SPARK?!

Meaningful Conversations.

project

Development Log

created for

Orbital 2021

stage

Milestone 3

Development Log

This is **SPARK**.

We believe that meaningful connections between people are possible with good questions. That's why we've decided to build an application to facilitate these connections.

We enjoyed the entire development process, and welcome you along the journey to see what we've discovered.

SPARK is a project developed for **ORBITAL 2021**

SPARK is created by **Jefferson Lim An Jian** and **Tan Rui Quan**

SPARK?!

1) What is SPARK?

SPARK?!

SPARK?!
Meaningful Conversations.

Level of Achievement

We are aiming for **Artemis** level of achievement in **Orbital 2021**.

Motivation

We love questions.

We are from Facilitators@NUS. In our club, we explore various techniques for asking good questions: from debriefs, to probing questions, to question cards. From our experiences, we enjoyed how so many **meaningful conversations could develop through the power of questions** — conversations that help to **bond people together, discover more about ourselves, and learn new perspectives.**

We wanted to make this toolkit available to others in the form of an easy-to-use web application, which we have called **SPARK**.

Aim

To achieve this, we have identified 3 major aims of our application.

SPARK aims to:

1. Facilitate **introspective reflection**
2. Spark **meaningful conversations with others**
3. Be **enjoyable** and **effortless**: all in a web platform, accessible on mobile and desktops, face-to-face or online, that has all the tools you need, with a great, smooth user experience.

Our application is hosted at: sparkorbital.herokuapp.com

Scope of Project

To meet our aims, as well as challenge ourselves to develop with technologies new to us, we have set out on the following project scope.

SPARK is deployed as a **full stack web application**, using **React** as our frontend, and **NodeJS** + **ExpressJS** as backend, using **MongoDB** as its database and **socket.io** for real-time chat. SPARK also integrates with Unsplash's web API.

We are also intentionally curating and documenting our learning process along this journey. This is achieved through an online blog, written with static site-generator **Hugo**, with both of us making weekly posts. (blog is at sparkblog.netlify.app)

We chose to make a web application, since we believe it is the most accessible for users, who can simply access it by entering a URL. SPARK is responsive to both desktop and mobile users, making a web application the straightforward choice for our project.

Ultimately, we wanted to **develop an application that was rich in features and functionality, providing users with many ways to use it**. Not only that, we wanted to experience the **process of software engineering**, which covers the **design**, **implementation**, **testing**, and **maintenance** of our application. This is why we placed additional emphasis on these areas of our application, challenging ourselves to build something of our own we could be proud of.

Tech Stack

We both wanted to learn React, and utilize some popular tools in the market (MongoDB, socket.io, Figma, Hugo etc.) for web development. Additionally, we wanted to have a better understanding of proper software development practices, such as **unit testing, E2E testing,** and **CI/CD using Github Actions.**

Thus, we chose to explore the following tech stack for the project.

1. React (frontend engine)
2. Unsplash API (integration for pictures)
3. NodeJS + ExpressJS + MongoDB + Socket.io (backend, database and real-time chat)
4. Git (our repository for frontend and backend), Github Actions
5. Jest, Cypress (testing library)
6. Figma + Adobe suite (prototyping)
7. Hugo (blogging our journey!)
8. Heroku + Netlify (web hosting)

Unique Selling Points

Admittedly, SPARK is not a new concept. There exists other products with the same idea of "using questions to spark conversations". These range from physical card games, to mobile apps, and digital implementations.

Nevertheless, we believe that SPARK is special. SPARK has the potential to help individuals discover more about themselves, and uncover new perspectives of friends you might have known for years.

1. Most other applications contain many 'surface level' questions, which suit a party setting, but we believe that conversations have the potential to be deep and meaningful. **SPARK has 600+ handcrafted questions that go deep, challenge assumptions, and make connections that matter.**

2. Other implementations focus primarily on group settings. We also think that there is room for introspection when meaningful questions are used Solo, as well as in a more intimate, one-on-one setting. **SPARK provides the tool for short, but purposeful, conversations with yourself and others.**

3. SPARK's innovative use of Picture Cards to ask questions provides a different way of connection, **which can lead to more creative and engaging reflection.**

4. SPARK cares about privacy. We realized that there was no reason to store user data, which includes their personal answers to questions, as well as private chat data between users, on any server of any sort. As such, **SPARK ensures that your reflections and your connections stay private.**

5. Finally, SPARK was created by ourselves! Orbital is meant to be a learning journey for us to create something from scratch, and this is pretty meaningful to us :)

Some of the tools that have inspired SPARK are: Speakeasy Cards and We're Not Really Strangers. We have taken the most meaningful portions of other games to incorporate into SPARK.

2) Design

SPARK?!

With our goals for SPARK set out, we embark on the design phase.

UI/UX Design

In designing our interface and user experience, we had to have clear goals of what we wanted to achieve and who we wanted to target.

In summary, our design goals are:

- For our **Youth** users to be **meaningfully engaged** and **curious** when using SPARK.
- SPARK should look **bold**, and **attention grabbing**

Target Audience

To aid our engineering and design process, we defined a specific target audience that SPARK is aimed at.

We chose the target audience of Youths. **Youths are tech-savvy, familiar with the online world, and regularly reach for it as a source of entertainment, information, and social connection.** SPARK fits perfectly in this world. We also feel that there is much to be gained when Youths can **conduct meaningful conversations, as well as facilitate introspection**, which are useful skills that will come in handy in the future.

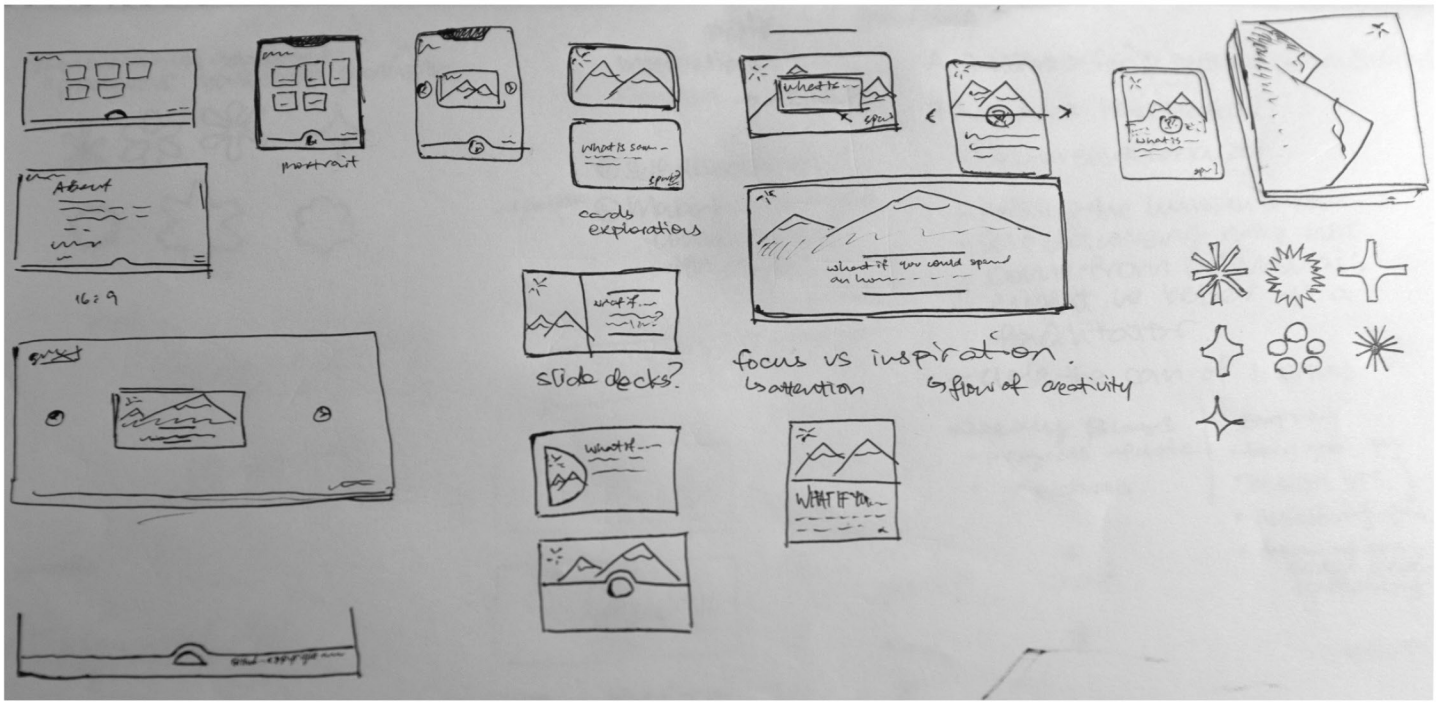
With a target audience clearly defined, it was easier to **empathize with the potential user**, and **prioritize features according to the suitability** with this target audience.

Look & Feel

Throughout the ideation phase, a core idea stuck: for SPARK to be **bold, meaningful, and curious.** We wanted the user experience and look of the app to convey this idea.

To achieve this, we focused on using contrasting colours, and applying a minimalistic design for the application. Our full design system can be viewed in the **Appendix**.

Initial Sketches

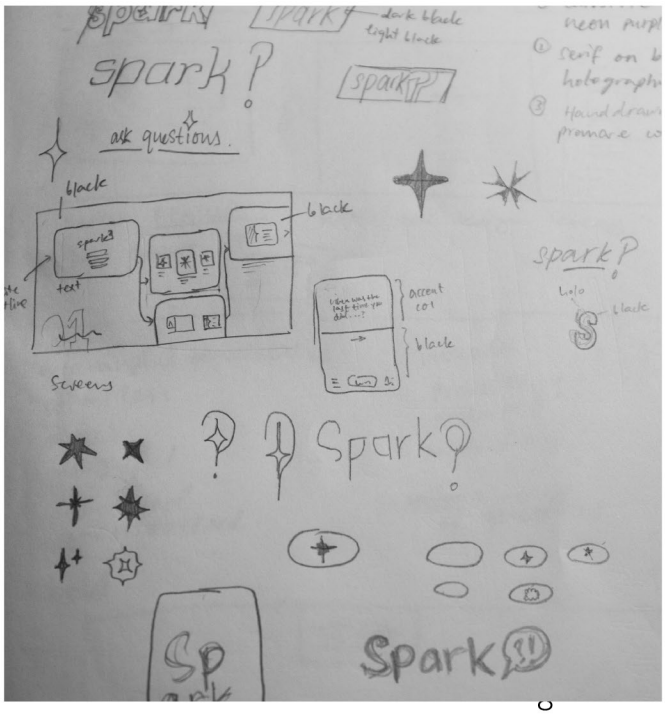


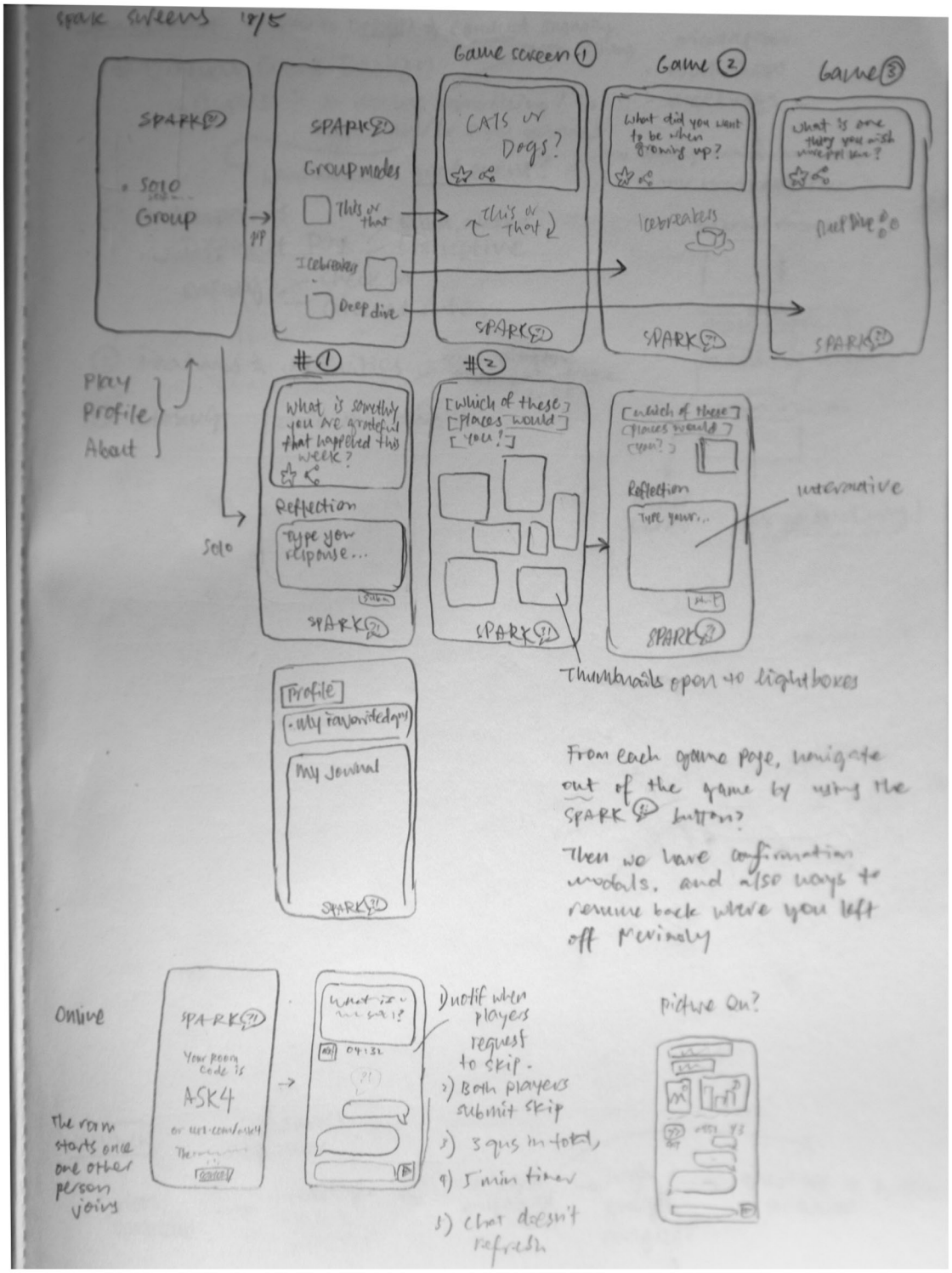
Firstly, we sketched our ideas for the layout of the application.

We wanted to have a low-fidelity way to **quickly iterate through possible layouts** and **discover the optimal user experience.**

Early on, we also explored various ideas for a thematic element to tie the whole app together.

