
lightmdb Documentation

Release 1.0

ITU itucsd1630

December 30, 2016

| | | |
|----------|------------------------|----------|
| 1 | User Guide | 3 |
| 2 | Developer Guide | 9 |

Team itucsd1630

Members

- Emin Mastizada
- Emre Eroglu
- Memduh Gokirmak
- Onur Can Carkci
- Seda Bilgin

Light Movie DataBase

We love watching movies, make lists for different emotions and times.

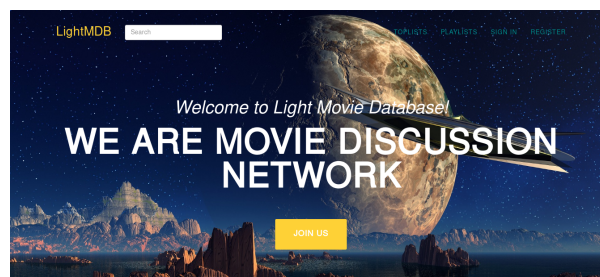
Current movie database sites became very complex and interaction between users are minimum. Our idea is to create social network on movies for movie lovers like us. Share ideas, follow professionals, reviewers and friends, review movies, make your playlists. Get notification when new movies added by your favorite director or in your lovely categories.

We are Movie Discussion Network.

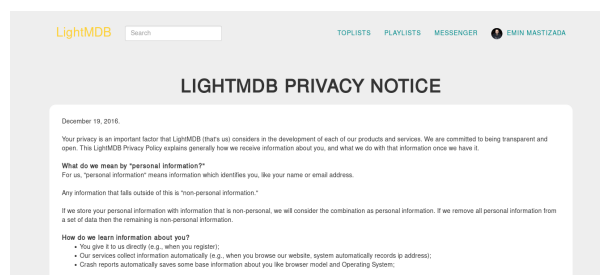
Contents:

User Guide

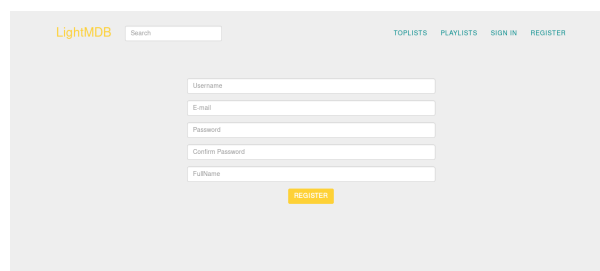
- In homepage there is intro about website, Top Movies and Top Users:



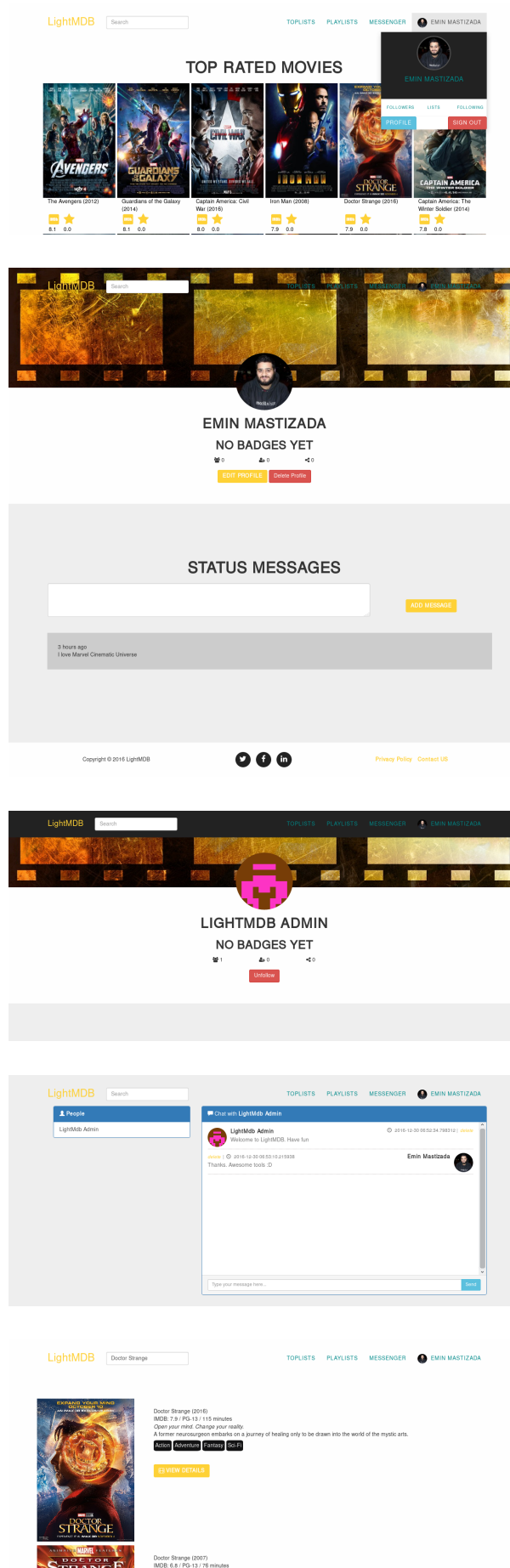
- Check Privacy Notice to learn details of information that will be used after registration:

