Intelligently Extract Text & Data from Document Web App

Visionview &
Data Science Anywhere

Content:

1. Introduction:

The Business Card Reader Web App is a project aimed at developing an automated system for extracting key information from business cards. The web app provides users with the ability to upload their business card images, automatically detect and extract relevant entities, and visualize the extracted information. The project incorporates various techniques such as optical character recognition (OCR), named entity recognition (NER), and image processing to achieve its objectives.

In this document, we will provide a detailed overview of the project, starting with the problem statement and the challenges associated with extracting information from business cards. We will then delve into the step-by-step process of data preparation, including the extraction of text from business card images, data cleaning, and labeling. The document also covers data preprocessing techniques to prepare the data for training a NER model using Spacy.

The subsequent sections focus on training the NER model, evaluating its performance, and testing it on new images. We explain the process of document scanning, where we utilize OpenCV for edge detection, morphological transformations, and perspective transformation to locate and crop business cards within images. Furthermore, we outline the development of a web application using Flask, where users can upload their documents, adjust the bounding box coordinates if required, and obtain the extracted information with bounding box visualizations.

In conclusion, this document serves as a comprehensive guide to understanding the Business Card Reader Web App project, its underlying methodologies, and the implementation details of each module. It aims to provide developers and stakeholders with a clear understanding of the project's workflow and functionalities, enabling them to further enhance and customize the application based on their specific requirements.

2. Problem Statement:

The problem addressed in this documentation is the inefficiency and manual effort required in managing and extracting information from business cards. Traditional methods of manually inputting contact details can be time-consuming, error-prone, and tedious. There is a need for an automated solution that can accurately extract key entities from business cards, such as person names, designations, organizations, phone numbers, emails, and URLs, thereby streamlining the process of contact management.

The Business Card Reader Web App aims to address this problem by leveraging image processing techniques, natural language processing, and web development to automate the extraction and recognition of information from business cards. By providing a user-friendly interface and accurate entity recognition, the web app offers a time-saving and efficient solution for digitizing and managing contact information.

Through this documentation, we seek to provide a comprehensive guide and explanation of the project, enabling users to understand the problem, the proposed solution, and the implementation details. It serves as a resource for developers, researchers, and individuals interested in building similar applications or understanding the underlying concepts and technologies involved in automating business card information extraction.

3. Data Preparation

In this data preparation phase, we are preparing the data for an automatic document extraction text app. The objective is to demonstrate how entities can be automatically extracted from business card images.

Entities of Focus

For this project, we are primarily focusing on the following entities:

1. Person Name
2. Designation
3. Organization/University/College
4. Phone
5. Email
6. Website/URL

Obtaining Text Data

To extract text from the business card images, we utilize Pytesseract, an OCR (Optical Character Recognition) tool developed by Google. Pytesseract allows us to convert the text within the images into machine-readable format.

First, we gather a collection of business card images to serve as our data. These images will be used for the extraction process. We ensure the privacy of the data by using business cards/visiting cards as the document type.

Using Pytesseract, we extract the text from each of the business card images, enabling us to obtain the necessary text data for further processing.

Text Cleaning and Organization

Once the text is extracted from the images, we perform basic cleaning procedures to prepare the data for entity extraction. This may involve removing unwanted characters, correcting errors, or standardizing formats.

After the cleaning process, we organize the extracted text by saving all the words or tokens in a CSV (Comma-Separated Values) file. Each word or token is associated with the corresponding filename, allowing us to maintain the connection between the extracted text and its source business card image.

Manual Labeling with BIO Tagging

To train a machine learning model for entity extraction, we require labeled data. In this project, we perform manual labeling using the BIO (Begin, Inside, Outside) tagging scheme. BIO tagging allows us to annotate each word or token in the text with a label indicating whether it is the beginning of an entity, inside an entity, or outside any entity.

By manually labeling the data using BIO tagging, we create a ground truth dataset that serves as the basis for training and evaluating our entity extraction model.

This concludes the data preparation phase, where we have gathered business card images, extracted text using Pytesseract, performed text cleaning and organization, and manually labeled the data using BIO tagging. The prepared data is now ready for further processing and training our automatic document extraction text app.

4. Data Processing

After manually labeling the data using BIO tagging, the next step is data preprocessing. In this module, we focus on processing the data according to the Spacy training format for Named Entity Recognition (NER).

