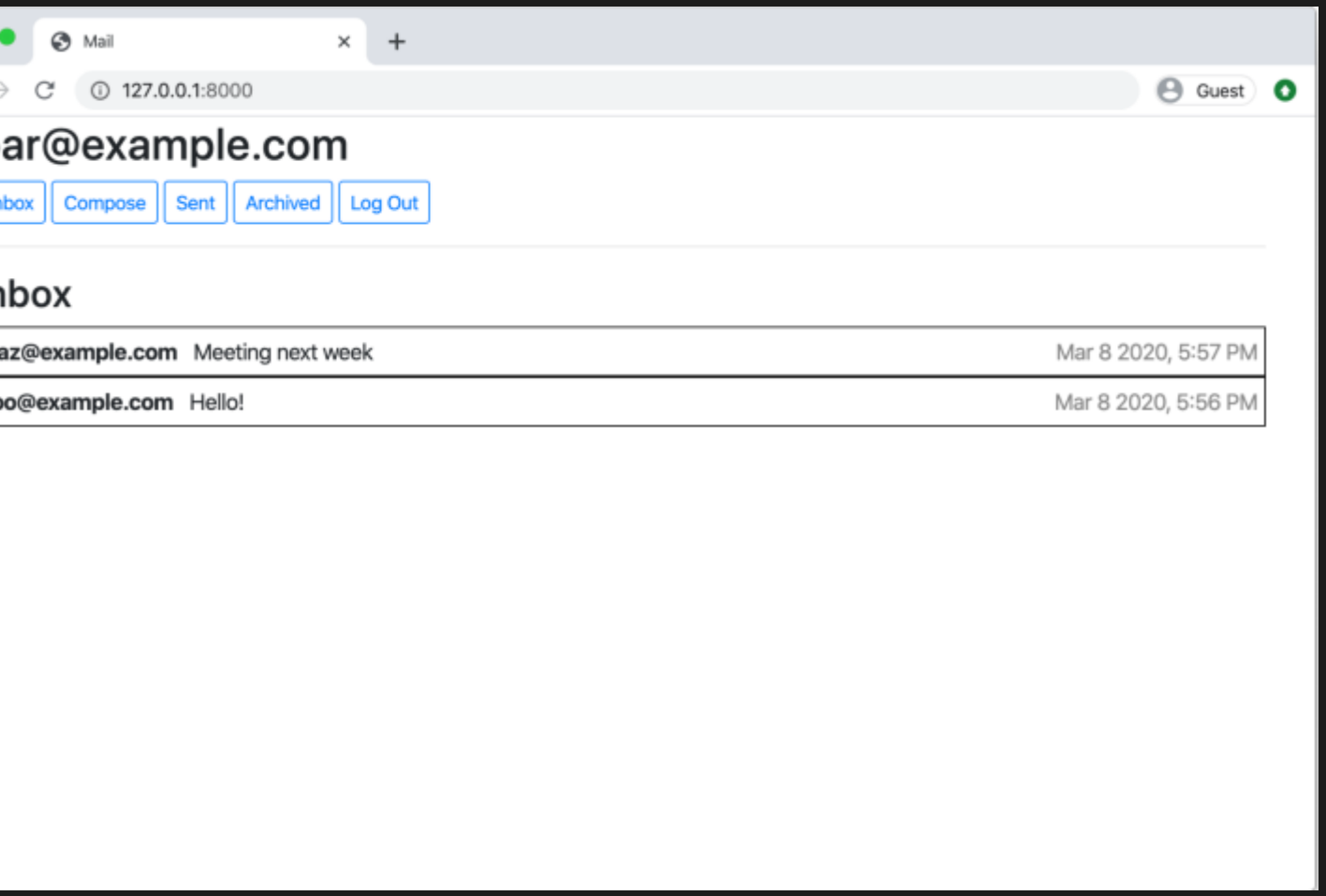
CS50x Iran

راهنمای پروژه email

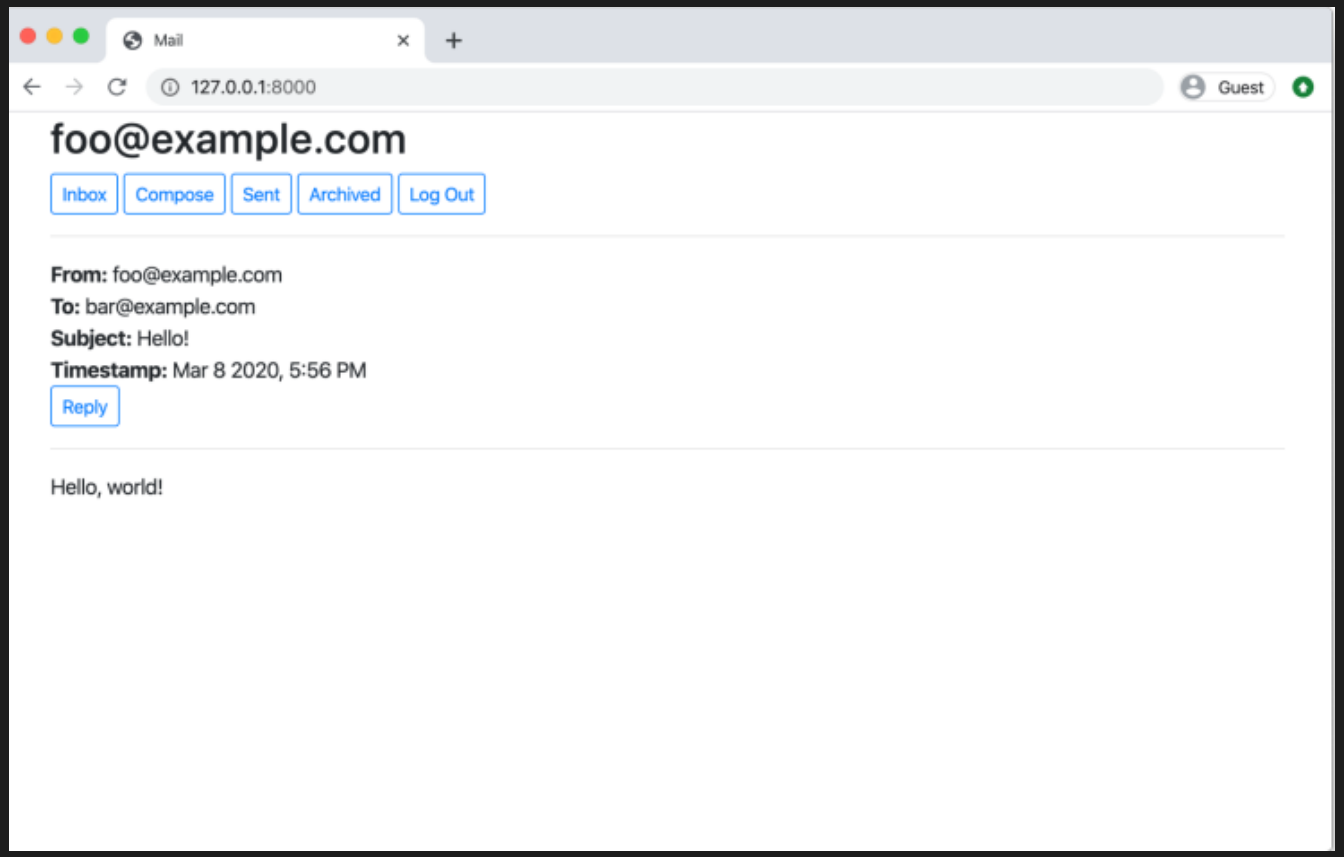
HARVARD
UNIVERSITY

ایمیل



یک سرویس گیرنده ایمیل که تماس های API را برای ارسال و