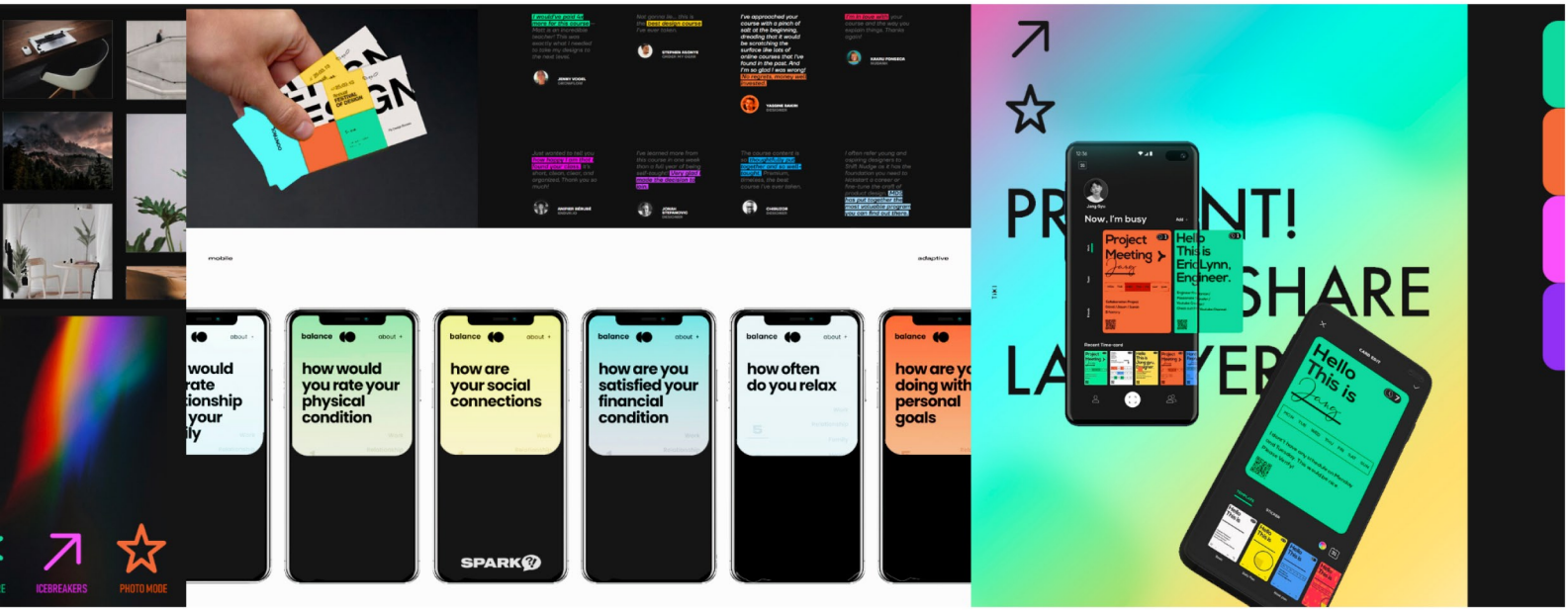
We decided on an **interrobang** as a symbol of asking questions. This eventually turned into the logo for our application, appearing throughout the app **to encourage users to be curious.**





Exploration of how various pages should link to each other, and understanding what the user experience is like.

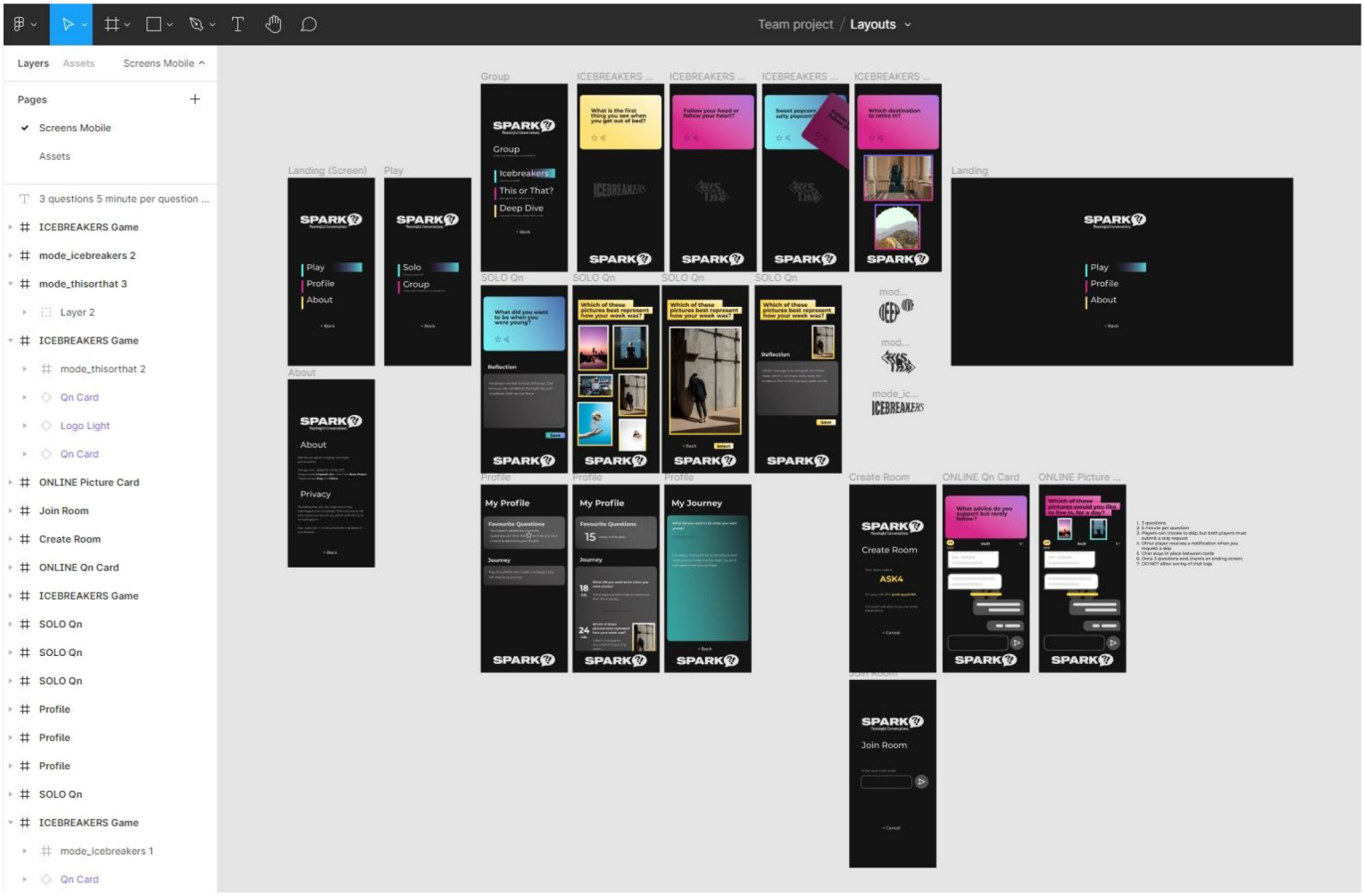
Stylescape



Next, we collected inspiration from **Pinterest** and decided on a **colour scheme and design library.**

This was collected into a **Stylescape**, which is a design document that guides the design of the whole application. It contains information about **colours, typography** and the overall **feel** of SPARK.

Figma Mockups



Finally, a **high fidelity mockup of the UI** was created on **Figma**.

Early in Milestone 1, this was the mockup that was shown to potential users, such as our friends and family members.

We received feedback about how the UI could be improved to be more appealing, as well as feedback about the colours and overall usability of SPARK. The initial comments also gave us inspiration for more features and adjustments we could make to our app.

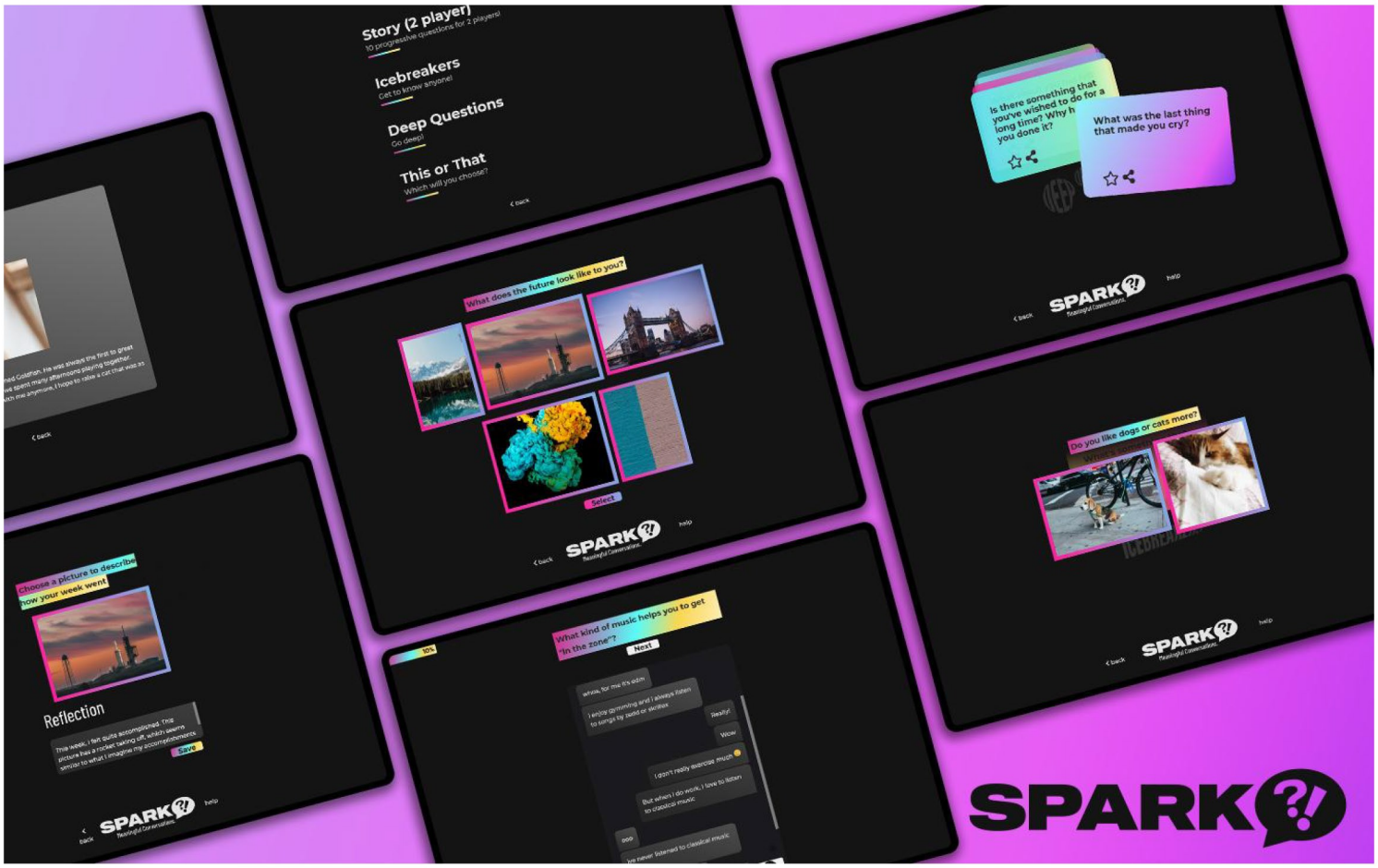
Final product

“interrobang” logo

SPARK?!
Meaningful Conversations.

Our logo consists of two elements: **conversations** and the **interrobang**.

The **conversations** (chat bubble) represents **purpose, dialogue & reflection**, while the **interrobang symbol** represents being **bold, surprise & meaning**, which all align with the key design goals of SPARK.



Final look & feel of SPARK

Feature Design

In Milestone 1, in tandem with developing the identity and design of SPARK, we also came up with a list of features for SPARK.

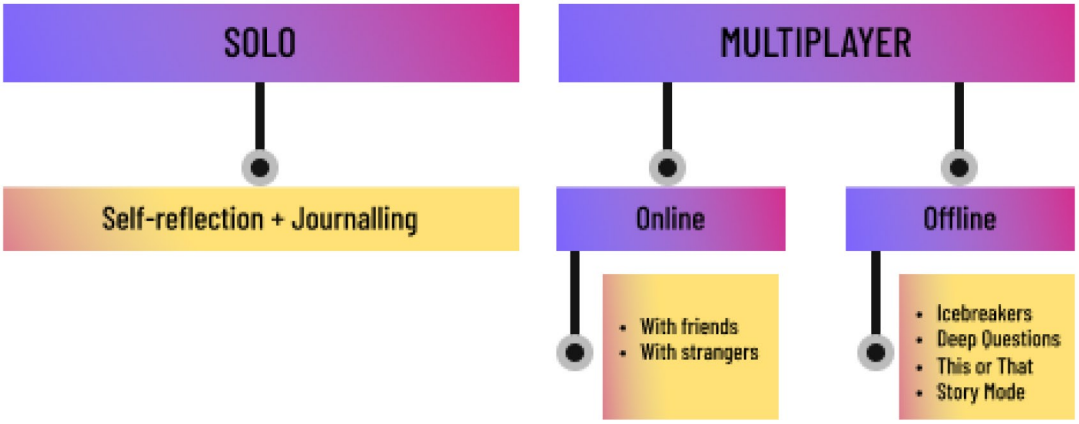
Originally, our app was only going to solely focus on questions that users could utilize in their conversations with friends. However, we decided to add in an online chat feature, which could drastically increase the utility and usefulness of our application.

User Stories

We created a matrix of User Stories to better understand the features we needed for SPARK.

As a ...	I want to be able to...	So that I can...	Feature to develop
User	Select a mode	Use the app	Main Menu, Navigation
User in solo mode	Receive questions about myself	Discover myself through self-reflection	Solo Mode
User in solo mode	Save my reflections into my profile	Keep track of my progress	Solo Mode
User in group mode	Select a category in group mode, based on what I want to play	Choose which direction the conversation would steer towards	Main Menu, Navigation
User in group mode	- Receive deep questions - Receive fun/interesting questions	- Have meaningful conversations with others - Learn more about others	Group Mode Question Cards Picture Cards
User in online mode	Generate a room code	Invite my friend to join and have a chat remotely	Online Mode Real-time Matchmaking
User in online mode	Enter a room code	Join the room that my friend created	Online Mode Real-time Matchmaking
User in online mode	Join a random room	Converse with a stranger	Online Mode Real-time Matchmaking
User	Add questions to a list of Favourites	Access them easily in the future	User profile
User	Share questions onto my social media	Get my friends involved in answering the questions	User profile

Game Modes



Following the User Stories, we decided on this structure for our game modes in SPARK.

We wanted to provide users with **a variety of choices** for how they wanted to use SPARK in ways that best suited them.

User Journeys

Finally, we charted out a few user journeys that potential users could take, which helps in designing our **tests** and **implementing the application flow**.

Journey 1: Introspection

This user enjoys alone time and self-reflection. They enjoy keeping a journal and writing down their own personal thoughts.

With SPARK, the feature they use primarily is **Solo Mode** and **Journey** mode. Being able to receive 3 introspective questions a day in **Solo Mode** helps this user self-facilitate their own experiences, and pen down their thoughts. The **Journey** mode within the **Profile Page** helps the user check back on their past experiences, growing in their own journey.

