

Boop Design

JIB Team 0318

Grace Rarer
Jiale Zheng
Gunnar Andra-Thomas
Sanjana Jampana
Juan Caicedo
Porter Hunley

Client: Professor Stallworth and Dr. Shelley

Table of Contents

Boop Design	1
Table of Contents	2
Table of Figures	2
Terminology	3
Introduction	4
System Architecture	5
Data Storage Design	9
Component Design Detail	12
User Interface Design	15
References	23

Table of Figures

Figure 1: Static System Architecture	6
Figure 2: Dynamic System Architecture for Accepting a Friend Request	7
Figure 3: Dynamic System Architecture for Spontaneous Push Notifications	8
Figure 4: Database Entity Relationship Diagram	9
Figure 5: Static Component Design Diagram	13
Figure 6: Dynamic Component Design Diagram	14

Terminology

Angular: a frontend framework created by Google for creating web-apps in TypeScript.

Bcrypt: the password hashing and validation algorithm used by Boop.

CRUD: the four basic actions of most data-access APIs are create, read, update, and delete.

ExpressJS: the NodeJS web-server library used for Boop's backend.

Gravatar: a service that connects your email address to your profile image across different websites and apps.

NodeJS: a runtime environment that allows server-side software to be written using JavaScript, or languages in the JavaScript ecosystem such as TypeScript.

PostgreSQL: an open-source implementation of the SQL relational database standard.

Progressive Web App (PWA): A website that uses a Service Worker script to behave like a native application. PWAs use caching so they do not have to reload the entire webpage for every session, and support Push Notifications.

RESTful API: web service that utilizes the Representational State Transfer (REST) architecture to handle requests on the frontend.

Web Push Notification: push notifications sent to a user's web browser.

Universally Unique Identifier (UUID): a set of standards for producing identifiers or primary keys that are intended to be globally unique.

Introduction

Background

Our project is Boop, the digital equivalent to crossing paths with a friend in the dining hall. We aim to have users utilize our platform to encourage spontaneous social messaging in their lives. Users registered to the service will receive spontaneous push notifications, reminding them to talk to a friend, encouraging a friendly, lively environment. Other features include profile creation and editing and control over notifications. We utilize a Progressive Web App (PWA) to present this platform on both desktop web browsers and mobile devices. We are hopeful that this idea will help users feel just a bit more connected in this increasingly online world.

Document Summary

The **System Architecture** section will highlight the static and dynamic elements of the Boop application, providing context into the structure and relationship between all components available.

The **Data Storage Design** section will describe the database section of our app, which consists of PostgreSQL integration and stores profile and account information, as well as other features.

The **Component Design Detail** section will show a more detailed glance into the components of our design, highlighting in greater depth the relationships between all available features.

The **UI Design** section presents the current state of the User Interface, and details the visual design decisions to make the app experience seamless.

System Architecture

Introduction

We have included some diagrams explaining the components of our system architecture and showing examples of how these components work together in some of Boop's features.

Most of the features of Boop are user-initiated; users open our web-app to update information about themselves, manage their friends lists, and view information about their friends. However, Boop's headline feature, push notifications prompting users to chat with their friends, occurs spontaneously when the backend server sends a push notification without being prompted by a user's actions.

Static System Architecture

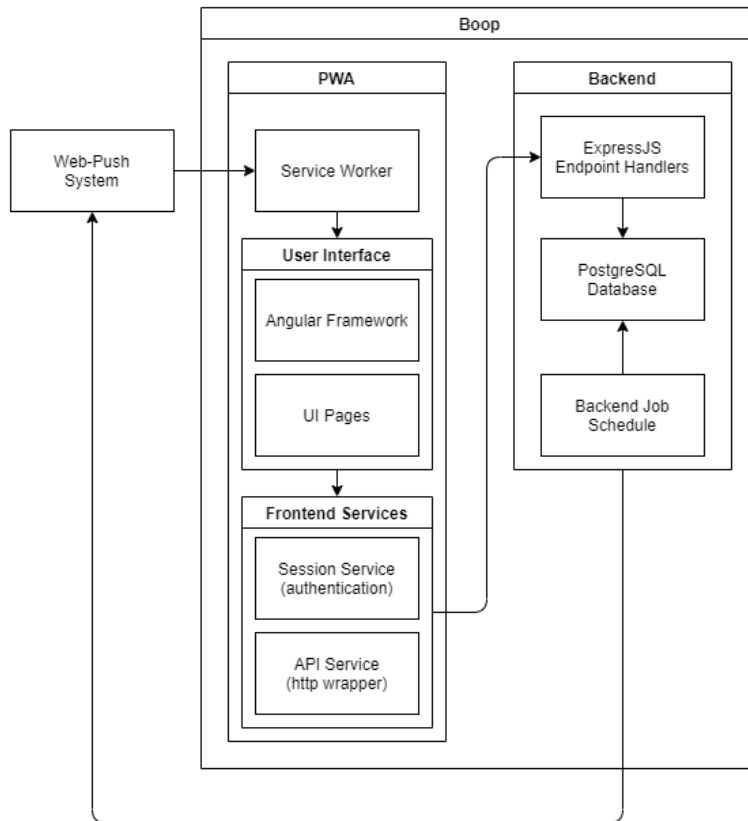


Figure 1: Static System Architecture

The frontend of our application is a Progressive Web App (PWA), meaning that it has a service worker script that is always running, even when the app is closed. This service worker improves performance by caching the application so that the browser does not have to download the app every time the user visits the page, but more importantly, the service worker is always available to receive push notifications [1]. The visible parts of our user interface are components created with Angular, and behind the scenes we have custom frontend services that wrap HTTP requests on behalf of the UI components and automatically handle authentication of each request. The frontend communicates with the backend through a RESTful API by sending JSON objects over HTTP.

HTTP requests made by the frontend are handled by our backend server using the ExpressJS framework. The handler functions rely on a Postgres database to store and retrieve information in response to the HTTP requests. In addition to handlers triggered by HTTP requests, the backend also has a cycle of scheduled jobs that it repeats every few minutes. This includes clearing out expired tokens from the database, and is also how we schedule the sending of spontaneous push notifications.

Push notifications are sent using the web-push protocol. Our server sends a request to the web-push service created by the organization that created the user's browser, which then forwards the push to the user's device. The service worker is able to receive these notifications, show them to the user, and open the app when the notification is clicked.

Dynamic System Architecture

This first sequence diagram shows how the user can check for and respond to friend requests. This is a straightforward application of CRUD principles (create, read, update, delete), with the backend server being an intermediary between the web-app and the database. Services on the frontend and Express handlers on the backend server act as intermediaries between the user interface and the database.