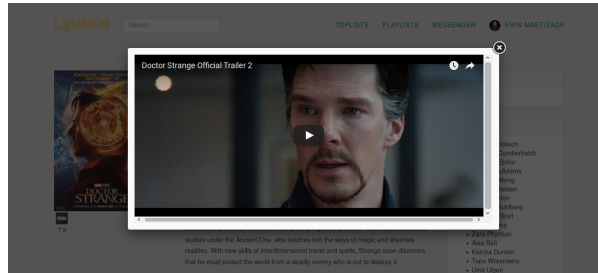
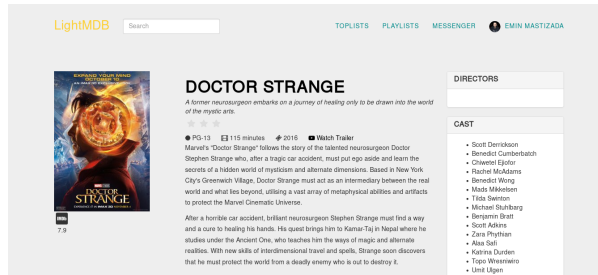


- There is registration and sign in button at navigation, register for website:

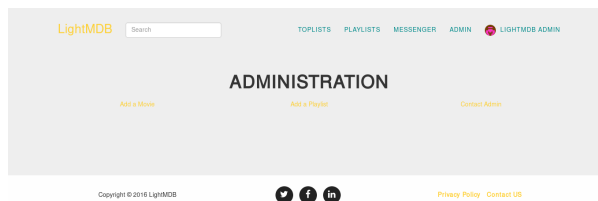


- After registration go to profile and check details:
- Profile Page where you can edit details, delete account and write status messages:
- You can navigate to other profile pages and follow people you know:
- When users follow each-other they will be able to chat over direct messaging:
- Search for movies using search bar on top:
- View movie details:





- Also watch movie trailer:
- Administration members will have one extra button in navigation for admin page:



1.1 Parts Implemented by Emin Mastizada

1. Home page
2. Admin page
3. Registration page
4. Login page
5. User profile page
6. Follow, unfollow ajax (javascript)
7. Status Message
8. Movie Search
9. Movie page
10. Privacy Policy and License Compability
11. Gravatar Images
12. LightBox for trailers

