github.com/STreeChin/contactapi/internal/controller/contact.go (96.5%) ▾ not tracked not covered covered

```
package controller

import (
	"encoding/json"
	"net/http"
	"strconv"
	"strings"
	"time"

	"github.com/STreeChin/contactapi/pkg/entities"
	"github.com/go-playground/validator/v10"
	"github.com/google/uuid"
	"github.com/gorilla/mux"
	"github.com/pkg/errors"
	"github.com/sirupsen/logrus"
	"go.mongodb.org/mongo-driver/mongo"
)

//ContactService interface
type ContactService interface {
	AddOrUpdateContact(contact *entities.Contact) (string, error)
	GetOneContact(key, value string) (*entities.Contact, error)
}

type contactController struct {
	log            *logrus.Logger
	contactService ContactService
}

//NewContactController instance
func NewContactController(log *logrus.Logger, cs ContactService) *contactController {
	return &contactController{log, cs}
}

//GetOneContactCtrl:
func (cc *contactController) GetOneContactCtrl(w http.ResponseWriter, r *http.Request) {
	if w == nil || r == nil {
		cc.log.Warningln("http input nil")
		return
	}

	contactIDOrEmail := mux.Vars(r)["contact_id_or_email"]

	key, check := cc.checkContactIDOrEmail(contactIDOrEmail)
	if !check {
		cc.log.Infoln("Invalid contact_id_or_email value provided")
		cc.handleError(w, http.StatusBadRequest, "Invalid contact_id_or_email value provided.")
		return
	}

	respContact, err := cc.contactService.GetOneContact(key, contactIDOrEmail)
	if err != nil {
		if errors.Cause(err) == mongo.ErrNoDocuments {
			cc.log.Infof("GetOneContactCtrl: %+v", err)
			cc.handleError(w, http.StatusNotFound, "Contact could not be found.")
			return
		}

		cc.log.Errorf("GetOneContactCtrl: %+v", err)
		cc.handleError(w, http.StatusInternalServerError, err.Error())
		return
	}

	cc.buildResponse(w, respContact)
}

//AddOrUpdateContactCtrl: add or update contact
```

github.com/STreeChin/contactapi/internal/controller/contact.go (96.5%) ▾ not tracked not covered covered

```
                cc.log.Println("Warning: http input nil")
                return
        }

        contact, err := cc.parseContactFromReq(r)
        if err != nil {
                cc.log.Errorf("AddOrUpdateContactCtrl: %+v", err)
                cc.handleError(w, http.StatusInternalServerError, "Internal Error")
                return
        }

        if contact.Email == "" {
                cc.log.Infoln("AddOrUpdateContactCtrl: request email is nil")
                cc.handleError(w, http.StatusBadRequest, "No contact details provided.")
                return
        }

        contactID, err := cc.contactService.AddOrUpdateContact(contact)
        if err != nil {
                cc.log.Errorf("AddOrUpdateContactCtrl: %+v", err)
                cc.handleError(w, http.StatusInternalServerError, err.Error())
                return
        }

        body := map[string]string{"contact_id": contactID}
        cc.buildResponse(w, body)
}

func (cc *contactController) checkContactIDOrEmail(value string) (key string, check bool) {
        err := validator.New().Var(value, "required,email")
        if err == nil {
                key = "email"
                check = true
                return key, check
        }

        l := strings.SplitAfterN(value, "-", 2)
        if l[0] == "person_AP2-" && len(l) > 1 {
                _, err := uuid.Parse(l[1])
                if err == nil {
                        key = "contactid"
                        check = true
                        return key, check
                }
        }

        return key, check
}

func (cc *contactController) parseContactFromReq(r *http.Request) (*entities.Contact, error) {
        var err error
        dst := new(entities.ReqContact)

        err = json.NewDecoder(r.Body).Decode(dst)
        for k, v := range dst.Contact.Custom {
                l := strings.SplitAfterN(k, "--", 2)
                l[0] = strings.Replace(l[0], "--", "", -1)
                l[1] = strings.Replace(l[1], "--", " ", -1)
                var newValue interface{}
                switch l[0] {
                case "integer":
                        newValue, err = strconv.Atoi(v.(string))
                case "boolean":
                        newValue, err = strconv.ParseBool(v.(string))
                case "string":

                case "date":
                        loc, _ := time.LoadLocation("Local")
```

github.com/STreeChin/contactapi/internal/controller/contact.go (96.5%) ▾ not tracked not covered covered

```
                        newValue, err = strconv.ParseFloat(v.(string), 32)

                default:
                        cc.log.Warnln("The type of custom field is error")
                        return nil, errors.New("the type of custom field is error")
                }

                dst.Contact.Custom[l[1]] = newValue
                delete(dst.Contact.Custom, k)
                break
        }

        return &dst.Contact, err
}

func (cc *contactController) buildResponse(w http.ResponseWriter, body interface{}) {
        w.Header().Set("Content-Type", "application/json")
        w.WriteHeader(http.StatusOK)
        _ = json.NewEncoder(w).Encode(body)
}

func (cc *contactController) handleError(w http.ResponseWriter, code int, msg string) {
        body := map[string]string{}

        switch code {
        case http.StatusBadRequest:
                body["error"] = "Bad Request"
        case http.StatusNotFound:
                body["error"] = "Not Found"
        case http.StatusInternalServerError:
                body["error"] = "Internal Server Error"
        default:
                body["error"] = "Internal Server Error"
        }

        body["message"] = msg
        w.Header().Set("Content-Type", "application/json")
        w.WriteHeader(code)
        _ = json.NewEncoder(w).Encode(body)
}
```