# Monster Reservation Group Coding Challenge

# Brief

The challenge is stated as:

The task is to make an Angular web app deployed to Firebase. Firebase hosting is free and should be straightforward to use. Feel free to reach out if you run into trouble here. The app should include some flavor of authentication. Dealer's choice - user/pass, google, encrypted link, something else, your call. It should deny access to the form unless authentication is successful.

Delivery should come with working credentials if applicable. Once authenticated, the user should be taken to a form where they enter flight details. Upon submission, the app should send a post request with the below specs and inform the user of their success (or failure). The UI should inform the user whether their request was successful and they should know they're done. Any additions or enhancements you want to make here are acceptable but not required.

* The app should be placed in a github repo I can share with the team.
* URL: https://us-central1-crm-sdk.cloudfunctions.net/flightInfoChallenge
* Request header "token" should contain the value "WW91IG11c3QgYmUgdGhlIGN1cmlvdXMgdHlwZS4gIEJyaW5nIHRoaXMgdXAgYXQgdGhlIGludGVydmlldyBmb3IgYm9udXMgcG9pbnRzICEh".
* Request header "candidate" should contain your name.
* Request payload should follow the below interface.
* Comments are not required, but should be passed along if entered. Missing properties will cause the request to fail.
* Your arrivalDate can be in any format convertible to a Date object.

interface FlightInfoPayload {

airline: string

arrivalDate: string

arrivalTime: string

flightNumber: string

numOfGuests: number

comments?: string

}

# TLDR – Deliverables

The following pages in this document describe the design, architecture, and testing for this project.

The parts of the deliverable are:

* Design document (this document).
* Flight Info web app (core challenge).
* Token Guard web API (additional project for authentication).
* Customer’s email (additional UX).

The web app source code will be available at: “<https://github.com/HaydnFidler/MonsterChallenge-FlightInfo>”.

The web app will be hosted at: “<https://monster-challenge-b.web.app>”.

The web API source code will be available at: “<https://github.com/HaydnFidler/MonsterChallenge-TokenGuard>”.

The web API will be hosted at: “<https://studio--studio-9761992685-e3ea1.us-central1.hosted.app>”.

Supporting infrastructures for the deliverables are:

* Firebase Authentication (database of users).
* Firebase hosting (web app and web API).
* SendGrid (email campaign for emails to customers).
* Github (public repos).

User accounts for testing will be described in the Testing section.

# Design

## Assumptions

It is assumed that the user has previously created an account and supplied their email address and a password.

It is assumed that the user’s email address has been verified.

## User Experience / Storyboard

The user will receive an email.

The email will invite them to share their upcoming visit details by clicking a link. The link will direct them to our web app, where they will be prompted to enter their email address and password.

If a user arrives at the site without using the link (i.e, navigates there directly), they will be shown a page that informs them that they need to arrive here via their “personalized invitation”.

If the user enters an email that matches the link used to arrive here and their password matches the one stored in their account, they will be shown a page where they can enter their vacation arrival details.

If either the email or the password is incorrect, the user will be prompted to try again.

On the vacation arrival details page, the form will request their airline, arrival date, arrival time, flight number, number of guests, and any additional comments they would like to provide.

Upon submitting the vacation arrival details, the user will be directed to a page thanking them for their participation.

If the submission fails (due to a server error, for example), they will be notified in an event and asked to try again.

### Normal flow

Email -> landing page (auth) -> vacation details -> thank you

### Error flows

Email -> landing page (auth) -> (no auth) failed.

Email -> landing page (auth) -> vacation details -> (server failure) failed.

(no email) landing page (auth) -> failed.

# Architecture

The web app will be written in Angular v20, utilizing Firebase for hosting, data storage, and authentication.

## General

An email template will be generated, inviting the user to share their upcoming arrival plans. The email will contain a link that directs them to our web app and includes their email address inside a one-way encrypted token.

