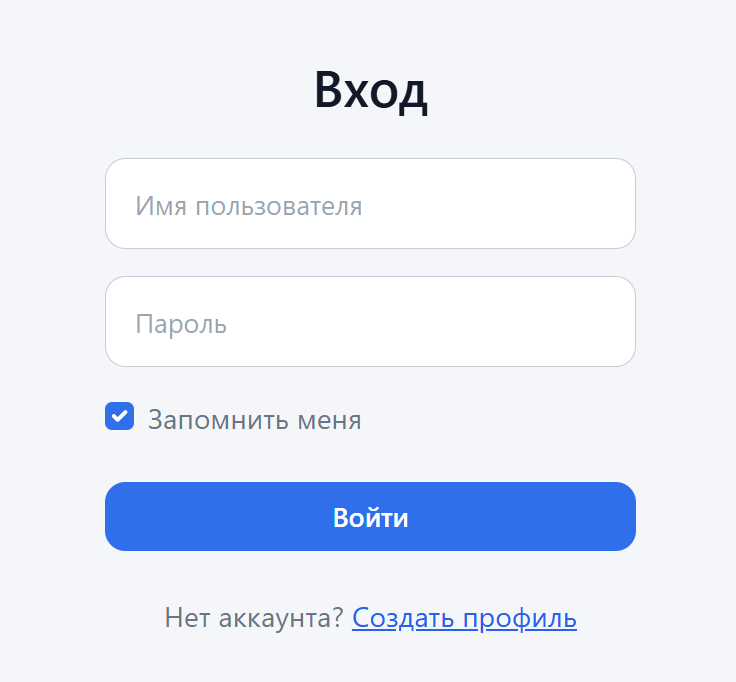


Рисунок 7 – Вид формы авторизации

После успешной авторизации осуществляется перенаправление на начальную страницу. Интерфейс обновляется под авторизованного пользователя: становятся доступными кнопки «Перейти к занятиям» и «Мой словарь». Пользователю в роли «Admin» доступна кнопка «Админ-панель» и видно наименование его роли на панели навигации. Пример внешнего вида панели навигации представлен на рисунке 8.

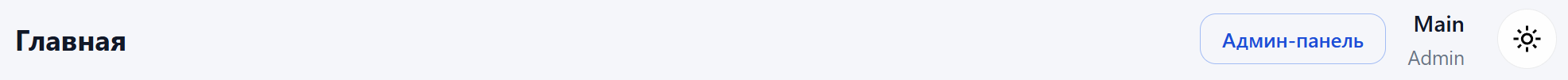


Рисунок 9 – Вид панели навигации

Для перехода к списку уроков требуется нажать на кнопку «Перейти к занятиям», после чего пользователь перенаправляется на страницу с уроками текущего раздела курса, где отображается название раздела, его содержание и описание текущего урока. Пример внешнего вида этой страницы представлен на рисунке 10.