دریافت ایمیل برقرار میکند ، طراحی کنید.

ایمیل



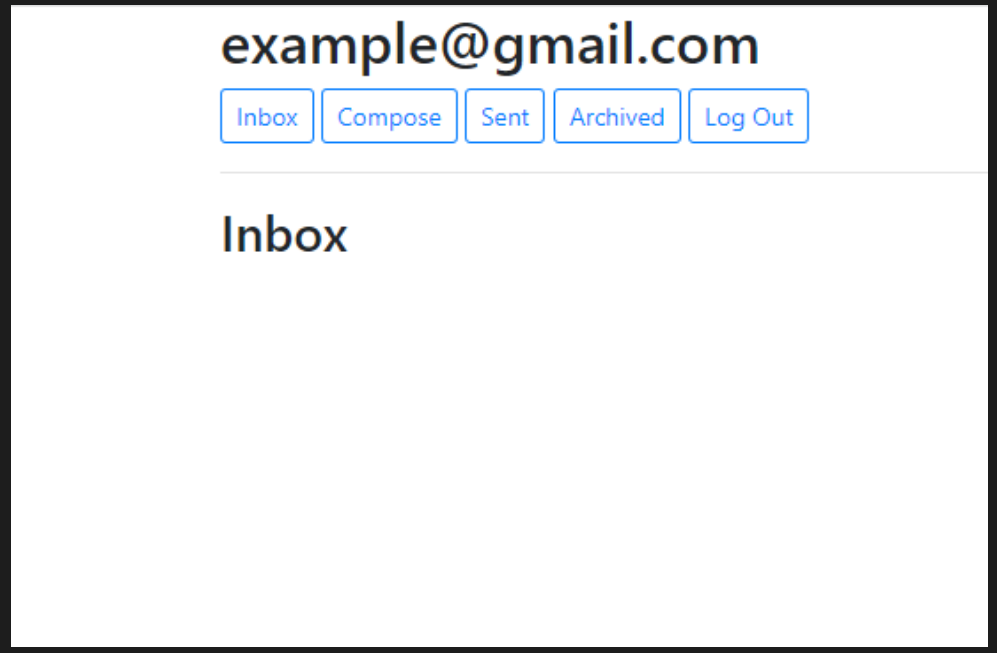
کد توزیع را از
https://cdn.cs50.net/web/2020/spring/projects/3/mail.zip
دانلود کرده و از حالت فشرده خارج کنید.