The web app will be a single-page application, using Angular’s built-in routing. Four pages will be created: three for each part of the user’s normal workflow, and one for an unauthorized landing.

A web api will be created to produce and verify the encrypted token.

### Styling

Styling of the web app and the email will be inspired by the Monster Reservation website (monsterrg.com).

# Email

The mail will be personalized for each customer, featuring their name and a link that identifies their account.

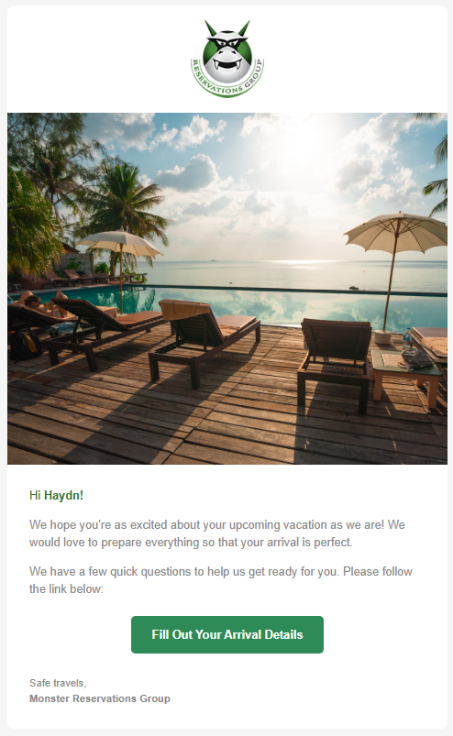


Figure - Email template

The link will include the customer’s email address as an encrypted token, using HMAC-SHA256.

# Web App Components

This app will adopt a general style that is sympathetic to both the company's branding and that of the email the customer originally received. The intention is to create a deliberate sense of continuity for the customer as they transition from their email app to their browser. The style target is shown below:

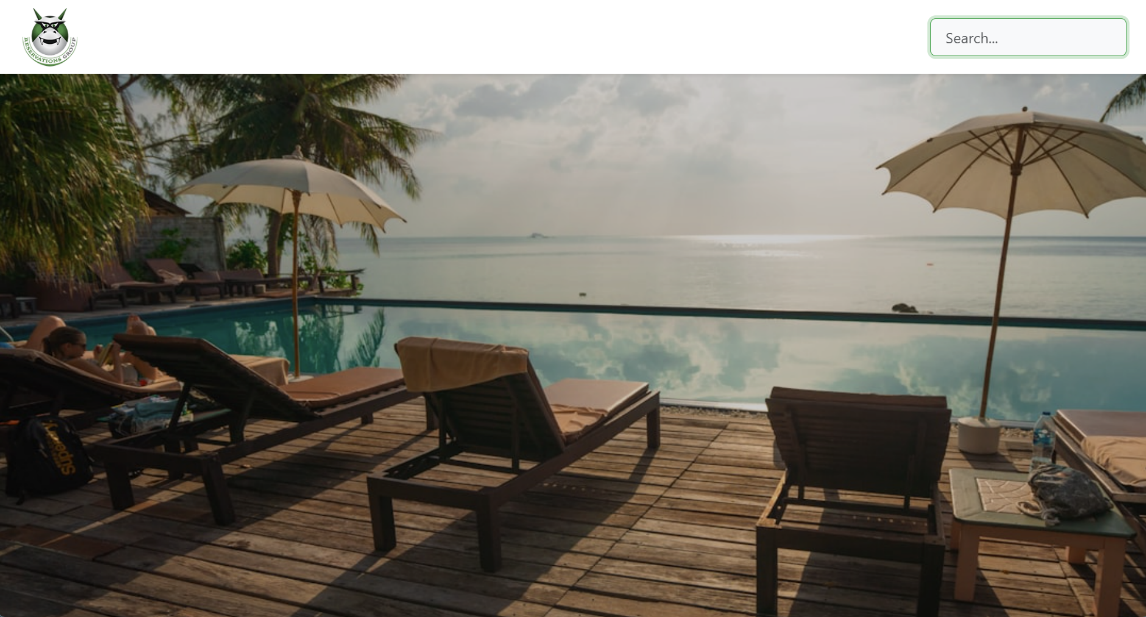


Figure - Root page style target

The root page will fill the browser and remain static while the routed components will float above, scrolling underneath the header when required.