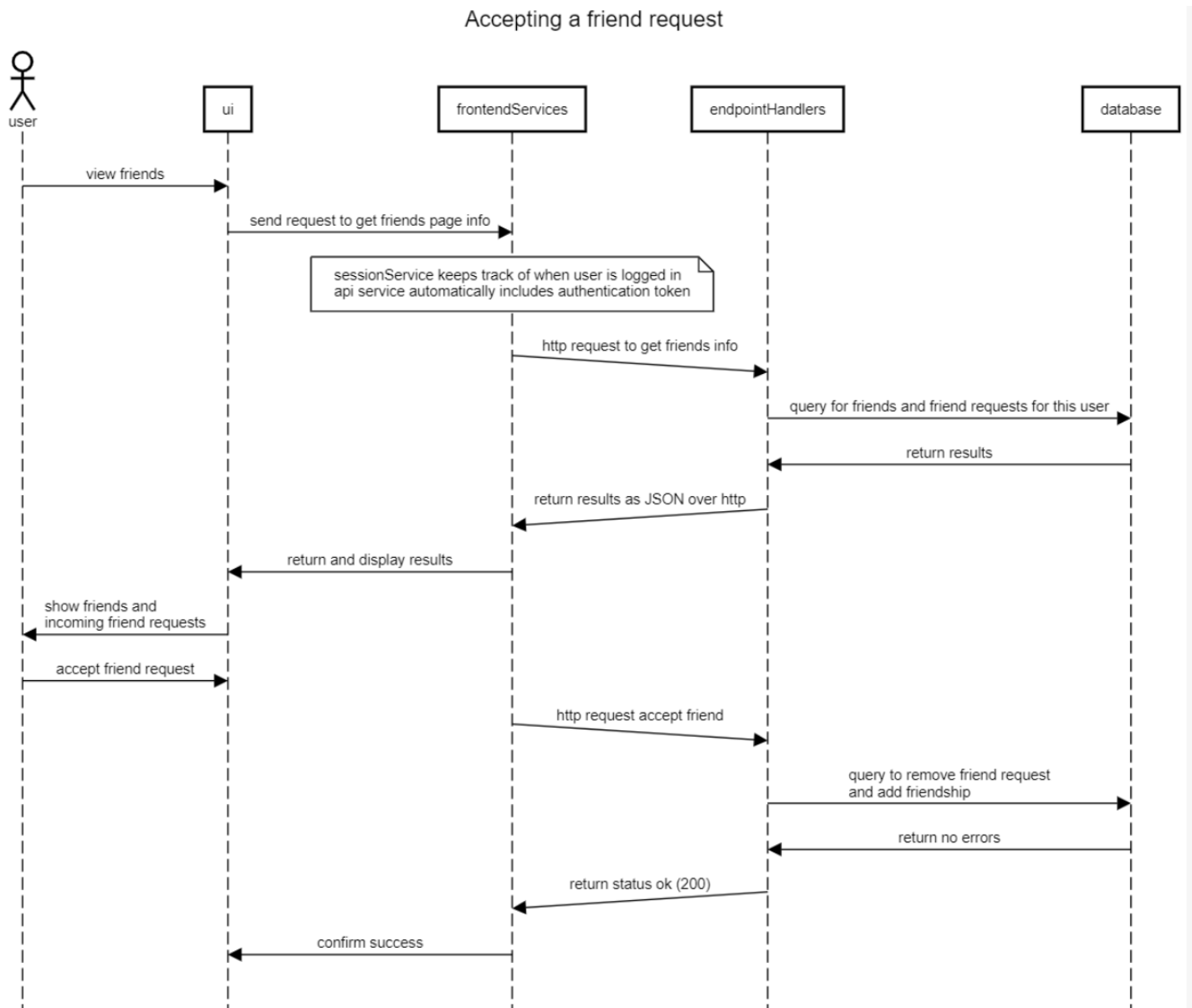


Figure 2: Dynamic System Architecture for Accepting a Friend Request

This second sequence diagram shows the process of spontaneous push notifications, which are started by a scheduled job rather than by a user action.

