***Know your words***

***A dictionary notebook project***

**Simeon Orlinov Rolev**

**№ 25630**

Know your words Docs

Contents

[Intro and purpose 2](#_Toc515569165)

[Requirements analysis 3](#_Toc515569166)

[Functional requirements 3](#_Toc515569167)

[Non-functional requirements 3](#_Toc515569168)

[Used technologies 3](#_Toc515569169)

[Features 4](#_Toc515569170)

[Google Translate 4](#_Toc515569171)

[The list view 4](#_Toc515569172)

[Developer docs and implementation 5](#_Toc515569173)

[Backend 5](#_Toc515569174)

[Registration and Authentication 5](#_Toc515569175)

[Database models 5](#_Toc515569176)

[REST API 5](#_Toc515569177)

[Frontend 6](#_Toc515569178)

[References 7](#_Toc515569179)

# Intro and purpose

Know your word is a simple language learning tool that supports saving new words, adding notes to them and a layout that allows you to test yourself. Its purpose is to give you a simple interface that allows you to learn new words. It is integrated with **Google Translate** which makes it possible to translate and save whole phrases from many to many languages. It is implemented as a **website**, which gives you the option to use it from any device as long as you have internet access. Its responsiveness allows for using it on mobile devices. The app has a simple database which allows for user authentication and personalization.

# Requirements analysis

## Functional requirements

The app should be able to let you create accounts and save words for it. This requires a database for the users and the associated with them translations. It should be a responsive and one-page app to make it faster and usable from any device. The app should allow you to hide your input or output of your translations in order to test your vocabulary. It should support multi-language and use externally the Google Translate API. It should be able to register both single words and whole phrases.

## Non-functional requirements

The performance of the app should be fast, especially the frontend, since it is a webapp. The backend should provide a reliable, secure and fast authentication system for the user accounts, as well as an API that will connect us to the database. The app should be responsive and any-device-compatible. It should be cross-browser compatible. The JS code that will be used for the frontend should be minimized and encrypted before production deployments.

# Used technologies

This project uses Django, Django Rest Framework, React JS, Node JS, SQLite, SASS + bootstrap, Webpack and a bonus server that has the Google Translate token generator deployed in order to make the Google Translate option available for free.

Django is a good choice for any webapp that requires authentication, stability and database management. It is a tested framework that has proven to be reliable and has a great documentation and built-in tools. Its REST Framework is a simple way to connect to the database and setup an API.

ReactJS is a leading JavaScript platform that is here to stay. It provides a very fast VirtualDOM technology that fastens the HTML element rendering drastically. It relies on component based approach which makes many modules easy to test and reuse. It supports tons of features that can be imported later on.

SQLite is the default “small” database that comes with Django when you initiate a new project, but for this one it is completely enough, as we store only translations and users.

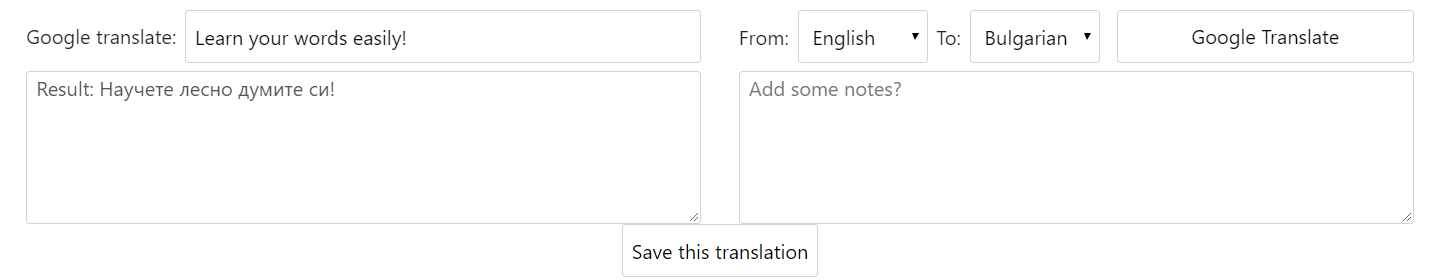
SASS is a tool that allows us to use tons of syntactic sugar to our style implementation.

Webpack is a leading technology for modern web development and takes care of all bundling of files for both production and development environments.