در ترمینال خود، cd mail وارد کنید.

Python makemigrations mail را اجرا کنید تا با اپ ایمیل اتصال برقرار کند.
Python manage.py migrate را برای اعمال مهاجرت به پایگاه داده خود اجرا کنید.

درک کردن

در کد توزیع، پروژه جنگویی به نام project3 است که یک برنامه به نام mail دارد.
در ابتدا، بعد از آماده سازی و اعمال انتقال های پروژه، python manage.py runserver را برای راه اندازی وب سرور اجرا کنید. وب سرور را در مرورگرتان باز کنید و حساب جدیدی را با استفاده از پیوند"Register" بسازید. ایمیل هایی که در این پروژه میفرستید و دریافت میکنید همه در پایگاه داده تان ذخیره خواهد شد این ایمیل ها در حقیقت به سرورهای ایمیل واقعی ارسال نمیشوند، بنابراین، میتوانید آدرس ایمیل مثلاً، (foo@example.com) و رمز عبوری که برای این پروژه میخواهید را انتخاب کنید: نیاز نیست که اطلاعات کاربری همان اطلاعات کاربری معتبر آدرس های ایمیل واقعی باشد .



درک کردن

وقتی وارد شدید، می بینید که به صفحه Inbox سرویس گیرنده پستی می روید، هرچند که بیشتر این صفحه (فعلاً) خالی است. برای پیمایش در صندوق های پستی Sent و ،Archived روی دکمه ها کلیک کنید؛ میبینید که این ها هم فعلاً خالی اند. روی دکمه "Compose" کلیک کنید تا به فرمی بروید که به شما امکان نوشتن ایمیل جدید را میدهد. با این حال، هر بار که روی دکمه ای کلیک میکنید، به مسیر جدید نمیروید یا درخواست ایمیل جدید ساخته نمیشود: در عوض، کل برنامه فقط یک صفحه است که از جاوا اسکریپت برای کنترل رابط کاربری استفاده میکند. حالا بیایید کد توزیع را دقیق تر بررسی کنیم و ببینیم چطور کار میکند .

درک کردن

نگاهی به mail/urls.py بکنید تا ببینید که مسیر پیشفرض باعث بارگیری تابع index در views.py میشود.
پس بیایید views.py را بالا بیاوریم و تابع index را نگاه کنیم. میبینید که تا وقتی کاربر ورود کرده است، این تابع قالب mail/inbox.html را نشان میدهد .

```python
urlpatterns = [
    path("", views.index, name="index"),
    path("login", views.login_view, name="login"),
    path("logout", views.logout_view, name="logout"),
    path("register", views.register, name="register"),
]
```

```python
def index(request):

    # Authenticated users view their inbox
    if request.user.is_authenticated:
        return render(request, "mail/inbox.html")

    # Everyone else is prompted to sign in
    else:
        return HttpResponseRedirect(reverse("login"))
```

Network/models.py

بیایید این قالب را که در mail/templates/mail/inbox.htmlذخیره شده است ببینیم.
مشاهده میکنید که در بدنه صفحه ، آدرس ایمیل کاربر ابتدا در عنصر h2 نمایش داده میشود. بعد از آن، دنباله ای از دکمه ها در صفحه برای پیمایش بین صفحات مختلف برنامه وجود دارد. پایین آن، میبینید که این صفحه دو بخش اصلی دارد که هر کدام با عنصر divتعریف شده اند. اولی (با idبرابر با emails-view) محتوای صندوق پستی ایمیلها را دارد (که در ابتدا خالی است).
دومی (با idبرابر با compose-view) فرمی دارد که کاربر با آن ایمیل جدیدی مینویسد. بنابراین، دکمه های بالا باید این view ها را به صورت گزینشی نمایش دهند یا پنهان کنند: برای مثال، دکمه compose باید emails-view را پنهان کند و view_compose را نمایش دهد؛ در عین حال، دکمه inbox باید compose-view را پنهان کند و emails-view را نمایش دهد.
چطور اینکار را میکند؟.