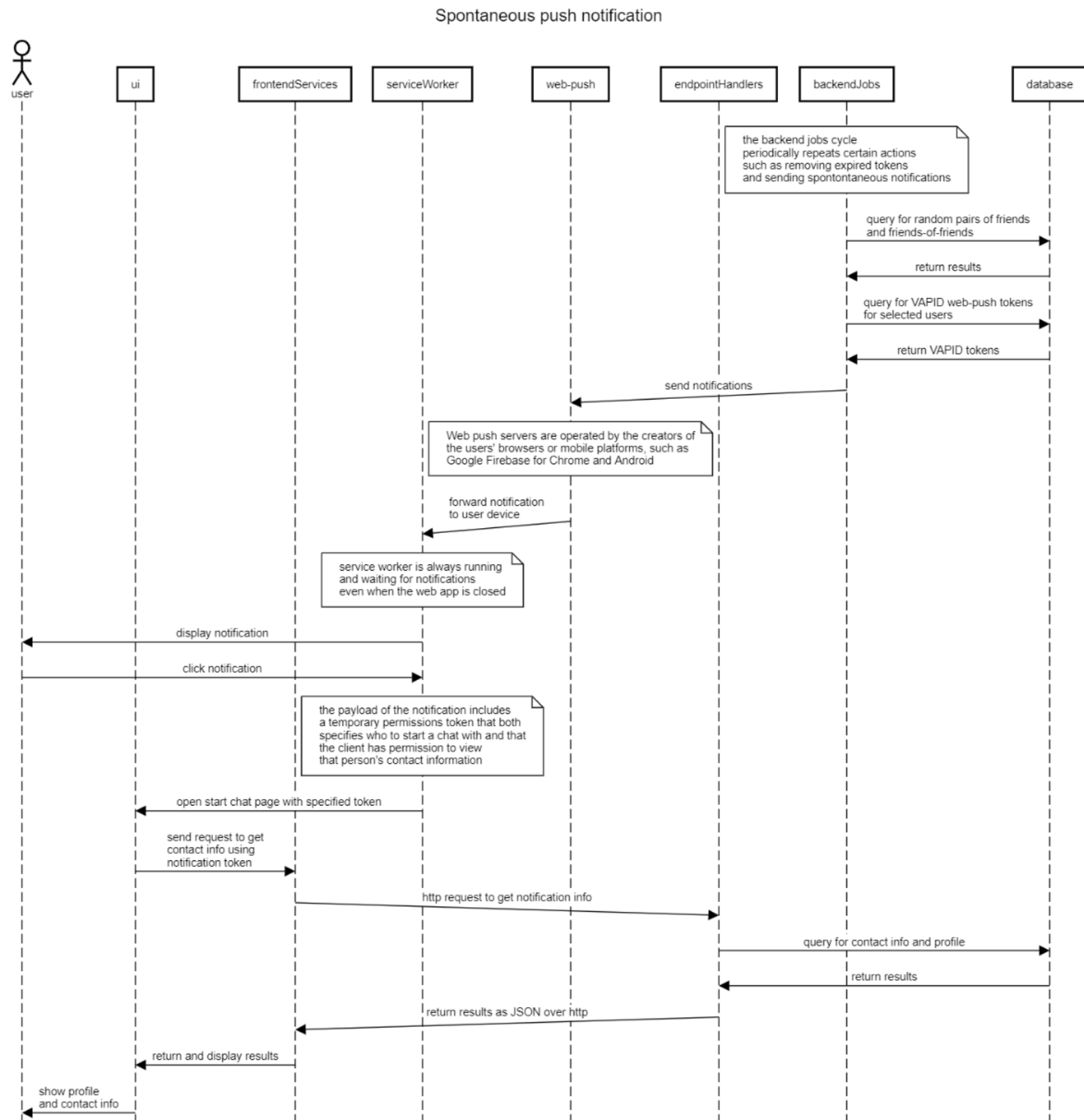


Figure 3: Dynamic System Architecture for Spontaneous Push Notifications

Data Storage Design

Introduction

Although Boop is all about communication, we do not host users' conversations on our own platform. Boop's backend server only has to keep track of users' information and the friendships between users. To store this data, we use a PostgreSQL database.

Entity Relationship Diagram

This diagram shows the schema used by Boop's PostgreSQL database.

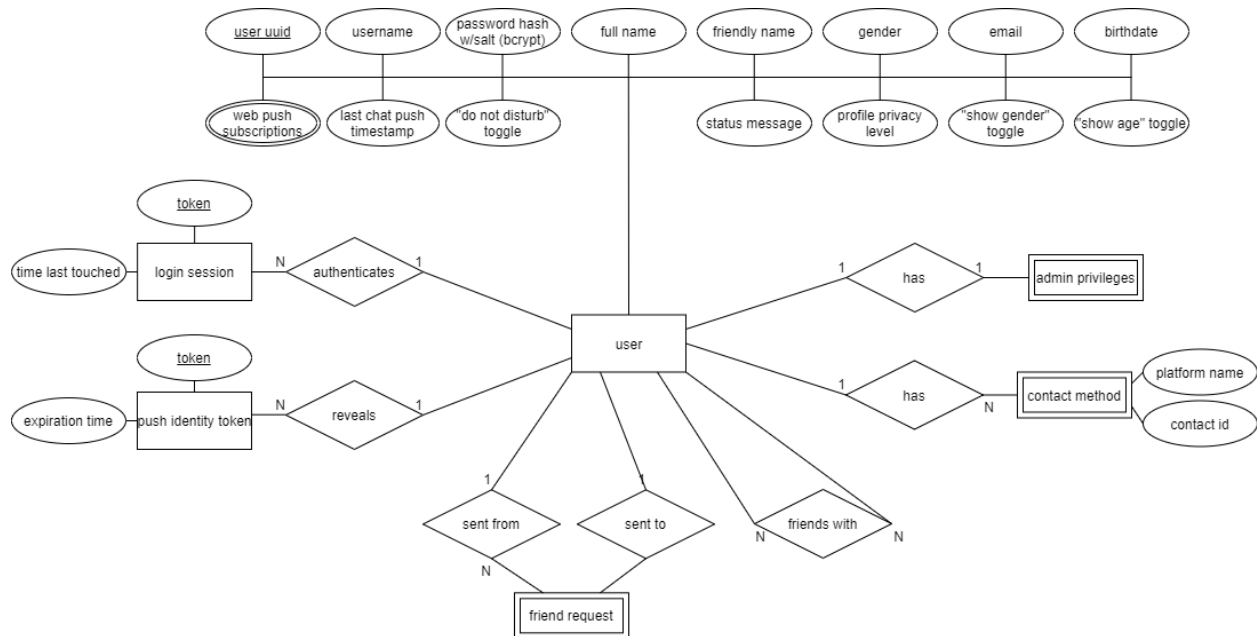


Figure 4: Database Entity Relationship Diagram

Since users are allowed to change their username, a universally unique identifier (generated with UUID v4) is used as the user's permanent primary key.

Since web push notifications are always sent to all of the devices associated with a given user, we always retrieve a user's notification subscriptions as a group rather than querying for just one. Therefore, web push subscriptions are represented as a multi-value column of JSON objects rather than as separate entities.

Our implementation of the database denormalizes the "friends with" relationship by representing each friendship as two rows. For example, if Alice is friends with Bob, we would have one row for "Alice is friends with Bob" and another row for "Bob is friends with Alice". We must create or delete both rows at the same time to maintain consistency, but this duplication makes it faster and easier to treat the network of friendships as an undirected graph, such as when calculating the "friends of friends" data.

Login session tokens are used to both identify and authenticate users who are logged in. Push identity tokens are temporary tokens that only allow a client to view the "start chat" page for a particular person. For both of these tokens, the keys are based on universally unique identifiers.

Files

Our backend does not store any files. User profile images are processed, stored, and served by Gravatar, a third-party service, so when we display an image the backend only needs to generate the Gravatar URL using a user's email address [2]. The frontend asset bundle will include images such as the Boop logo and the icons for popular social messaging platforms.

Europe's GDPR and other data privacy laws include a "right to data portability" which requires that users should be able to export their data in a format that can be easily understood [3]. To comply with this requirement, we give users the ability to download their information as a JSON file. This file is generated on-demand using information in the database.

Data Exchange

Our frontend and backend communicate via a RESTful API using JSON over HTTP. Both the Angular frontend and ExpressJS backend use middleware that automatically converts data to and from a JSON representation. The frontend and backend are both written in TypeScript and both depend on a separate NPM package called boop-core that provides them with shared type definitions for the objects that are sent through the API.

For example, here is the type definition for the request and response objects for logging in:

```
1  export type LoginRequest = {
2    username: string;
3    password: string;
4  };
5
6  export type LoginResponse = {
7    userUUID: string;
8    sessionToken: string; // used to verify identity when making requests to backend
9  };
10
```

You can see all of these shared type definitions under core/src/datatypes in our GitHub repo [4].

We also use a wrapper around our backend ExpressJS handlers so that exceptions thrown by backend endpoints can be automatically caught, transmitted with an appropriate HTTP error code, and handled by the frontend.

Security

Security best practices require that passwords never be saved in plain text; instead, they should be validated using an approach called "hashing and salting" [5]. This is where a random component called a salt is used to produce a different password hash for each user even when two users have the same password. We use the "bcrypt" password-hashing algorithm. The database stores the bcrypt "hash", which is a combination of the password hash and the salt as one string, and this hash is used to validate the password when a user logs in. Bcrypt's unique trait is that it is intentionally slow in order to prevent attackers from being able to easily crack password hashes [6]. By being a relatively expensive hashing algorithm that can be made even more expensive as computing hardware becomes more powerful, bcrypt is a future-proof defense against brute-force hashing attacks as well as "rainbow table" attacks.

