Twine Software Design Document

Steeve Joseph

December 13, 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose

Currently, a problem with dating sites are the lack of genuine connections. Using heteronormative examples, for men the issues include:

- Suffering from poor picture quality
- Lack of ability to leverage vocal tonality, body language, and eye contact

Issues that women face on these apps listed below:

- Safety concerns
- An overwhelming number of suitors to choose from
- An apparent lack of substance from said options

1.2 Scope

The scope of this project thus far is to be used as a way to allow men and women to connect based on genuine attributes, spearheaded by the use of content uploaded by trusted friends.

Specifically, the product would allow friends of a user to upload content such as videos, voice recordings, pictures that describe how the friend feels about the user, what the friend thinks, etc.

1.3 Overview

A representative use-case for the product is given below:

1. The user signs into Twine with their email address and password.
2. The user defines a list of people that they would like to get references from.
3. The references receive an email asking them to upload content onto the requester's profile.
4. The reference uploads the content.
5. The reference sees profiles that may be good fits for the requester and makes recommendations.
6. The recommendee's references get notifications that one of their friends may have a match.
7. The recommendee's reference confirms the mutual fit.
8. The user and the match can now message each other.

1.4 Reference Material

1.5 Definitions and Acronyms

Term	Definition
MVP	Minimum Viable Product
LDA	Leading Dating apps
MS	Matrimony Sites
UGC	User Generated Content

Table 1.1: Table of Definitions and acronyms used

Chapter 2

System Overview

2.1 Similar Software

Currently, similar products in this space include:

- Tinder/Bumble
- Hinge
- Omegle
- Various Matrimony Sites (MS)

Comparison of Twine with above sites

- Tinder/Bumble
  - Twine offers more information on long term compatibility than Tinder.
  - Twine removes the dehumanizing swipe aspect.
- Hinge
  - Twine adopts Hinge features like prompts on UGC, including voice responses
  - Twine brings voice prompts and communications to the front
- Omegle
  - Twine allows users to video chat in a similar way to Omegle
  - However, Twine allows the video chat only after there has been a mutual match, as determined by friends of both users
  - Twine also deprioritizes the text based chat aspect

Chapter 3

System Architecture

3.1 API

3.1.1 Backend Language

ExpressJS was chosen to be the backend language of choice for Twine . Advantages:

- written in JS
- simple, flexible, extensible
- "One language to rule them all"

3.1.2 Authentication

PassportjS will be used for Authentication.

3.1.3 Routes

At the very least, the necessary routes are:

- Login: Allows users to register/log in
- Add content: Allows friend to upload content on user's behalf and (potentially) allows user to upload their own content
- Add friend: Allows a user to add a friend, meaning that the friend will be able to post content on the user's behalf
- Make Recommendation: Allows a friend to make a recommendation on behalf of the user

3.1.4 External APIs/Libraries

Notable APIs that will be used server-side include:

- Imgur API: used to handle uploading of images, videos, audio
- Nodemailer: Used to send emails to users
- PeerJS: used to establish video/audio calls

3.2 User Interface

ReactJS will be used as the UI framework Twine will primarily be web-based. Advantages

- Large community support
- Flexible and extentsible
- Allows for rapid development/iteration

Chapter 4

Data Design

4.1 Database Design

MongoDB is used for Twine , using MongoDB Atlas as the database provider.
The database currently looks like this:

Figure 4.1: Twine ERD

4.2 User Model

The User model will look like this:

Figure 4.2: Twine user model

4.3 Data Interaction

Twine will read and write to a MongoDB database via an API written in ExpressJS. The API will communicate with a ReactJS frontend.

Chapter 5

Component Design

Chapter 6

User Interface Design

6.1 UI Overview

6.2 Screenshots

6.3 Screen Objects and Actions

Chapter 7

Requirements

Chapter 8

Milestones

Chapter 9

Testing

9.1 API Testing

9.1.1 Postman

Postman was chosen as the API testing framework, for its balance between functionality, and freeness. The main features it brings are the use of **Collections**, used to run multiple requests (and their coresponding tests) simultaneouslty. Similarly, fields from the response of one request can be used in another request via chaining.

9.2 Unit Testing

9.2.1 Jest

Jest was voted the #1 testing framework, according to The 2020 State of JS Developer Survey. Other options under consideration were KarmaJs, MochaJS, and Jasmine.

9.3 End to End Testing

9.3.1 Playwright

Playwright was chosen as the end-to-end tool, for its support, its ability to run headless, and its ability to accept config as code. Other options under consideration were Squish Tools, Selenium, and Pupeteer (with its Jest integration)