1.2 Parts Implemented by Member Name

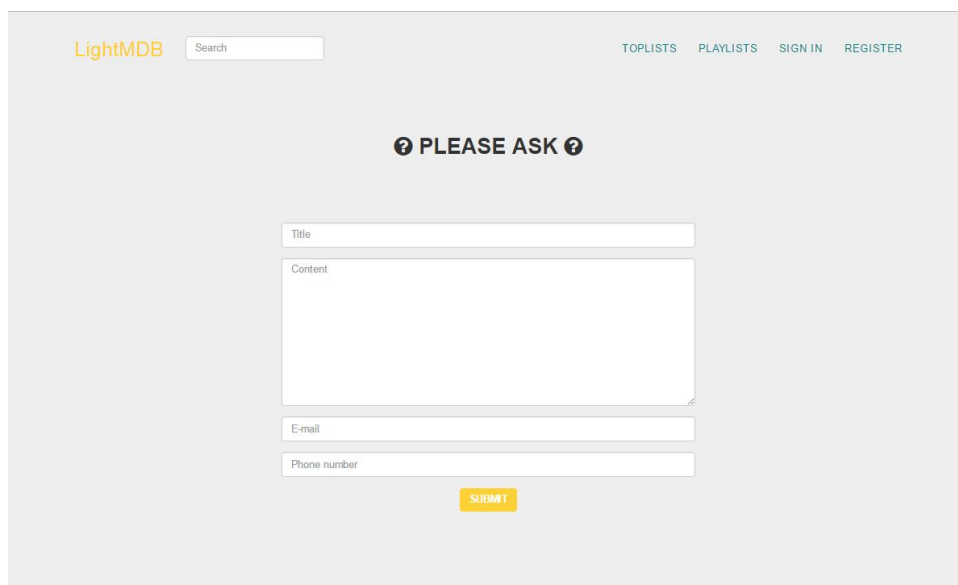
1.3 Parts Implemented by Memduh Gökırmak

1. Playlists page
2. Various former revisions of movie page
3. Single avg file for camera shape

1.4 Parts Implemented by Seda Bilgin

1.4.1 CONTACT US PAGE

Users can ask questions by submitting the form on “Contact Us” page. There are some validation controls while submitting the form. For example users could not submit without an e-mail or enter characters to phone number.

The screenshot shows the 'PLEASE ASK' contact form on the LightMDB website. At the top left is the 'LightMDB' logo and a search bar. At the top right are links for 'TOPLISTS', 'PLAYLISTS', 'SIGN IN', and 'REGISTER'. The main heading is 'PLEASE ASK' with question mark icons on either side. Below this is a form with four input fields: 'Title', 'Content' (a larger text area), 'E-mail', and 'Phone number'. A yellow 'SUBMIT' button is located at the bottom right of the form.

1.4.2 CONTACT US ADMIN PAGE

On Admin page users could select sent questions and search and list questions by their status. Sent questions with “New” status are listed by default.

Users also update the status of sent questions by clicking “ADD”.

1.4.3 Comments

On Contact admin page users could add comments on sent questions while updating their status.

After clicking “Show” users could see previous comments and update or delete comments.

1.5 Parts Implemented by Member Name

LightMDB

[TOPLISTS](#) [PLAYLISTS](#) [SIGN IN](#) [REGISTER](#)

SENT QUESTIONS

Choose types

☒ New ☐ Replied ☐ Waiting ☐ Spam ☐ Closed [QShow](#)

Copyright © 2016 LightMDB

[Privacy Policy](#) [Contact US](#)

| Title | Content | Email | Phone | Status | Sent Time | Comment | Delete |
|-------------|------------------------------|------------------|--------------|--------|----------------------------|---|------------------------|
| Want to ask | How can I change my password | user@example.com | 555111222333 | new | 2016-12-30 06:41:45.942885 | Show Add | Delete |

LightMDB

[TOPLISTS](#) [PLAYLISTS](#) [SIGN IN](#) [REGISTER](#)

SENT QUESTIONS

Choose types

☒ New ☐ Replied ☐ Waiting ☐ Spam ☐ Closed [QShow](#)

Copyright © 2016 LightMDB

[Privacy Policy](#) [Contact US](#)

| Title | Content | Email | Phone | Status | Sent Time | Comment | Delete |
|-------------|------------------------------|------------------|--------------|--------|----------------------------|---|------------------------|
| Want to ask | How can I change my password | user@example.com | 555111222333 | new | 2016-12-30 06:41:45.942885 | Show Add | Delete |

Replied

Replied

Waiting

Spam

Closed

☐ Is Comment sent to user?

[Save](#)

[Close Comment Adder](#)

LightMDB

[TOPLISTS](#) [PLAYLISTS](#) [SIGN IN](#) [REGISTER](#)

SENT QUESTIONS

Choose types

☒ New ☐ Replied ☐ Waiting ☐ Spam ☐ Closed [QShow](#)

Copyright © 2016 LightMDB

[Privacy Policy](#) [Contact US](#)

| Title | Content | Email | Phone | Status | Sent Time | Comment | Delete |
|-------------|------------------------------|------------------|--------------|--------|----------------------------|---|------------------------|
| Want to ask | How can I change my password | user@example.com | 555111222333 | new | 2016-12-30 06:41:45.942885 | Show Add | Delete |

Replied

Comment

☐ Is Comment sent to user?

