DB-based email service

Team:

Annam Saivardhan, 200050008

Cheerla Vinay Kumar, 200050027

Dendukuri Sandeep Varma, 200050032

Dwarapudi Harshavardhan, 200050038

Rahul Dathathreya G, 180050034

Overview:

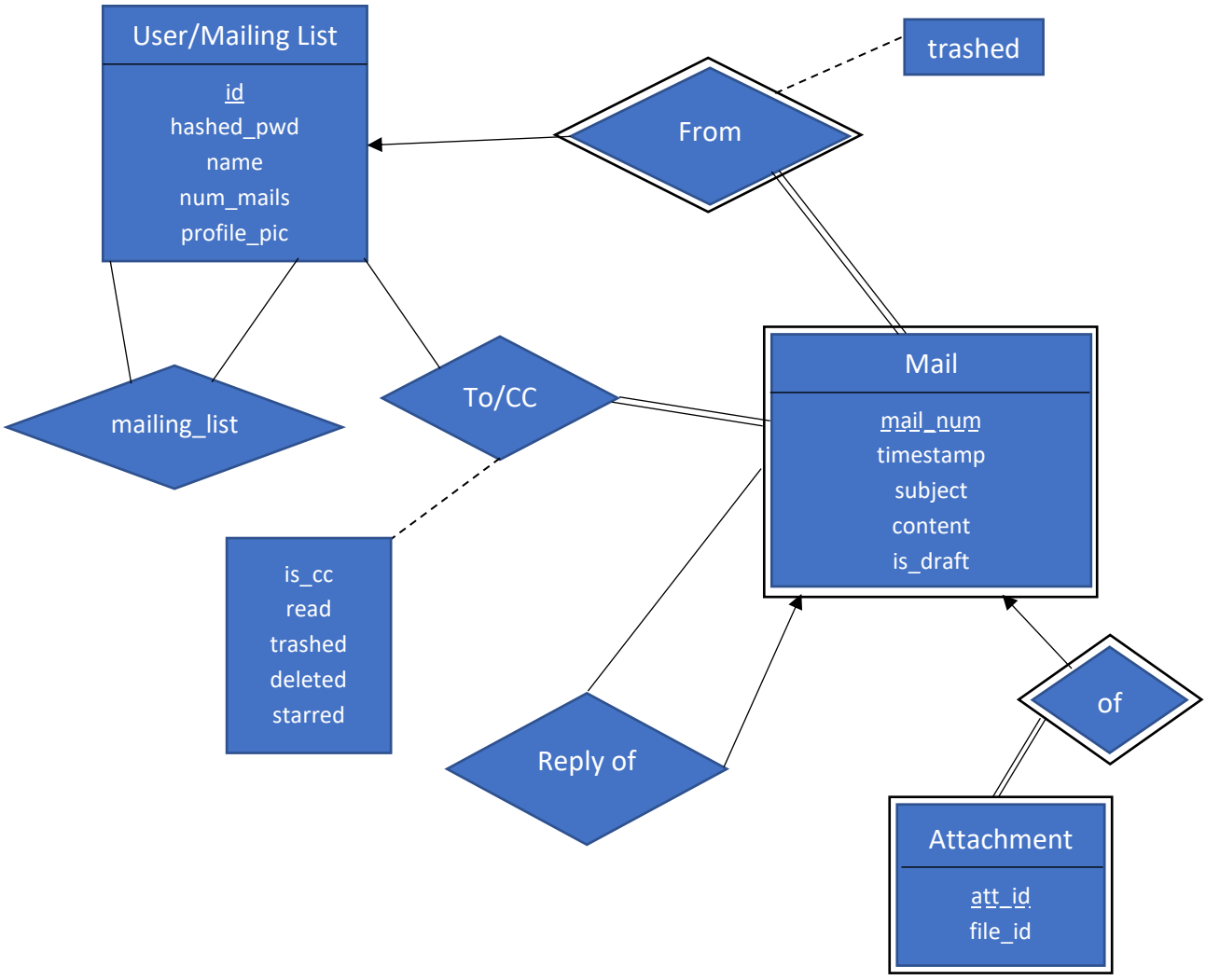
We create a website which serves the purpose of a generic email service.

Users:

An enterprise, university, or community

Proposed database usage design (how data is stored):

ER Design



Relational Design/Schema

From the above ER design, the following relational schema can be inferred to be optimal:

The relation mail_admin wasn't shown in the above ER design as it doesn't relate with any other relations and is only for administration purposes.

Primary key for each relation is marked in bold.