# Features

## Google Translate

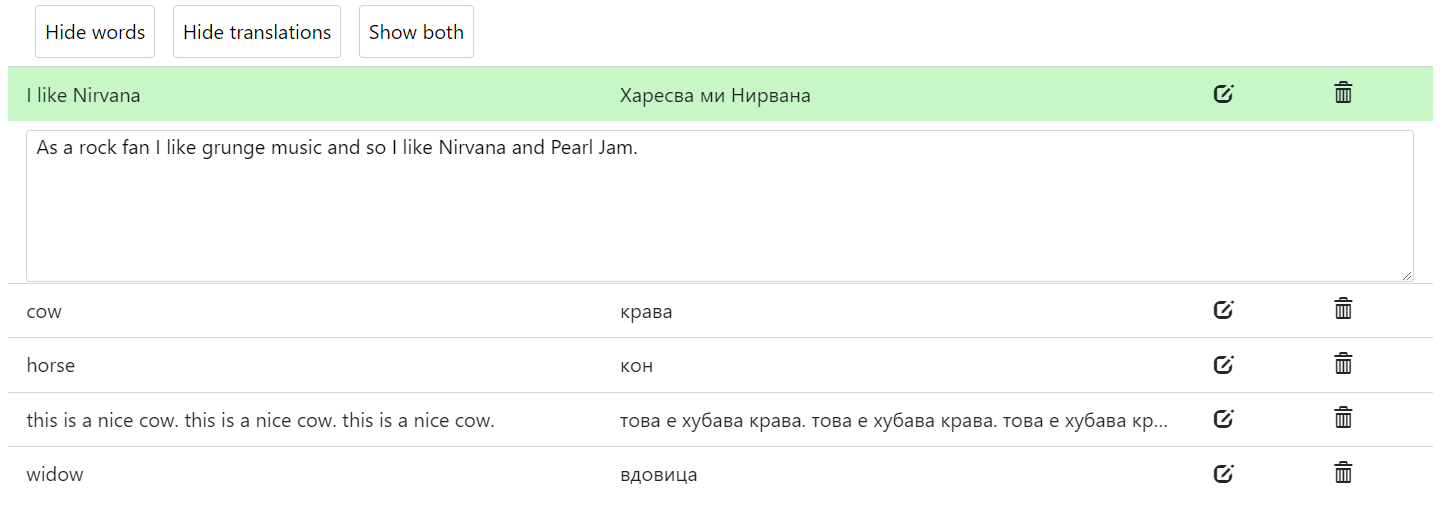
This website uses Google Translate services for free. This allows for translating phrases from all the supported by Google languages.



Steps:

1. Type your word/phrase you’d like to translate.
2. Select ‘From’ and ‘To’ languages.
3. Click ‘Google Translate’.
4. Check out the result the Google returned.
5. (Optional) Add some notes if you want. For example, if you translated a word from a song, you can add the context from the lyrics where you heard this word being used.
6. You can now Save the translation and it will appear in your list view, described below.

## The list view



In the list view you will now be able to see your last saved translations. When you click on one, you can see the notes that you added when you saved the word or phrase. At the top of the list view you have the buttons ‘Hide words’, ‘Show words’ and ‘Show both’. When you click on ‘Hide words’, the first column gets hidden and you can test yourself. Respectively, ‘Hide translations’ does the opposite and ‘Show both’ displays back the initial list view with both columns visible.

# Developer docs and implementation

## Backend

### Registration and Authentication

Registrations and login system uses the default Django User.

### Database models

The translations’ code for models and database entries is available under the folder ‘sampleapp/models.py’. The Translations are being ordered by last modified then requested. They include input, output, notes, languages and last modified fields. Each translation entry is being associated with the user that creates it using a foreign key. Deleting the user will cascade the deletion of all his translation entries as well. The only field that is allowed to be blank is the “notes” field as we assume that the user may not want to save any. Django automatically sets unique <id> property to each entry.

### REST API

The REST API service uses the Django REST API framework. This current implementation uses a customized rest\_framework.viewsets.ModelViewSet and it supports:

* List all translations for the current user: ‘/api/translations/’
* POST new translations at: ‘api/translations/’
* Manage GET/PUT/PATCH/DELETE for individual entries at: ‘api/translations/<id>’.

The code for the REST API can be found under ‘sampleapp/api.py’. The serializers.py in the same directory defines the way we manage JSON information and pass it back and forth with the frontend. For now, we just parse the whole thing to JSON. The urls.py file contains the definition of the described above api url paths, automatically generated by the Django REST Framework’s SimpleRouter().

## Frontend

As we leave the login/signup page, we go to the main home page. This is where we use ReactJS and the whole site is a single page app so we do not use the Django navigation anymore. We implement functionalities and view as components. For example, the whole Google Translate form and functionality is implelemted under src/js/components/google-translator.jsx and the list view under list-view.js. All the references to the backend API are defined in the src/js/api.js module.

The components:

1. GoogleTranslator

Renders the HTML component and sets the login behind it. Its interface handles the changes of languages, saving notes and getting the translation itself done.

1. TranslationsList

Renders the list of translations that we loaded from the API. Displays the edit and delete options for a certain translation entry. Gives us the options to hide the input or output columns for our translations ( hideWords / hideTrans / showWords methods ). It supports some real time updates of the list view, for example editing and deleting of items without requesting them from the API. It contains a single reference of the EditTranslationDialog that we will describe below

1. EditTranslationDialog

Component for editing a word that we already added. We can open it from the edit icon that is displayed on the right side of each translation in the list view.

1. Translation

The root component that just matches the API responses and displays the items in the list view. Communicates directly with the API with its delete and edit methods that work with IDs.

We handle all the styles using BOOTSTRAP and SCSS compilers. Code is available under src/scss folder. It contains:

* \_common.scss – Defines some common visual styles of elements that we use often, like buttons, inputs, textareas etc.
* \_mixins.scss – Defines some common reusable visual parts that we may include in elements at any point.
* \_variables.scss - Defines some global constants for colors, borders that we may reuse in the project styles and replace easily.
* Base.scss – loads all the css for the project
* Dialog.scss – adds some styles for the dialog that we may use in the site, for example the “Edit word” dialog. Sets the background layover that covers the whole screen and positions the dialog in the middle.
* Google-translator.scss – defines the styles for the Google Translate component.
* List-item.scss – defines the styles we use for the list items that represent out translations.

# References

GitHub: <https://github.com/SimeonRolev/KnowYourWords>

Django: <https://www.djangoproject.com/>

Django REST Framework: <http://www.django-rest-framework.org/>

React JS: <https://reactjs.org/>

Create-react-app: <https://www.npmjs.com/package/create-react-app>

Bootstrap: <https://getbootstrap.com/>