github.com/STreeChin/contactapi/internal/service/contact.go (94.9%) ▼ not tracked not covered covered

```
package service

import (
        "github.com/STreeChin/contactapi/pkg/config"
        "github.com/STreeChin/contactapi/pkg/entities"
        "github.com/gomodule/redigo/redis"
        "github.com/google/uuid"
        "github.com/pkg/errors"
        "github.com/sirupsen/logrus"
)

//Repository interface
type Repository interface {
        GetOneContact(key, value string) (*entities.Contact, error)
        InsertOneContact(contact *entities.Contact) error
        UpdateOneContact(contact *entities.Contact) error
        GetContactIDByAPIKey(apiKey string) (string, error)
}

//Cache interface
type Cache interface {
        GetEmailByContactID(key string) (string, error)
        SetEmailByContactID(key, value string) error
        GetOneContact(value string) (*entities.Contact, error)
        SetOneContact(value string, contact *entities.Contact) error
        DelOneContact(value string) error
}

type contactService struct {
        log   *logrus.Logger
        cfg   config.Config
        cache Cache
        rep   Repository
}

//NewContactService instance
func NewContactService(log *logrus.Logger, cfg config.Config, rs Cache, cr Repository) *contactService {
        return &contactService{log, cfg, rs, cr}
}

//GetOneContact: get one contact
func (c *contactService) GetOneContact(key, value string) (*entities.Contact, error) {
        var err error
        var email string
        contact := new(entities.Contact)

        if key == "contactid" {
                //get the contactID by email from cache, cache: contactID->email->contact{}
                email, err = c.cache.GetEmailByContactID(value)
                if err == nil {
                        contact, err = c.cache.GetOneContact(email)
                }
        } else if key == "email" {
                contact, err = c.cache.GetOneContact(value)
        } else {
                return nil, errors.New("invalid key")
        }

        if err != nil && errors.Cause(err) == redis.ErrNil {
                contact, err = c.rep.GetOneContact(key, value)
                if err != nil {
                        return contact, errors.Wrap(err, "service getOneContact")
                }
                //cache: contactID->email->contact{}
                err = c.cache.SetOneContact(contact.Email, contact)
                if err != nil {
                        c.log.Error("service SetOneContact", err)
                }
```

github.com/STreeChin/contactapi/internal/service/contact.go (94.9%) ▼ not tracked not covered covered

```
                        c.log.Error("service SetOneContact", err)
                }
                //response to the the http if getting db success but setting cache fail, log the fail
                err = nil
        }

        return contact, err
}

//AddOrUpdateContact: add or update contact
func (c *contactService) AddOrUpdateContact(contact *entities.Contact) (string, error) {
        var err error

        if contact.ContactID == "" {
                contact.ContactID = "person_AP2-" + uuid.New().String()
        }

        _, err = c.rep.GetOneContact("email", contact.Email)
        if err != nil {
                err = c.rep.InsertOneContact(contact)
                if err != nil {
                        return "", errors.Wrap(err, "service AddOrUpdateContact")
                }
        } else {
                err = c.rep.UpdateOneContact(contact)
                if err != nil {
                        return "", errors.Wrap(err, "service AddOrUpdateContact")
                }
        }

        //invalid cache: contactID->email->contact{}
        err = c.cache.DelOneContact(contact.Email)
        if err != nil {
                return "", errors.Wrap(err, "service AddOrUpdateContact")
        }
        err = c.cache.DelOneContact(contact.ContactID)

        return contact.ContactID, errors.Wrap(err, "service AddOrUpdateContact")
}
```