The search box will forward the search text to monsterrg.com, reinforcing the customer’s sense of relationship between the two.

## Authentication

The default page will be for authentication. It is assumed that the user has previously created an account with an email address and password.

The page can only be accessed by clicking the link in the invitation email. The link will contain an encrypted token that represents their email address. Any URL missing the token will be routed to the Unknown Landing page.

The user will be asked to enter their email address and password.

Their email address entry must match the one in the link’s token. Both their typed entry and the token will be sent to a web API (Token Guard) for verification. The web API will encrypt the email address they entered and match it with the encrypted token from the link. If they match, this indicates that the email originated from an approved source and that the recipient is likely the email’s owner.

The email and password they enter will then be sent to the Firebase Authentication service for verification.

The style target for this page is shown below:

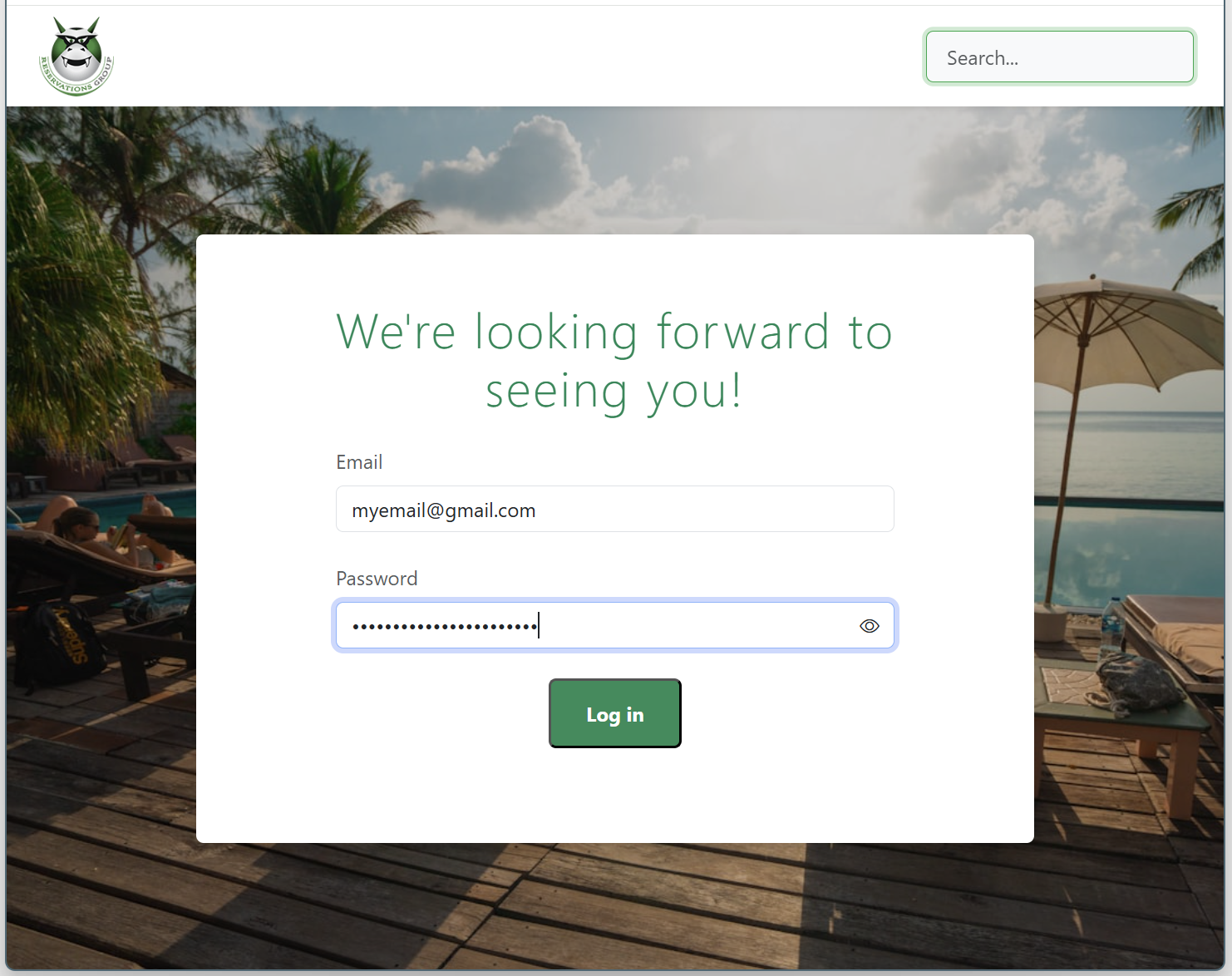


Figure - Auth page style target

Upon passing both types of verification, the session will be considered “authorized”, and the user will be routed to the Flight Info page.

## Flight Info

The Flight Info page will be shown after a successful authentication. If a successful authentication hasn’t happened for the active user, attempting to route to this page will re-route to the Unknown Landing page.

The page’s intent is to capture the user’s flight information:

* Airline
* Arrival date
* Arrival time
* Flight number
* Number of guests
* Any additional comments.

The user can leave the comments field blank if they choose to, but the other fields are required.

The style will be similar to the following guide:

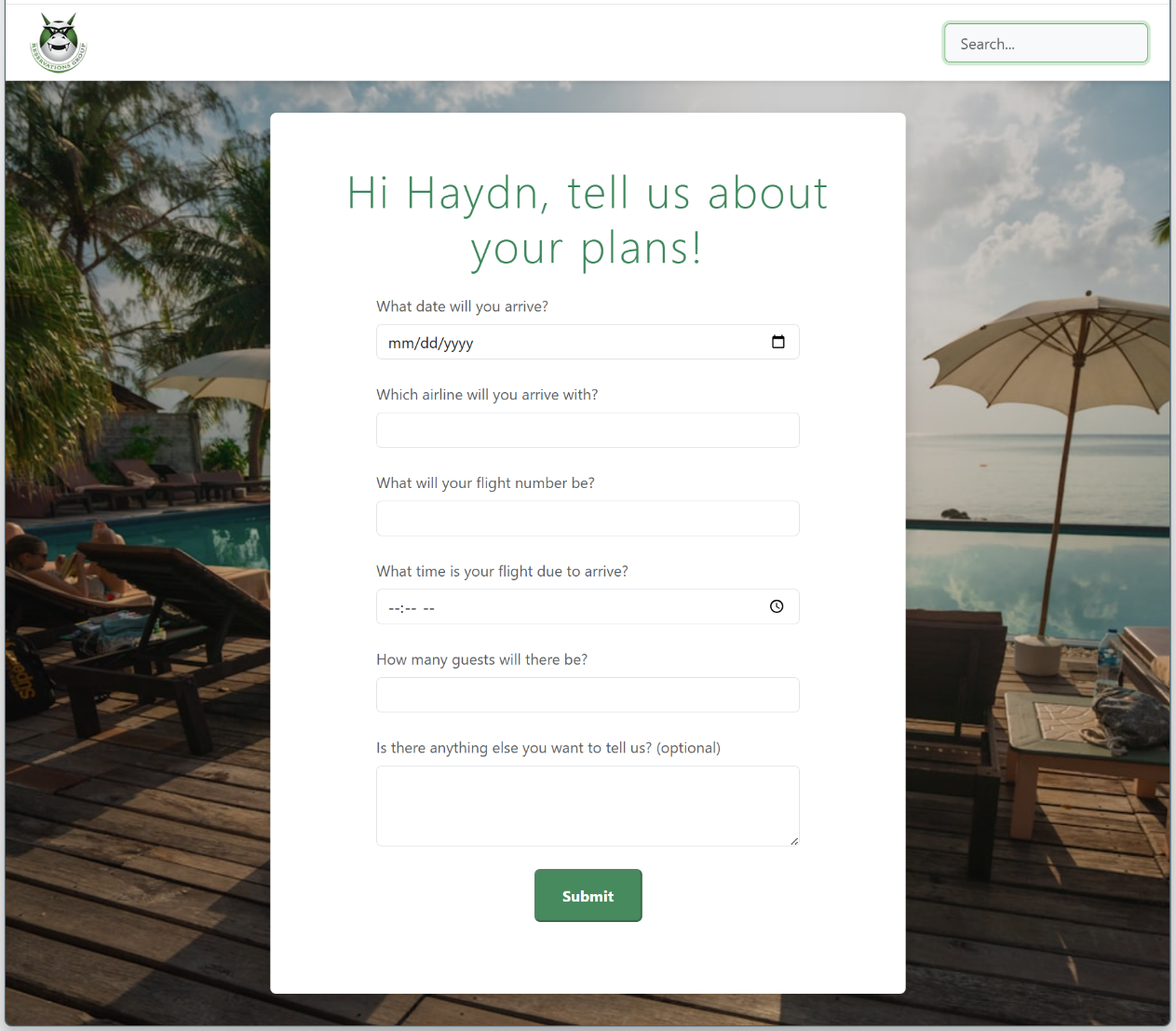


Figure - Flight info input form style target

The form will validate to ensure the required fields have been populated. Additionally, each field will only accept information of the correct type, where possible.

The user will be shown indicators on fields they have failed to populate correctly.

The Submit button will only become enabled when the form is valid and ready to post to the Web API.

## Completion

The completion page will indicate to the user that they have successfully completed the task. It will display a simple, yet personalized, thank-you message and prompt them to move forward with their lives.