درک کردن

EXPLORER

MAIL

- mail
  - __pycache__
  - migrations
  - static\mail
    - inbox.js
    - styles.css
  - templates\mail
    - inbox.html
    - layout.html
    - login.html
    - register.html
  - __init__.py
  - admin.py
  - apps.py
  - models.py
  - tests.py
  - urls.py
  - views.py
- project3
- db.sqlite3
- manage.py

inbox.html

mail > templates > mail > inbox.html

```
1  {% extends "mail/layout.html" %}
2  {% load static %}
3
4  {% block body %}
5      <h2>{{ request.user.email }}</h2>
6
7      <button class="btn btn-sm btn-outline-primary" id="inbox">Inbox</button>
8      <button class="btn btn-sm btn-outline-primary" id="compose">Compose</button>
9      <button class="btn btn-sm btn-outline-primary" id="sent">Sent</button>
10     <button class="btn btn-sm btn-outline-primary" id="archived">Archived</button>
11     <a class="btn btn-sm btn-outline-primary" href="{% url 'logout' %}">Log Out</a>
12     <hr>
13
14     <div id="emails-view">
15     </div>
16
17     <div id="compose-view">
18         <h3>New Email</h3>
19         <form id="compose-form">
20             <div class="form-group">
21                 From: <input disabled class="form-control" value="{{ request.user.email }}">
22             </div>
23             <div class="form-group">
24                 To: <input id="compose-recipients" class="form-control">
25             </div>
26             <div class="form-group">
27                 <input class="form-control" id="compose-subject" placeholder="Subject">
28             </div>
29             <textarea class="form-control" id="compose-body" placeholder="Body"></textarea>
30             <input type="submit" class="btn btn-primary"/>
31         </form>
32     </div>
33 {% endblock %}
34
35 {% block script %}
36     <script src="{% static 'mail/inbox.js' %}"></script>
37 {% endblock %}
```

درک کردن

توجه کنید که در پایین ،inbox.html فایل جاوااسکریپت mail/inbox.js گنجانده شده است.
این فایل را باز کنید، آن را در mail/static/mail/inbox.js ذخیره کنید، و آن را نگاه کنید. مشاهده میکنید که وقتی محتوای DOMصفحه بارگیری میشود، شنونده های رویداد را به هر کدام از دکمه ها الصاق میکنیم. برای مثال، وقتی دکمه inbox کلیک میشود، تابع load_mailbox را با آرگومان "inbox" صدا میزنیم؛ وقتی هم دکمه compose کلیک میشود، تابع compose_email را صدا میزنیم. این توابع چه کار میکنند؟ تابع compose_email ابتدا emails-view را پنهان میکند (به ویژگی style.display آن مقدار none میدهد و compose-view را نمایش میدهد) به ویژگی style.display آن مقدار block میدهد(. بعد از آن، این تابع تمام فیلدهای input فرم را میگیرد (کاربر ممکن است در این فیلدها آدرس ایمیل گیرنده، عنوان، و متن اصلی ایمیل را تایپ کرده باشد) و برای آنکه پاکشان کند، رشته خالی '' را در مقدارشان قرار میدهد. این یعنی که هر بار روی دکمه "Compose" کلیک میکنید، باید فرم ایمیل خالی به شما نمایش داده شود: برای امتحان آن میتوانید مقادیری را در فرم تایپ کنید، view را به Inbox تغییر دهید و سپس به view با نام Compose بروید.

static > mail > JS inbox.js > ...

```
document.addEventListener('DOMContentLoaded', function() {

  // Use buttons to toggle between views
  document.querySelector('#inbox').addEventListener('click', () => load_mailbox('inbox'));
  document.querySelector('#sent').addEventListener('click', () => load_mailbox('sent'));
  document.querySelector('#archived').addEventListener('click', () => load_mailbox('archive'));
  document.querySelector('#compose').addEventListener('click', compose_email);

  // By default, load the inbox
  load_mailbox('inbox');
});
```

درک کردن

در عین حال، تابع load_mailbox ابتدا emails-view را نشان میدهد و compose-view را پنهان میکند. تابع load_mailbox نیز آرگومانی میگیرد که نام صندوق پستی ای است که کاربر میخواهد ببیند. برای این پروژه، یک سرویس گیرندۀ ایمیل با سه صندوق طراحی خواهید کرد: ،inboxصندوق sent شامل تمام نامه های ارسال شده، و آرشیو ایمیلهایی که قبلاً در inbox بوده اند اما بعد آرشیو شده اند. بنابراین، آرگومان load_mailbox یکی از آن سه مقدار خواهد بود و تابع load_mailbox نام صندوق انتخاب شده را با به روزرسانی innerHTML در emails-view نمایش میدهد)بعد از تبدیل کاراکتر اول آن به حرف بزرگ(. به همین علت، وقتی نام صندوق را در مرورگرتان انتخاب میکنید، نام آن صندوق)تبدیل شده به حروف بزرگ(در DOM ظاهر میشود. تابع load_mailbox به به روزرسانی emails-view میپردازد تا متن مناسب را در آن قرار دهد.

```
...

function compose_email() {

  // Show compose view and hide other views
  document.querySelector('#emails-view').style.display = 'none';
  document.querySelector('#compose-view').style.display = 'block';

  // Clear out composition fields
  document.querySelector('#compose-recipients').value = '';
  document.querySelector('#compose-subject').value = '';
  document.querySelector('#compose-body').value = '';
}
```