Features to develop

Solo Mode *(receive daily questions)* ➜ user reflections end up in **Journey** on the **Profile Page**

Journey 2: Social Butterfly

This user loves making new friends, and cherishes the time spent with all of them. They're always on the lookout for fun activities to play together and bond.

With SPARK, this user enjoys the **Group Mode** and **Online Mode** features. Having a database of questions to bring out during parties and gatherings helps this user create conversations easily and effortlessly. When the user is not physically with their friends, they can create **Private Rooms** which their friends can join, and also chat with strangers in **Online Rooms**.

Features to develop

Group Mode *(ability to receive varied and interesting questions)* ➜ **Question Bank**

Online Mode with **Real-Time Chat** ➜ Matchmaking with **Friends** and **Strangers**

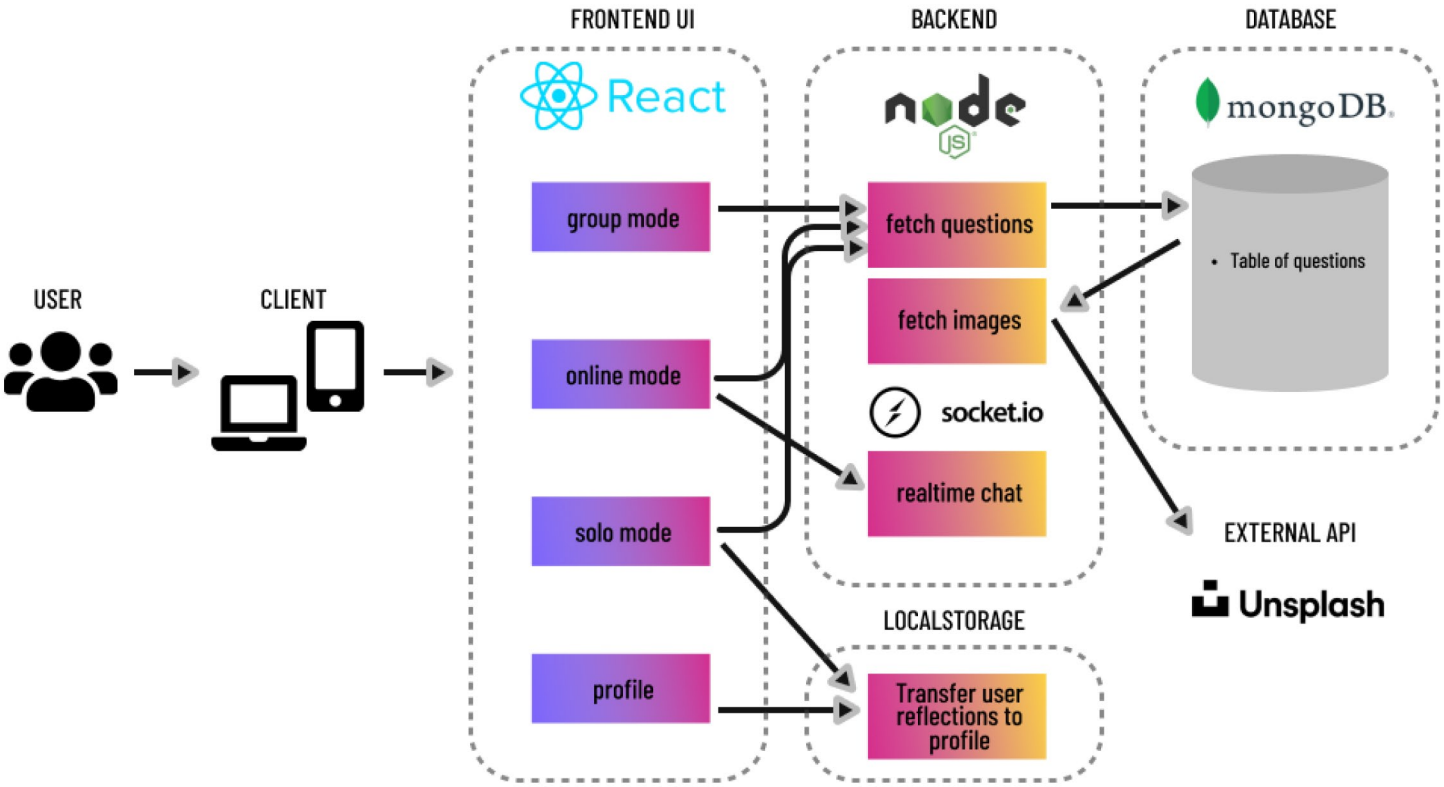
3) Implementation

SPARK?!

Overview

SPARK is a full-stack web application. We wrote the frontend in React, as well as a backend API using Express and NodeJS.

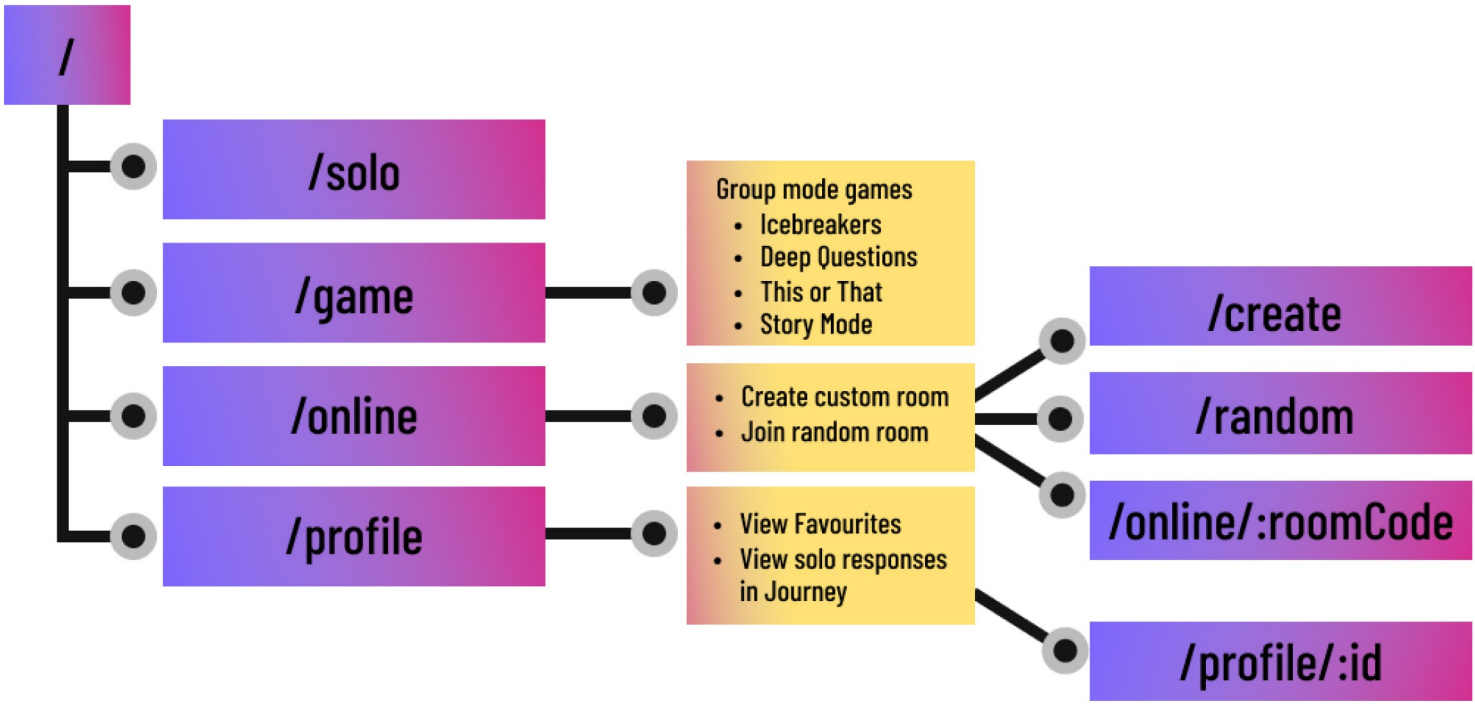
Both the frontend and backend were bootstrapped in **Milestone 1**, with a simple proof of concept. **Milestone 2** was spent developing all the core components and features of SPARK in parallel. Finally, **Milestone 3** was used to polish and conduct thorough testing of SPARK
.



Overall design architecture of SPARK

Sitemap

Pages



Since we were using **React** as a frontend, routing was kept simple using **react-router** to manage client-side routing for users.

Even though the sitemap only has a few possible routes, depending on the user state and options chosen by the user, we are able to **dynamically render different modes** and **content** for the user.

One example is the ability to display different groups of cards, depending on the game mode selected for **Group Mode**.

Thus, the benefit of this dynamic routing is the ability for SPARK to function like a **single page application (SPA)**, with very little reloading or downtime on the client's view.

(/) - Home Page

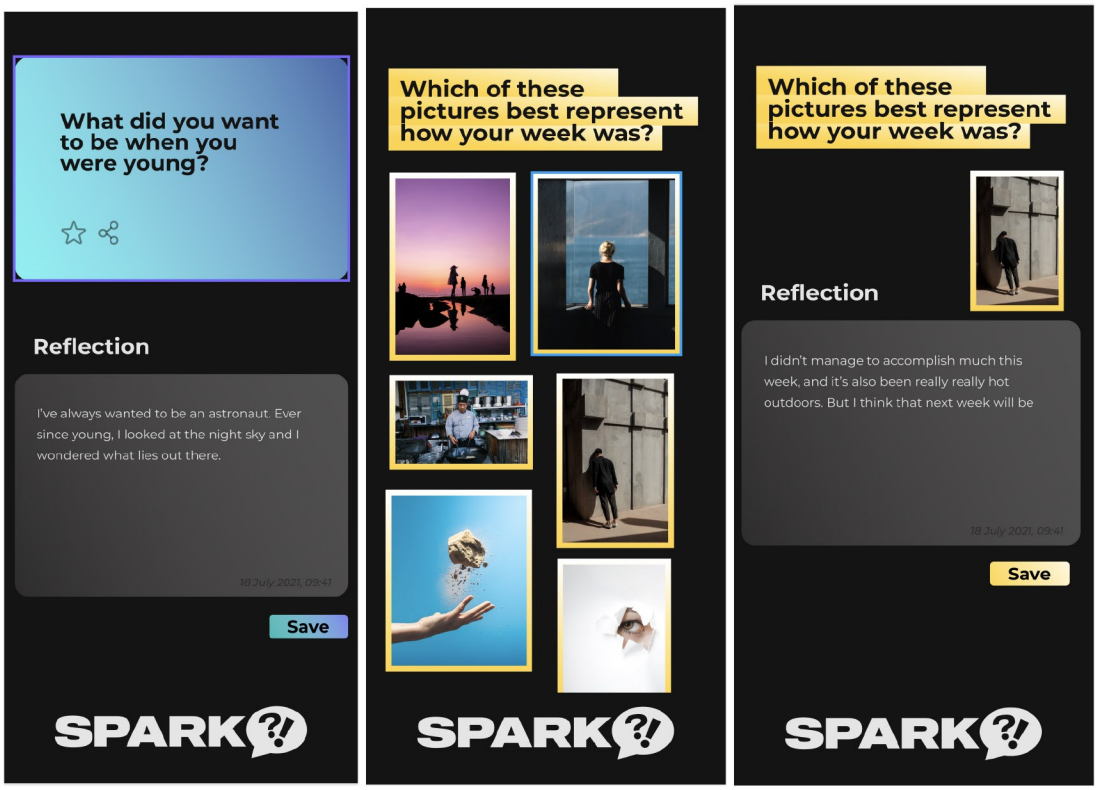
The homepage consists of a simple menu where users can click through to access the rest of the application.

react-router is used to send users to the appropriate page, and use of **React useState** hook allowed us to pass the appropriate properties to track the choices made by the user.

(/solo) - Solo mode

Solo mode is made for reflection! Every day, users receive 3 questions that they can write an answer for. The reflections are saved in the user Journey which users can revisit.

In our application, we do not intend to store any user's 'sensitive data' on any server, for privacy reasons. All data is stored solely on the user's browser local storage.

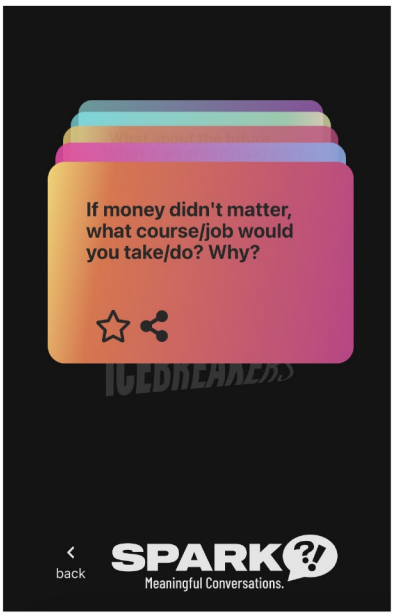


- **Question Cards & Picture Cards**
  Question Cards & Picture Cards are utilised here as a tool for introspective reflections.
- **Reflection**
  Users will write a short self-reflection, which is stored in their Journey.

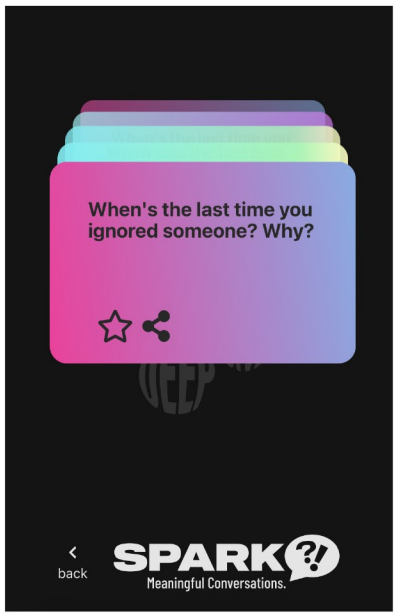
(/game) - Group mode

Group mode contains questions from different categories. This mode is best played in-person, where one player accesses the app, and swipes through to pick questions for the rest of the group.

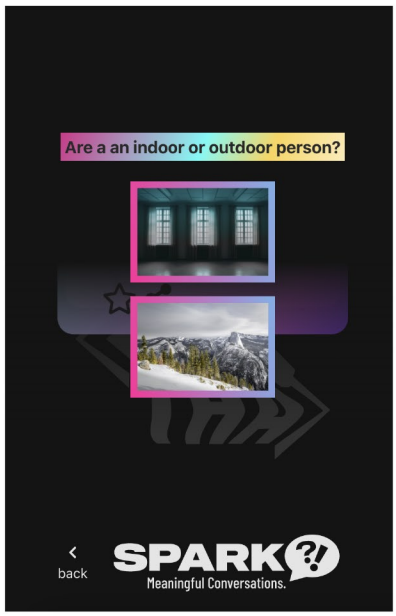
- **Icebreakers**
  Question cards & Picture cards that help to warm you up with anyone! The wide variety of topics is sure to get any conversation going.
- **This or That**
  A simple but effective party game! Which option would you or your friends pick?
- **Deep Questions**
  Questions that challenge assumptions and go deep into what's normally left unsaid. Find out what you and your friends are truly like, and bond together in the process!
- **Story**
  Story Mode is for 2 players to embark on a journey to get to know each other better. Players take turns to ask questions from a pool of 10 which get increasingly personal.