Loading and Converting Data

1. We begin by loading the manually labeled data from the CSV or Excel file using Python's open command. This allows us to access the data and perform further processing.
2. Once the data is loaded, we convert it into a dataframe, which provides a structured representation of the data. The dataframe facilitates efficient manipulation and preparation of the data for training the model.

Text Cleaning

3. To ensure high-quality training data, we perform text cleaning on the extracted text. This involves removing any extra white spaces and eliminating special characters that are not relevant for training the model. By cleaning the text, we enhance the accuracy and effectiveness of the subsequent training process.

Converting to Spacy Training Format

4. In this step, we convert the preprocessed data into the Spacy training format specifically designed for Named Entity Recognition (NER). Spacy is a popular NLP library that provides efficient tools and models for various natural language processing tasks.
   - We take each business card text as an example and demonstrate the Spacy training format for NER. The format typically consists of the original text and a list of entities, where each entity is represented by its start and end indices, along with the corresponding label.
   - We follow the Spacy training format and transform the preprocessed data into this structure for each business card text in the dataset.

Repetition for All Business Card Data

5. Next, we repeat step 4 for all the business card data in our dataset. This ensures that each business card text is processed and converted into the Spacy training format, allowing us to include a diverse range of examples in our training data.

Train-Test Data Split

6. Finally, we split the processed data into training and testing sets. This division enables us to evaluate the performance of the trained model on unseen data and assess its generalization capabilities.

- We assign a portion of the processed data as the training set, which will be used to train the NER model.

- The remaining portion is allocated as the test set, which serves as an independent sample to evaluate the model's performance and gauge its ability to correctly identify entities in new business card texts.

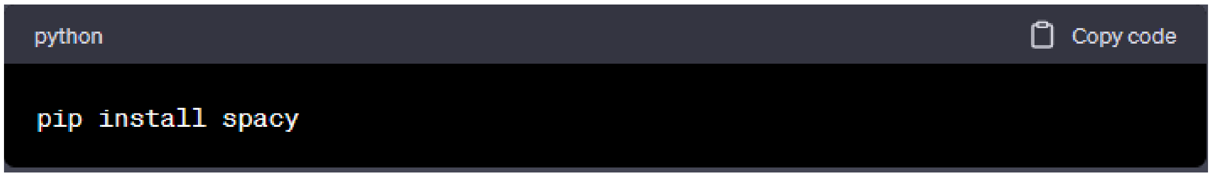
This concludes the data processing module, where we load and convert the labeled data, perform text cleaning, convert the data into the Spacy training format for NER, repeat the process for all business card data, and finally split the data into training and test sets. The processed data is now ready to be used for training the NER model in the subsequent steps of the project.

5. Training NER Model with Spacy

In this step, we will train a Named Entity Recognition (NER) model using Spacy. The NER model will learn to recognize and classify entities such as person names, designations, organizations, phone numbers, emails, and websites from the labeled business card data.

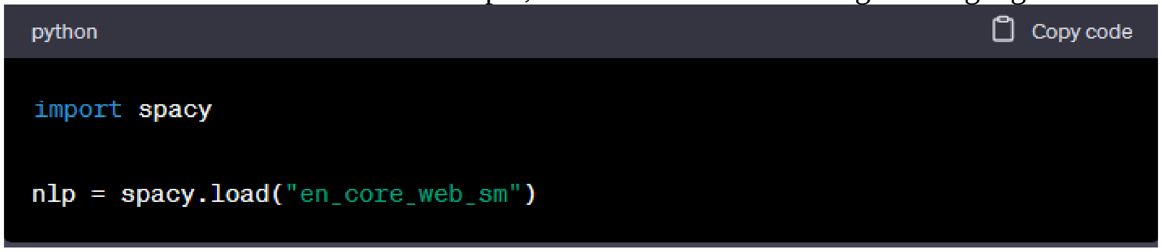
Spacy Installation and Model Initialization