[Save](#)

[Close Comment Adder](#)

LightMDB

Search

TOPLISTSPLAYLISTSSIGN INREGISTER

SENT QUESTIONS

Choose types

☒ New☒ Replied☐ Waiting☐ Spam☐ Closed

Show

Copyright © 2016 LightMDB

Privacy PolicyContact US

| Title | Content | Email | Phone | Status | Sent Time | Comment | Delete |
|-------------|------------------------------|------------------|--------------|---------|----------------------------|--------------------|-------------------|
| Want to ask | How can I change my password | user@example.com | 555111222333 | replied | 2016-12-30 06:41:45.942885 | <div>ShowAdd</div> | <div>Delete</div> |

Close Comment Box

| Comment | Mail Sent | Time | Update | Delete |
|--------------|-----------|----------------------------|-------------------|-------------------|
| test comment | False | 2016-12-30 07:21:22.005631 | <div>UPDATE</div> | <div>Delete</div> |

LightMDB

Search

TOPLISTSPLAYLISTSSIGN INREGISTER

SENT QUESTIONS

Choose types

☒ New☒ Replied☐ Waiting☐ Spam☐ Closed

Show

Copyright © 2016 LightMDB

Privacy PolicyContact US

| Title | Content | Email | Phone | Status | Sent Time | Comment | Delete |
|-------------|------------------------------|------------------|--------------|---------|----------------------------|--------------------|-------------------|
| Want to ask | How can I change my password | user@example.com | 555111222333 | replied | 2016-12-30 06:41:45.942885 | <div>ShowAdd</div> | <div>Delete</div> |

Close Comment Box

| Comment | Mail Sent | Time | Update | Delete |
|--------------|-----------|----------------------------|-------------------|-------------------|
| test comment | False | 2016-12-30 07:21:22.005631 | <div>UPDATE</div> | <div>Delete</div> |

Waiting

change comment

☐ Is Comment sent to user?

Save

Close Comment Adder

Developer Guide

2.1 INSTALL

Project uses postgres as database. You need to install postgresql locally or use vagrant.

By default application uses vagrant settings for database. If you installed it locally you will need to set DSN in *local_settings*.

Prepare Environment:

- **Create virtual environment in venv folder:** `$ virtualenv venv -p python3`
- **Install project requirements:** `$ pip install -r requirements.txt`
- **Initialize database:** `$ python manage.py migrate`
- **Run application:** `$ python manage.py runserver`

local_settings File:

```
* Create "local_settings.py" file in project root directory.
* Add custom settings to file.
```

Available Settings:

```
>>> DEBUG=True           # Debug mode
>>> DSN="..."          # Postgres credentials, check `DEFAULT_DSN` in `application.py` file.
>>> HOST="127.0.0.1"     # Application host
>>> PORT=5000            # Application port
>>> SENTRY_DSN="..."    # Sentry dsn setting
```

Environment Variables:

```
* 'VCAP_APP_PORT' - Bluemix application port
* 'VCAP_SERVICES' - Bluemix settings for services
* 'SENTRY_DSN' - Sentry DSN (logging)
* 'CI_TESTS' - Travis CI environment
* 'SECRET_KEY' - Secret key for cookies
```

2.2 Database Design

- Main tables are *users* and *movies*.
- *user_followers* connects to *users* table as ManyToMany using *follower_id* and *following_id*.
- *status_messages* connects both to *movies* and *users* as ManyToOne, stores movie comments and personal status messages in timeline.

- *user_messages* connects to *users* as ManyToMany using *sender_pk* and *receiver_pk*. Stores private messages.
- *celebrities* table created for storing celebrities like actors, directors and etc.
- *casting* stores movie cast information, connects to *celebrities* using *celebrity_pk* (ManyToOne).
- *directors* acts same as casting.
- *playlists* connects to *users* as ManyToOne using *user_id*
- *playlist_movies* stores movies for playlists, connects both to *playlists* and to *movies*. Connects playlists to movies as ManyToMany.
-

include the E/R diagram(s)

2.3 Code

Creating new models:

- Create your new models inside lightmdb/models/ folder.
- As everyone in team should write few sql command, we will not use Base objects.
- Each object should have *get*, *filter*, *save* and *delete* methods:

```
.. code-block:: python

class Movie:
    @classmethod
    def get(cls, pk=None):
        # Use Unique keys as possible parameters for function
        # Fetch movie from database as dictionary
        # Return Movie object with database values
        # If there is no matching result, return None

    @classmethod
    def filter(cls, limit=100, order="id DESC", **kwargs):
        # Fetch movie using parameters (filters) in kwargs
        # Use limit and order with default values
        # Return list of Movie objects
        # If there is no matching result, return empty list ([])

    def save(self):
        # if self.pk is present, update database with current values
        # if it is new object, insert into database
        # add "RETURNING id" to sql if you need pk after execution (see Movie)
        # Return call to get method:
        return Movie.get(identifier=self.identifier)

    def delete(self):
        # If object is not populated from database, ie. self.pk is None:
        raise ValueError("Movie is not saved yet.")
        # Delete Movie using identifier from database.
        # Do not return anything
```

- *__init__* section of object should take parameters in same order as in schema.sql file.
- After creating model add it to lightmdb/models/*__init__.py* file.

Create Form for each Model:

- In order to safely create Model using user's request parameters create Form based on Model.

- Forms should be under lightmdb/forms/ folder.
- After creating form add it to lightmdb/forms/__init__.py file.

Creating View:

- To show your new model in client you will need add view for it.
- Create view file under lightmdb/views/ folder.
- Make Blueprint variable for you view, add all views for that Blueprint.
- Add your Blueprint variable to lightmdb/views/__init__.py file.

Add view to urls:

- In order to enable views we need to add them in lightmdb/application.py file.
- Add your new view to *DEFAULT_BLUEPRINTS* parameter.
- Run server and check your view

Writing Tests:

CI will test project in each pull request using /tests.py file. Add your new model and view tests in that file.

2.4 Work done by each team member:

List with issues for each team member can be found on [Github Projects](#)

2.4.1 Parts Implemented by Emin Mastizada

1. Project Syntax (application, management, models, views, forms)
2. Base User Model
3. Session Management
4. Github, Bluemix, Travis, RTD setup
5. Base Template
6. Developer Guide
7. Status Messages Model
8. Followers Model
9. IMDB Search and Save (with Emre)
10. Youtube Trailer Search

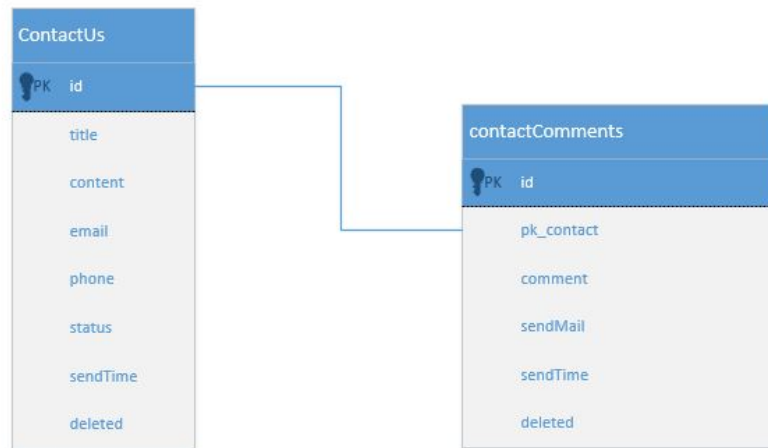
2.4.2 Parts Implemented by Emre Eroglu

- Database Integration

2.4.3 Parts Implemented by Memduh Gökırmak

1. Tables for Movie, Playlist, Playlist_Movie, Celebrity, Casting, Director
2. Views for Movie, Playlist
3. Models for all above
4. UI Implementation full for Movie, creation and reading for Playlist, only reading Celebrity, Casting, Director on pages of relevant movies