When a user logs in, a session token is stored in the database and sent back to the client. The client stores this token in local storage on the user's browser and automatically transmits the token as an HTTP header when making API requests to our backend. The server compares this token to the sessions stored in the database to determine which user made the request. Sessions expire if they are not used for a long period of time (currently 30 days).

The "start chat" page required a different authentication system, since users might receive and open a Boop notification while not logged in to the app. When these notifications are sent, we create a special temporary token that allows the client to view the "start chat" page for a specific user, and this token is transmitted in the metadata of the push notification, which is encrypted before being sent to the web-push servers.

Component Design Detail

Static Component Diagram

This diagram shows the static relationships between the components of our web-app frontend and Node backend. On the frontend, most major components of the UI are standalone screens, with the Angular Router determining which screen to show based on the current URL. The controller code for these UI screens accesses data from the backend via the API Service and Session Service, which automatically handle authentication for logged-in users.

On the backend, the ExpressJS server that handles requests from the frontend is divided into units called Express routers, which consist of handler functions for specific API endpoints. The Express app and its handler functions are stateless, with all data being stored in our Postgres database. A few tasks that are not based on user input, such as removing expired tokens from the database and sending spontaneous push notifications, are scheduled to repeat every few minutes.

Since Boop does not use an object-oriented design architecture and does not have a class-based data model of interacting objects, we chose not to use a standard UML diagram. The diagram below shows the connections between the functional components of our application, but most of the backend components are stateless collections of functions rather than true objects. For simplicity and legibility, this diagram omits smaller sub-components, such as loading bars and confirmation dialog pop-ups that are used as shared components of UI screens, or SQL query scripts that are used by multiple Express routers.

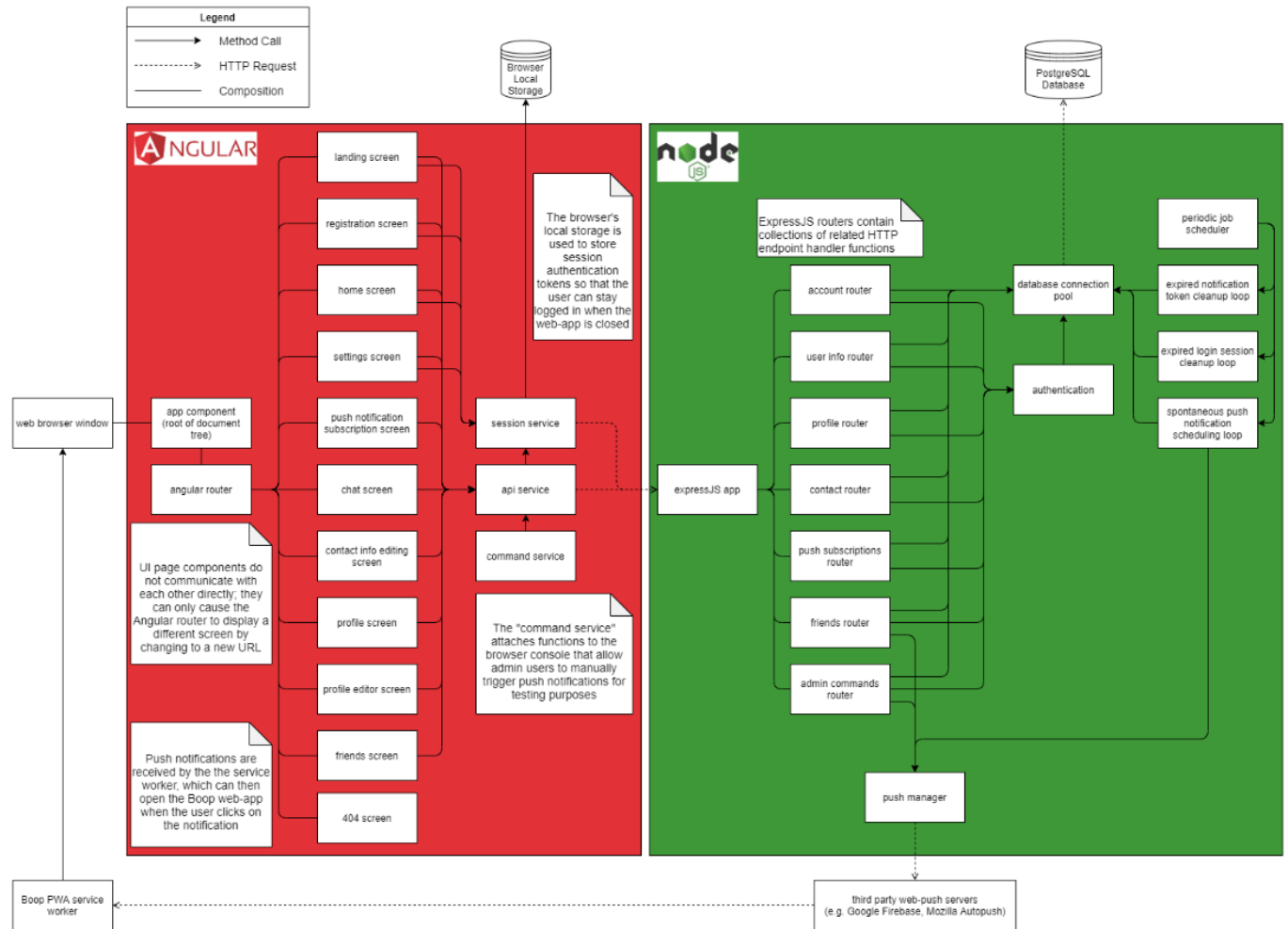


Figure 5: Static Component Design Diagram

Dynamic Component Diagram

When we develop features that are based around user interaction, we can safely ignore several layers of indirection and middleware to get a clearer picture of the flow of information. For example, although the static diagram shows that most UI screens do not directly communicate, it is more useful to think about the user interface in terms of links between pages rather than thinking about the implementation details of the Angular routing.

The diagram below depicts components related to viewing and editing user profiles and shows how these components interact from the perspective of user-driven interactions. For example, a user who is logged into the home screen could update their status message, navigate to their own profile, edit their profile description, navigate through their friends lists to see other users' profiles and send a friend request to another user who would be notified by a web-push notification. These pages form a vertical slice of our application that represents how most of Boop's features function.

