# Purpose

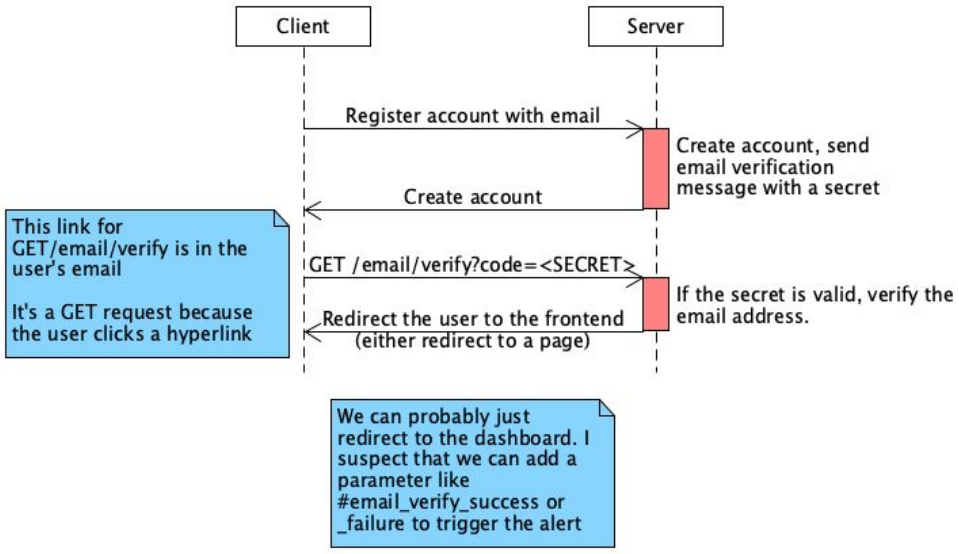
This is an internal design/bootstrap document that has a few purposes

- Explain the feature to a developer that has near-zero knowledge of specific tools and technologies used to create this feature
- Settle on a high-level design for the feature in Affordable
- Provide a framework for said developer to begin work on the feature

This document in particular is focused on implementing email validation in Affordable.

# Backend

A typical flow for email validation looks something like this:




## API

The registration API can remain unchanged
We would need to add an endpoint like GET /email/verify with a query parameter "code" that will carry the secret. This endpoint will return a redirect (probably 302) to a frontend URL.

## Implementation

On the backend, the method AuthenticationService.registerUser should be altered to send the email verification message.

Currently the code to add an email address to the database looks like

```
registerUser {
	// …
	// What happens now
	await this.emailDao.createEmailRecord(<EmailRecord>{
		user: newUser.id,
		email: newUser.email,
		primary_ind: true,
		verified: false
	});

	// instead do
	await this.emailService.addEmail(newUser.id, newUser.email, true);

	// ...
}
```

In this case, I recommend creating a new class, EmailService, to handle registering, storing and validating email addresses. (There already exists an AffordableEmailService for actually sending emails. I recommend renaming this to AffordableEmailClient) This class could look something like this:

```
class EmailServiceImpl implements EmailService {
	// ...
	public async addEmail(userId, emailAddress, isPrimary): Promise<EmailRecord> {
		await this.emailDao.createEmailRecord(...);
		// A random uuid is probably cryptographically secure enough
		const secret = uuidv4();
		const hyperlink = BACKEND_URL + `/email/verify?code=${secret}`
		// There exists a usable message body already in the existing
		// email verification service. Just change the link.
		const messageBody = ...hyperlink...
		await this.emailClient.sendVerificationEmail(messageBody);
	}

	/**
	 * @returns URL to redirect to
	 */
```

```
	public async verifyEmail(secret: string): Promise<string> {
		let emailRecord;
		try {
			emailRecord = await this.emailDao.getEmailRecordBySecret(secret);
			emailRecord.isVerified = true;
			await this.emailDao.update(emailRecord);
		} catch (error) {
			return FRONTEND_URL + "#email_verify_failure"
		}
		return FRONTEND_URL + "#email_verify_success"
	}
}
```

Some notes about this implementation approach

- Secret needs to be stored in the database. I see no reason why this couldn't be added as a column to the emails table, but there may be another preferable approach.
- The constants FRONTEND_URL and BACKEND_URL need to be set somewhere. I think the desired approach would be as environment variables (we already use the library dotenv). At this point, I think it would be acceptable to hard code these as constants with values "localhost:3000" and "localhost:4000", to save time.
- At the end of the verify email call, we need to redirect the user to the appropriate frontend link. I don't know exactly how to do this in Express, but am happy to help if needed.

# Frontend

Immediately, we can clean up the registration page and delete the security questions that trigger email verification.

One simple way to provide feedback to the user about their email verification is to redirect the user to the dashboard (or whatever the redesigned front page should be), and have anchor links trigger alerts. The user could be redirected to "localhost:3000/dashboard#email_verify_success" or "failure" and that would trigger the appropriate alert (using sweetalert, which is already used throughout the React app). I don't personally know how to do this in React, but am happy to research/assist as needed.