2.4.4 Parts Implemented by Seda Bilgin



ContactUS

Table Definition

The ContactUs table is defined to have id, title, content, email, phone, status, sendTime, deleted columns. The id increases serially, and the status assigned “New” by default. Status column data in ContactUs table is a type of contactStatus and contactStatus is defined with an enum type. ContactUs table is in schema.sql file.

```

CREATE TYPE contactStatus AS ENUM ('new', 'replied', 'waiting', 'spam', 'closed');
CREATE TABLE contactUs (
  id SERIAL PRIMARY KEY,
  title VARCHAR(100) NOT NULL,
  content VARCHAR(255) NOT NULL,
  email VARCHAR(50) NOT NULL,
  phone VARCHAR(50) NOT NULL,
  status contactStatus DEFAULT 'new',
  sendTime timestamp DEFAULT CURRENT_TIMESTAMP,
  deleted BOOLEAN DEFAULT FALSE
);
  
```

Class

“ContactMessage” class is under models/contact.py file. Class __init__() is shown in below.

```

def __init__(self, cid=None, title=None, content=None, email=None, phone=None):
    if cid:
        self.cid = int(cid)
    if title:
        self.title = str(title)
    if content:
        self.content = str(content)
    if email:
        self.email = str(email)
    if phone:
        self.phone = str(phone)
    if cid:
  
```



```

self.saved_message=True
contact_info=self.get_info_by_id()
if contact_info:
    self.title = contact_info[0]
    self.content = contact_info[1]
    self.email = contact_info[2]
    self.phone = contact_info[3]
    self.status =contact_info[4]
    self.sendTime = contact_info[5]
else:
    self.saved_message=False

```

Methods

1. Select

- Select operation gets data from ContactUs table by id.

```

def get_info_by_id(self):
    try:
        db = get_database()
        cursor = db.cursor()
        cursor.execute("SELECT title,content,email,phone,status,sendtime FROM {table} "+
            "WHERE id=%s".format(table=self.TABLE_NAME), [self.cid])
        contact_info= cursor.fetchone()
        if contact_info:
            return contact_info
    except:
        return False

```

- Selecting of rows with desired status or deleted marked of table.

```

@staticmethod
def get_messages(desired_status=None,get_deleted=False):
    accepted_status=[]
    all_status = ['new', 'replied', 'waiting', 'spam', 'closed']
    if not desired_status:
        desired_status = ['new','replied','waiting']
    for one_status in desired_status:
        if one_status in all_status:
            accepted_status.append(one_status)
    where = ''
    if not get_deleted:
        where = 'deleted=False and (status=\'\'
    if len(accepted_status) > 0:
        where += '\\' or status = \\''.join(accepted_status)
        where += '\')'
    try:
        db = get_database()
        cursor = db.cursor()
        cursor.execute("SELECT id,title,content,email,phone,status,sendtime from"+
            " contactUs where "+where)
        return cursor.fetchall()
    except:
        return []
return []

```

2. Insert

- Insert operation adds data to ContactUs table.

```

def save(self):
    try:
        db = get_database()

```

```
cursor = db.cursor
cursor.execute("INSERT INTO {table} (title,content,email,phone) VALUES"+
    " (%s,%s,%s,%s)".format(table=self.TABLE_NAME),
    [self.title,self.content,self.email,self.phone])
cursor.close()
db.commit()
db.close()
return True
except:
    return False
```

3. Update

- Update operation changes status of row by identified id if new status is one of contactStatus type.

```
def change_status(self,new_status):
    all_status=['new','replied','waiting','spam','closed']
    if new_status in all_status:
        self.status = new_status
        try:
            db = get_database()
            cursor = db.cursor
            cursor.execute("UPDATE {table} SET status=%s WHERE"+
                " id=%s".format(table=self.TABLE_NAME),
                [new_status,self.cid])
            cursor.close()
            db.commit()
            db.close()
            return True
        except:
            return False
    return False
```

4. Delete

- Delete operation delete row by identified id.

```
def delete_message(self):
    db = get_database()
    cursor = db.cursor
    ##cursor.execute("UPDATE {table} SET deleted=1 WHERE
    ##    id=%s".format(table=self.TABLE_NAME), [self.cid])
    cursor.execute("DELETE FROM {table} WHERE id=%s".format(table=self.TABLE_NAME),
        [self.cid])
    cursor.close()
    db.commit()
    db.close()
```