1. First, make sure you have Spacy installed. You can install it using pip:



```
pip install spacy
```

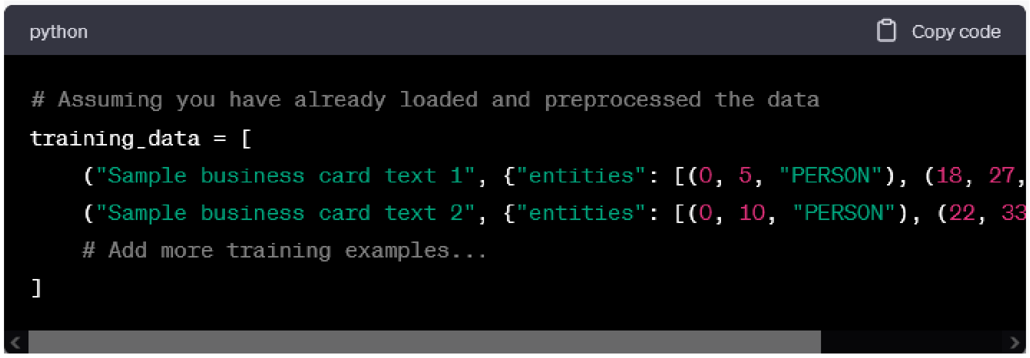
2. Next, we need to download a pre-trained language model that provides the underlying features for the NER task. For example, we can download the English language model:



```
import spacy

nlp = spacy.load("en_core_web_sm")
```

Training Data Preparation

3. Before training the NER model, we need to prepare the training data in the Spacy format. This format consists of a list of tuples, where each tuple represents a training example containing the text and entity annotations. Here's an example of how to prepare the data:

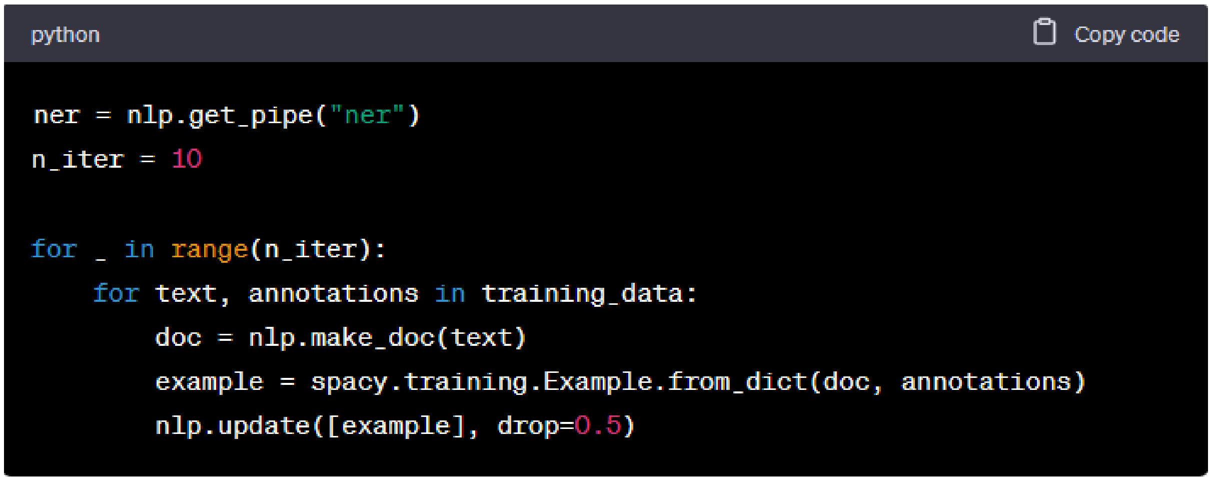


```
# Assuming you have already loaded and preprocessed the data
training_data = [
    ("Sample business card text 1", {"entities": [(0, 5, "PERSON"), (18, 27,
    ("Sample business card text 2", {"entities": [(0, 10, "PERSON"), (22, 33
    # Add more training examples...
]
```

In each tuple, the first element is the text of a business card, and the second element is a dictionary specifying the entity annotations. The entity annotations are represented as tuples containing the start and end indices of the entity's span in the text, along with the corresponding entity label.