An example question from Icebreakers



An example question from Deep Questions



An example question from This or That

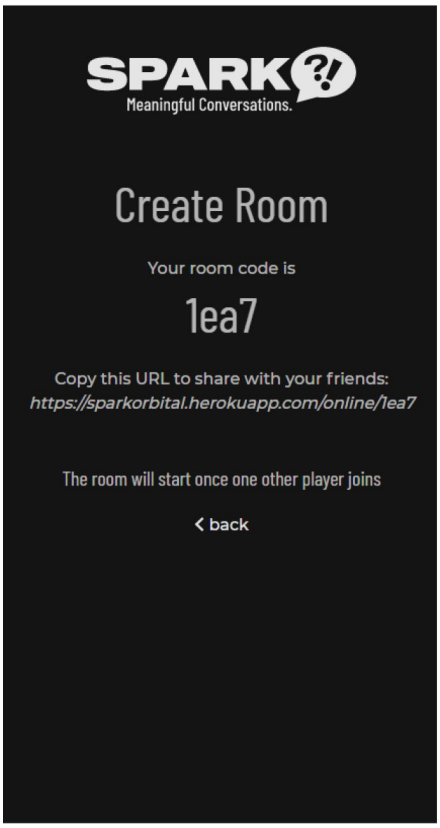
(/online) - Online mode

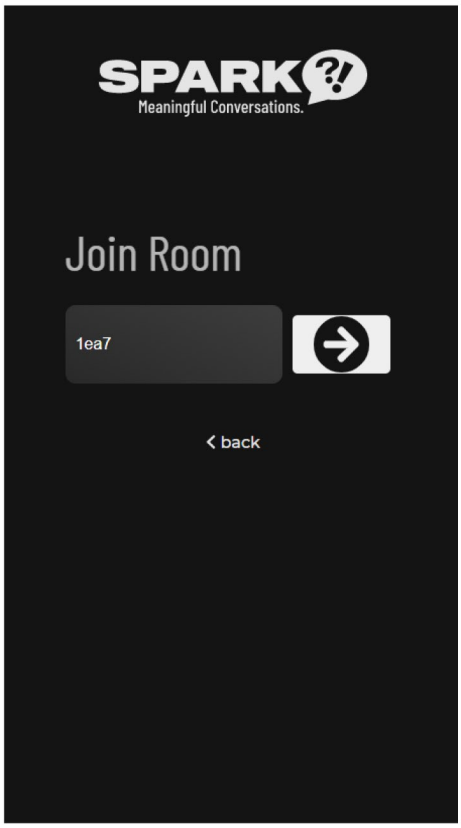
Take your conversations and questions online! Pair up with a stranger, or someone you know, and enjoy learning more about one another!

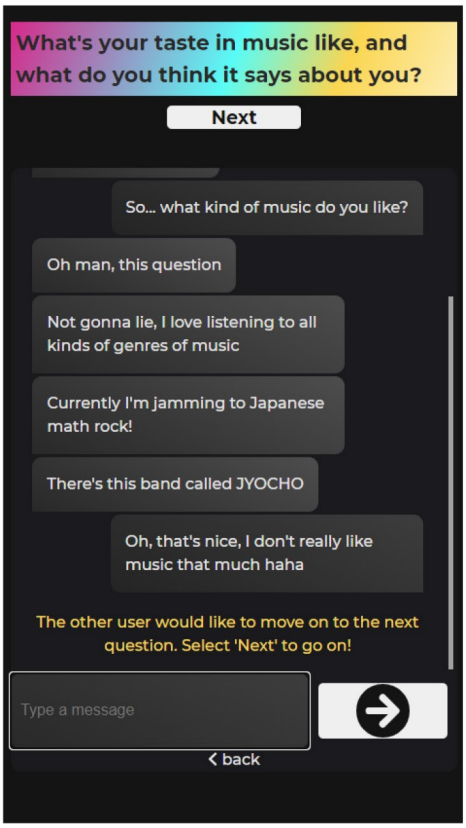
- **Create room**
  A unique room code will be generated for the user, which can be shared
- **Join room**
  Users with a room code can join an existing room
- **Real-time chat**
  The Question and Picture Cards feature is combined with a chat function to facilitate a meaningful conversation. There are 10 questions, each with different 'levels' of familiarity, for users to progress through.
- **Random room**
  Players are matched up with a random stranger, and given a set of questions

Usage of **socket.io** in the backend allowed for real-time chatting functionality in Online Mode. With **socket.io**, instead of having the client constantly send requests to the backend server for updates, the client simply subscribes to the server via a websocket, and there can be **continuous communication between client and server**.

Thus, our frontend is able to create a persistent connection with the server, as well as other users, and **display content in real-time** which is **synced between users**.

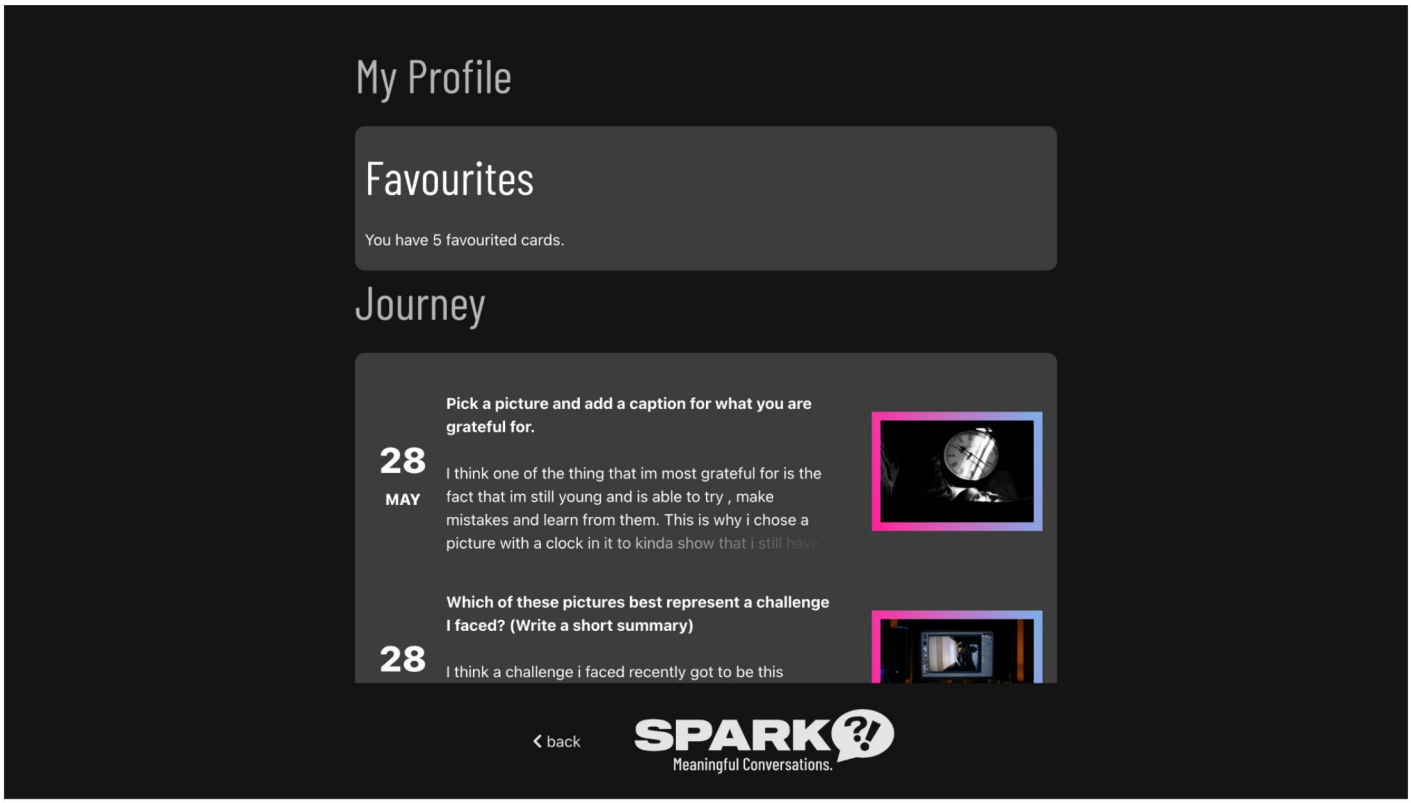






(/profile) - Profile page

My Profile



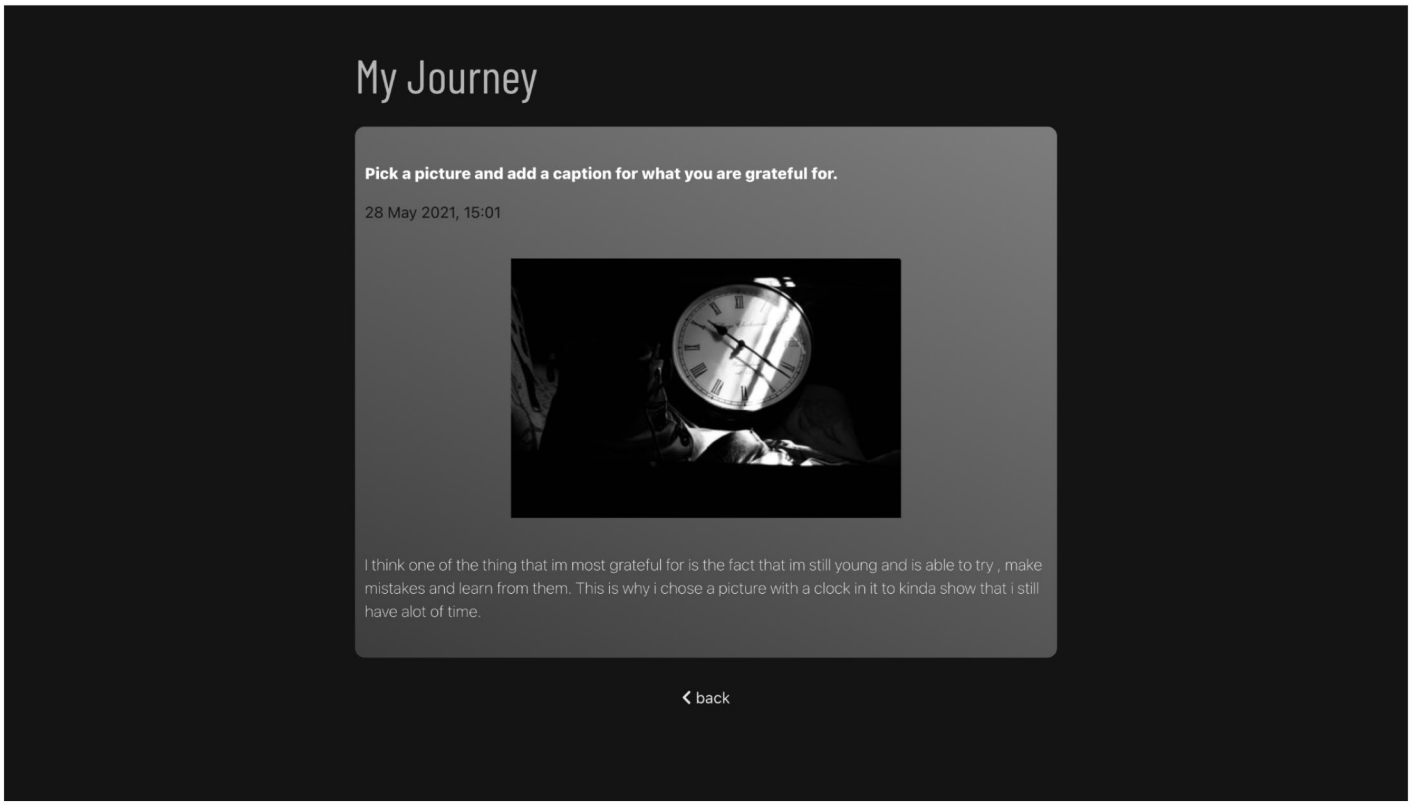
My Profile *contains the* ***Favourites*** *and the* ***Journey*** *sections for users to click into. Over here, the user would see a summary of the number of favourited cards they have as well as the latest reflection.*

Favourites

*Previously favourited questions can be easily accessed in the **Favourites** section.*

Users' favourites are stored in the browser's localStorage so as to protect users' privacy. We don't want user data with any revealing information to be sent and stored in online servers

Journey



The Journey segment contains all questions that the user has answered in the Solo Mode. It serves as a record of their answers, and a way to look back and reflect on their responses.

Each of the user's answers are timestamped, such that users can look back on their progress and changes in thoughts over time.

Using dynamic routing in **react-router-dom,** we are able to direct users to `/profile/:id`, where `:id` refers to each unique entry in the user's journal.

Backend

Dependencies

- express — for web application purposes
- mongoose — for integrating with MongoDB
- socket.io — for real-time chat functionality
- socket.io-client — for unit testing
- supertest — for unit testing

Backend Architecture

```
├─ controllers/     routers for API endpoints
│  ├─ questions.js  questions router
│  └─ pictures.js   pictures router
├─ models/          Schema for models
│  └─ question.js   question Schema
├─ socket.io/       socket connection for online mode
├─ tests/           code for testing API calls
├─ utils/           utility code
├─ app.js           Node.js server
└─ index.js         entry point of backend
```

Despite being very light (~720KB!), we have still kept our folder structure of our backend **neat** and **organized**, to make the **application easier to extend in the future**.

This meant segregating out the routing, testing, database Schemas, tests, and so on.

Our backend serves 2 major roles:

1. As an API to fetch questions from our **MongoDB**, and pictures from **Unsplash API**.
2. As a server to manage **real time chat using websockets** for clients in Online Mode.

Key Backend API Endpoints

In this section, we document a few key API Endpoints that are used in our app. This list is not exhaustive. For a full documentation, view the **Appendix**.

API Design considerations

We designed our backend API to follow REST API conventions.