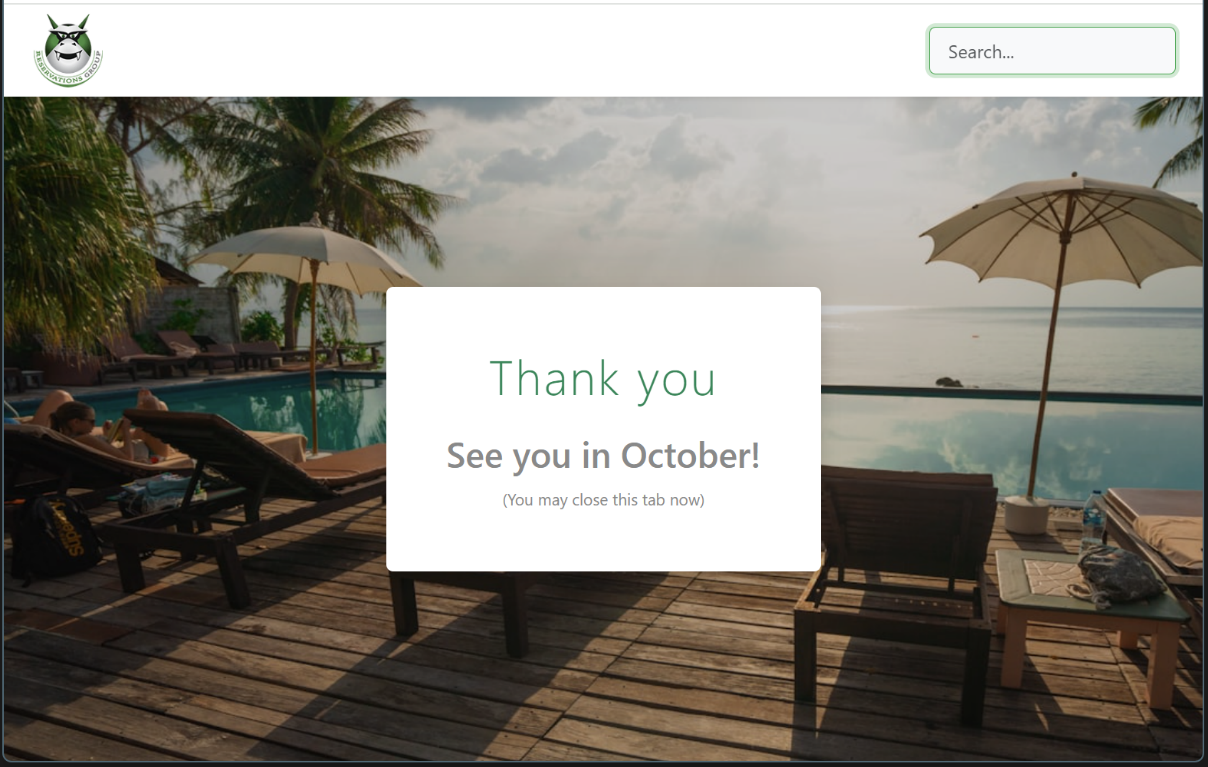


Figure - Completion page style target

## Unknown Landing

The Unknown Landing page is where users are directed when they are outside the intended workflow. These can include:

* Arriving at the web app via means other than a personalized invitation link.
* Attempting to land on any page other than the Authentication page without authenticating first.

It is intended that accidental visitors to the web app will land here.

The style will be similar to the following guide:

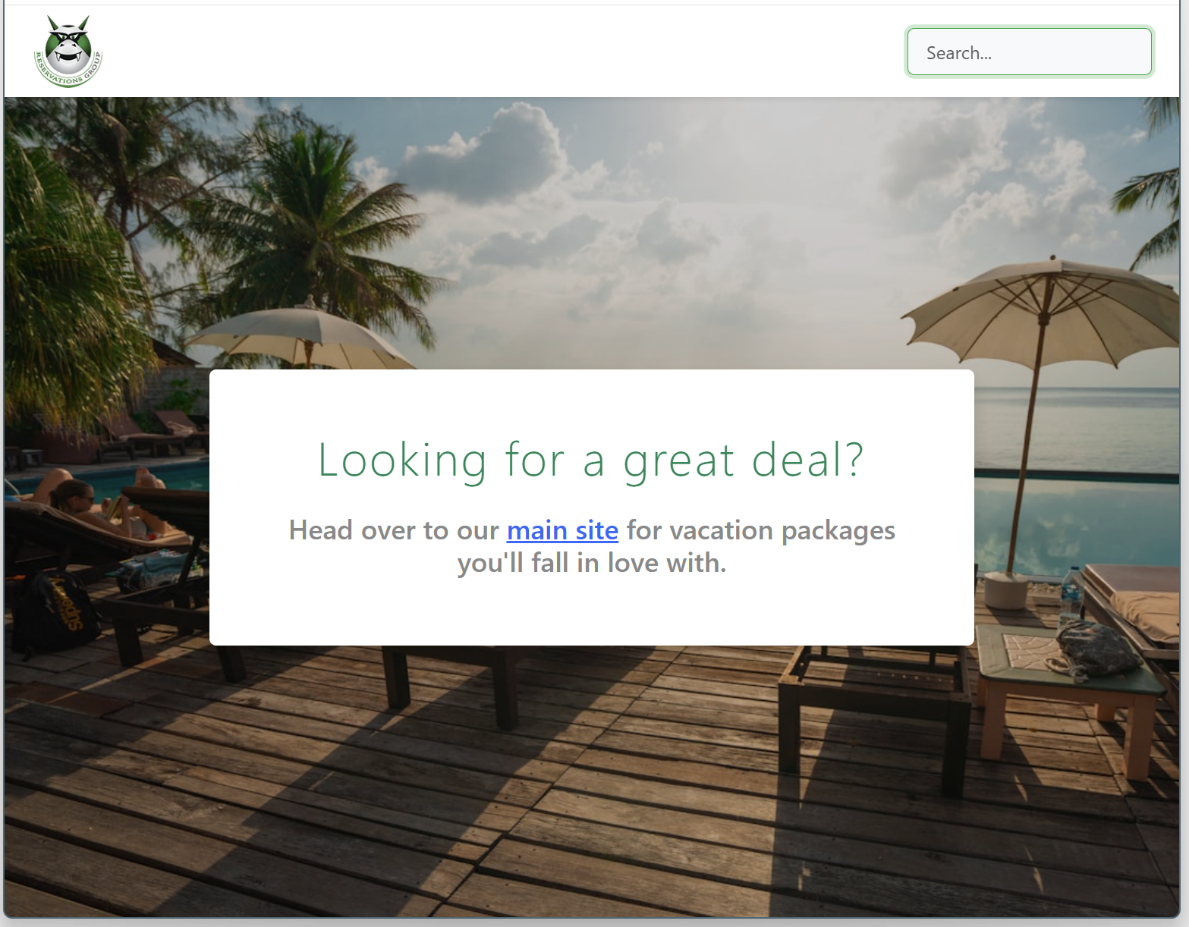


Figure - Unknown Landing page style target

# Web APIs

## Email Token API (Token Guard)

A new web API will be created. It will have two primary purposes:

* Create a token from the user’s email address.
* Verify that a plaintext version of the user’s email address encrypts to the same (or equivalent) token.

This web API will be used at two points during the process:

* During the creation of the email.
* During the authentication process in the web app.

The web API will be hosted in Firebase.

## Password API

This web API will be part of Firebase’s services. It will centre around the signInWithEmailAndPassword() function within the Firebase libraries.

## Flight Info API

This web API will be called during the form submission on the Flight Info page. Details of this API were provided in the challenge’s description.

# Testing

## Users

A number of test users will be created in Firebase:

| Email address | Password (minimum 6 characters) | displayName |
| --- | --- | --- |
| haydn@haydnfidler.com | haydnhaydn | Haydn |
| haydn2@haydnfidler.com | haydn2 | Haydn |
| matthews@monsterrg.com | matthew | Matt |

Each user will receive a personalized email that includes their secure link to the Flight Info web app.