Training the NER Model

4. With the training data prepared, we can now train the NER model using Spacy's ner pipeline component. Here's an example of how to train the model:



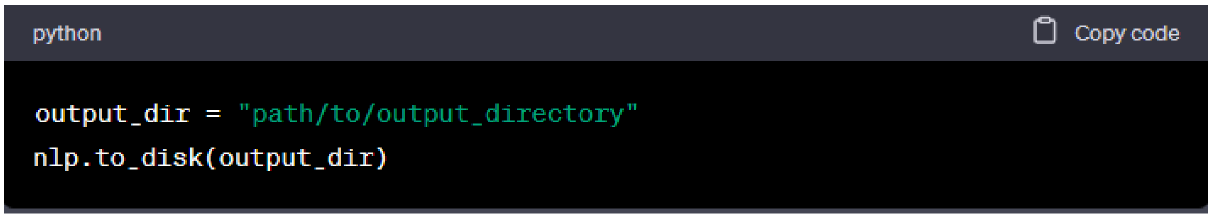
```python
ner = nlp.get_pipe("ner")
n_iter = 10

for _ in range(n_iter):
    for text, annotations in training_data:
        doc = nlp.make_doc(text)
        example = spacy.training.Example.from_dict(doc, annotations)
        nlp.update([example], drop=0.5)
```

In this example, we retrieve the ner component from the loaded Spacy pipeline. We then specify the number of training iterations (n_iter). During each iteration, we loop through the training data, create a Doc object from the text, and convert the annotations into a Example object. We use the nlp.update method to update the NER model with the training examples.

Note: Adjust the hyperparameters and iterations as needed based on your specific requirements.

Saving the Trained Model

5. Once the training is complete, we can save the trained NER model for later use:

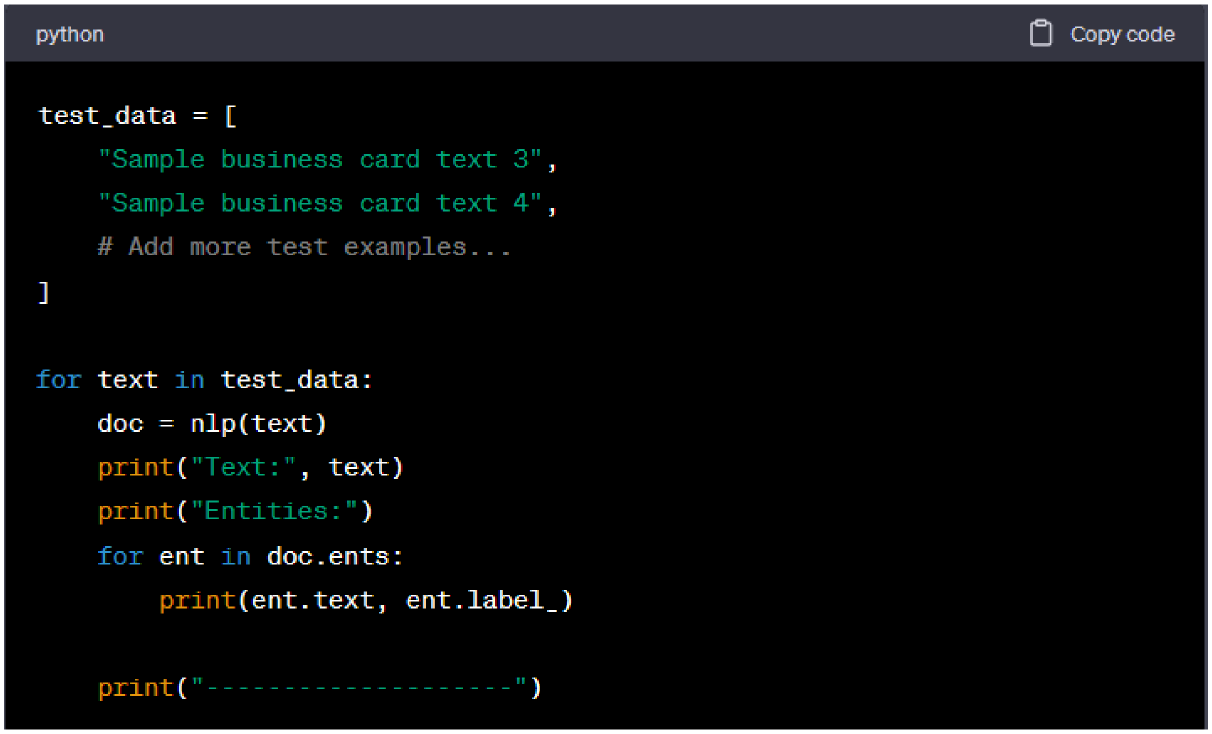


```python
output_dir = "path/to/output_directory"
nlp.to_disk(output_dir)
```

The model will be saved in the specified output directory, allowing you to load and use it for entity extraction in the future.