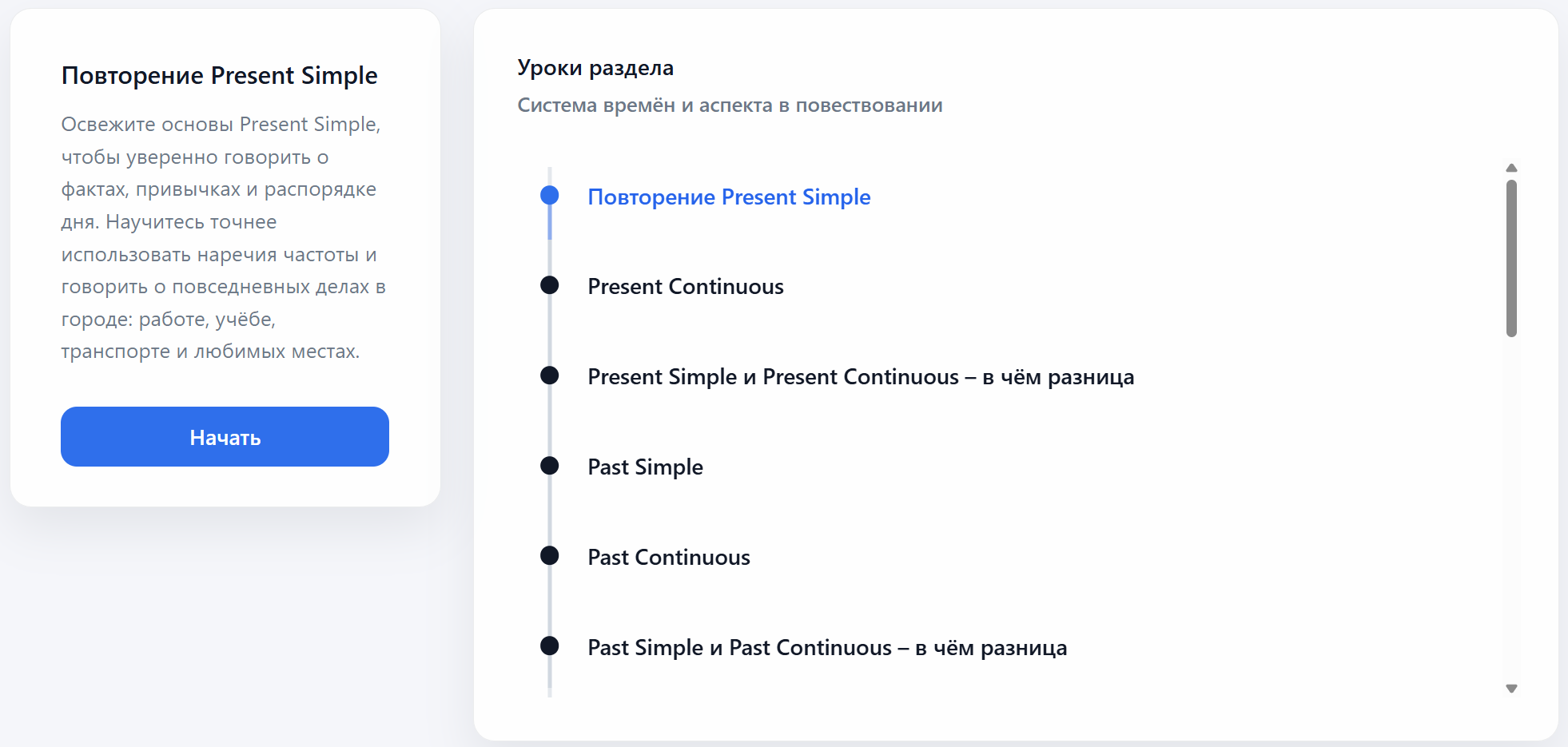


Рисунок 10 – Вид страницы со списком уроков

По мере прохождения уроков личный словарь пользователя автоматически пополняется новой лексикой. Для отслеживания прогресса и управления словарем необходимо нажать на имя пользователя в панели навигации. После нажатия осуществляется перенаправление на страницу профиля пользователя. В профиле отображается статистика обучения и список выученных слов, сгруппированных по датам. Навигация по датам осуществляется с помощью стрелок в верхнем правом углу, либо путем выбора конкретного дня через иконку календаря с последующим нажатием кнопки «Показать». Пример внешнего вида профиля представлен на рисунке 11.