1) mail_admin (**id**, hashed_pwd)
2) mail_user (**id**, hashed_pwd, name, num_mails, profile_pic)
3) mail (**sender_id, mail_num**, timestamp, subject, content, is_draft, trashed)
    a. sender_id references id in user
4) recipient (**sender_id, mail_num, id**, is_cc, read, starred, trashed, deleted)
    a. (sender_id, mail_num) references (sender_id, mail_num) in mail
    b. id references id in mail_user
5) mailing_list (**user_id, list_id**)
    a. user_id references id in mail_user
    b. list_id references id in mail_user
6) reply (**id, mail_num**, p_id, p_mail_num)
    a. (id, mail_num) references (sender_id, mail_num) in mail
    b. (p_id, p_mail_num) references (sender_id, mail_num) in mail
7) attachment (**sender_id, mail_num, att_id**, file_id)
    a. (sender_id, mail_num) references (sender_id, mail_num) in mail

Points to note:

- Even mailing lists are stored in the user relation but with not pwd and num_mails -1
- Scheduled mails are also stored in the mail table but with timestamp in future
- The attribute trashed in the mail relation denotes whether the mail is in sent box or trash box of sender where as the same attribute in to relation denotes the same but of recipient. The attributed deleted denotes deleted from trashed too.

More detailed SQL DDL file at the end of the document (Appendix 1).

Application Backend:

Based on Express, Node JS. Connected to PostgreSQL database. Following are the major APIs that are provided to the frontend general users:

1) Authentication – login, logout, change password and checking whether logged in
2) Send a mail or Save as Draft– inserts rows to the relations ‘mail’, ‘to’ and ‘attachment’ if any and also in ‘reply’ if it a reply to a previous mail.
3) Some APIs to star, mark as read/unread, archive, move to trash or may be a single API for all of these
4) One API for each of inbox, unread, starred, archived, bin, sent, scheduled, drafts.

Major APIs to administrators (along with above):

1) Create a mailing list or add a user to a mailing list
2) Create or remove a user

A list of tentative API endpoints (a code snippet) to be provided is attached at the end of the document (Appendix 2). Some more endpoints might be added in the future.

Application Frontend:

Based on React. It would be an email website which would have most of the features in any common email like viewing inbox, sent, unread, drafts, starred, archived, scheduled, bin pages and compose, reply, forward, move to trash, star, archive, schedule mail, etc.

For this too, a code snippet is attached at the end (Appendix 3) which gives a brief idea on the frontend endpoints.

Appendix 1: SQL Data Definition Language – to create database with required schema