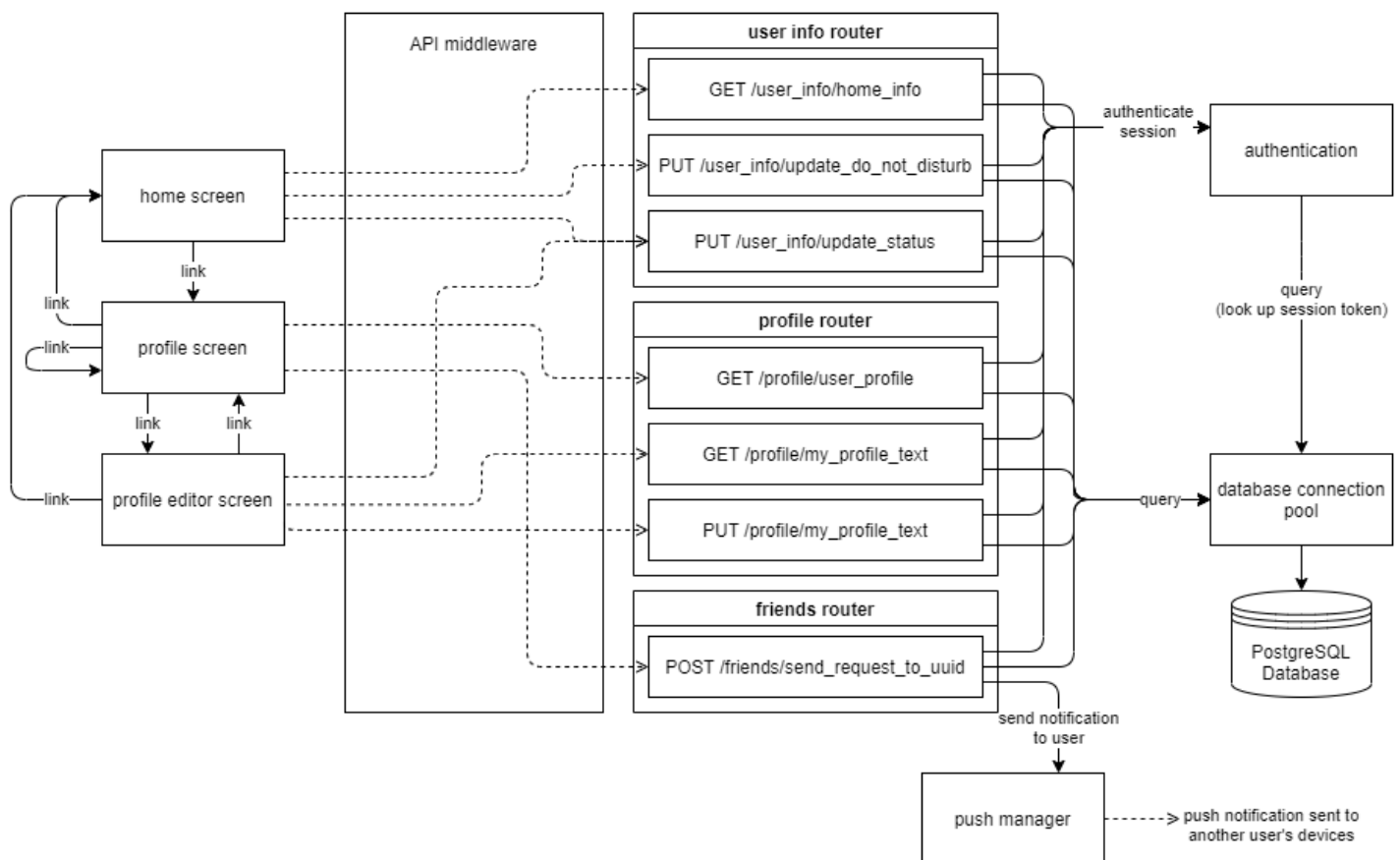


Figure 6: Dynamic Component Design Diagram

User Interface Design

Introduction

We created the user interface for Boop with the Angular framework and Google's Material Design component library. From the start, we designed the layout of our screens to work well at the size and aspect ratio of mobile devices. Our HTML is designed to work well on full-sized computer monitors as well, but Android devices are our primary target.

Toolbar

The toolbar is visible at the top of every Boop page. Clicking on either the Boop logo or the "home" icon takes the user to the home page if they are logged in, or to the landing page otherwise. We chose to use an orange color for the toolbar as well as on our favicon and other brand materials. In color psychology, orange is considered a "social color, making people open up and enhance their communication" [7]. This matches the friendly and playful feeling that we hope users will associate with Boop.

Landing Page

The landing page is the first thing that new visitors will see. Here, users can sign in or begin registering an account. The "Learn More" button opens a dialog box containing the Boop FAQ, which explains Boop's main features and how to use them.

The image displays two side-by-side screenshots of the Boop user interface. Both screenshots feature an orange toolbar at the top with the Boop logo and a home icon. The left screenshot shows the landing page with the text "Boop sends notifications to you and your friends to encourage spontaneous social interaction online." and a "Learn More" link. Below this is a sign-up section titled "Sign up to receive Boops!" with input fields for Username, Password, and Confirm Password, and a "Sign Up" button. Below the sign-up section is a login section titled "Log In" with input fields for Username and Password, and a "Log In" button. The right screenshot shows the same landing page but with a "Boop FAQ" dialog box open. The dialog box contains the title "Boop FAQ" and two sections: "What does Boop do?" and "What's the point of spontaneous reminders?". The "What does Boop do?" section explains that Boop sends notifications to a pair of friends to remind them to check in on each other, and that it will also occasionally send notifications to friends-of-friends. The "What's the point of spontaneous reminders?" section explains that shared physical spaces are full of spontaneous opportunities to meet people and catch up with friends, and that catching up and exchanging small-talk helps you form connections. The dialog box also has a "Close" button at the bottom.

Boop sends notifications to you and your friends to encourage spontaneous social interaction online.

[Learn More](#)

Sign up to receive Boops!

Username

Password

Confirm Password

Sign Up

OR

Log In

Username

Password

Log In

Boop FAQ

What does Boop do?

At random intervals, Boop sends notifications to a pair of friends to remind them to check in on each other. When you click on a Boop reminder, you can see a list of your friend's contact info or usernames for social messaging apps like WhatsApp, Messenger, Discord, and Snapchat, so you can use those platforms to start a conversation.

Boop will also occasionally send notifications to friends-of-friends, helping you connect with other people in your social circle who you might know or want to be introduced to. If you want, you can disable friend-of-friend notifications by setting your privacy level to "friends only".

What's the point of spontaneous reminders?