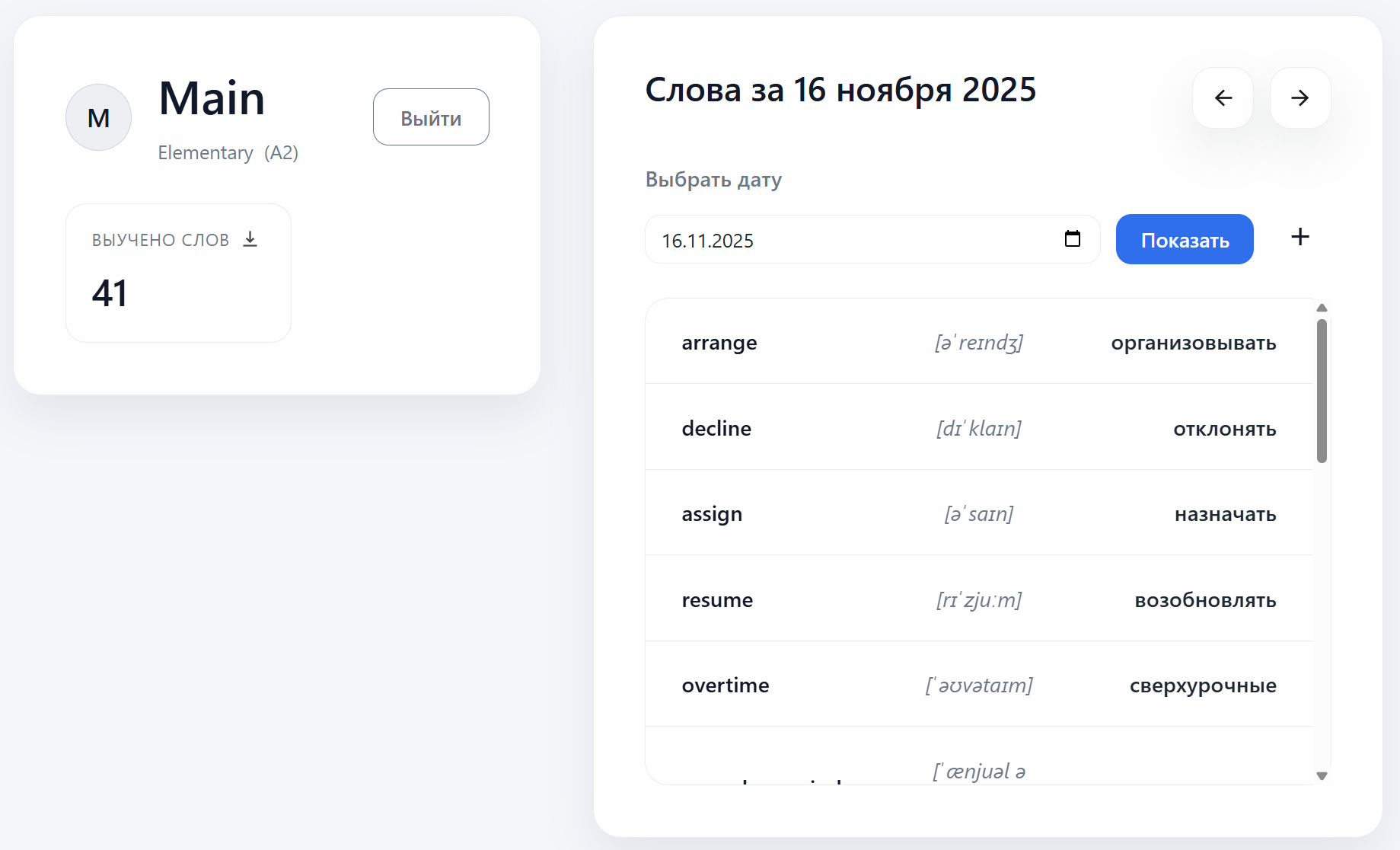


Рисунок 11 – Вид страницы профиля пользователя

Помимо автоматического пополнения словаря из пройденных уроков, в системе предусмотрена возможность ручного добавления лексики. Это позволяет пользователю самостоятельно расширять базу изученных слов для их последующего использования в упражнениях на закрепление.

Для добавления нового слова необходимо нажать кнопку с пиктограммой «плюс», расположенную справа от фильтра дат. После нажатия открывается модальное окно поиска, представленное на рисунке 12.