The main requests to our backend are GET requests to retrieve data from various databases and services. We also implemented a POST request to write questions into the database to cut down on development time.

questions - Endpoint to fetch questions from database

GET /api/questions/{category}/{level}	
Returns questions from the given category and level If level value is not given, returns questions from all levels If level and category values are not given, returns all questions from database	
Parameters	
Name	Description
category string	Category values that need to be considered for filter Available values: *icebreakers* *this-or-that* *deep* *solo* *online-stranger* *online-room* *story*
level string	Level values to need to be considered for filter Available values: *1, 2, 3*
Responses	
Code	Description
200	Successful operation **GET** /api/questions/icebreaker/1

	<pre>[
	  {
	    "category": [
	      "icebreakers",
	      "online-stranger",
	      "online-room",
	      "story"
	    ],
	    "question": "What's something that you've always wanted to try/test out?",
	    "level": "1",
	    "canPicture": false,
	    "id": "60d9d3979b9fc39777971ebe"
	  },
	...other questions
	]</pre>

400	Invalid category or level values supplied

pictures - Endpoints to fetch pictures from Unsplash

GET /api/pictures/topic/{topics}/{count}	
Returns an array of pictures from the given topics Multiple topic values can be provided with comma separated strings If count value is not given, returns one picture	
Parameters	
Name	Description
topics ***required** string (path)	Topic values that need to be considered for filter Available values: *animals* *athletics* *architecture* *arts-culture* *business-work* *current-events* *experimental* *fashion* *film* *food-drink* *health* *history* *interiors*

	nature *people* *spirituality* *technology* *textures-patterns* *travel* *wallpapers*
count string (path)	The number of pictures to return Default: 1, Max: 30
Responses	
Code	Description

200	Successful operation
	GET /api/pictures/topic/people,health/3
	
	[
	  {
	    "id": "wQk0pSCyw94",
	    "created_at": "2021-01-18T17:36:36-05:00",
	    "updated_at": "2021-07-08T00:48:05-04:00",
	    "promoted_at": null,
	    "width": 2784,
	    "height": 4176,
	    "color": "#d9d9d9",
	    "blur_hash": "LaOVy1R+oexZ_4oLj[kCAIxZNGNG",
	    "description": "model: @yulyapulya262 ",
	    "alt_description": "woman in gray hoodie standing near body of water during daytime",
	    "urls": {
	
	      "raw": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1",
	      "full": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=srgb&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=85",
	      "regular": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=1080",
	      "small": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=400",
	      "thumb": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=200"
	    },

	...other information about the picture }, ...other pictures]
400	Invalid topic value(s) supplied

GET /api/pictures/query/{query}/{count}	
Returns array of pictures with the given search query If count value is not given, returns one picture	
Parameters	
Name	Description
query ***required** string (path)	Filter pictures matching a search term
count string (path)	The number of pictures to return Default: 1, Max: 30
Responses	
Code	Description

200	Successful operation
	GET /api/pictures/query/female/3
	[
	{
	"id": "wQk0pSCyw94",
	"created_at": "2021-01-18T17:36:36-05:00",
	"updated_at": "2021-07-08T00:48:05-04:00",
	"promoted_at": null,
	"width": 2784,
	"height": 4176,
	"color": "#d9d9d9",
	"blur_hash": "LaOVy1R+oexZ_4oLj[kCAIxZNGNG",
	"description": "model: @yulyapulya262 ",
	"alt_description": "woman in gray hoodie standing near body of water during daytime",
	"urls": {
	"raw": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1",

```
      "full":
"https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=srgb&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=85",
      "regular":
"https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=1080",
      "small":
"https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=400",
      "thumb":
"https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=200"
    },
  ...other information about the picture
  },
...other pictures
]
```

Question DB Schema

We have a single database to store our questions. The Schema for the database contains the following fields (* = required):

- category*: The category that this question belongs in. Question can belong to multiple categories, thus this field is an Array of Strings. Available values: *icebreakers, deep, this-or-that, online-stranger, online-room, story*
- question*: The question itself
- questionAlt: An alternative phrasing of the question
- level*: An indicator for the 'deepness' of the question. Available values: *1, 2, 3*
- canPicture: A boolean to indicate if the question can be presented as a PictureCard
- this: The first option for this-or-that. Used as a search term if canPicture is true
- that: The second option for this-or-that. Used as a search term if canPicture is true
- topic: The topics to search for questions which have a picture

```
const questionSchema = new mongoose.Schema({
 category: {
   type: [String],
   required: true,
 },
 question: {
   type: String,
   required: true,
 },
 questionAlt: String,
 level: {
   type: String,
   required: true,
 },
 canPicture: {
   type: Boolean,
   required: true,
 },
 this: String,
 that: String,
 topic: String,
})
```

Socket.io

We used Socket.io to enable **continuous communication** between client and server, which is required in the **Online Mode real-time chat** feature.

Both the client and server emit **events**, which are continuously monitored, and result in various state changes and content being rendered.

Socket.io is able to group users into rooms, and events can be tailored to be sent only to certain rooms. In SPARK, we put 2 users into a single room, and the private messages sent between them are only emitted to one another.

When rooms are created, they are appended to an array in the server, which keeps track of users and rooms available.

Events emitted by client	
Event	Purpose
create	Client sends a request to create new game room
privateMessage	Client sends a private chat message
leaveWaitingRoom	Client reports that user has left waiting room
leave	Client reports that user has left chat room

Events emitted by server	
Event	Purpose
session	Create details of session such that user info persists on refresh
joinRoom	Server confirms that user has joined a room
message	Server sends a chat message to a specific room
setQuestions	Server sends a list of questions a particular room

Frontend

Dependencies

- react-emoji — for rendering emojis in online chat
- react-fontawesome — used for icons throughout the app
- react-router-dom — routing
- react-swipeable-cards — component for swipeable cards
- react-testing-library — unit testing & integration testing library
- react-toastful — toast notification component
- redux — global state management for react

React

Frontend Architecture

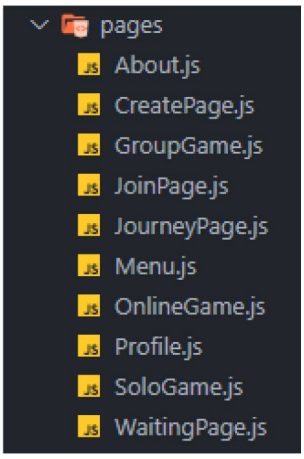
```
├─ public/          html boilerplate
├─ src/             React source code
│  ├─ assets/       static images/logos
│  ├─ components/   React Component library
│  │  ├─ cards/
│  │  ├─ chat/
│  │  ├─ navigation/
│  │  ├─ utils/
│  │  └─ profile/
│  ├─ mocks/        mock data for testing
│  ├─ pages/        pages of the site
│  ├─ reducers/     Redux reducers
│  ├─ services/     code to interface with API
│  ├─ styles/       stylesheets
│  ├─ utils/        utility scripts
│  ├─ App.js/       React boilerplate
│  ├─ index.js      entry point of frontend
└─ └─ socket.js/    code for real-time connection
```

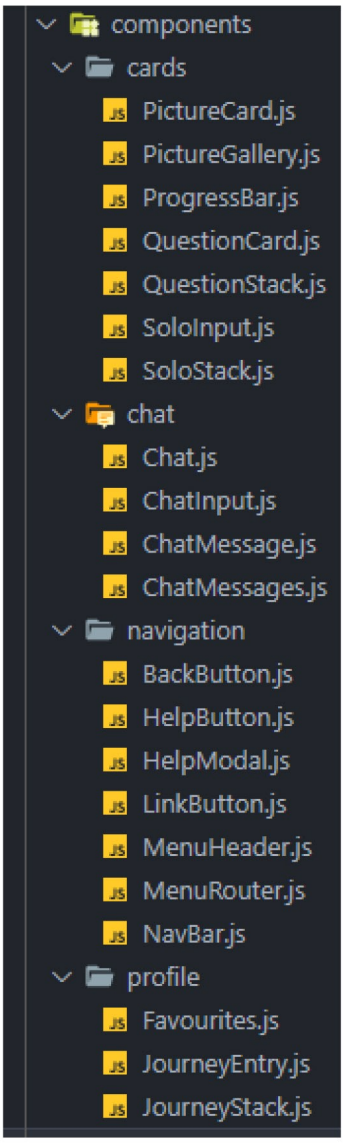
Unit tests are situated in `__tests__` directories within the `component` directory.

Component Library

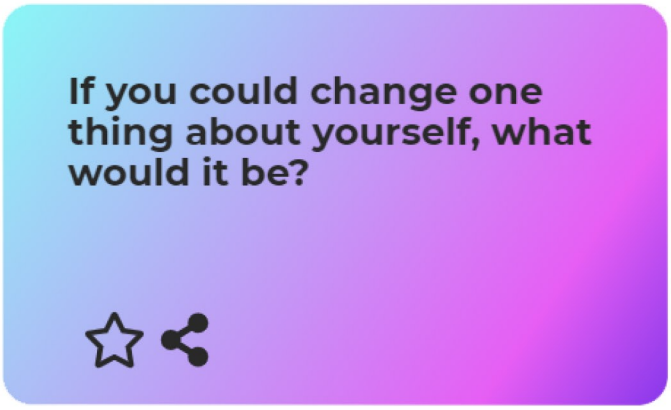
Pages are high level Components which handle the layout and structure of each page the user visits. Each page contains functional Components.

Functionality is abstracted out into separate Components, for features such as **Cards, Chat, Profile,** overall site **Navigation**, and **Utilities** (a small little `Confetti` component).





Question Cards



Question Cards are the main interactive component of our app.

- Each Question Card has the functionality of Favourites, which adds the Card to your favourites for easy access, and a Share feature to share this question on social media.
- The **QuestionCard.js** Component receives the following props:
  - data, Object which contains the actual question itself
  - mode, String which represents current game mode. Controls rendering of this-or-that questions.
  - isFavouritible, Boolean value to control rendering of Favourites
  - color, String to represent colour of card
  - displayToast, Function that's passed from parent to display toast message

```
return (
  <div className={`game__question-card --${color}`}>
    <p className="game__question-card--title"> {question} </p>

    {isFavoritible
      ? // render favourite + share icons
      : null
    }
  </div>
);
```

Picture Cards

Picture Cards are randomly selected photos from the Unsplash API, paired with a question..

- The **PictureCard.js** component receives the following props:
  - data, Object which contains the actual question itself
  - mode, String representing current game mode
  - topic, String representing search query on Unsplash
  - isActive, Boolean to handle display of Picture cards
  - isSelectable, Boolean value to control rendering of selection elements
  - handleSelect, Function that's passed from parent component to manage selection of pictures

- Then, information about the number of pictures to render are calculated and retrieved from the Backend. The relevant props are passed to the Presentational Component **PictureGallery.js**, which is responsible for rendering the Pictures.

Snippet from **PictureCard.js**, which generates pictures to be rendered

```
return (
    <div className={`card__picture ${isActive ? 'active' : 'inactive'}
${isSelectable ? '' : 'disable--select'}`}>
      <div>
        <p className="card__picture--question"> {question} </p>
      </div>

        <PictureGallery pictures={pic} isSelectable={isSelectable}
handleSelect={handleSelect}/>

    </div>
);
```

Snippet from **PictureGallery.js**, which handles rendering of images

```
<div className="card__picture--gallery">
    {pictures.map(pic =>
      <div key={pic.id}
        className={`card__picture--container ${isSelectable ? 'selectable' : ''}
            ${(select) ? select.id==pic.id ? 'selected' : '' : ''}`}
            onTouchEnd={(e) => toggleSelect(e, pic)}
            onMouseUp={(e) => toggleSelect(e, pic)}>

            <img src={pic.urls.small} alt={pic.alt_description}
              className={`card__picture--thumb ${(pictures.length === 1) ?
'card__picture--solo' : ''}`}/>

      </div>
    )}
</div>
```

Description:
The PictureGallery needs to handle multiple images. Furthermore, in Solo Mode, users should be able to click to select certain images. Thus, there needs to be an event handler when PictureCards are selectable

Question Stack

QuestionStack.js is a Component that is responsible for rendering both types of cards, and manages the interactivity of the Cards.

- The Component takes an array of questions, and displays a subset of 5 questions
- If the Card is a Picture Card, it is rendered as a **PictureCard** component

Card and **CardWrapper** are components from react-swipeable-cards library, that manages interactivity and swiping of the cards

```
return (
  <CardWrapper className="game__question-card--container">
    {qnList.map((q, idx) =>
      <Card key={q.id} onSwipe={handleSwipe}>
        {q.canPicture
          ? <PictureCard topic={q.topic} data={q} isActive={idx===0} mode={mode}/>
          : <QuestionCard data={q} isFavoritible={isFavoritible}
displayToast={displayToast color={colors[Math.floor(Math.random() *
colors.length)]} mode={mode}/>
          }
      </Card>
      )}
  </CardWrapper>
);
```

Redux

We used react-redux along with react-thunk to handle global state management of the app.

Redux Ducks

For the Redux store, we used **“Redux Ducks”** as an engineering pattern to structure our Redux reducers. Redux ducks suggests that we put related **actions**, **actions creators** and **reducers** into a single file. This makes it really easy to see how each action is called and how it affects the state of the store.

Since all related functions reside in close proximity to one another, yet are clearly segregated into their own independent responsibilities, it made working and reasoning with the code a lot clearer.

Stores

Next, the store of react-redux allowed us to manage global state, without having to pass down props to each component.

There are 3 stores configured:

1. Solo Store
2. Questions Store
3. Mode Store

Solo Store

The Solo store manages information about the Solo mode of the app, including user responses to the daily Solo questions, and whether Solo mode is available for the day.

On each launch of the application, information about Solo mode is read from the localStorage and cached in this store. Then, subsequent components which rely on this information (**Profile**, **SoloGame** components) will have access to the values.

```
const initialState = {
  soloReady: true,  // whether Solo Mode is available (resets every day)
  loading: false,   // for rendering
  hasErrors: false, // for rendering
  journey: [],      // fetches questions stored in user Journey
  questions: []     // fetches daily questions to render
};
```

Questions Store

Since users can only play in one game mode at a time, a global state is used to store the questions retrieved from the question bank.

When users enter a different mode, a new set of questions is queried asynchronously. Otherwise, questions are retrieved from the cache.

```
const initialState = {
  loading: false,   // for rendering
  hasErrors: false, // for rendering
  questions: []     // contains current array of questions
};
```

Mode Store

Finally, there is a global state to store the user's current game mode. Initially during development, this state was passed as a prop throughout the application (since Components such as **NavBar** etc. relies on it.)

However, it was refactored in Milestone 3 into a global state store, which made the code a lot clearer and easy to reason about.

```
const initialState = {
 mode: 'profile'
};
```

CSS

As a challenge, no libraries or CSS frameworks were used. We chose to write the CSS using BEM (Block-Element-Modifier) methodology, which is a semantic way to write naming conventions, while keeping CSS modular. Following this method sped up development time tremendously.

Writing CSS without libraries also made the JSX markup in React a lot neater and easier to reason with.

The following rules were used:

Block - a block is a layout or a container that does not handle state. These refer to UI elements. For example, `.modal`, `.menu`, `.question-card` are examples of block components

Element - elements refer to the stateful components within a layout. If a `question-card` is a UI layout container, then it contains elements such as `.question-card__title`, `.question-card__icons-container` and `.question-card__icon`

Modifier - a modifier is some sort of visual state that a component can have. For example, `.msg__bubble--send` and `.msg__bubble--receive` are examples of components with modifiers.

A variety of modern CSS elements were used, including **flexbox**, **css variables** and **css animations**, ultimately keeping the code clean and light.

Because each component is logically named and descriptive, it is easily searchable within the stylesheet. Thus, we have decided to keep all styling within a single index.css file.

Nevertheless, the stylesheet is appropriately sectioned such that it can be compartmentalized in the future.