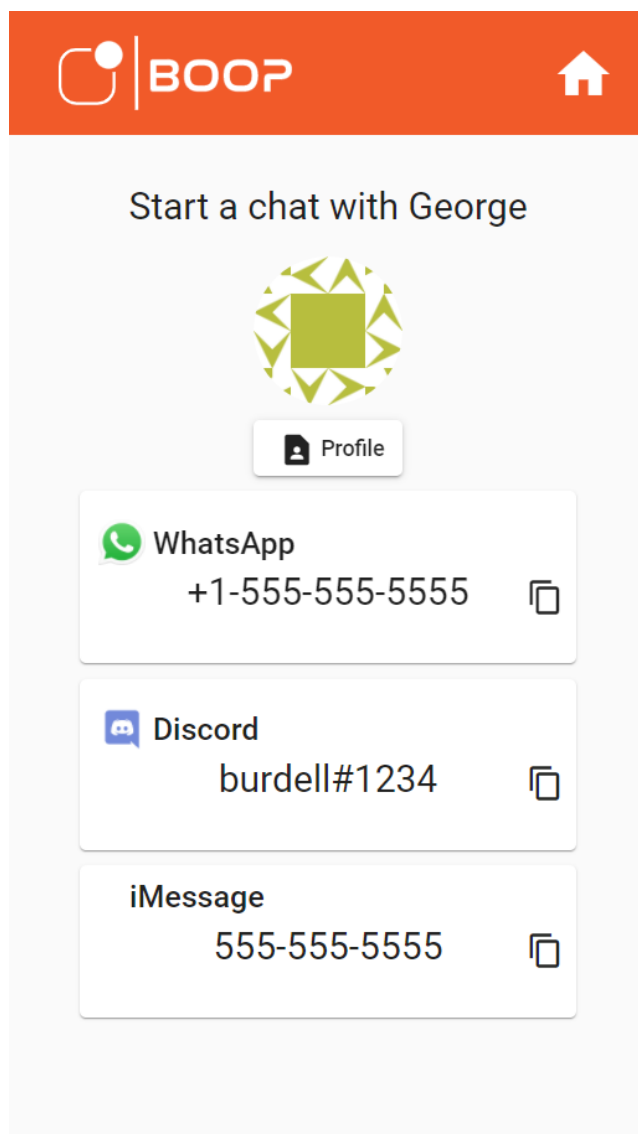
Shared physical spaces are full of spontaneous opportunities to meet people and catch up with friends. When you see your classmate eating in the dining hall or cross paths with your friend on the sidewalk, catching up and exchanging small-talk helps you form connections.

Close

Chat Page

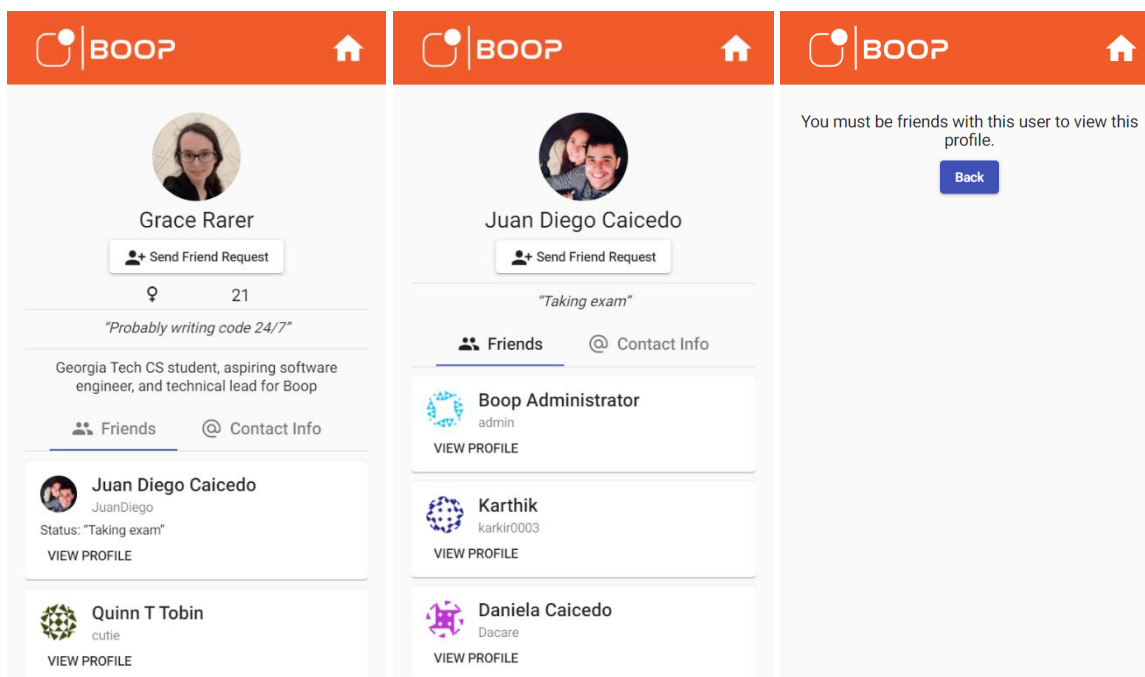
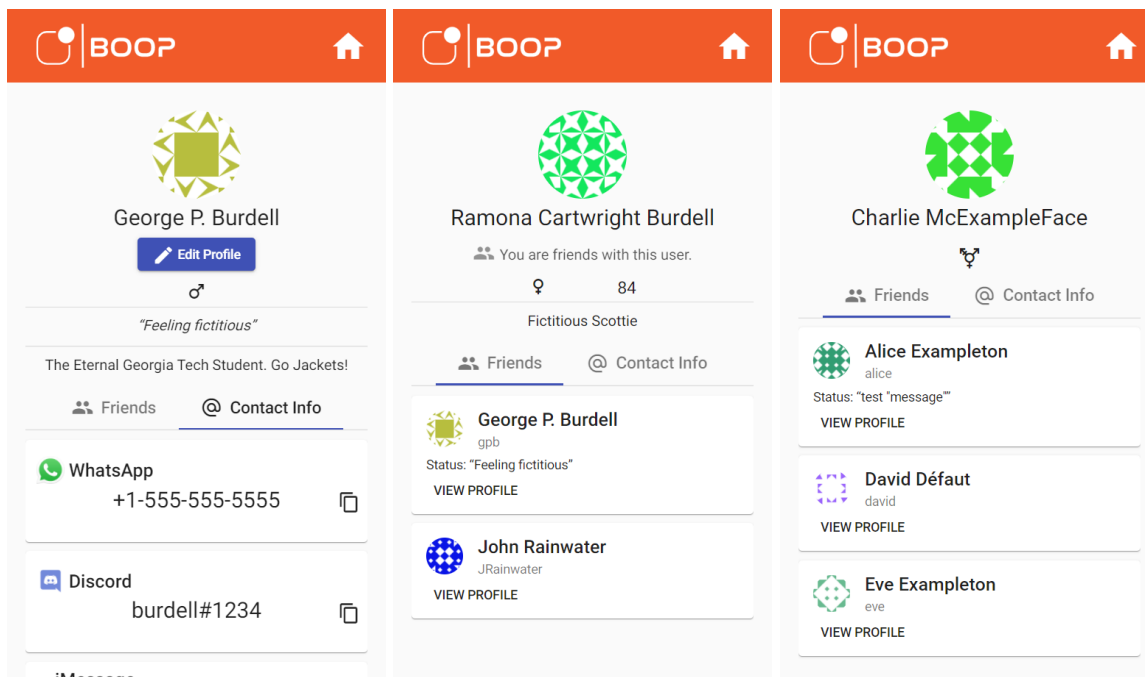
Boop reminder notifications prompt users to chat with a specific friend or friend-of-friend. The page shows that person's profile image and a link to their profile alongside a list of the contact methods that can be used to start a conversation with them.