Template Operations

Route of website pages with POST,GET methods are in views/contactus.py

This class uses wtforms module to prepare form of page.

```
class ContactForm(Form):
    """Form to be used in contactus page."""
    title = StringField('title', [
        validators.Length(min=5, max=100),
        validators.DataRequired("Please, enter title.")
    ],
        render_kw={
            "placeholder": "Title",
            "class": "form-control"
```

```

    }
)
content = TextAreaField('Content', [
    validators.Length(min=10, max=255),
    validators.DataRequired("Please, enter content.")
],
render_kw={
    "placeholder": "Content",
    "type": "textarea",
    "class": "form-control",
    "rows": "10",
    "cols": "50"
})
email = EmailField('Email', [
    validators.Email("Please, enter correct email address."),
    validators.DataRequired("Please, enter your email address.")
],
render_kw={
    "placeholder": "E-mail",
    "class": "form-control"
})
phone = StringField('phone', [
    validators.Length(min=5, max=50),
    validators.DataRequired("Please, enter phone number.")
],
render_kw={
    "placeholder": "Phone number",
    "class": "form-control"
})
)

```

‘/contact/’ website is running with the code below.

By default, page creates a form with a class under form/contactus.py file. if there is not any post to website, app sends a page using contact/contact.html template with prepared form. else form will be validated and ContactMessage object will be created then saved and app sends a page using contact/thanks.html template.

View of Contact Admin page

Saved messages are shown in contact/admin page.

This page shows all messages with desired status, default desired status is ‘new’. If Page form is posted desired types can be select. If pages posted data has ‘deleted’, ContactMessage object will be created by id and will be deleted. If page posted data has ‘update’, ContactMessage object will be created by id and status of message is updated.

```

@contactus.route("/admin/", methods=["GET", "POST"])
def contact_admin():
    desired_types = ['new']
    comments=[]
    pk_contact=0
    notpost=0
    if request.method == 'POST':
        flash(request.form)
        if 'update' in request.form and 'status' in request.form:
            message=ContactMessage(request.form['update'])
            message.change_status(request.form['status'])
            if 'sendMail' in request.form:
                send_mail = True
            else:
                send_mail = False
            if 'commentUpdate' in request.form:

```

```
        comment=ContactComment(pk=request.form['commentUpdate'])
        comment.update_comment(request.form['comment'], send_mail)

    else:
        comment=ContactComment(pk_contact=message.cid,
                                comment=request.form['comment'],
                                send_mail=send_mail)

        comment.save()

    if 'delete' in request.form:
        message = ContactMessage(request.form['delete'])
        message.delete_message()
        comment=ContactComment(pk_contact=message.cid)
        comment.delete_comments_by_contact_id()

    elif 'deletecomment' in request.form:
        comment = ContactComment(pk=request.form['deletecomment'])
        comment.delete_comments_by_id()

    desired_types=[]
    all_types=['new', 'replied', 'waiting', 'spam', 'closed']
    for one_type in all_types:
        if one_type in request.form:
            desired_types.append(one_type)

    if 'showComments' in request.form:
        pk_contact=request.form['showComments']
        contact_comment=ContactComment(pk_contact=pk_contact)
        comments=contact_comment.get_comments_by_contact_id()
        if not comments:
            comments=[]

    else:
        notpost=1

    messages=ContactMessage.get_messages(desired_types)
    return render_template(
        'contact/contactadmin.html',
        table=messages,
        comments=comments,
        pk_contact=int(pk_contact),
        post=request.form,
        len=len(comments),
        notpost=notpost,
        thead=[
            'Title', 'Content', 'Email',
            'Phone', 'Status', 'Sent Time', 'Comment', 'Delete'
        ]
    )
```

ContactComments

Table Definition

The ContactComments table is defined to have id, pk_contact, comment, sendMail, sendTime, deleted columns. The id increases serially, by default send time will be inserted time and deleted will be false. ContactComments table is in schema.sql file.

Reference sql queries:

Class

“ContactComment” class is under models/contactComment.py file. Class `__init__()` is shown in below.

