

Abstract:

This invention introduces an innovative system for web development, enabling users to dynamically position and manipulate HTML elements within a web document through an intuitive drag-and-drop interface. This system significantly enhances the efficiency and flexibility of web form creation and user interface design, leveraging advanced recognition algorithms to identify HTML elements and seamlessly integrate them into the desired locations within the web document.

Claims:

A method for dynamically arranging HTML elements in a web document, comprising:

Detecting a drag action initiated by a user on a selectable HTML element;

Identifying a drop target within the web document where the HTML element is to be placed;

Embedding the HTML element into the drop target location, maintaining the web document's structural integrity.

The system of claim 1, wherein the HTML elements include, but are not limited to, text boxes, checkboxes, radio buttons, and custom-designed elements.

A computer-implemented system for enabling drag-and-drop functionality in web documents, comprising:

A user interface allowing the selection and dragging of HTML elements;

A recognition module for identifying permissible drop target locations within the web document;

An integration module for embedding the dragged HTML element into the drop target location upon release.

This sample is intended to guide your discussion with a patent attorney, who can refine and expand upon these ideas to ensure comprehensive protection for your invention.

let's also include how this will revolutionize tech by disrupting the status quo technology DocSign, because it's Native to HTML, thereby integrating into the Fortune 500 companies Operating Software and ERP, Oracle Etc. Thus allowing for ease of Database integration for Big Data Analytics on every form field, let alone the Simplicity of Creating FORMs in Microsoft Word, verses needing to Pay high rate Software Development to Code each form, which equates to a Massive Reduction in Burn Rate and Speed of Implementation of Ongoing Priorities, Regulatory, Compliance Chances, Revenue Cycle Management, Controls etc.

This invention not only simplifies form creation and integration but also disrupts traditional document signing and management technologies such as DocSign by being native to HTML. Its seamless integration into Fortune 500 companies' Operating Software and ERP systems, including Oracle, transforms database integration for Big Data Analytics. By enabling form creation directly in Microsoft Word without the need for extensive software development, this system significantly reduces burn rates and accelerates the implementation of ongoing priorities, including regulatory compliance, revenue cycle management, and controls.

This abstract highlights the invention's potential to revolutionize technology by offering a simpler, more integrated solution that could disrupt existing market standards.

The Ease of Integration, Native HTML vs the bulkiness of separately working with PDF files and the associated routing, the lack of access to the data sets, and especially the implied dependency on software developers for native form integration vs Microsoft Word instant upload, is the future of Form and Contract Integration with Enterprise Software.

Furthermore, the Integration of this Functionality creates massive efficiency in workflow operations as it enables the customization of an enterprises (SOPs) Standard Operating Procedures, within the native framework to be triggered from any identified instances of interest in order to Prompt the user to Read the Specific Guidance SOP in order to avoid a deviation in the proper protocol, additionally a Form utilizing our same technology can be triggered to collect specific data relevant to the Instance, thereby providing a fully

integrated, holistic approach to Informing Necessary Actions and Collecting Critical Data in an automated process which is fully customizable on a codeless administrative panel, thereby achieving degrees of magnitude savings in software engineering and timeframes for customization for each enterprise client, to achieve database integrated customized form field capture.

I invented a new protocol which allows for instant form creation layered on top of Word Documents, here is how we coded it:

The provided code outlines a Vue.js component designed for creating interactive forms on top of Word documents. It features drag-and-drop functionality, allowing users to place form elements like text fields, date pickers, and signatures within table cells identified by "TD" tags. This innovative system also supports dropping elements into various HTML tags for flexible form design. Elements are dynamically rendered using Vue components, and the state is managed via a Vuex store, demonstrating a sophisticated approach to form creation and manipulation in web applications.

The architecture described leverages Vue.js to enhance Word documents with interactive form capabilities. It uniquely identifies table cells ("TD") and other HTML elements to serve as drop targets for form elements like text inputs and signatures. Through drag-and-drop interactions, users can dynamically place form fields within the document. The system uses Vue components for each form field type, managed by Vuex for state management, enabling real-time updates and interactions within the document. This approach combines the flexibility of document editing with the power of web application interactivity.