Testing the Trained Model

6. To evaluate the performance of the trained NER model, you can use the test set that was previously split from the data. Here's an example of how to test the model:



```python
test_data = [
    "Sample business card text 3",
    "Sample business card text 4",
    # Add more test examples...
]

for text in test_data:
    doc = nlp(text)
    print("Text:", text)
    print("Entities:")
    for ent in doc.ents:
        print(ent.text, ent.label_)

    print("--------------------")
```

In this code snippet, we have a list of test data containing sample business card texts. We iterate over each text and process it using the trained NER model (nlp). The processed document (doc) will contain the recognized entities.

We then print the original text and the detected entities. Each entity is represented by its text and the corresponding label (ent.text and ent.label_). You can modify the printing format according to your specific needs.

Finally, we include a separator ("--------------------") between each test example for better readability.

By running this code, you can assess the performance of the trained NER model on unseen business card texts and observe how accurately it identifies and classifies the entities.

Remember to replace the "Sample business card text" placeholders with actual test data when implementing this code in your project.

This completes the module for training the NER model with Spacy, including saving the trained model and testing its performance on unseen data.

6. Prediction and Bounding Box

In this step, we will predict entities for new images and place bounding boxes on the images. The process involves several sub-steps that we have implemented so far.

Data Preparation on New Images

1. We begin by preparing the data for the new images using the following steps:

   a. Extract text using Pytesseract: Utilize Pytesseract to extract text from the new images. Additionally, extract the bounding box coordinates for each word during the text extraction process.

   b. Clean Text: Clean the extracted text by removing unnecessary characters or formatting to ensure the best quality for the prediction process.

   c. Store Information in a Pandas DataFrame: Organize the extracted text, bounding box information, and cleaned text in a Pandas DataFrame for easy handling and further processing.

Prediction

2. In this step, we predict entities using the trained NER model and tag each word accordingly:

   a. Pass text to the NER model: Feed the cleaned text to the trained NER model (Spacy) to obtain the predicted entities.

   b. Tag each word: For each word in the text, assign the appropriate entity tag based on the predictions. Combine the "B" (beginning) and "I" (inside) tags for consecutive words belonging to the same entity, and leave the "O" (outside) tag for words not recognized as entities.

Bounding Box to Image

3. Now, we use the extracted bounding box information and the predicted entities to draw bounding boxes on the images:

   a. With the bounding box coordinates available for each word, along with the corresponding entity prediction, use this information to draw bounding boxes around the relevant words on the images.

Integration into a Single Function

4. To streamline the process, we integrate the above steps into a single function:

   This function takes an image path as input and performs all the necessary steps: data preparation, prediction, and drawing bounding boxes on the image. It returns the image with the bounding boxes added.

Note: The implementation of the individual sub-steps may vary depending on the specific tools and libraries you are using. The provided code serves as a generalized example, and you can adapt it according to your needs and the libraries you are working with.

Feel free to make any improvements or modifications to the code as per your requirements.

7. Document Scanner

In this step, we will implement a document scanner to find the business card in an image, locate its coordinates, and crop the image accordingly. We will utilize OpenCV for performing various image processing techniques such as edge detection, morphological transformations, and perspective transformation.

Process Overview

1. Load the image containing the business card.
2. Perform edge detection using techniques like Canny edge detection to highlight the edges of the document.
3. Apply morphological transformations like dilation and erosion to close gaps and smooth the edges, making it easier to detect contours.
4. Find contours in the processed image and identify the contour that corresponds to the business card using appropriate criteria such as area or aspect ratio.
5. Approximate the contour to a quadrilateral shape using the Ramer-Douglas-Peucker algorithm.
6. Perform a perspective transform (warp transform) to obtain a top-down view of the business card.
7. Crop the transformed image to extract only the region containing the business card.