Methods

1. Select

- Select operation gets data from contactComments table by id.

```
def get_comments_by_contact_id(self):
    try:
        db = get_database()
        cursor = db.cursor()
        cursor.execute("SELECT id,comment,sendmail,sendtime FROM {table} WHERE pk_contact=%s".format(
            table=self.TABLE_NAME, pk_contact=id))
        contact_info= cursor.fetchall()
        if contact_info:
            return contact_info
    except:
        return False
```

2. Insert

- Insert operation adds data to contactComments table.

```
def save(self):
    try:
        db = get_database()
        db.cursor().execute("INSERT INTO {table} (pk_contact,comment,sendmail) VALUES (%s,%s,%s)".format(
            table=self.TABLE_NAME, pk_contact=id, comment=comment, sendmail=sendmail))
        db.commit()
        return True
    except:
        return False
```

3. Update

- Update operation updates comment and sendMail columns of row identified by object id.

```
def update_comment(self, comment, sendmail):
    try:
        db = get_database()
        cursor = db.cursor()
        cursor.execute("UPDATE {table} set comment=%s , sendmail=%s WHERE id=%s".format(table=self.TABLE_NAME,
            comment=comment, sendmail=sendmail, id=id))
        contact_info= cursor.fetchall()
        if contact_info:
            return contact_info
    except:
        return False
```

4. Delete

- delete_comments_by_id method deletes row by identified id.

```
def delete_comments_by_id(self):
    try:
        db = get_database()
        cursor = db.cursor()
        cursor.execute("DELETE FROM {table} WHERE pk_contact=%s".format(table=self.TABLE_NAME), [self.id])
        contact_info= cursor.fetchall()
        if contact_info:
            return contact_info
    except:
        return False
```

- delete_comments_by_contact_id method deletes rows by contact_id

```
def delete_comments_by_contact_id(self):
    try:
        db = get_database()
        cursor = db.cursor()
        cursor.execute("DELETE FROM {table} WHERE pk_contact=%s".format(table=self.TABLE_NAME), [self.pk_contact])
        contact_info= cursor.fetchall()
```

```
    if contact_info:
        return contact_info
except:
    return False
```

Template Operations

Route of website pages with POST,GET methods are in views/contactus.py

View of Contact Admin page

Comments adding deleting inserting updating and selecting are in contact/admin page.

By default page do not prepare any object for comments. If pages posted data has 'showComments', comments of contact will be prepared by contact id. If pages posted data have 'update' and 'commentUpdate', comment and send mail columns will be updated. If pages posted data have 'update' but not 'commentUpdate', ContactMessage object will be created and will be save. If pages posted data has 'deletecomment', object will be created by comment_id and will be deleted.

App run for this page with 'contact/contactadmin.html' template.

```
@contactus.route("/admin/", methods=["GET", "POST"])
def contact_admin():
    desired_types = ['new']
    comments=[]
    pk_contact=0
    notpost=0
    if request.method == 'POST':
        flash(request.form)
        if 'update' in request.form and 'status' in request.form:
            message=ContactMessage(request.form['update'])
            message.change_status(request.form['status'])
            if 'sendMail' in request.form:
                send_mail = True
            else:
                send_mail = False
            if 'commentUpdate' in request.form:
                comment=ContactComment(pk=request.form['commentUpdate'])
                comment.update_comment(request.form['comment'],send_mail)
            else:
                comment=ContactComment(pk_contact=message.cid,comment=request.form['comment'])
                comment.save()
        if 'delete' in request.form:
            message = ContactMessage(request.form['delete'])
            message.delete_message()
            comment=ContactComment(pk_contact=message.cid)
            comment.delete_comments_by_contact_id()
        elif 'deletecomment' in request.form:
            comment = ContactComment(pk=request.form['deletecomment'])
            comment.delete_comments_by_id()
        desired_types=[]
        all_types=['new','replied','waiting','spam','closed']
        for one_type in all_types:
            if one_type in request.form:
                desired_types.append(one_type)
        if 'showComments' in request.form:
            pk_contact=request.form['showComments']
            contact_comment=ContactComment(pk_contact=pk_contact)
            comments=contact_comment.get_comments_by_contact_id()
            if not comments:
                comments=[]
    else:
```

```
        notpost=1

    messages=ContactMessage.get_messages(desired_types)
    return render_template(
        'contact/contactadmin.html',
        table=messages,
        comments=comments,
        pk_contact=int(pk_contact),
        post=request.form,
        len=len(comments),
        notpost=notpost,
        thead=[
            'Title', 'Content', 'Email',
            'Phone', 'Status', 'Sent Time', 'Comment', 'Delete'
        ]
    )
```

2.4.5 Parts Implemented by Onur Can Carkci