Our stylesheet is organized according to these sections

1. Global variables, CSS reset
2. Site-wide styles and classes
3. Menu
4. Profile
5. Game
6. Solo Mode
7. Picture Card
8. Question Card
9. Online

Statistics from CSSStats.com

SPARK

File size
21KB

Gzipped file size
4KB

	SIZE	GZIPPED
Basscss	10kb	2kb
Tachyons	72kb	13kb
Foundation	119kb	16kb
Primer	140kb	22kb
Bootstrap	123kb	23kb
Bulma	186kb	24kb

Overview

RULES	SELECTORS	DECLARATIONS	PROPERTIES
126	145	550	78

Total Selectors by Type

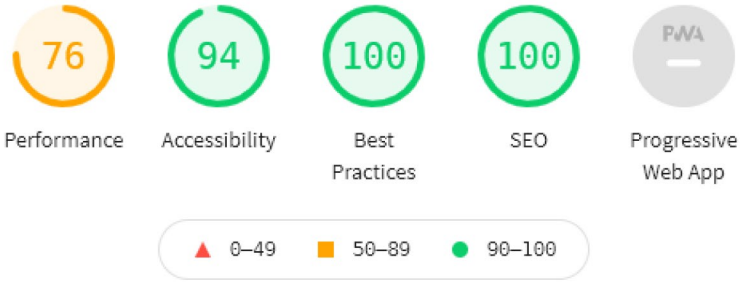
Selectors are the part of a CSS ruleset that describes what elements in a document the rule will match.

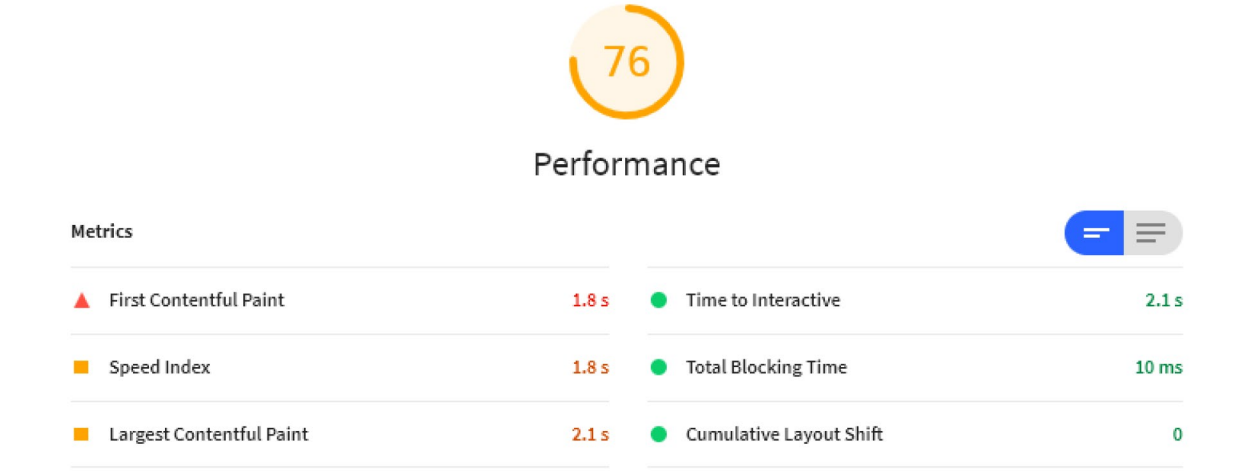
ID	CLASS	PSEUDO CLASS	PSEUDO ELEMENT
0	127	12	7

Performance

Using **Google Lighthouse** to audit our application, we found out that site metrics such as **First Contentful Paint** could be improved further.

Better optimizations could be done in the future, such as using lazy loading, and perhaps even using Next.js as a way to serve the page faster.

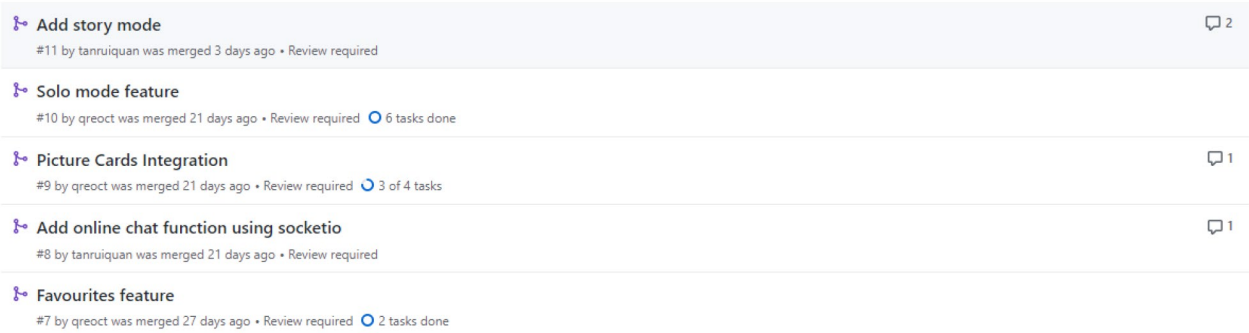




CI/CD

To ensure a **collaborative software engineering process**, we used **Github** in order to have version control. The following practices were used:

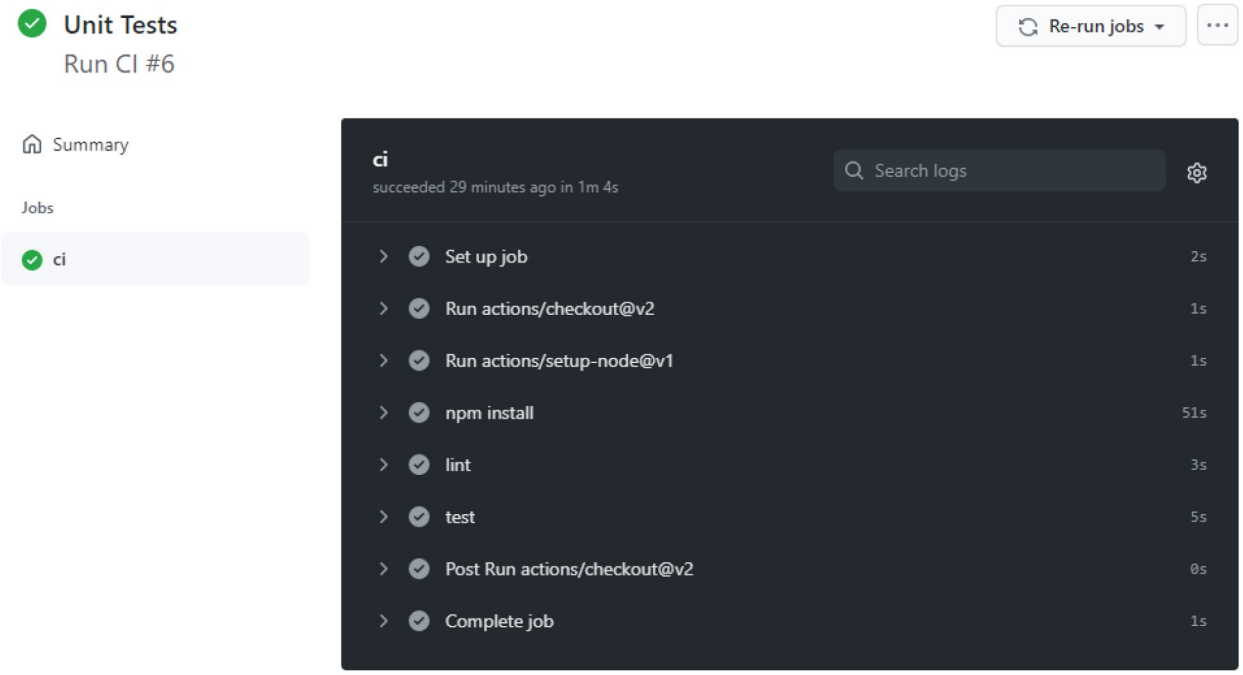
1) **Pull Requests & Branching**



We made use of **pull requests** to organize our changes. Furthermore, direct pushes to the master branch were not allowed, ensuring that no breaking changes could be made.

By developing based on **features required**, we were able to create **new branches** for each new feature developed. Then, by consolidating the changes in pull requests, it was easy to manage **code reviews** and **version control** of the application

2) **Continuous Integration with Github Actions**



Using **Github Workflows**, we **automated running linting checks** and **unit tests** on each push to both frontend and backend repositories. This allowed us to ensure that all code was acceptable quality before merging to master.

3) **Continuous Deployment with Github Actions**
Lastly, the deployment process was streamlined with Github Actions and npm commands. Scripts were written in the package.json file to run the build process of the frontend, and then bundle everything together with the backend to deploy onto **Heroku**.

Using Github's environment secrets, we were able to connect our heroku credentials, and allow **Github Actions** to **automatically build and deploy our application** whenever changes are made.

Implementation Challenges Faced

User Login decision

We deliberated on the decision to implement a login system for SPARK. We opted for an absence of a login system as our design goals for SPARK was to be effortless and users should not have to go through trouble of signing up and logging in to connect with others. Furthermore, we did not want to store 'sensitive' information (such as journal entries) on a database, as we felt that it was a violation of privacy.

However, this came at the expense of users being unable to have their journey and favourited cards saved across multiple devices.

Takeaway: After user feedback (see Section **TESTING: User Feedback**), having saved content was an important part of the experience to many users. We could have implemented a guest login feature, which allows users to experience the app, and **only save their data to an account if they opt in for it.**

This also highlights the importance of having a clear target audience, gathering user feedback, and prioritizing features that would benefit the target audience.

Creating the cards

Both the Question Cards and Picture Cards were important to get right. We tested a number of libraries before settling on react-swipeable-cards. Although it worked for most cases, late in testing we realized that it was difficult to swipe on iPhones (we initially only tested with Android devices).

Takeaway: Although we did not have time to migrate to another library, or write one from scratch, it is important to have options when using external libraries. Furthermore, **it's necessary to test with as many cross-platform devices, and as early as possible**.

Styling

Writing all CSS from scratch instead of using frameworks (like Tailwind, Bootstrap, etc), was a challenge. Specifically, ensuring cross-platform and cross-device consistency, as well as layout issues when using external components. There remains some interaction inconsistencies, especially with extremely small viewports, which make the application not truly responsive.

Takeaway: **Thankfully, clear design guidelines as well as following a methodology like BEM made compartmentalizing the stylesheets easy**. Nevertheless, it might be prudent to use libraries for certain key components (such as navigation bars), which have been tried-and-tested to work.

Having a design idea in mind before creating the frontend is important. Even something low fidelity like wireframes is more than enough in helping to keep the CSS clean.

Online Chat

The real-time chat system is a more advanced feature that required a deeper understanding of networking. Although socket.io abstracted out a lot of the lower implementation details, there was still a rather steep learning curve.

It certainly would have been better if we could implement more quality of life features (eg. notifying the user if the other user is typing) but implementing simple send and receive messages proved to be difficult enough.

Takeaway: Reading the **official documentation** and searching for **online demos/tutorials** helped us a lot in the process of developing the online chat system. In addition, knowing where to search for solutions to your problems (no shame in visiting stackoverflow if it means solving your issues) is really important. Of course, it is always good to dedicate time to learn the fundamentals and go into the details behind the tools we are using which would be beneficial in the long run.

4) Testing

SPARK?!

Unit Tests

Unit Tests are a way to ensure that individual components of the application works well. We wrote Unit Tests for both the Backend and the Frontend. **Full list of unit tests written are listed in the Appendix.**

For all tests, we created a separate testing environment, which is linked to a **testing database** in the backend. This ensures that all tests are **reproducible**, and **start from the same state**.

Backend Unit Testing

Our Backend API endpoints were written using Test Driven Development (TDD). In other words, writing unit tests to describe the functionality of the API before writing the logic of the endpoint.

All 13 Backend API endpoints have Unit Tests written to test for both positive and negative functionality. As an example, `GET /topic/:topics` returns a list of pictures from the given topics. Since the topics are case sensitive, an invalid query such as `GET /topic/NaTuRe` correctly gives a 400 Bad Request response.

Similarly, **tests were written for the socket.io components** of the backend, to ensure that the server and client communication works as intended. socket.io-client library was used to aid this process.

One limitation of testing the backend API was that making calls to the Unsplash API counts towards our API call limit. Although we have applied for Production status, which allows 5,000 API calls per hour, heavy loads of testing during CI/CD might eat into this. Creating a separate API key is also not feasible since it simply postpones the problem..

Frontend Unit Testing

Not many unit tests were written for the frontend. Using react-testing-library, tests were written as a proof-of-concept to check whether elements were rendered properly and whether internal states worked well.

For example, the **PictureCard, PictureGallery, QuestionCard, QuestionStack** components were tested to ensure that information was properly rendered with the given props.

One limitation of our frontend testing was that only the **Cards** components were tested, with a **68.75% code coverage** as reported by jest. While unit tests were important, we felt that Integration Tests were especially important to ensure the components of the frontend worked well together. Future work would have to dedicate more resources to writing Unit Tests in the frontend.

Integration Tests

An Integration Test was written as a proof-of-concept for the Frontend in order to verify that components work well together. Specifically, the **SoloInput** component was thoroughly tested using **react-testing-library** and **Jest** to ensure 100% code coverage for the Component.

The **SoloInput** component is used for Solo mode. It receives data, which is an object that contains a question and optional pic field. Within this component is a textbox that users can fill in their reflections. Lastly, there is a submit button which saves the reflection to user's Journey.

To fully test this component, we wrote test cases for:

1. Rendering **SoloInput** with only a question
2. Rendering **SoloInput** with pictures
3. Interacting with pictures
    a. Ensuring that picture must be selected before proceeding
4. Interacting with textbox & button
    a. Ensuring that empty textbox will not be submitted
    b. Ensuring that contents of textbox are cleared after submission

```
PASS src/components/cards/__tests__/SoloInput.test.js
 SoloInput Component
   √ renders question with default props (176 ms)
   √ interaction with buttons and reflection textarea (63 ms)
   √ renders picture card (11 ms)
   √ interactions with picture card and reflection textarea (60 ms)
```

This Integration Test is fully automatic and can be triggered with `npm run test`. It is also automatically triggered when pushes are made to our Github repository.

End-to-End Testing

Finally, End-to-End (E2E) Testing was also employed in our application. Early on in development, E2E testing was done manually. **In Milestone 3, we focused on creating automated tests and an automated workflow with CI/CD.** This also meant having E2E tests for our app.

E2E tests were written using the **Cypress** library. The objective of these tests are to ensure that **UI elements** are rendered correctly for the user, and that they are able to **achieve basic tasks** based on the features and functionality they would like to use.

Based on the **User Journeys** we defined, it was easy to set up the tasks for the tests to run.