درک کردن

```
function load_mailbox(mailbox) {

  // Show the mailbox and hide other views
  document.querySelector('#emails-view').style.display = 'block';
  document.querySelector('#compose-view').style.display = 'none';

  // Show the mailbox name
  document.querySelector('#emails-view').innerHTML = `<h3>${mailbox.charAt(0).toUpperCase() + mailbox.slice(1)}</h3>`;
}
```

درک کردن

البته، این برنامه ناقص است. تمام صندوقها نام صندوق (Inbox، Sent، Archive) را نمایش میدهند، اما در حقیقت هیچ ایمیلی را نشان نمیدهند. هنوز هیچ ویویی برای مشاهدهٔ محتوای ایمیلها وجود ندارد. و فرم composeامکان تایپ محتوای ایمیل را به شما میدهد، اما دکمه ارسال ایمیل در واقع هیچ کاری نمیکند. اینجاست که باید نقش ایفا کنید

با استفاده از APIاین برنامه، نامه دریافت میکنید، نامه میفرستید، و ایمیلهایتان را به روز میکنید. ما کل APIرا برایتان نوشته ایم)و در زیر مستند کرده ایم(تا بتوانید آن را در کد جاوا اسکریپت تتان استفاده کنید.(توجه کنید که در واقع که کل کد پایتون را برایتان در این پروژه نوشته ایم. باید بتوانید این پروژه را با نوشتن HTMLو جاوا اسکریپت تکمیل کنید).

این برنامه از مسیرهای APIزیر پشتیبانی میکند:

GET

GET /emails/<str:mailbox>

ارسال درخواست GET به /emails/<mailbox>، که در آن <mailbox> یا inbox است یا sent یا archive، باعث میشود با ترتیب زمانی معکوس به فهرست تمام ایمیل های آن صندوق برگردید (در قالب JSON). برای مثال ، اگر درخواست GET را به /emails/inbox بفرستید ، شاید پاسخ JSON شبیه زیر دریافت کنید (دو ایمیل را نشان می دهد):

```
{
"id": 100,
"sender": "foo@example.com",
"recipients": ["bar@example.com"],
"subject": "Hello!",
"body": "Hello, world!",
"timestamp": "Jan 2 2020, 12:00 AM",
"read": false,
"archived": false
}
```

توجه کنید که در هر ایمیل، id(شناسه منحصربه فرد)، آدرس ارسال کنندهٔ (sender) ایمیل، آرایه ای از دریافت کنندگان ، (recipients) رشته ای برای عنوان ایمیل ، (subject) متن اصلی، و برچسب زمانی ، (timestamp) و همینطور دو مقدار بولی مشخص میشود که نشان میدهند آن ایمیل خوانده شده است یا خیر و آیا آن ایمیل archive شده است یا خیر.

چطور در جاوااسکریپت به این مقادیر دسترسی دارید؟ اگر از جاوا اسکریپت به یاد داشته باشید، میتوانید از fetchبرای ایجاد درخواست وب استفاده کنید. بنابراین، کد جاوا اسکریپت زیر

```
fetch('/emails/inbox')
.then(response => response.json())
.then(emails => {
// Print emails
console.log(emails);
// ... do something else with emails ...
});
```

درخواست GETبه emails/inbox / ایجاد میکند، پاسخ حاصل را به JSON برمیگرداند، و سپس آرایه ای از ایمیلهای درون متغیر emails را در اختیارتان میگذارد. میتوانید آن مقدار را با استفاده از console.log در کنسول مرورگر چاپ کنید(اگر ایمیلی داخل inboxشما نباشد، این آرایه خالی خواهد بود) یا کار دیگری با آن آرایه کنید.