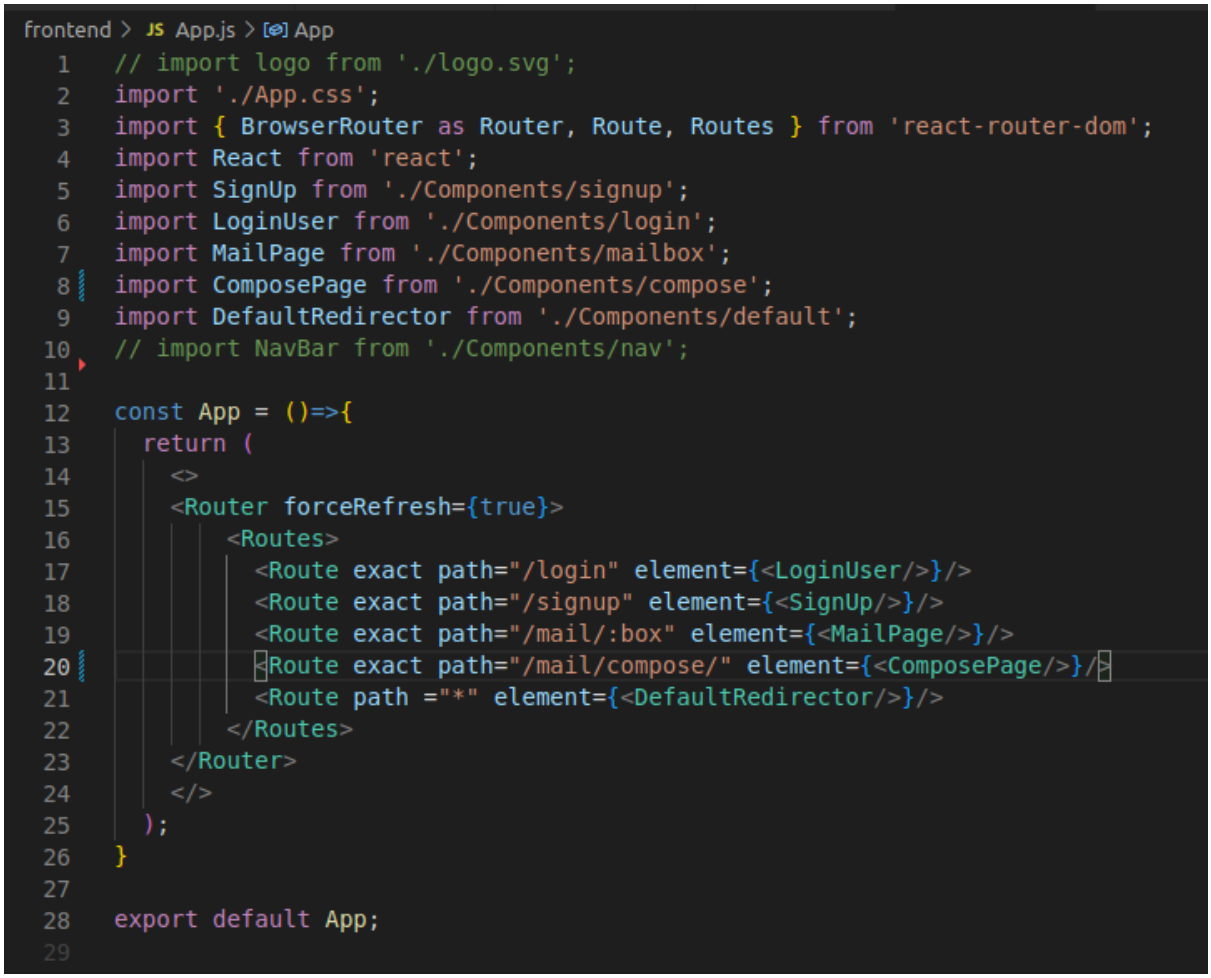
database > ddl.sql

```sql
drop table if exists attachment;
drop table if exists reply;
drop table if exists mailing_list;
drop table if exists recipient;
drop table if exists mail;
drop table if exists mail_user;
drop table if exists mail_admin;

create table mail_admin(
    id varchar(25) not null,
    hashed_pwd varchar(80) not null,
    primary key(id)
);

create table mail_user(
    id varchar(25),
    hashed_pwd varchar(80) not null,
    name varchar(40) not null,
    profile_pic varchar(30),
    num_mails int not null default 0 check (num_mails >= 0),
    -- number of mails sent by the user
    -- or is -1 for a mailing list
    primary key(id)
);

create table mail(
    sender_id varchar(25) not null,
    mail_num int not null,
    time timestamptz not null,
    subject varchar(200),
    content varchar(1000),
    is_draft boolean not null default false,
    trashed boolean not null default false,
    -- in trash of sender
    deleted boolean not null default false,
    -- deleted permanently
    primary key(sender_id, mail_num),
    foreign key(sender_id) references mail_user(id)
);

create table recipient(
    sender_id varchar(25) not null,
    mail_num int not null,
    id varchar(25) not null,
    is_cc boolean not null default false,
    read boolean not null default false,
    starred boolean not null default false,
    trashed boolean not null default false,
    -- in trash of recipient
    deleted boolean not null default false,
    -- deleted permanently
    primary key(sender_id, mail_num, id),
    foreign key(sender_id, mail_num) references mail(sender_id, mail_num),
    foreign key(id) references mail_user(id)
);

create table mailing_list(
    id varchar(25) not null,
    list_id varchar(25) not null,
    primary key(id, list_id),
    foreign key(id) references mail_user(id),
    foreign key(list_id) references mail_user(id)
);

create table reply(
    id varchar(25) not null,
    mail_num int not null,
    p_id varchar(25) not null,
    p_mail_num int not null,
    primary key(id, mail_num),
    foreign key(id, mail_num) references mail(sender_id, mail_num),
    foreign key(p_id, p_mail_num) references mail(sender_id, mail_num)
);

create table attachment(
    sender_id varchar(25) not null,
    mail_num int not null,
    att_num int not null check (att_num >= 0),
    file_id varchar(30) not null,
    primary key(sender_id, mail_num, att_num),
    foreign key(sender_id, mail_num) references mail(sender_id, mail_num)
);

```

Appendix 2: API endpoints in backend

```
> app.get('/logout',…
  )

> app.post('/login',…
  )

> app.post('/signup',…
  )

> app.get('/check_login',…
  )

> app.get('/mail/:box',…
  )

> app.post('/send_mail',…
  )

> app.post('/get_mail',…
  )

> app.post('/mark_as_read',…
  )

> app.post('/move_to_trash',…
  )

> app.post('/mark_star',…
  )

> app.post('/new_mailing_list',…
  )

> app.post('/add_to_mailing_list',…
  )

> app.listen(port,…
  )
```

Appendix 3: Endpoints in frontend



```
frontend > JS App.js > App
1   // import logo from './logo.svg';
2   import './App.css';
3   import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
4   import React from 'react';
5   import SignUp from './Components/signup';
6   import LoginUser from './Components/login';
7   import MailPage from './Components/mailbox';
8   import ComposePage from './Components/compose';
9   import DefaultRedirector from './Components/default';
10  // import NavBar from './Components/nav';
11
12  const App = ()=>{
13    return (
14      <>
15      <Router forceRefresh={true}>
16          <Routes>
17            <Route exact path="/login" element={<LoginUser/>}/>
18            <Route exact path="/signup" element={<SignUp/>}/>
19            <Route exact path="/mail/:box" element={<MailPage/>}/>
20            <Route exact path="/mail/compose/" element={<ComposePage/>}/>
21            <Route path ="*" element={<DefaultRedirector/>}/>
22          </Routes>
23      </Router>
24      </>
25    );
26  }
27
28  export default App;
29
```

The endpoint ‘MailPage’ contains the UI and necessary implementation to show all mail pages such as inbox, sent, starred, draft, scheduled, trash, etc.

The endpoint ‘ComposePage’ includes both composing new mails and editing drafts too.

Some more endpoints might be added based on needs and for admin purposes.