The links generated for each user are as follows. They are denoted by (S)ite, (E)mail, and (P)assword:

S: <https://monster-challenge-b.web.app?token=MTc1ODEzNTA5Njovdle2_JZvDJTxeRYANZO10tDaijAC4dvDCYO8EqWHCQ>

E: haydn@haydnfidler.com

P: haydnhaydn

S: <https://monster-challenge-b.web.app?token=MTc1ODQ3NDQ1NjoQuX-dB6Db_Ts1kUfdU2ajFMe-1ZLIqXetom7GKl-fBw>

E: haydn2@haydnfidler.com

P: haydn2

S: <https://monster-challenge-b.web.app?token=MTc1ODQ3NDQ4MjqADnH5miA21iGHKIApT9ktXVNMFLt2L1zk293rPPgpUg>

E: matthews@monsterrg.com

P: matthew

### Testing Access

#### Link Verification

Testing scenarios should include:

* Correct token.
* Incorrect token (different user).
* Invalid token (corrupt token).
* No token.

Incorrect-token testing will be performed by entering valid credentials for one user after arriving from a link of another user.

#### Password Verification

Normal email/password testing should be performed against the known user credentials that exist in Firebase.

### Unit Testing

The Token Guard web API project will include some unit testing that parallels the scenarios mentioned above in the Link Verification section. This will use the “Jest” framework.