GET

توجه کنید کـi اگر صندوق نامعتبری را درخواست کنید (هر چیزی غیر از ، inbox، sentیا ،archive)این پاسخJSON را دریافت خواهید کرد:

{“error”:”Invalid mailbox”}

GET /emails/<int:email_id>

با ارسال درخواست GETبه emails/email_id / که در آن email_id شناسه ایمیل از نوع عدد صحیح است، نمایش JSON آن ایمیل مانند زیر برگردانده خواهد شد :

```
{
"id": 100,
"sender": "foo@example.com",
"recipients": ["bar@example.com"],
"subject": "Hello!",
"body": "Hello, world!",
"timestamp": "Jan 2 2020, 12:00 AM",
"read": false,
"archived": false
}
```

توجه کنید که اگر ایمیل وجود نداشته باشد، یا اگر کاربر به آن ایمیل دسترسی نداشته باشد، مسیری که برگردانده میشود خطای Not Found 404 همراه با این پاسخ JSON است:

{"error": "Email not found"}

برای مثال، برای دریافت ایمیل شمارۀ 100، ممکن است کد جاوا اسکریپتی مانند زیر بنویسید

```
fetch('/emails/100')
.then(response => response.json())
.then(email => {
// Print email
console.log(email);
// ... do something else with email ...
});
```

تاکنون نحوۀ دریافت ایمیل‌ها را دیدیم: یا همۀ ایمیل‌های صندوق پستی یا فقط یک ایمیل.

POST/emails

برای ارسال ایمیل، می‌توانید درخواست POSTبه مسیر emails/ بفرستید. این مسیر مستلزم ارسال سه قطعه داده است: مقدار recipients (رشته‌ای جدا شده با ویرگول از تمام کاربرانی که به آنها ایمیل ارسال می‌شود)، رشتهٔ subject ، رشتهٔ Body .برای مثال ، می‌توانید کد جاوا اسکریپتی مانند زیر بنویسید:

```
fetch('/emails', {
  method: 'POST',
  body: JSON.stringify ({
      recipients: 'baz@example.com',
      subject: 'Meeting time',
      body: 'How about we meet tomorrow at 3pm?'
  })
})
.then(response => response.json())
.then(result => {
    // Print result
    console.log(result);
});
```

POST/emails

اگر ایمیل با موفقیت ارسال شود، این مسیر با کد وضعیت 200 پاسخ می‌دهد و پاسخ JSONآن عبارت خواهد بود از: {"message": "Email sent successfully."}.

توجه کنید که حداقل باید یک گیرندهٔ ایمیل داشته باشیم: اگر گیرنده‌ای ارائه نشود، آن مسیر با کد وضعیت 400 پاسخ می‌دهد و پاسخ JSON عبارت خواهد بود از {"error": "At least one recipient required."}.

PUT /emails/<int:email_id>

مسیر آخری که نیاز خواهید داشت نشانه‌گذاری ایمیل به عنوان خوانده‌شده/خوانده‌نشده یا آرشیوشده/آرشیو نشده است. برای این کار، درخواست PUT (به جای درخواست GET) به emails/<email_id>/ بفرستید که در آن email_id شناسهٔ ایمیلی است که می‌خواهید اصلاحش کنید. برای مثال، کد جاوا اسکریپتی شبیه زیر:

```
fetch('/emails/100', {
  method: 'PUT',
  body: JSON.stringify({
      archived: true
  })
})
```

ایمیل شمارهٔ 100 را به‌عنوان آرشیوشده نشانه‌گذاری می‌کند. بدنهٔ درخواست PUT ممکن است {archived: false} باشد تا پیام را از آرشیو بیرون بیاورد، و همین‌طور ممکن است {read: true} یا {read: false} باشد و ایمیل را، به ترتیب، خوانده‌شده یا خوانده‌نشده کند.

با استفاده از این چهار مسیر (APIدریافت تمام ایمیل‌های صندوق پستی، دریافت یک ایمیل، ارسال ایمیل، و به‌روزرسانی ایمیل موجود)، همهٔ ابزارهای لازم را برای تکمیل این پروژه خواهید داشت.

مشخصات

با استفاده از جاوا اسکریپت ، HTML، و CSS، سرویس‌گیرندهٔ ایمیل را در برنامهٔ تک‌صفحه‌ای‌تان در inbox.js (و نه هیچ فایل اضافهٔ دیگری ؛ برای نمره دادن، ما فقط inbox.jsرا در نظر می‌گیریم!) پیاده‌سازی کنید.
باید این نیازمندی‌ها را محقق کنید:

مشخصات

ارسال نامه: وقتی کاربر فرم ایجاد ایمیل را می‌فرستد، کد جاوا اسکریپت را برای ارسال واقعی نامه اضافه کنید.

ـ احتمالاً می‌خواهید درخواست POSTبه emails/ داشته باشید و مقادیر مربوط به recipients ، subject، و body را منتقل کنید.

- بعد از اینکه ایمیل ارسال شد، صندوق ایمیل‌های ارسال‌شدهٔ کاربر را بارگیری کنید.

