Email Service

Namazu Elements Core Features · https://namazustudios.com/docs/namazu-elements-core/email-service/

Elements version 3.8+

`EmailService` is a first-class platform service for sending transactional email via SMTP. It is available to platform code through normal Guice injection and to plugin developers through the standard element injection mechanism.

Interface

```
package dev.getelements.elements.sdk.service.email;

public interface EmailService {
    void send(String from, String to, String subject, String body, boolean html);
}
```

Parameter	Description
`from`	Sender address. Pass `null` or blank to use the configured `DEFAULT_FROM`.
`to`	Recipient address.
`subject`	Subject line.
`body`	Message body - plain text or HTML depending on the `html` flag.
`html`	`true` -> `text/html` `false` -> `text/plain`

Throws `InvalidDataException` if SMTP is not configured (i.e. `SMTP_HOST` is blank).

Platform configuration

Configure the platform-level SMTP connection via system defines (ELM attributes or JVM properties). These apply to the entire server and to any code running outside an element context.

Constant key	Default	Description
`dev.getelements.elements.email.smtp.host`	*(blank - disabled)*	SMTP hostname. Leave blank to disable email at the platform level.
`dev.getelements.elements.email.smtp.port`	`587`	SMTP port.
`dev.getelements.elements.email.smtp.starttls`	`true`	Enable STARTTLS.
`dev.getelements.elements.email.smtp.user`	*(blank)*	SMTP username.
`dev.getelements.elements.email.smtp.password`	*(blank)*	SMTP password.
`dev.getelements.elements.email.default.from`	*(blank)*	Default sender address used when `from` is null.

Example (passing as JVM system properties on startup):

```
-Ddev.getelements.elements.email.smtp.host=smtp.sendgrid.net
-Ddev.getelements.elements.email.smtp.port=587
-Ddev.getelements.elements.email.smtp.starttls=true
-Ddev.getelements.elements.email.smtp.user=apikey
-Ddev.getelements.elements.email.smtp.password=SG.xxxxx
-Ddev.getelements.elements.email.default.from=noreply@mygame.com
```

Using EmailService in platform code

Once SMTP is configured, inject `EmailService` anywhere in the platform service layer:

```
import dev.getelements.elements.sdk.service.email.EmailService;
import jakarta.inject.Inject;

public class MyPlatformService {
```

```
    private EmailService emailService;

    public void notifyUser(String userEmail, String message) {
        emailService.send(null, userEmail, "Notification", message, false);
    }

    @Inject
    public void setEmailService(EmailService emailService) {
        this.emailService = emailService;
    }
}
```

For UNSCOPED platform code inject with `@Named(Constants.UNSCOPED)`:

```
@Inject
public void setEmailService(@Named(Constants.UNSCOPED) EmailService emailService) { ... }
```

Using EmailService inside a custom Element

`EmailService` is bound in the platform Guice injector and is visible to element child injectors. Inject it directly - no additional ELM dependency is needed.

```
import dev.getelements.elements.sdk.service.email.EmailService;
import jakarta.inject.Inject;

public class WelcomeEmailService {

    private EmailService emailService;

    public void sendWelcome(String toAddress, String displayName) {
        final var body = "<h2>Welcome, " + displayName + "!</h2>"
                       + "<p>Thanks for joining. Good luck out there.</p>";
        emailService.send(null, toAddress, "Welcome to the game!", body, true);
    }

    @Inject
    public void setEmailService(EmailService emailService) {
        this.emailService = emailService;
```

```
    }
}
```

SMTP is configured at the platform level (see above) and shared across all elements.

Graceful degradation

If `SMTP_HOST` is blank the service logs a warning and any call to `EmailService.send()` throws `InvalidDataException` with the message `"Email service is not configured (SMTP_HOST is blank)."` - no NPE, no silent failure.

This means you can start the server and configure SMTP later without any crash on startup.

> Warning
>
> If `DEFAULT_FROM` is not configured and `null` or blank is passed as the `from` argument, the message will be sent with a blank `From` header. Most SMTP servers will reject this with a `5xx` error. Always set `DEFAULT_FROM` in production, or ensure every call to `send()` provides an explicit `from` address.

See also

- Email Verification - built-in email-based UID verification using `EmailVerificationService`, including token lifecycle, REST endpoints, and custom templates.

Overriding the mail transport (advanced)

To replace the default SMTP implementation entirely - for example, to use a cloud-native sending SDK - rebind `EmailService` in your element's Guice module:

```
public class MyGameElementModule extends AbstractModule {
    @Override
    protected void configure() {
        bind(EmailService.class).to(MyCustomEmailService.class);
    }
```

```
}
```

`MyCustomEmailService` implements `dev.getelements.elements.sdk.service.email.EmailService` and can use any transport mechanism. The binding in the element's child injector takes precedence over the platform binding for all code within that element.

Email Provider Setup

This shows how to configure the `EmailService` SMTP settings for the most common transactional email providers. All providers use the same set of platform configuration keys.

