Golang Developer Intern Task

👨‍💻 Submitted by: NITYA NAND JHA

> **Organization name:** Trademarkia
> **Role:** Go-lang backend developer
> **Date:** 🗓 Feb 17, 2023

TABLE OF CONTENTS

Task Summary 👀

The developer needs to show an understanding of data, data types, and be able to develop features using GoLang on Scale. The task will incorporate an understanding of XML and JSON type data formats as well as APIs using GoFiber.

Tasks 🐣

The task has been divided into four parts.

Part 1 (Data)

1. Download the XML data from the given google drive
2. Write the schema to Unmarshal XML data
    a. Import "encoding/json" "encoding/xml" "fmt" "io/ioutil" "log" "os" modules.

b. Create a struct for version, action-key, and transection-date.
c. Create a struct proceeding-information and start proceeding-entry array in it.
d. Create a proceeding-entry struct and take number, type-code, filling-date, employee-number, interlocutory-attorney-name, location-code, day-in-location, status-update-date, status-code.
    i. Create a struct inside proceeding-entry named party-information and create another instruct 'party' in it.
        1. Add elements identifier, role-code, name
        2. Create another struct Address information and inside that create another instruct proceeding-address.
            a. Inside proceeding address add identifier, type-code, name, address-1, city, state, country, and postcode.
    ii. Create a struct prosecution-history and start an array ProsecutionEntry in it.
3. Create a struct Prosecution-entry to accept data.

```
type XMLData struct {
        Version struct {
                VersionNo   string `xml:"version-no" json:"version-no"`
                VersionDate string `xml:"version-date" json:"version-date"`
        } `xml:"version" json:"version"`
        ActionKey       string `xml:"action-key-code" json:"action-key-code"`
        TransectionDate uint32 `xml:"transaction-date" json:"transaction-date"`

        ProceedingInformation ProceedingInformation `xml:"proceeding-information" json:"proceeding-information"`
}

type ProceedingInformation struct {
        ProceedingEntry []ProceedingEntry `xml:"proceeding-entry" json:"proceeding-entry"`
}

type ProceedingEntry struct {
```

```
        Number                      uint32 `xml:"number"
json:"number"`
        TypeCode                    string `xml:"type-code"
json:"type-code"`
        FilingDate                  uint32 `xml:"filing-date"
json:"filing-date"`
        EmployeeNumber              uint32 `xml:"employee-number"
json:"employee-number"`
        InterlocutoryAttorneyName string `xml:"interlocutory-
attorney-name" json:"interlocutory-attorney-name"`
        LocationCode                string `xml:"location-code"
json:"location-code"`
        DayInLocation               uint32 `xml:"day-in-location"
json:"day-in-location"`
        StatusUpdateDate            uint32 `xml:"status-update-
date" json:"status-update-date"`
        StatusCode                  uint32 `xml:"status-code"
json:"status-code"`

        PartyInformation struct {
                Party struct {
                        Identifier          uint32
`xml:"identifier" json:"identifier"`
                        RoleCode            string `xml:"role-
code" json:"role-code"`
                        Name                string `xml:"name"
json:"name"`
                        PropertyInformation struct {
                                Property struct {
                                        Identifier   string
`xml:"identifier" json:"identifier"`
                                        SerialNumber string
`xml:"serial-number" json:"serial-number"`
                                        MarkText     string
`xml:"mark-text" json:"mark-text"`
                                } `xml:"property"
json:"property"`
                        } `xml:"property-information"
json:"property-information"`
                        AddressInformation struct {
```

```
                                ProceedingAddress struct {
                                        Identifier uint32
    `xml:"identifier" json:"identifier"`
                                        TypeCode   string
    `xml:"type-code" json:"type-code"`
                                        Name       string
    `xml:"name" json:"name"`
                                        Address1   string
    `xml:"address-1" json:"address-1"`
                                        City       string
    `xml:"city" json:"city"`
                                        State      string
    `xml:"state" json:"state"`
                                        Country    string
    `xml:"country" json:"country"`
                                        Postcode   string
    `xml:"postcode" json:"postcode"`
                                } `xml:"proceeding-address"
    json:"proceeding-address"`
                        } `xml:"address-information"
    json:"address-information"`
                } `xml:"party" json:"party"`
        } `xml:"party-information" json:"party-information"`
        ProsecutionHistory struct {
                ProsecutionEntry []ProsecutionEntry
    `xml:"prosecution-entry" json:"prosecution-entry"`
        } `xml:"prosecution-history" json:"prosecution-history"`
    }

    type ProsecutionEntry struct {
        Identifier  uint32 `xml:"identifier" json:"identifier"`
        Code        uint32 `xml:"code" json:"code"`
        TypeCode    string `xml:"type-code" json:"type-code"`
        Date        uint32 `xml:"date" json:"date"`
        HistoryText string `xml:"history-text" json:"history-
    text"`
    }
```

Main Function

- Open XML file
- Check if it empty
- Read the XML file
- Check if it empty
- Unmarshal the data
- Indent the data
- Write indented data into JSON file named result.

```
func main() {

        xmlfile, err := os.Open("tt230101.xml")
        if err != nil {
                fmt.Println(err)
        }

        defer xmlfile.Close()

        byteValue, _ := ioutil.ReadAll(xmlfile)

        var entry XMLData
        err = xml.Unmarshal(byteValue, &entry)
        if err != nil {
                log.Fatalf(err.Error())
        }
        jfile, err := json.MarshalIndent(entry, "", "  ")
        if err != nil {
                log.Fatalf(err.Error())
        }
        err = os.WriteFile("result.json", jfile, 0644)
        if err != nil {
                log.Fatalf(err.Error())
        }
}
```