مشخصات

Mailbox: وقتی کاربر به صندوق inbox، Sent، یا Archiveخود می‌رود، صندوق مناسب را بارگیری کنید.

- احتمالاً می‌خواهید درخواست GET به /emails/<mailbox> بفرستید تا ایمیل‌های آن صندوق خاص را درخواست کنید.

- وقتی به صندوق پستی می‌روید، برنامه باید ابتدا API آخرین ایمیل‌های آن صندوق پستی را کوئری بگیرد.

- وقتی به صندوق پستی می‌روید، نام صندوق پستی باید بالای صفحه نمایان شود (این بخش برای شما انجام شده است).

مشخصات

ـ بنابراین، هر ایمیلی باید در صندوق مخصوصش ارائه شود (برای مثال، به صورت همراه با خط دور) و مشخص شود که ایمیل از طرف کیست، موضوعش چیست، و مهرزمانی آن ایمیل چیست.

ـ اگر ایمیل خوانده‌نشده باشد، باید با پس‌زمینهٔ سفید نمایان شود.
اگر خوانده شده است، باید با پس‌زمینهٔ خاکستری نمایان شود.

مشخصات

مشاهدهٔ ایمیل: وقتی کاربر روی ایمیلی کلیک می‌کند، باید به ویویی برده شود که در آنجا محتوای آن ایمیل را مشاهده کند.

- احتمالاً می‌خواهید درخواست GET به /emails/<mailbox> داشته باشید تا آن ایمیل را درخواست کنید.
- برنامه‌تان باید فرستنده، گیرنده‌ها، عنوان، برچسب زمانی، و متن اصلی ایمیل را نمایش دهد.
- احتمالاً می‌خواهید divدیگری به inbox.html (علاوه‌بر emails-viewو compose-) viewبرای نمایش آن ایمیل اضافه کنید. حتماً کدتان را به روزرسانی کنید تا هنگام کلیک روی دکمه‌های پیمایش، ویوهای صحیحی پنهان شوند یا نمایش یابند.

مشخصات

- راهنمایی مندرج در بخش راهنمایی را ببینید تا بدانید چطور به عنصر HTML که به DOM اضافه کرده‌اید شنوندهٔ رویداد بیفزایید.
- بعد از اینکه روی ایمیل کلیک شد، باید آن ایمیل را به عنوان خوانده‌شده علامت بزنید. اگر یادتان باشد، برای به‌روزرسانی اینکه ایمیل خوانده شده است یا نه، می‌توانید درخواست PUT به /emails/<email_id> بفرستید.

مشخصات

آرشیو کردن و از آرشیو درآوردن: به کاربران اجازه دهید ایمیل‌هایی که دریافت کرده‌اند را آرشیو کنند و از آرشیو بیرون بیاورند.

ـ هنگامی که کاربر ایمیلی در Inbox را مشاهده می‌کند، دکمه‌ای باید در اختیارش باشد که با آن بتواند ایمیل را آرشیو کند. هنگامی که کاربر ایمیلی در Archive را مشاهده می‌کند، باید دکمه‌ای در اختیارش باشد که بتواند با آن ایمیل را از آرشیو خارج کند. این نیازمندی در مورد ایمیل‌های صندوق Sent صدق نمی‌کند.

ـ اگر یادتان باشد، برای نشانه‌گذاری ایمیل به عنوان آرشیو شده یا آرشیو نشده، می‌توانید درخواست PUT به /emails/<mailbox> بفرستید.

ـ وقتی ایمیل آرشیو شده یا آرشیو نشده شد، صندوق inboxکاربر را بارگیری کنید.

مشخصات

پاسخ: به کاربران اجازه دهید به ایمیل پاسخ دهند.

- هنگامی که کاربر ایمیلی را مشاهده می‌کند، باید دکمهٔ "Reply" در اختیارش قرار بگیرد تا بتواند به ایمیل پاسخ دهد.
- فرم ایجاد ایمیل را از قبل پر کنید، به این ترتیب که در فیلد recipient اسم کسی قرار بگیرد که ایمیل اصلی را فرستاده است.
- خط subject را از قبل پر کنید. اگر عنوان ایمیل اصلی foo بوده است، عنوان جدید باید Re: foo باشد (اگر عنوان با Re : شروع می‌شود، نیازی نیست که دوباره آن را اضافه کنید.)
- بخش body ایمیل را با خطی شبیه این از قبل پر کنید:
«On Jan 1 2020, 12:00 AM foo@example.com wrote:»،
و بعد متن اصلی ایمیل را بیاورید.