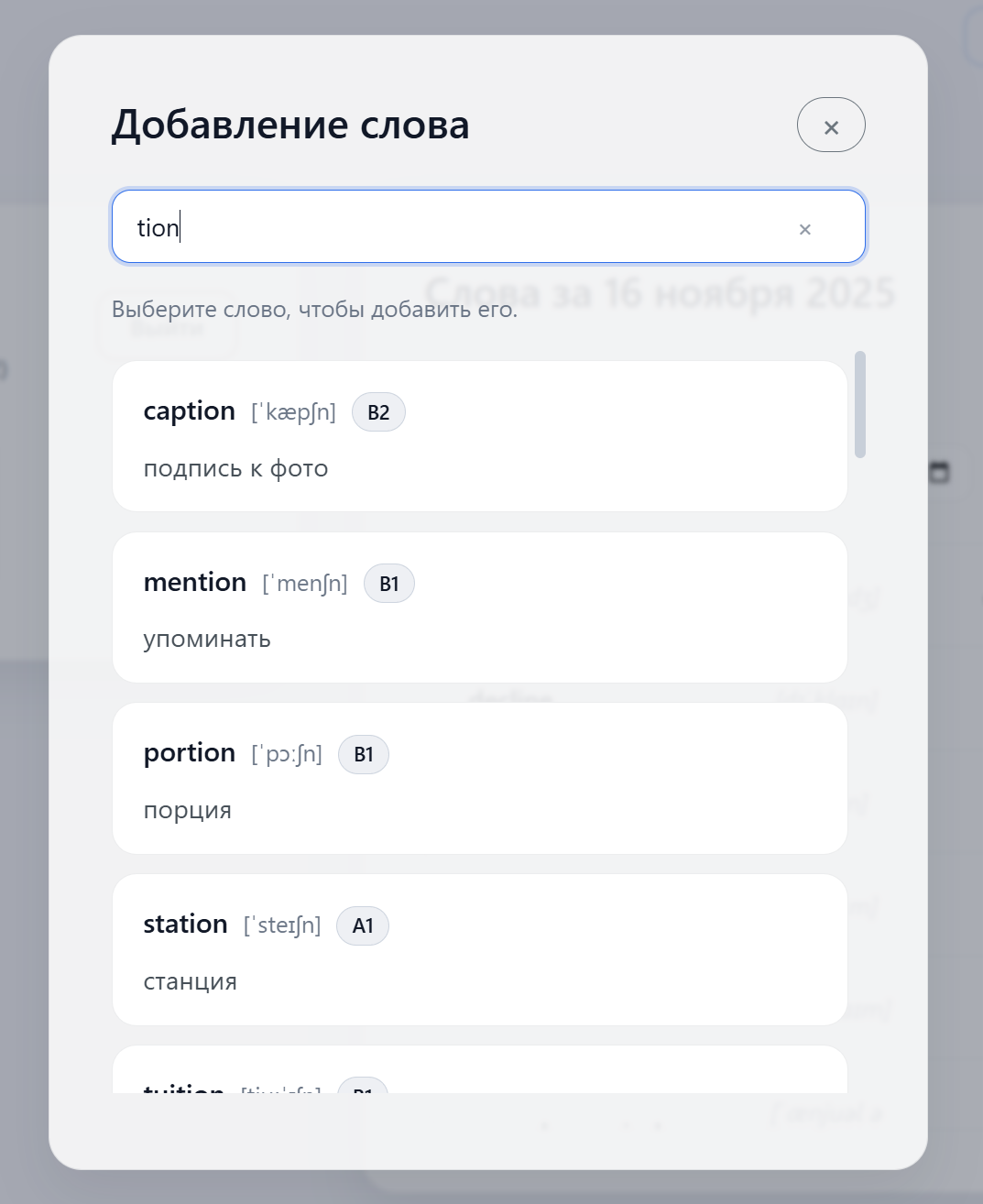


Рисунок 12 – Вид окна поиска слов

В этом окне пользователю требуется ввести слово или его часть в строку поиска. Система автоматически предложит список подходящих вариантов с указанием транскрипции, перевода и языкового уровня. Для добавления записи в словарь необходимо нажать на подходящее слово из списка.