When we first created this page, we experimented with “deep-linking” into chat platforms’ web clients so that clicking on a contact method would open a chat with that user on that platform. This worked in some cases, but not reliably. Each chat platform has a different format for these links, and many do not support deep links at all. Even for those that do support them, they may be unreliable if users provide their contact info in an inconsistent format, which is common if the platform uses phone numbers. Since it would not be possible to provide a consistent support for deep links across different chat platforms, we opted for a simpler design where clicking on a contact method copies that contact information to the user’s clipboard.



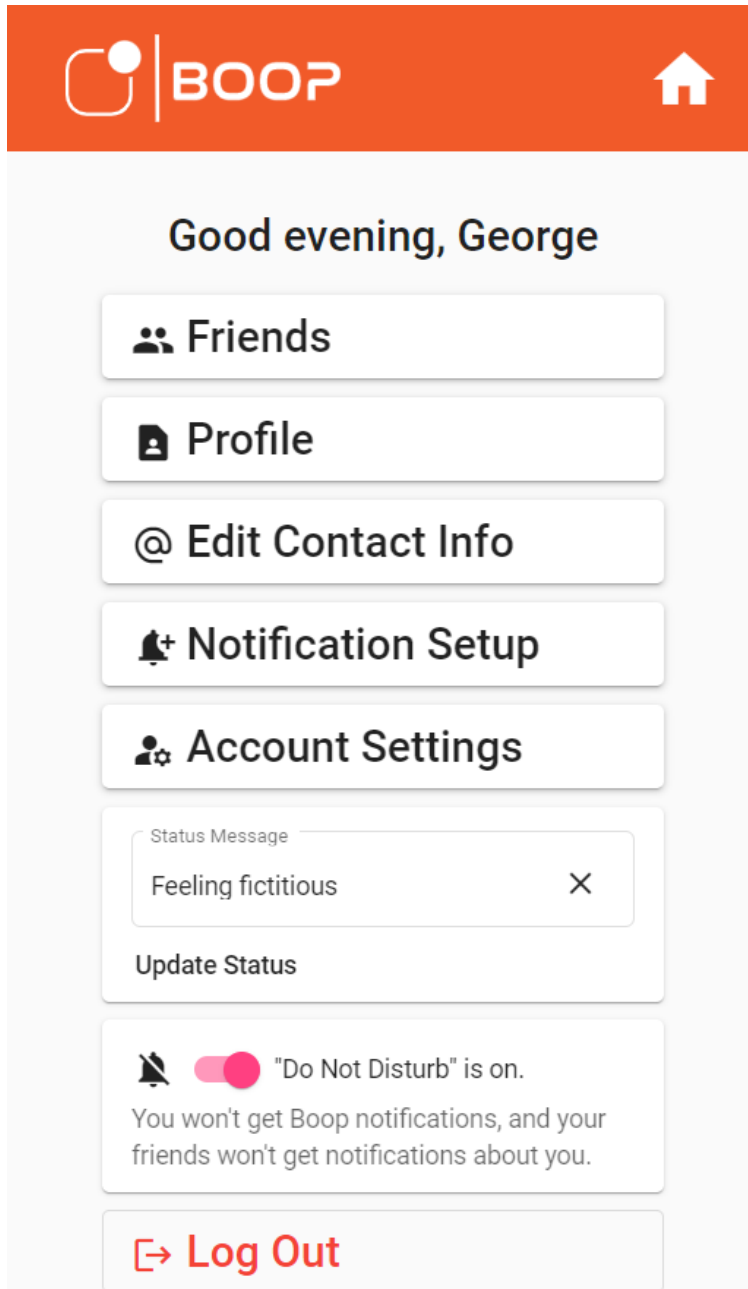
Profile

Profile pages are a way for users to introduce themselves and connect with potential friends in their social circles. Users can control which information is shown on their profile and who can view it. Profiles may include the profile image, gender, age, status message, and profile description. Users whose profile privacy is set to “everyone” can be viewed by users who are not logged in by navigating to `boopboop.app/profile/<username>`.