Journey 1

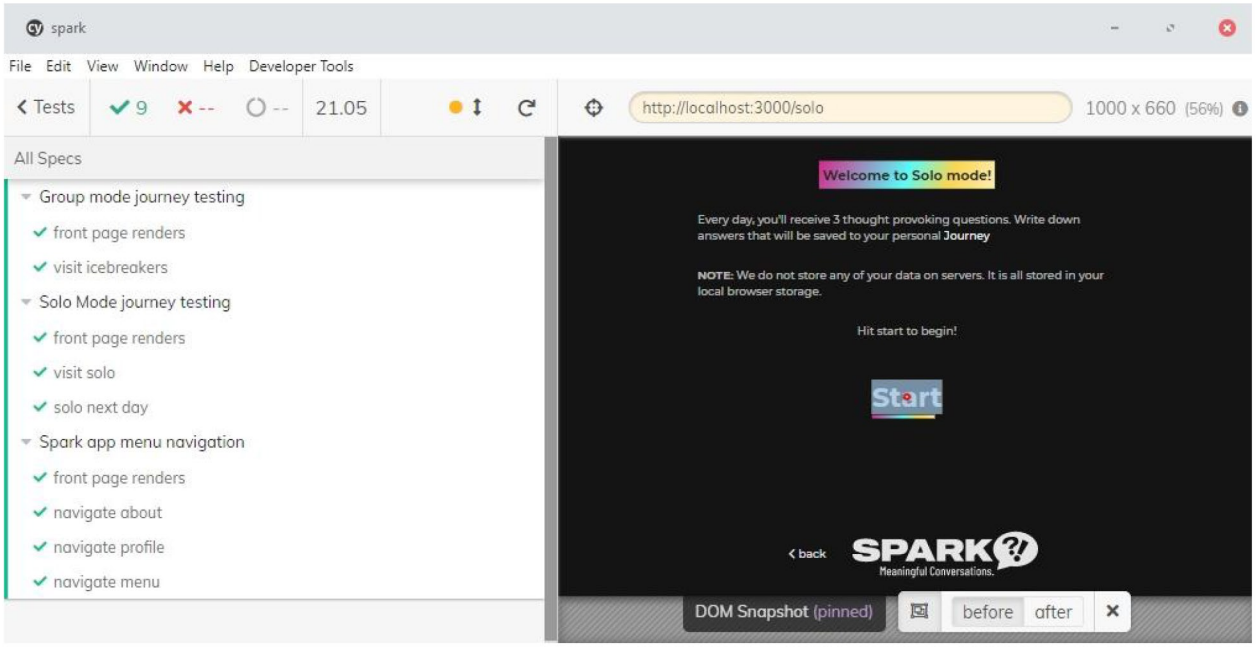
Solo Mode *(receive daily questions)* ➜ user reflections end up in **Journey** on the **Profile Page**

Testing Plan 1

1. Open app
2. Navigate to Profile, ensure that user Journey is empty
3. Ensure that there is a prompt for users to attempt Solo Mode
4. Navigate to Solo Mode
5. Ensure that onboarding modal component renders
6. Ensure that onboarding modal component can be re-opened
7. Attempt 1st Question and Save
8. Navigate to Profile, ensure that 1st response is saved
9. Navigate to Solo Mode, attempt remaining questions
10. Ensure that no prompt for users to attempt Solo Mode once all 3 questions are complete
11. Navigate to Profile, ensure that user Journey contains new entries

Testing Plan 2

1. Run previous Testing Plan (TP)
2. Navigate to Profile, ensure that user Journey contains previous entries
3. Ensure that no prompt for users to attempt Solo Mode
4. Navigate to Solo Mode, ensure that Mode is disabled
5. Modify localStorage that tracks user daily attempts
6. Navigate to Solo Mode, user should be able to attempt it
7. Attempt all 3 questions from Solo Mode
8. Navigate to Profile, ensure that user Journey contains new entries



Screenshot of in-progress automated E2E Testing using Cypress

Journey 2

Group Mode ➜ Question Bank ➜ Favourites feature

Testing Plan

1. Open app
2. Navigate to Profile, ensure that 0 cards are favourited
3. Navigate to Group Mode and select Icebreakers
4. Ensure that onboarding modal component renders
5. Ensure that onboarding modal component can be re-opened
6. Ensure that Question Card is rendered with Favourites button
7. Ensure that clicking on Favourites button renders Toast notification component
8. Ensure that Favourites icon is changed correctly
9. Navigate to Profile, ensure that 1 card is favourited
10. Open up Favourited cards Component
11. Unfavourite the card
12. Ensure that card is no longer in Favourites

Journey 3

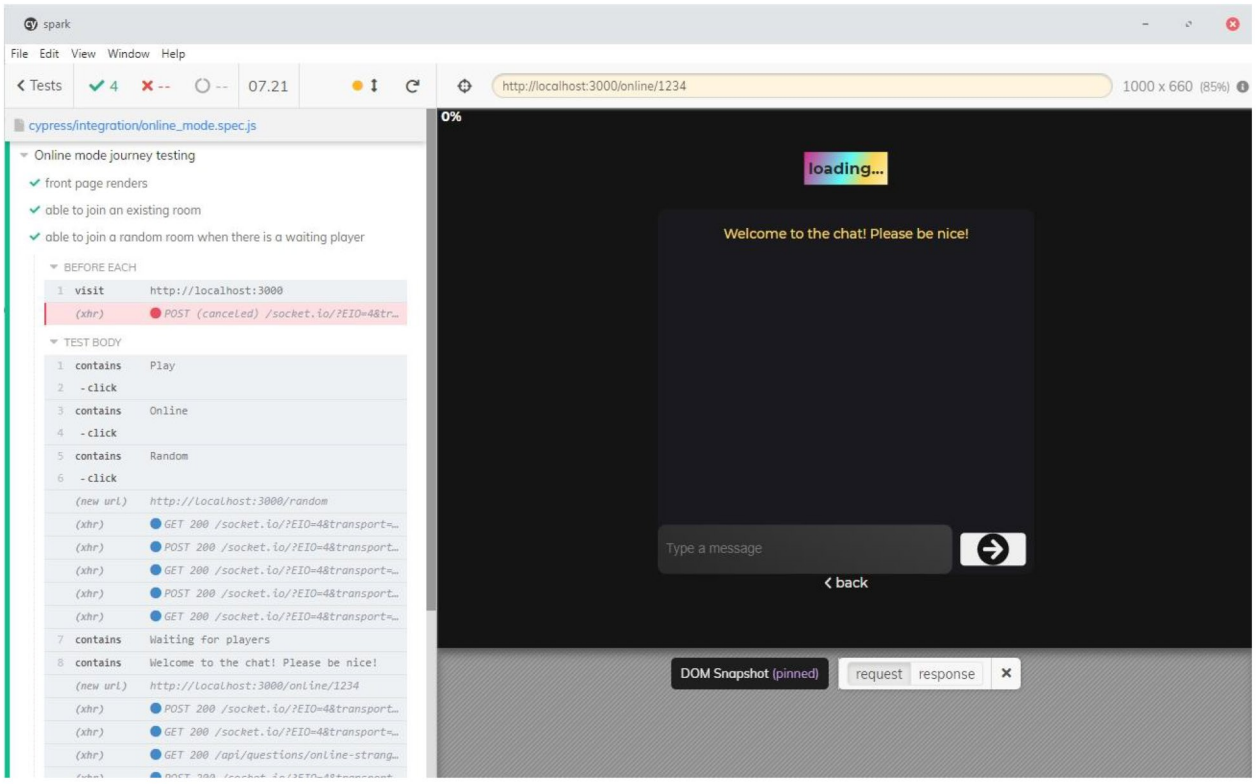
Online Mode with **Real-Time Chat** ➜ Matchmaking with **Friends** and **Strangers**

Testing Plan 1

1. Open app
2. Navigate to Online Mode and select Join
3. Enter the room code '1234' (Room created in test environment)
4. Ensure player enters the chat
5. Ensure questions are generated
6. Ensure player is able to type a message
7. Ensure player is able to click next
8. Ensure player receives a prompt
9. Player leaves room

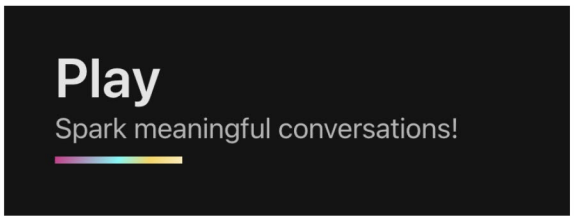
Testing Plan 2

1. Open app
2. Navigate to Online Mode and select Random
3. Ensure that "Waiting for players..." animation is rendered
4. Ensure player enters the chat
5. Ensure questions are generated
6. Ensure player is able to type a message
7. Ensure player is able to click next
8. Ensure player receives a prompt
9. Player leaves room



User Feedback

For Milestone 3, as part of Testing, we engaged 2 potential users with the latest working prototype to gather feedback on SPARK.

User 1	
Likes	He likes the minimalist style and the black background. He finds the app aesthetically pleasing. He pointed out that having the dates shown in the user journey is a great way for users to look back at previous reflections and see how their thoughts evolved over time. He particularly likes this function. He really likes the questions. Overall, he enjoyed using this app and requested for SPARK when it is ready for production.
Dislikes	(Both feedbacks refers to the button below)  He stated that the buttons do not look like buttons as there are no clear borders. He did not realise it was a button at first. The colour bar can easily be misunderstood as a progress bar. He thought it was some kind of progress bar at first.
Suggestions	In solo mode, consider including a congratulations message or confetti upon completing the 3 questions. This helps to build positive reinforcement. **(Implemented thereafter)** Would be nice to have icons beside each button in the menu page. In solo mode, when filling in the reflection, it would be better for the enter button to serve the purpose of saving the response as opposed to adding a new line. Would be nice to have an account so that the reflections saved in the user journey can be shared across devices.

<table>
<tr><th colspan="2">User 2</th></tr>
<tr><td>Likes</td><td>Of all the modes available, he enjoyed solo mode the most. He thinks that the questions are thought-provoking.

He thinks that the words are clear. Words can be read from far away as the font size is big enough.</td></tr>
<tr><td>Dislikes</td><td>Contrary to user 1, he thinks the black background is too dark, noting that most webapps have a lighter background.

He did not enjoy the experience of swiping the cards (on laptop) as it is too tedious.

He mentioned that “The fonts (on the menu pages) makes me feel like I’m doing a survey”.

He thinks the novelty of this game would wear off pretty soon.</td></tr>
<tr><td>Suggestions</td><td>Clicking the SPARK logo should bring the user back to the homepage. Currently, it does nothing. (Implemented thereafter)

Clicking the back button when in game (be it solo, group or online) should not bring users back to the homepage. Instead, it should bring users back to the respective menu page.

Light mode or some way to customise the UI

Considering making the play button bigger as he thinks the play button should capture the user’s attention. He suggested the following layout for the homepage:

<table><tr><td colspan="2">Play</td></tr><tr><td>Profile</td><td>About</td></tr></table>
He also suggested the following layout for the group menu page:

<table><tr><td>Story (2 players)</td><td>Icebreakers</td></tr><tr><td>Deep Questions</td><td>This of That</td></tr></table></td></tr>
</table>

After gathering the suggestions from our potential users, we added the **congratulations message after completing the 3 reflections** and **routing the SPARK logo to the homepage.** As for the rest of the suggestions, we either did not implement them as we felt that it was not necessary or that we did not have enough time to implement them.

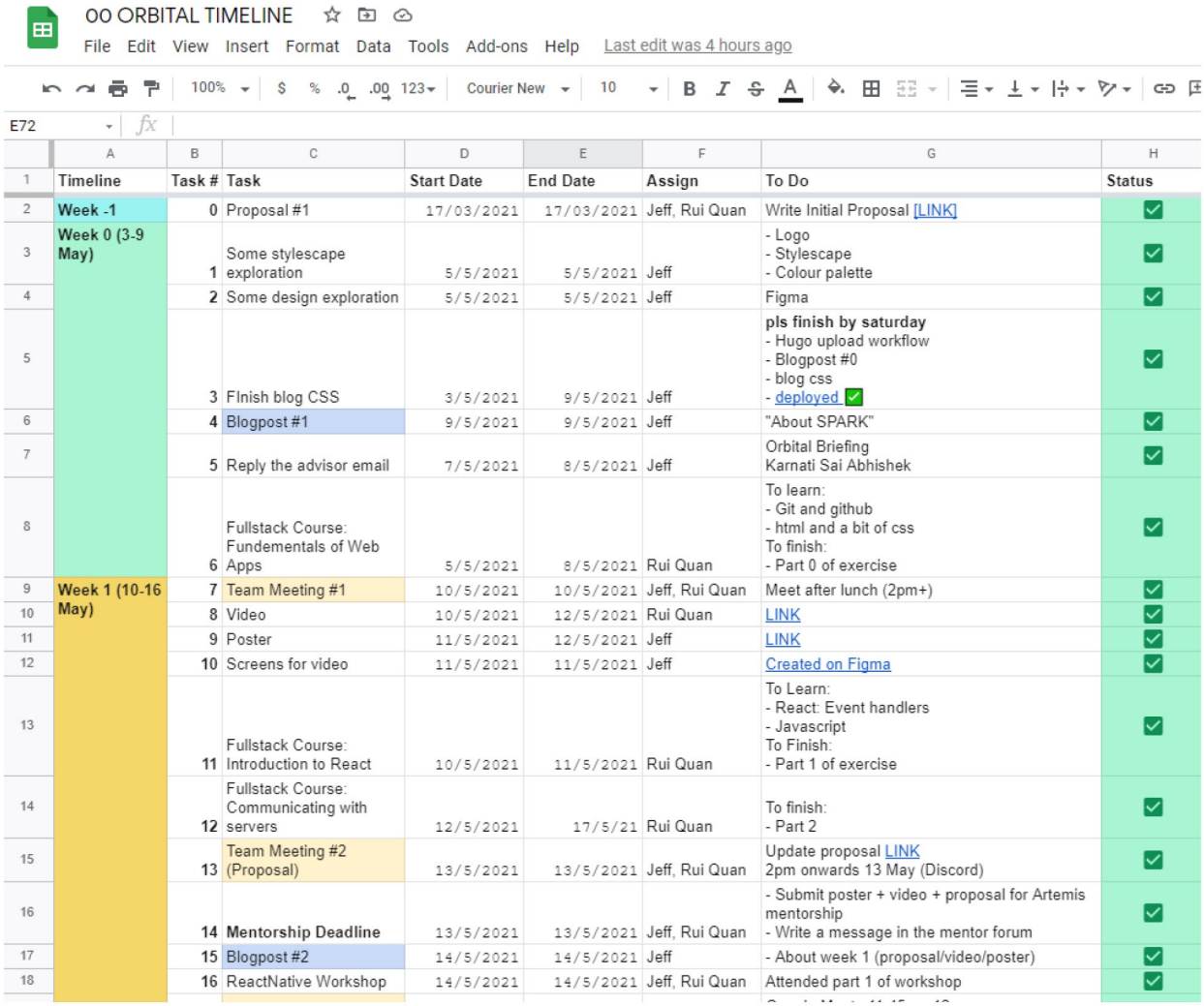
Nevertheless, we appreciate the responses and recognize the importance of **iteration** and **user testing** as a process.

5) Documentation

SPARK?!

Project Management

In order to track our progress and tasks internally, we used a collaborative **Google Sheets** document. This allowed both of us to keep track of what we were doing, and also serves as a record of our project log and timeline.



