This is a fascinating project! To build an app that automates downloading your daily newspaper subscription and emailing it to your mailing list, running from your Linux terminal, here’s a breakdown of the best methods and options, keeping in mind your preference for open-source, security, and your technical expertise:

**Core Components and Technologies:**

1. **Programming Language:** Python is an excellent choice due to its extensive libraries for web interaction, email handling, and scripting.
2. **Website Login and Content Download:**
   * **requests library:** This is a powerful and user-friendly library for making HTTP requests. You can use it to simulate logging into the newspaper website by sending POST requests with your credentials.
   * **Beautiful Soup 4 library:** Once logged in, you can use requests to download the newspaper content (likely in PDF or HTML format). Beautiful Soup helps parse the HTML content if needed to locate the download link or extract specific articles.
   * **Selenium (if needed):** If the website uses complex JavaScript rendering for login or content loading, Selenium might be necessary. It allows you to automate a web browser (like Chrome or Firefox) to interact with the website. This is generally more resource-intensive than requests and Beautiful Soup.
3. **Newspaper Content Handling:**
   * **PDF:** If the newspaper is a PDF, you can use libraries like PyPDF2 or pdfminer.six for potential processing (though for simply emailing, this might not be needed beyond downloading).
   * **HTML:** If it's HTML, Beautiful Soup can help extract the main content if you want to format it differently for the email body.
4. **Thumbnail Generation:**
   * **For PDF:** You can use pdfminer.six or Pillow (with Ghostscript installed) to render the first page of the PDF as an image and then create a thumbnail.
   * **For HTML:** You could use a headless browser like wkhtmltoimage or Puppeteer (Node.js based, but callable from Python) to take a screenshot of the first page and then use Pillow to create a thumbnail.
5. **Email Templating:**
   * **Python's email module:** This built-in module allows you to construct email messages with different parts (plain text, HTML, attachments).
   * **Jinja2 or Mako:** These are powerful templating engines that let you create reusable HTML templates for your emails, making it easy to format the newspaper content and include the thumbnail and links.
6. **Email Sending:**
   * **Python's smtplib module:** This module allows you to send emails using the Simple Mail Transfer Protocol (SMTP). You'll need to configure it with your email provider's SMTP server details (host, port, username, password). Consider using a dedicated email service for better deliverability if your mailing list is large.
7. **Past 7 Days' Papers:**
   * **Local Storage:** You can create a directory on your Linux machine to store the downloaded newspapers for the past 7 days.
   * **Filename Convention:** Use a consistent naming convention (e.g., YYYY-MM-DD\_newspaper\_name.pdf) to easily manage and identify the files.
   * **Updating the Template:** Your Python script will need to check the last 7 days of downloaded files and generate links to them within the email template.
8. **Scheduling:**
   * **cron (Linux):** This is the standard Linux utility for scheduling tasks to run automatically at specific times. You can set up a cron job to execute your Python script daily at the time the newspaper is usually released.

**Detailed Steps and Code Structure (Conceptual Python):**

import requests  
from bs4 import BeautifulSoup  
from datetime import date, timedelta  
import smtplib  
from email.mime.multipart import MIMEMultipart  
from email.mime.text import MIMEText  
from email.mime.image import MIMEImage  
from jinja2 import Environment, FileSystemLoader  
import os  
# Potentially import other libraries like PyPDF2 or Selenium as needed  
  
# Configuration (store securely - environment variables or a dedicated config file)  
NEWSPAPER\_URL = "your\_newspaper\_login\_url"  
USERNAME = "your\_username"  
PASSWORD = "your\_password"  
DOWNLOAD\_LINK\_SELECTOR = "css selector or xpath to the download link"  
EMAIL\_SUBJECT = f"Daily Newspaper - {date.today().strftime('%Y-%m-%d')}"  
SMTP\_SERVER = "your\_smtp\_server"  
SMTP\_PORT = 587 # Or 465 for SSL  
SMTP\_USERNAME = "your\_email\_address"  
SMTP\_PASSWORD = "your\_email\_password"  
MAILING\_LIST = ["email1@example.com", "email2@example.com"]  
TEMPLATE\_PATH = "path/to/your/email\_template.html"  
NEWSPAPER\_STORAGE\_PATH = "/path/to/store/newspapers"  
THUMBNAIL\_PATH = "/path/to/save/thumbnail.jpg"  
  
def login\_and\_download():  
 session = requests.Session()  
 # Simulate login (adjust based on website's login mechanism)  
 login\_data = {'username': USERNAME, 'password': PASSWORD}  
 response = session.post(NEWSPAPER\_URL, data=login\_data)  
 response.raise\_for\_status() # Raise an exception for bad status codes  
  
 # Find the download link (inspect the website's HTML)  
 soup = BeautifulSoup(response.content, 'html.parser')  
 download\_link\_element = soup.select\_one(DOWNLOAD\_LINK\_SELECTOR) # Use CSS selector  
  
 if download\_link\_element and 'href' in download\_link\_element.attrs:  
 download\_url = download\_link\_element['href']  
 if not download\_url.startswith('http'):  
 download\_url = response.url.split('/')[0] + '//' + response.url.split('/')[2] + download\_url # Handle relative URLs  
  
 newspaper\_response = session.get(download\_url, stream=True)  
 newspaper\_response.raise\_for\_status()  
  
 filename = f"{date.today().strftime('%Y-%m-%d')}\_newspaper.pdf" # Adjust based on actual file type  
 filepath = os.path.join(NEWSPAPER\_STORAGE\_PATH, filename)  
 with open(filepath, 'wb') as f:  
 for chunk in newspaper\_response.iter\_content(chunk\_size=8192):  
 f.write(chunk)  
 return filepath  
 else:  
 print("Download link not found.")  
 return None  
  