راهنمایی

ـبرای ایجاد عنصر HTMLو افزودن کنترل‌کنندهٔ رویداد به آن، می‌توانید از کد جاوا اسکریپتی شبیه زیر استفاده کنید:

```
const element = document.createElement('div');
element.innerHTML = 'This is the content of the div.';
element.addEventListener('click', function() {
    console.log('This element has been clicked!')
});
document.querySelector('#container').append(element);
```

این کد عنصر div جدیدی می‌سازد، InnerHTML آن را مقداردهی می‌کند، کنترل‌کنندهٔ رویدادی اضافه می‌کند تا وقتی روی آن div کلیک شد، تابع خاصی اجرا شود، و بعد آن را به آن عنصر HTML اضافه می‌کند که Id آن برابر با container است (در این کد فرض می‌شود که یک عنصر HTMLداریم که که id آن برابر با container است: احتمالاً می‌خواهید آرگومان querySelector را تغییر دهید تا همان عنصری باشد که می‌خواهید به آن عنصری اضافه کنید.

راهنمایی

ـ شاید خوب باشد mail/static/mail/styles.css را ویرایش کنید و CSS مورد نیازتان در این برنامه را اضافه کنید.

ـ اگر یادتان باشد، در صورتی که آرایهٔ جاوا اسکریپت داشته باشید، با استفاده از forEach می‌توانید حلقه را به ازای تمام عناصر آن آرایه تکرار کنید.

ـ اگر یادتان باشد، معمولاً به ازای درخواست‌های POST و PUT ، جنگو برای مقابله با حملات احتمالی جعل درخواست میان وب‌گاهی[1] به توکن CSRF نیاز دارد. برای این پروژه، ما مسیرهای API را عامدانه خالی از CSRF کرده‌ایم، بنابراین نیازی به توکن ندارید. با این حال، در پروژهٔ واقعی، همیشه بهتر است مراقب چنین آسیب‌پذیری‌های بالقوه‌ای باشید!

نحوهٔ ارسال

https://cs50x.ir/winter/static/web/weeks/submit-web-project.pdf

ـ درصورتی که مرحله 1 را برای پروژه صفر انجام داده اید ، به مرحله 2 بروید.
1. به این پیوند بروید، وارد حساب گیت‌هاب شوید، و روی Authorize cs50 کلیک کنید. سپس کادری را علامت بزنید که نشان می‌دهد اجازهٔ دسترسی به مطالب ارسالی‌تان را به کارمندان دوره می‌دهید؛ روی Join course کلیک کنید.

نحوهٔ ارسال

2-به ide.cs50.io بروید و بر روی "Sign in with GitHub" کلیک کنید تا به CS50 IDE خود دسترسی پیدا کنید.
2-فایل اصلی پروژه خود را آپلود کنید.
3- cd network را اجرا کنید تا به مسیر پروژه بروید.
4- دستور submit50 web50/projects/2020/x/network
را اجرا کنید تا پروژه هفته هفتم شما با موفقیت سابمیت شود.

#submit50 از شما نام کاربری اکانت گیت هابتان و Personal Access Token میخواهید که میتوانید طی مراحل ذکر شده نحوه سابمیت کردن (موجود در جلسه صفر) Personal Access Tokenخود را دریافت کنید.
ـدر یک ویدیو حداکثر 5 دقیقه ای نحوه کارکرد پروژه خود را نشان دهید و حتما تایم استمپ گذاری کنید .
این فرم را سابمیت کنید

برای مشاهدهٔ روند پیشرفت فعلی‌تان، می‌توانید به https://cs50.me/cs50w بروید.

نکته ارسال

وقتی پروژه‌تان را ارسال می‌کنید، محتوای شاخهٔ web50/projects/2020/x/mail باید با ساختار فایل کد توزیع غیر فشرده، همان‌طور که در ابتدا دریافت شده بود، مطابقت داشته باشد. به عبارتی، فایل‌هایتان نباید داخل هیچ‌کدام از دایرکتوری‌هایی که خودتان ایجاد کرده‌اید بنشینند. شاخهٔ شما همچنین نباید به جز این پروژه، حاوی کد پروژه‌های دیگر باشد. عدم رعایت این ساختار فایل احتمالاً باعث رد شدن کار ارسالی‌تان می‌شود.

برای مثال، در این پروژه، اگر کارمندان نمره‌دهی به آدرس https://github.com/me50/USERNAME/tree/web50/projects/2020/x/mail بروند (جایی که USERNAME برابر است با نام کاربری‌تان در گیت‌هاب که در فرم زیر ارائه شده است)، باید دو زیر دایرکتوری (project3، و mail) همین‌طور فایل manage.py را ببینیم. بعد از بررسی این موضوع، اگر کدتان به این شکل سازماندهی نشده است، مخزن کارهایتان را مطابق با این پارادایم دوباره ساماندهی کنید.

CS50x Iran

which uses Harvard University's introduction to the
intellectual enterprises of computer science and the art of programming course

CS50x.ir