SendGrid

1. Sign in to sendgrid.com and go to **Settings -> API Keys**.
2. Create an API key with at least the **Mail Send** permission.
3. Verify your sender domain or a single sender address under **Sender Authentication**.

```
dev.getelements.elements.email.smtp.host=smtp.sendgrid.net
dev.getelements.elements.email.smtp.port=587
dev.getelements.elements.email.smtp.starttls=true
dev.getelements.elements.email.smtp.user=apikey
dev.getelements.elements.email.smtp.password=SG.<your-api-key>
dev.getelements.elements.email.default.from=noreply@yourdomain.com
```

> The SMTP username is always the literal string `apikey`; the API key itself goes in the password field.

Mailgun

1. Sign in to mailgun.com and select or create a **Domain**.
2. Under the domain's **SMTP credentials**, note the hostname and create or copy an SMTP password for the default `postmaster@<domain>` user (or create a new SMTP user).

3. If your domain is in the Mailgun sandbox, add your recipient addresses to the **Authorized Recipients** list before testing.

```
dev.getelements.elements.email.smtp.host=smtp.mailgun.org
dev.getelements.elements.email.smtp.port=587
dev.getelements.elements.email.smtp.starttls=true
dev.getelements.elements.email.smtp.user=postmaster@mg.yourdomain.com
dev.getelements.elements.email.smtp.password=<mailgun-smtp-password>
dev.getelements.elements.email.default.from=noreply@yourdomain.com
```

> EU-region accounts should use `smtp.eu.mailgun.org` as the host.

Amazon SES (SMTP interface)

1. In the AWS console, verify your sending domain under **Configuration -> Verified identities**.
2. If your account is in the **SES sandbox**, also verify each recipient address, or submit a production access request to lift the restriction.
3. Under **Account dashboard -> SMTP settings**, note the regional SMTP endpoint.
4. Go to **SMTP settings -> Create SMTP credentials** to generate an IAM user and download the SMTP username and password (these are not your AWS access keys).

```
dev.getelements.elements.email.smtp.host=email-smtp.<region>.amazonaws.com
dev.getelements.elements.email.smtp.port=587
dev.getelements.elements.email.smtp.starttls=true
dev.getelements.elements.email.smtp.user=<ses-smtp-username>
dev.getelements.elements.email.smtp.password=<ses-smtp-password>
dev.getelements.elements.email.default.from=noreply@yourdomain.com
```

Replace `<region>` with your AWS region, e.g. `us-east-1`.

> SES also supports port 465 (implicit TLS) and port 2587. Port 587 with STARTTLS is the most broadly compatible choice.

Postmark

1. Sign in to postmarkapp.com and create or open a **Server**.
2. Under **API Tokens**, copy the **Server API token** for SMTP auth.
3. Add and verify your sending domain under **Sender Signatures** or **Domains**.

```
dev.getelements.elements.email.smtp.host=smtp.postmarkapp.com
dev.getelements.elements.email.smtp.port=587
dev.getelements.elements.email.smtp.starttls=true
dev.getelements.elements.email.smtp.user=<server-api-token>
dev.getelements.elements.email.smtp.password=<server-api-token>
dev.getelements.elements.email.default.from=noreply@yourdomain.com
```

> For Postmark, the SMTP username and password are both set to the same server API token.

Generic SMTP server

Any SMTP relay that supports STARTTLS on port 587 (or port 465 for implicit TLS) works without additional changes. Common examples include self-hosted Postfix/Exim relays and corporate mail gateways.

```
dev.getelements.elements.email.smtp.host=mail.yourdomain.com
dev.getelements.elements.email.smtp.port=587
dev.getelements.elements.email.smtp.starttls=true
dev.getelements.elements.email.smtp.user=user@yourdomain.com
dev.getelements.elements.email.smtp.password=<password>
dev.getelements.elements.email.default.from=noreply@yourdomain.com
```

For port 465 (implicit TLS) set `starttls=false` - the underlying Jakarta Mail session will use `smtps` transport automatically.

Passing settings at startup

Settings can be provided as JVM system properties or as environment variables. For containers the environment variable form is usually more convenient:

JVM property	Environment variable
`dev.getelements.elements.email.smtp.host`	`DEV_GETELEMENTS_ELEMENTS_EMAIL_SMTP_HOST`
`dev.getelements.elements.email.smtp.port`	`DEV_GETELEMENTS_ELEMENTS_EMAIL_SMTP_PORT`
`dev.getelements.elements.email.smtp.starttls`	`DEV_GETELEMENTS_ELEMENTS_EMAIL_SMTP_STARTTLS`
`dev.getelements.elements.email.smtp.user`	`DEV_GETELEMENTS_ELEMENTS_EMAIL_SMTP_USER`
`dev.getelements.elements.email.smtp.password`	`DEV_GETELEMENTS_ELEMENTS_EMAIL_SMTP_PASSWORD`
`dev.getelements.elements.email.default.from`	`DEV_GETELEMENTS_ELEMENTS_EMAIL_DEFAULT_FROM`