def generate\_thumbnail(filepath, output\_path):  
 # Implementation using a library like pdfminer.six and Pillow  
 # (This will require more detailed code depending on the file type)  
 print(f"Generating thumbnail for {filepath} to {output\_path}")  
 # Placeholder  
 pass  
  
def get\_past\_week\_papers():  
 past\_papers = []  
 for i in range(7):  
 day = date.today() - timedelta(days=i)  
 filename = f"{day.strftime('%Y-%m-%d')}\_newspaper.pdf" # Adjust format  
 filepath = os.path.join(NEWSPAPER\_STORAGE\_PATH, filename)  
 if os.path.exists(filepath):  
 past\_papers.append((day.strftime('%Y-%m-%d'), filepath)) # Store date and path  
 return past\_papers  
  
def send\_email(newspaper\_path, thumbnail\_path, past\_papers):  
 env = Environment(loader=FileSystemLoader(os.path.dirname(TEMPLATE\_PATH)))  
 template = env.get\_template(os.path.basename(TEMPLATE\_PATH))  
  
 with open(newspaper\_path, 'rb') as f:  
 newspaper\_content = f.read()  
  
 with open(thumbnail\_path, 'rb') as f:  
 thumbnail\_content = f.read()  
  
 msg = MIMEMultipart('related')  
 msg['Subject'] = EMAIL\_SUBJECT  
 msg['From'] = SMTP\_USERNAME  
 msg['To'] = ', '.join(MAILING\_LIST)  
  
 # Render the HTML template  
 html\_body = template.render(  
 daily\_paper\_link="link\_to\_your\_newspaper\_online\_or\_a\_local\_link", # Consider how you want to link the full paper  
 past\_papers=past\_papers,  
 thumbnail\_cid="thumbnail" # For embedding the image  
 )  
 msg.attach(MIMEText(html\_body, 'html'))  
  
 # Attach the thumbnail  
 image = MIMEImage(thumbnail\_content)  
 image.add\_header('Content-ID', '<thumbnail>')  
 msg.attach(image)  
  
 # Consider attaching the full newspaper PDF as well if desired  
 # with open(newspaper\_path, 'rb') as f:  
 # attachment = MIMEApplication(f.read(), \_subtype="pdf")  
 # attachment.add\_header('Content-Disposition', 'attachment', filename=os.path.basename(newspaper\_path))  
 # msg.attach(attachment)  
  
 try:  
 with smtplib.SMTP(SMTP\_SERVER, SMTP\_PORT) as server:  
 server.starttls() # Use TLS encryption  
 server.login(SMTP\_USERNAME, SMTP\_PASSWORD)  
 server.sendmail(SMTP\_USERNAME, MAILING\_LIST, msg.as\_string())  
 print("Email sent successfully!")  
 except Exception as e:  
 print(f"Error sending email: {e}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 newspaper\_filepath = login\_and\_download()  
 if newspaper\_filepath:  
 generate\_thumbnail(newspaper\_filepath, THUMBNAIL\_PATH)  
 past\_papers = get\_past\_week\_papers()  
 send\_email(newspaper\_filepath, THUMBNAIL\_PATH, past\_papers)

**Email Template (Example email\_template.html):**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Daily Newspaper</title>  
</head>  
<body>  
 <h1>Today's Newspaper</h1>  
 <p>Here's a quick look at today's front page:</p>  
 <img src="cid:thumbnail" alt="Today's Front Page Thumbnail" width="300">  
 <p><a href="{{ daily\_paper\_link }}">Read the full newspaper online</a></p>  
 <hr>  
 <h2>Past 7 Days' Papers:</h2>  
 <ul>  
 {% for day, link in past\_papers %}  
 <li><a href="{{ link }}">{{ day }}</a></li>  
 {% endfor %}  
 </ul>  
 <p>This email was automatically generated.</p>  
</body>  
</html>

**Setting up the cron job:**

1. Open your terminal.
2. Type crontab -e to edit your cron table.
3. Add a line like this (adjust the time as needed, e.g., 6 AM):  
   0 6 \* \* \* python /path/to/your/script.py  
   * 0: Minutes (0-59)
   * 6: Hours (0-23)
   * \*: Day of the month (1-31)
   * \*: Month (1-12)
   * \*: Day of the week (0-6, where 0 is Sunday)
   * python /path/to/your/script.py: The command to execute. Replace with the actual path to your Python script.

**Important Considerations and Best Practices:**

* **Security:**
  + **Never hardcode credentials directly in the script.** Use environment variables, a secure configuration file (with appropriate permissions), or a secrets management solution to store your newspaper website login and email credentials.
  + Be mindful of the security of your Linux machine where you store the script and downloaded newspapers.
* **Website Structure Changes:** Newspaper websites often change their layout. Your script might break if the login mechanism or the location of the download link changes. You'll need to monitor it and update the selectors or logic accordingly.
* **Error Handling:** Implement robust error handling (using try...except blocks) to catch potential issues like network errors, login failures, or download problems. Log these errors to a file for debugging.
* **Rate Limiting:** Be respectful of the newspaper website's terms of service. Avoid making too many requests in a short period, which could lead to your IP address being blocked.
* **File Management:** Implement logic to manage the storage of past papers, potentially deleting files older than 7 days to save disk space.
* **Email Deliverability:** If you have a large mailing list, consider using a reputable email service provider (like SendGrid, Mailgun, or AWS SES) for better deliverability and to avoid your emails being marked as spam. These services often have Python libraries that simplify sending emails.
* **Testing:** Thoroughly test your script in a development environment before scheduling it to run automatically.

This comprehensive approach should provide you with a solid foundation for building your automated newspaper delivery app. Remember to break down the project into smaller, manageable steps and test each component as you develop it. Good luck!