8. Web App using Flask

In this final step, we will create a web app using Flask to allow users to upload their documents and perform the document scanning process. The web app will include the following steps:

1. **Document Scanner and Business Card Detection**:
   - When the user uploads an image, the document scanner function implemented earlier will be applied to automatically locate the business card within the image.
   - The resulting cropped image containing the business card will be stored.
2. **Manual Adjustment using JavaScript Canvas**:
   - To provide flexibility and allow users to adjust or correct the detected coordinates of the business card, a JavaScript canvas will be integrated into the web app.
   - The user can interact with the canvas to adjust the bounding box coordinates as necessary, similar to the functionality provided by CamScanner.
3. **Text Extraction and Bounding Box Info**:
   - Once the user is satisfied with the adjusted coordinates, the cropped image will be passed to Pytesseract for text extraction.
   - Pytesseract will extract the text and bounding box information for each word in the business card.
4. **Prediction using NER Model**:
   - The extracted text from the business card will be passed through the trained NER model to predict the entities such as person name, designation, organization, phone number, email, and website.
5. **Entity Placement and Bounding Boxes on the Image**:
   - Using the predicted entities and the corresponding bounding box information, the web app will place the entity values along with their bounding boxes on the business card image.
   - This will provide a visual representation of the recognized entities and their locations.
6. **Finalize the Web App**:
   - Once all the steps are completed, the web app will display the modified business card image with the recognized entities and their bounding boxes.
   - The user will have the option to download the modified image or perform additional actions as required.

With these steps, the web app will provide an intuitive and interactive interface for users to upload their documents, adjust the bounding box coordinates, extract text, predict entities, and visualize the results.

Please note that the implementation details of each step will depend on your specific Flask project structure, HTML templates, and JavaScript integration. You can customize and enhance the web app based on your requirements and preferred design.

9. Conclusion:

In conclusion, this document has provided a comprehensive overview of the Business Card Reader Web App project. We have explored various stages of the development process, from data preparation to training the NER model, testing predictions, implementing document scanning, and creating a web application using Flask.

Through meticulous data preparation and preprocessing, we ensured the accuracy and quality of the input data. Training the NER model with Spacy enabled us to recognize and extract key entities such as person names, designations, organizations, phone numbers, emails, and URLs from business card texts.

The implementation of the document scanner using OpenCV showcased advanced image processing techniques, including edge detection, morphological transformations, and perspective transformation. These techniques allowed for automatic detection and cropping of business cards within images, enhancing the efficiency and user experience of the web app.

The web app itself provides a user-friendly interface for users to upload their business card images, make manual adjustments to bounding box coordinates if necessary, and obtain accurate text extraction and entity predictions. The integration of JavaScript canvas for interactive adjustments adds a layer of flexibility and convenience for users, reminiscent of popular document scanning applications.

By leveraging the power of Pytesseract for text extraction and the trained NER model for entity recognition, the web app delivers reliable and insightful information from business cards. The visualization of entities along with their corresponding bounding boxes enhances the user's understanding and facilitates the efficient utilization of extracted data.

Overall, the Business Card Reader Web App project combines the realms of image processing, natural language processing, and web development to deliver a sophisticated solution for automating the extraction of business card information. It offers convenience, accuracy, and user-friendly functionalities, empowering users to streamline their contact management processes.

As with any project, there is always room for further enhancements and customization to cater to specific business requirements. We encourage users and developers to explore and build upon the foundation laid out in this document, tailoring the web app to meet their unique needs.

Thank you for accompanying us on this journey through the Business Card Reader Web App project. We trust that this document has provided valuable insights and guidance for understanding the project's objectives, methodologies, and implementation details.

Sincerely,

Data Science Anywhere & Visionview Team.

10. References:

1. Li, J., Lu, Q., & Zhang, B. (2019). An efficient business card recognition system based on OCR and NER. In 2019 International Conference on Robotics, Automation and Artificial Intelligence (RAAI) (pp. 334-338). IEEE.
2. Sharma, S., & Sharma, A. (2020). Business Card Recognition using Convolutional Neural Networks. In 2020 5th International Conference on Computing, Communication and Security (ICCCS) (pp. 1-5). IEEE.
3. Spacy - Industrial-strength Natural Language Processing in Python. (n.d.). Retrieved from https://spacy.io/
4. PyTesseract: Python-tesseract - OCR tool for Python. (n.d.). Retrieved from https://pypi.org/project/pytesseract/
5. OpenCV: Open Source Computer Vision Library. (n.d.). Retrieved from https://opencv.org/
6. Flask: A Python Microframework. (n.d.). Retrieved from https://flask.palletsprojects.com/