Additionally, in cases as needed we are able to recognize any HTML elements (ex: `<p>`, `<div>`, ``, `<h1>`, `<h2>`, ``, ``, `<hr>`, etc) as additional Targets enabling Draggable Elements to place as "inline" or "display block", "overflow" format, this will cover all potential positions that are in the field of possibilities, and thereby creates

an automatic draggable form creation from starting with any simple Document like Microsoft Word.

ChatGPT

To achieve the task of recognizing various HTML elements (like `<p>`, `<div>`, ``, etc.) as targets for draggable elements, you'll need to extend the drag and drop functionality. You can modify the `dragOver` and `drop` methods to accept these elements as valid targets. Additionally, you would adjust the styling logic to accommodate 'inline' and 'display block' placements based on the type of the dropped element. This involves dynamically setting CSS properties for each dropped element to ensure they adhere to the document's flow, either inline or as blocks, and handling overflow issues by setting appropriate CSS overflow properties on the container elements.

To enhance the drag-and-drop functionality to recognize various HTML elements as targets and adjust their display styles accordingly, you can modify the Vue component's methods. This code example extends the `dragOver` and `drop` methods to handle different HTML elements and dynamically sets their styles for inline or block display, including overflow handling:

```
// Inside your Vue component
```

```
methods: {  
  
  dragOver(e) {  
  
    const target = e.target;  
  
    const validTargets = ["TD", "P", "DIV", "SPAN", "H1", "H2", "UL", "OL", "HR"];  
  
    if (validTargets.includes(target.tagName)) {  
  
      this.removeHoveredClass();  
  
      target.classList.add("hovered");  
  
    }  
  
  }  
  
}
```

```
},  
  
drop(e) {  
  
    e.preventDefault();  
  
    const target = e.target;  
  
    const droppedField = e.dataTransfer.getData("field");  
  
    const validTargets = ["TD", "P", "DIV", "SPAN", "H1", "H2", "UL", "OL", "HR"];  
  
    if (!validTargets.includes(target.tagName)) return;  
  
    target.setAttribute("data-name", droppedField);  
  
    target.setAttribute("drop-item", true);  
  
    // Remove existing content if necessary or adjust it based on your requirements  
  
    // target.innerHTML = "";  
  
    // Define how to handle each dropped field  
  
    let component;  
  
    switch (droppedField) {  
  
        // Cases for each field type...  
  
        case "text":  
  
            component = document.createElement("input");  
  
            component.type = "text";  
  
            break;  
  
        // Add cases for other types of fields  
  
    }  
}
```

```

// Adjust styles for inline or block display

if (["SPAN", "P"].includes(target.tagName)) {

    component.style.display = "inline-block"; // Adjust for inline elements

} else {

    component.style.display = "block"; // Adjust for block elements

}

// Handle overflow if necessary

target.style.overflow = "auto";

if (component) {

    target.appendChild(component);

}

target.removeAttribute("drop-item");

this.removeHoveredClass();

},

// Additional methods...

}

```

Ensure you define the cases for handling different types of fields in the switch statement within the drop method. This code snippet is a foundational approach that needs to be adapted based on the specific requirements of your application, such as handling different input types and integrating with Vue's reactivity system.

The Reason this is a Superior Solution over the Status Quo of utilizing PDF signed documents is that they are:

Native to web applications and therefore seamless integrate into their backend database,
Thus allowing for full integration of all fields into all aspects/departments of analytics.

```
<template>

  <div

    ref="previewArea"

    v-html="code"

    class="drop-target"

    @dragenter.prevent

    @dragleave.prevent="dragLeave"

    @dragover.prevent="dragOver($event)"

    @drop="drop($event)"

  ></div>

</template>

<script>

import { createApp } from "vue";

import TextField from "./InputTypes/TextField.vue";

import TextArea from "./InputTypes/TextArea.vue";

import RadioGroup from "./InputTypes/RadioGroup.vue";
```

```
import CheckboxGroup from "./InputTypes/CheckboxGroup.vue";

import DatePicker from "./InputTypes/DatePicker.vue";

import NumberField from "./InputTypes/NumberField.vue";

import store from "../store";

import Signature from "./InputTypes/Signature.vue";

import AppModal from "./AppModal.vue";

import VueSignaturePad from "vue-signature-pad";

import TextInputModel from "../models/textInput";

import TextareaModel from "../models/Textarea";

import RadioModel from "../models/radio";

import CheckboxModel from "../models/checkbox";

import DatepickerModel from "../models/datepicker";

import NumberInputModel from "../models/NumberInput";

import SignatureModel from "../models/signature";

import SelectOption from "./InputTypes/SelectOption.vue";

import SelectModel from "../models/select";

export default {

  name: "PreviewArea",

  props: ["code"],

  computed: {

    pageMode() {
```

```
    return this.$store.getters.getPageMode;

  },

},

data() {

  return {

    pageState: "",

    readonly: false,

  }

},

methods: {

  removeHoveredClass() {

    document

      .getElementsByClassName("hovered")[0]

      ?.classList.remove("hovered");

  },

  dragOver(e) {

    const target = e.target;

    if (target.tagName === "TD" || target.closest("td")) {

      this.removeHoveredClass();

      e.target.closest("td").classList.add("hovered");

    }

  }

}
```

```
},  
  
drop: function (e) {  
  e.preventDefault();  
  
  const droppedField = e.dataTransfer.getData("field");  
  
  const dropTarget = e.target.closest("td");  
  
  this.deleteEmptyParagraph(dropTarget);  
  
  dropTarget.setAttribute("data-name", droppedField);  
  
  dropTarget.setAttribute("drop-item", true);  
  
  if (droppedField) {  
    let component;  
  
    let model;  
  
    switch (droppedField) {  
  
      case "textarea":  
  
        component = TextArea;  
  
        model = new TextareaModel({});  
  
        break;  
  
      case "text":  
  
        component = TextField;  
  
        model = new TextInputModel({});  
  
        break;  
  
      case "radio":
```

```
    component = RadioGroup;

    model = new RadioModel({});

    break;

case "select":

    component = SelectOption;

    model = new SelectModel({});

    break;

case "checkbox":

    component = CheckboxGroup;

    model = new CheckboxModel({});

    break;

case "datepicker":

    component = DatePicker;

    model = new DatepickerModel({});

    break;

case "number":

    component = NumberField;

    model = new NumberInputModel({});

    break;

case "signature":

    component = Signature;
```

```
    model = new SignatureModel({});

    break;
}

this.$store.dispatch("setFormModel", model);

if (dropTarget) {

    dropTarget.setAttribute("data-id", model.id);

    const id = e.dataTransfer.getData("id");

    // Create a new Vue application instance

    const app = createApp(component, {

        initialModel: model,

    })

    .use(store)

    .use(VueSignaturePad);

    app.component("AppModal", AppModal);

    // Mount the instance to a new element

    const el = document.createElement("div");

    document.body.appendChild(el);

    app.mount(el);

    // Append the element to the drop target

    dropTarget.appendChild(el);

    if (id) {
```

```

    const movedEl = document.getElementById(id);

    movedEl.style.display = "block";

    e.target.appendChild(movedEl);

    }

    }

    }

    dropTarget.removeAttribute("drop-item");

    document

        .getElementsByClassName("hovered")[0]

        ?.classList.remove("hovered");

    },

    dragLeave(e) {

        if (e.target.tagName === "TABLE") {

            this.removeHoveredClass();

        }

    },

    initComponents(droppedField, dropTarget) {

        let component;

        let model;

        dropTarget.setAttribute("drop-item", true);

        const droppedFieldId = dropTarget.getAttribute("data-id");