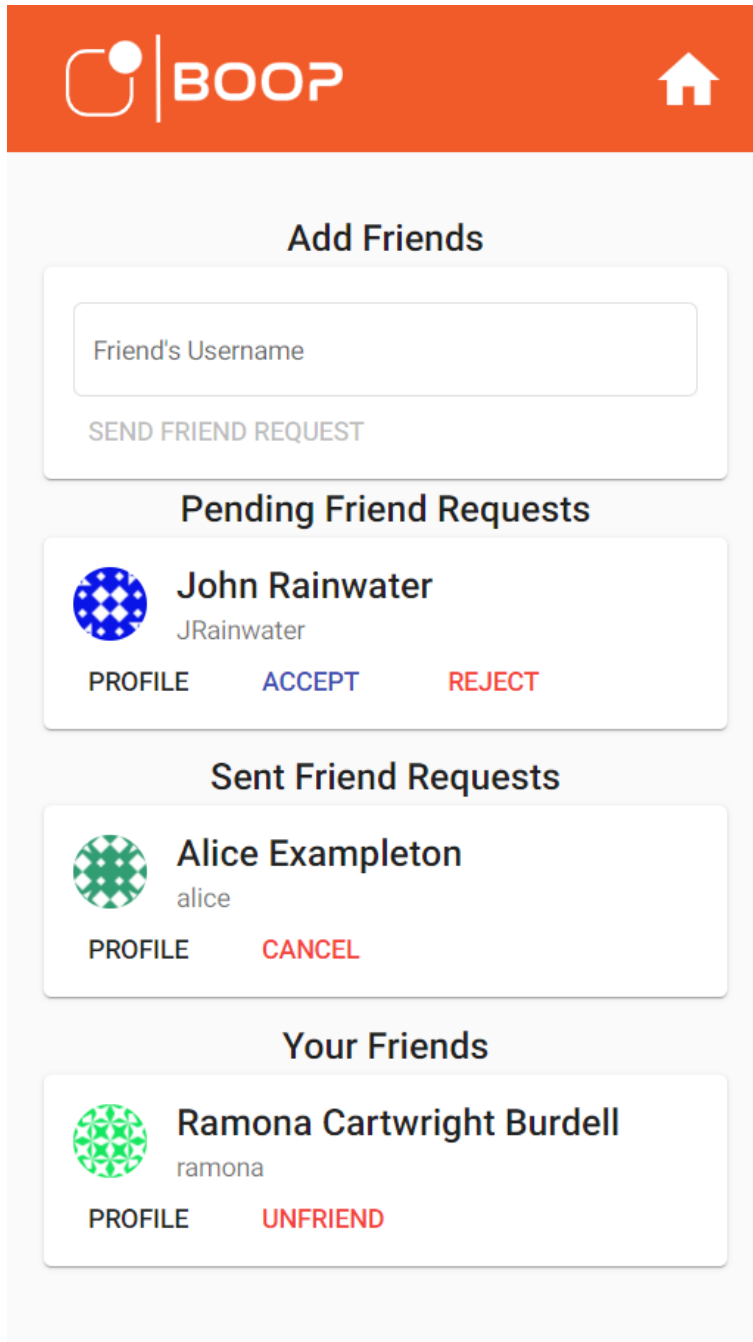
Home Page

The home page is the first thing that a logged-in user will see after opening Boop. The greeting at the top of the home page changes depending on the time of day in the user's time zone. In addition to linking to other screens, the home page contains the controls for the user's status message and "do not disturb" settings, since we want it to be easy for users to quickly change these.



Friends

This page allows users to view their friends list as well as send and receive friend requests. Note that the example users in the screenshot below all have the default “identicon”-style profile images because they do not have profile images set with Gravatar.



“Edit Contact Info” and “Edit Profile”

These pages allow users to change the information that is shown on their profile.

The dropdown list for chat apps shows the most common chat platforms, but users can also choose the “other” option and provide the name of any chat platform even if it is not in our list of options.

The image displays three screenshots of the Boop app's user interface, specifically the 'Edit Contact Info' and 'Edit Profile' sections.

The top row shows three instances of the 'Which chat apps do you use?' form. Each form has a title, a subtitle 'Let your friends know how to start a conversation with you', and a list of chat platforms. The first form shows 'WhatsApp' selected, with a contact ID of '+1-555-555-5555'. The second form shows 'Discord' selected, with a contact ID of 'burdell#1234'. The third form shows 'Other' selected, with a platform name of 'Example App' and a contact ID of 'Example Username'. Each form has a 'Save Changes' button at the bottom.

The bottom screenshot shows the 'Edit Profile' page. It has a title 'Edit Profile' and a subtitle 'Let your friends know how to start a conversation with you'. It features a 'Status Message' field with the text 'Feeling fictitious' and a 'Profile Description' field with the text 'The Eternal Georgia Tech Student. Go Jackets!'. Below these fields is an 'Update Status And Description' button. At the bottom, there are three buttons: 'Manage Personal Info, Avatar, and Privacy', 'Edit Contact Info', and 'View Profile'.

Settings

The settings page is arranged into “cards” that group various actions: viewing dialog boxes of information, updating account information, updating the Gravatar profile image, controlling privacy settings, changing passwords, exporting data to a prettified JSON file, and deleting the account.

The settings page is arranged into “cards” that group various actions: viewing dialog boxes of information, updating account information, updating the Gravatar profile image, controlling privacy settings, changing passwords, exporting data to a prettified JSON file, and deleting the account.

Information

- [Frequently Asked Questions](#)
- [Terms of Service](#)
- [Privacy Policy](#)

User Info

Username: gpb

Full Name: George P. Burdell

Nickname: George

Email Address: gburdell@gatech.edu

Birth Date: 4/1/1909

Gender: Male

Privacy Policy

User Info

Username: gpb

Full Name: George P. Burdell

Nickname: George

Email Address: gburdell@gatech.edu

Birth Date: 4/1/1909

Gender: Male

Profile Avatar

Boop uses Gravatar, a service that connects your email address to your profile image across different websites and apps. To change your Gravatar image, go to [gravatar.com](#) and sign in or create an account using the same email address that you use for Boop (gburdell@gatech.edu).

It might take around one minute for Gravatar changes to be applied.

Profile Privacy

Control who can see your profile, friends list, and contact info

Change Password

Old Password

New Password

Confirm New Password

Export Personal Data

Boop is transparent with the data we collect, feel free to take a look

Delete Account

Confirm Delete

Are you sure you want to delete your account? This action cannot be undone

CANCEL CONFIRM

Registration

Account registration is broken up into several steps. Users must complete one step before moving on to the next, but can go back to a previous step to change their information before submitting.

An earlier iteration of the registration process began with one page for collecting user information and then a sequence of “onboarding” pages for setting up contact details and notifications, but we found that users would get lost when trying to navigate or go back to the first page, so we decided to combine these steps into one page.

The registration process for Boop consists of the following steps:

- Personal Information**: Collects user details including Full Name, Nickname, Email Address, Birth Date, and Gender (optional).
- Profile**: Allows users to add Profile Text, set Profile Privacy (Everyone, Friends and Friends-of-Friends, Friends Only), and manage Profile Information (Show gender, Show age).
- Social Messaging Contact Info**: Asks which chat apps are used and collects a Contact ID (phone number) for text messages.
- Avatar**: Explains the use of Gravatar and provides instructions on how to change the profile image.
- Notifications**: Offers to activate push notification reminders to keep friends in touch.
- Submit**: Requires agreement to the Privacy Policy and User Agreement, followed by the final 'Create Account' button.

References

- [1] J. Medley, "Web Push Notifications: Timely, Relevant, and Precise," 12 February 2019. [Online]. Available: <https://developers.google.com/web/fundamentals/push-notifications>. [Accessed 2021].
- [2] Automattic, "Image Requests," [Online]. Available: <https://en.gravatar.com/site/implement/images/>. [Accessed 2021].
- [3] B. Welford, "A Guide to GDPR data privacy requirements," [Online]. Available: <https://gdpr.eu/data-privacy/>. [Accessed 2021].
- [4] Boop Team, "Boop Core Datatypes," 26 April 2021. [Online]. Available: <https://github.com/GRarer/boop/tree/main/core/src/datatypes>. [Accessed 2021].
- [5] D. Arias, "Adding Salt to Hashing: A Better Way to Store Passwords," 25 February 2021. [Online]. Available: <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>. [Accessed 2021].
- [6] C. Hale, "How To Safely Store A Password," 31 January 2010. [Online]. Available: <https://codahale.com/how-to-safely-store-a-password/>. [Accessed 2021].
- [7] H. v. Braam, "Orange Color Psychology," 9 September 2020. [Online]. Available: <https://www.colorpsychology.org/orange/>. [Accessed 2021].