```

```
const obj = this.$store.getters.getModelById(droppedFieldId);

if(!obj) return

switch (droppedField) {

  case "textarea":

    component = TextArea;

    model = new TextareaModel(obj);

    break;

  case "text":

    component = TextField;

    model = new TextInputModel(obj);

    break;

  case "radio":

    component = RadioGroup;

    model = new RadioModel(obj);

    break;

  case "select":

    component = SelectOption;

    model = new SelectModel(obj);

    break;

  case "checkbox":

    component = CheckboxGroup;
```

```
    model = new CheckboxModel(obj);

    break;

case "datepicker":

    component = DatePicker;

    model = new DatepickerModel(obj);

    break;

case "number":

    component = NumberField;

    model = new NumberInputModel(obj);

    break;

case "signature":

    component = Signature;

    model = new SignatureModel(obj);

    break;

}

if (this.pageState === 'submission') {

    model.readonly = this.readonly;

}

if (dropTarget) {

    dropTarget.innerHTML = "";

    // this.deleteEmptyParagraph(dropTarget);
```

```
// Create a new Vue application instance

const app = createApp(component, {

  initialModel: model,

})

  .use(store)

  .use(VueSignaturePad);

app.component("AppModal", AppModal);

// Mount the instance to a new element

const el = document.createElement("div");

document.body.appendChild(el);

app.mount(el);

// Append the element to the drop target

dropTarget.appendChild(el);

}

dropTarget.removeAttribute("drop-item");

},

deleteEmptyParagraph(target) {

  const paragraphs = target.getElementsByTagName("p");

  for (let i = 0; i < paragraphs.length; i++) {

    let child = paragraphs[i].getElementsByTagName("span")[0];

    if (child && child.innerHTML === "&nbsp;") {
```

```
        target.removeChild(paragraphs[i]);
    }
}
},
},
mounted() {
    this.pageState = document.getElementById("app").getAttribute("data-page");
    if (this.pageState === 'submission') {
        this.readonly = true;
        this.$store.state.formModel.map(el => el.readonly = true)
    }
    if (["edit", "preview"].includes(this.pageMode)) {
        this.$el.querySelectorAll("[data-name]").forEach((el) => {
            this.initComponents(el.getAttribute("data-name"), el);
        });
    }
},
};
</script>
<style scoped></style>
```

Additionally, we can teach the software to recognize checkbox elements automatically, thereby Elementizing them as Form Fil Objects with the following example logic:

```
<!DOCTYPE html>

<html>

<head>

  <title>Checkbox Interactor</title>

</head>

<body>

<!-- Example checkboxes -->

<div id="checkboxContainer">

  <input type="checkbox" id="box1" class="tiny-box">

  <label for="box1">Box 1</label>

  <input type="checkbox" id="box2" class="tiny-box">

  <label for="box2">Box 2</label>

  <!-- Add more checkboxes as needed -->

</div>

<script>

  // JavaScript to toggle checkboxes

  document.addEventListener("DOMContentLoaded", function() {

    const boxes = document.querySelectorAll('.tiny-box');
```

```
boxes.forEach(box => {  
    box.addEventListener('click', function() {  
        // Toggle checkbox checked status  
        this.checked = !this.checked;  
    });  
});  
});  
</script>  
</body>  
</html>
```

Illustrations:

flowchart TD

A [Source Document
(e.g., Microsoft Word)]

B [HTML Rendering Engine]

C [Draggable Elements
(Form Fields: Text, Date Picker, Signature, etc.)]

D [Target HTML Elements
(TD, DIV, P, SPAN, etc.)]

E [Integration Module
(Dynamic Form Placement & Styling)]

F [Enterprise Systems & ERP
(Database Integration & Big Data Analytics)]

A --> B

B --> D

C --> E

D --> E

E --> F

Description:

Source Document: The starting document (such as a Word file) that is rendered into HTML.

HTML Rendering Engine: Converts the document into HTML, enabling dynamic interactions.

Draggable Elements: Interactive form components (e.g., text fields, date pickers, signature fields) available for placement.

Target HTML Elements: Various HTML tags (TD, DIV, P, SPAN, etc.) that serve as valid drop zones.

Integration Module: Detects drag actions, embeds form elements into target locations, and adjusts styles (inline or block) while maintaining document integrity.

Enterprise Systems & ERP: Seamlessly connects the dynamic forms with backend databases and ERP systems (e.g., Oracle) for real-time Big Data analytics and enhanced workflow efficiency.

Furthermore the system will include a customized Machine Learning Model with the following parameters:

Title

System and Method for Continuous Adaptive Reweighting with Integrated Audit
Verification and Content Differentiation in Machine Learning Models