00 ORBITAL TIMELINE

File Edit View Insert Format Data Tools Add-ons Help Last edit was 4 hours ago

	Timeline	Task #	Task	Start Date	End Date	Assign	To Do	Status
2	Week -1	0	Proposal #1	17/03/2021	17/03/2021	Jeff, Rui Quan	Write Initial Proposal [LINK]	☑
3	Week 0 (3-9 May)	1	Some stylescape exploration	5/5/2021	5/5/2021	Jeff	- Logo - Stylescape - Colour palette	☑
4		2	Some design exploration	5/5/2021	5/5/2021	Jeff	Figma	☑
5		3	FInish blog CSS	3/5/2021	9/5/2021	Jeff	**pls finish by saturday** - Hugo upload workflow - Blogpost #0 - blog css - deployed ✅	☑
6		4	Blogpost #1	9/5/2021	9/5/2021	Jeff	"About SPARK"	☑
7		5	Reply the advisor email	7/5/2021	8/5/2021	Jeff	Orbital Briefing Karnati Sai Abhishek	☑
8		6	Fullstack Course: Fundementals of Web Apps	5/5/2021	8/5/2021	Rui Quan	To learn: - Git and github - html and a bit of css To finish: - Part 0 of exercise	☑
9	Week 1 (10-16 May)	7	Team Meeting #1	10/5/2021	10/5/2021	Jeff, Rui Quan	Meet after lunch (2pm+)	☑
10		8	Video	10/5/2021	12/5/2021	Rui Quan	LINK	☑
11		9	Poster	11/5/2021	12/5/2021	Jeff	LINK	☑
12		10	Screens for video	11/5/2021	11/5/2021	Jeff	Created on Figma	☑
13		11	Fullstack Course: Introduction to React	10/5/2021	11/5/2021	Rui Quan	To Learn: - React: Event handlers - Javascript To Finish: - Part 1 of exercise	☑
14		12	Fullstack Course: Communicating with servers	12/5/2021	17/5/21	Rui Quan	To finish: - Part 2	☑
15		13	Team Meeting #2 (Proposal)	13/5/2021	13/5/2021	Jeff, Rui Quan	Update proposal LINK 2pm onwards 13 May (Discord)	☑
16		14	**Mentorship Deadline**	13/5/2021	13/5/2021	Jeff, Rui Quan	- Submit poster + video + proposal for Artemis mentorship - Write a message in the mentor forum	☑
17		15	Blogpost #2	14/5/2021	14/5/2021	Jeff	- About week 1 (proposal/video/poster)	☑
18		16	ReactNative Workshop	14/5/2021	14/5/2021	Jeff, Rui Quan	Attended part 1 of workshop	☑

Timeline

Liftoff	Wk 1 10/5 - 16/5

Exploration stage
- Design exploration
  - Visual design of UI/UX
  - Feature design
- Learning about tools (React)
	Wk 1
	10/5 - 16/5

Initial Development - Set up production environment (front & backend) - Create page navigation	Wk 2 - 3 17/5 - 30/5
Question Cards - Proof of concept for question cards	Wk 3 23/5 - 30/5
Picture cards - Integration with Unsplash API - Picture Card component user interactions	Wk 3 17/5 - 30/5
Evaluation Milestone 1 - Proof of concept (able to view 1 randomly selected question card)	Wk 4 31/5 - 6/6
Group mode - Finalise question bank - Integrate question and picture cards into Icebreakers, This or That & Deep Conversations	Wk 4 - 5 31/5 - 13/6
Solo mode - Integrate cards into solo mode - Implement Reflection submission feature	Wk 4 - 5 31/5 - 13/6
Profile page - Implement Favourites feature - Implement Journey feature	Wk 5 - 6 7/6 - 20/6
Online mode - Implement create and join room features - Implement chat function - Implement random room function	Wk 6 - 8 14/6 - 4/7
Evaluation Milestone 2 - Proof of concept for SPARK	Wk 8 28/6 - 4/7
Responsive Design - Improve UI so that it works well with all devices	Wk 9 5/7 - 11/7
Testing - Create unit tests for React frontend using jest - User testing by letting people use the app - Refactoring code based on feedback/bugs - Implementing CI/CD	Wk 9 - 11 5/7 - 1/8
Evaluation Milestone 3 - All features of SPARK are fully tested and implemented!	Wk 11 26/7 - 1/8
Splashdown	Wk 15 25/8

Project Log

Liftoff (10/5 - 16/5)

Task	Date	Jeff (hrs)	Rui Quan (hrs)	Remarks
Advisor Meeting	9/5	1	1	
Publicity materials	10/5-12/5	8	2	Video, Poster, Design work
Team meeting	10/5	2	2	Laying out development plans
Total	-	11	5	

Milestone 1 (17/5 - 31/5)

Task	Date	Jeff (hrs)	Rui Quan (hrs)	Remarks
Learning React	5/5-31/5	10	40	https://fullstackopen.com/en/
Spark backend	25/5-31/5	1	5	Set up endpoints in backend, MongoDB, and integrate Unsplash API
Spark frontend	22/5-31/5	11	-	Set up routing, basic components, project structure & scope
Milestone 1 meeting	31/5	3	3	Standardised naming convention Decided file structuring convention Discuss logic between backend and frontend
Proposal & designs	22/5-31/5	10	-	Milestone 1 report, app mockups
Total		35	48	

Milestone 2 (1/6 - 28/6)

Task		Date	Jeff (hrs)	Rui Quan (hrs)	Remarks
Learning socket.io		19/6-20/6	1	10	In order to implement a real time chat feature, we need a bidirectional connection which socket.io provides
Spark frontend	Solo	7/6 - 20/6	25	-	Functionality to generate daily questions and save to user Journey
	Group	31/5 - 13/6	25	10	Complete Question Bank, ability to swipe Questions, like & share
	Online	14/6-28/6	8	20	Implement the routing for Create, Join and Random room Setup socket.io on the client side for the real time chat feature Integrate questions and the syncing of next button
Spark backend	Online mode	14/6-28/6	-	10	Setup socket.io for the server side
	Update endpoints	17/6-17/6	2	2	Add new endpoints for fetching for fetching questions and pictures
Milestone 2 meeting		14/6	5	5	Standardise design considerations
Proposal, documentation & designs		21/6 - 28/6	10	4	Milestone 2 report, app mockups, blogging
Publicity materials	Orbital Video	28/6	-	5	Video to explain the different features of our app
Total			76	66	

Milestone 3 (29/6 - 25/7)

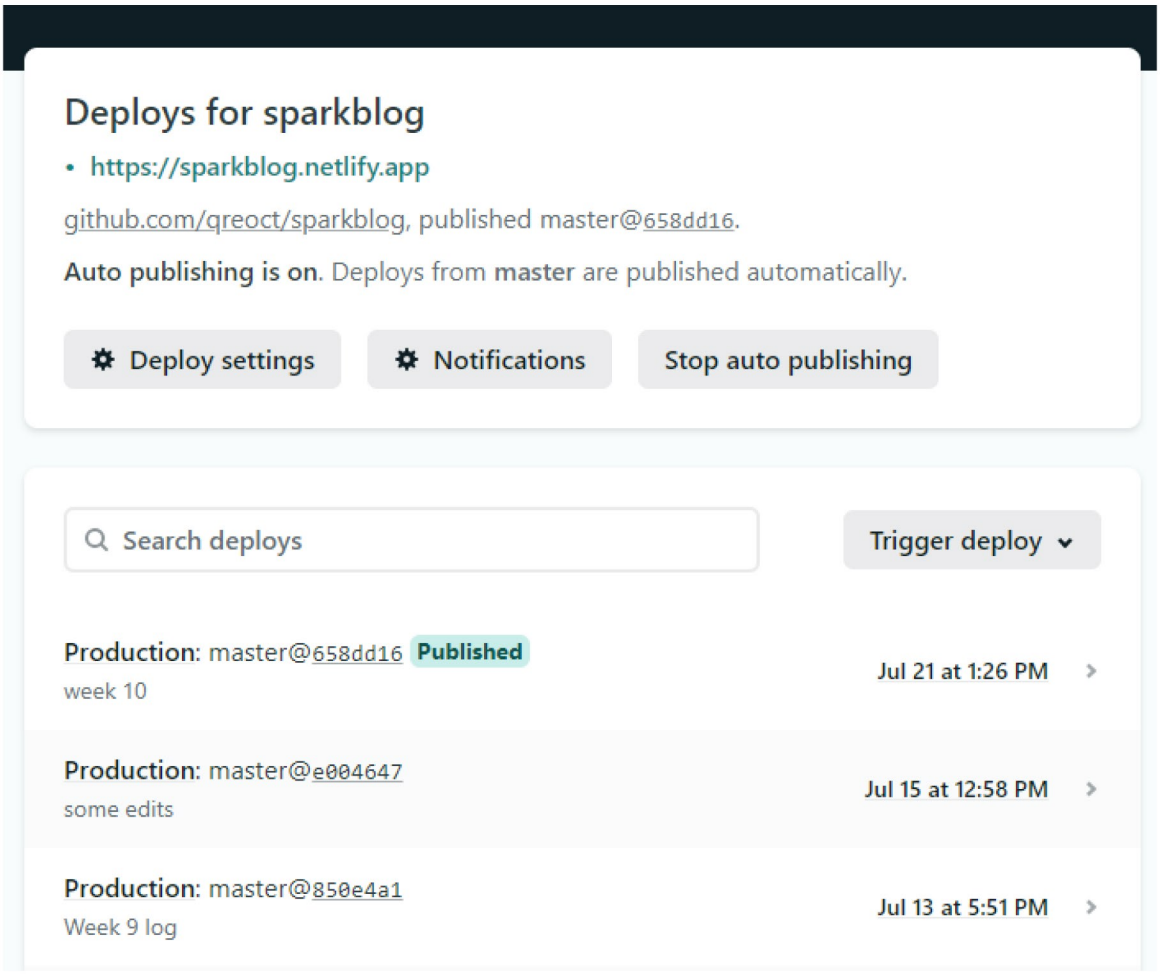
Task		Date	Jeff (hrs)	Rui Quan (hrs)	Remarks
Learning testing tools		29/6-18/7	2	8	react-testing-library, supertest, Cypress
Refactoring code		5/7 - 18/7	4	3	Cleaning up frontend and backend code for readability and performance
Writing and running tests	Unit Test	12/7 - 18/7	3	3	
	Integration Test	12/7 - 18/7	4	-	
	E2E Test	12/7 - 18/7	8	5	
	User Feedback	12/7 - 18/7	2	2	
Implementation of CI/CD		12/7 - 18/7	5	1	Discovered through testing
Proposal, documentation		29/6 - 25/7	20	12	Milestone 3 Report, blogging
Publicity materials	Orbital Video	18/7 - 25/7	-	5	Update our video to showcase new features, and polish content
Total			48	39	

Total hours for Orbital	170	158

Blog

Although it wasn't necessary to develop a blog for our Orbital project, we wanted a way to **document our development journey** and **personal learnings,** while also **embarking on something new**.

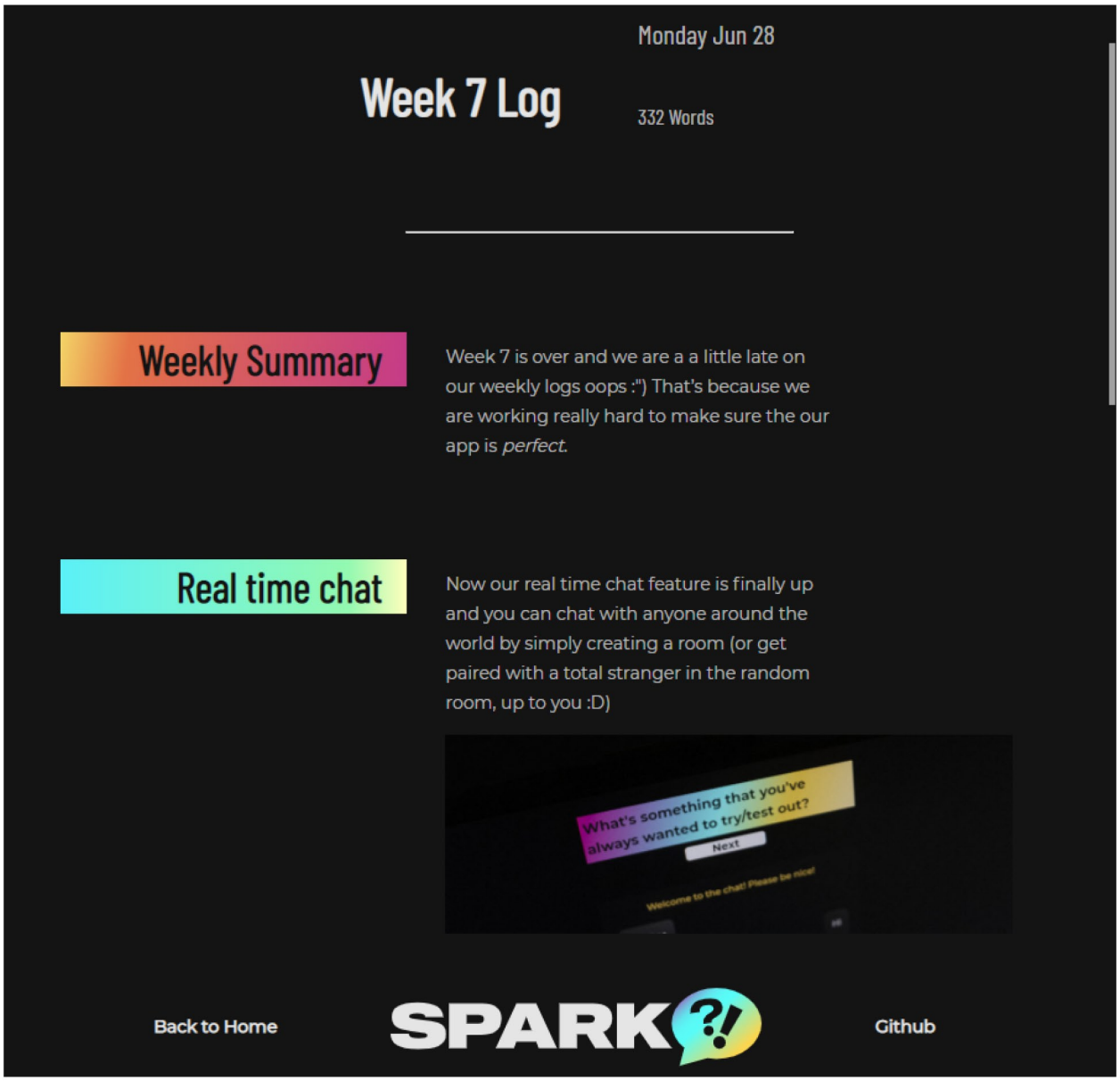
The blog was written using **Hugo**, a static site generator. We simply had to write our post content in Markdown, which would be rendered into HTML pages. Then, uploading the entire directory into Github, we were able to connect with **Netlify**'s automated deploys to continuously publish our blog onto the Internet at sparkblog.netlify.app



In our blog, we made **weekly updates of our team's progress throughout Orbital**. We were able to combine rich media like images, .gifs, and code blocks, to make the content more engaging.

We learnt how to use Hugo to write **content templates**, such that each of the posts had a structure that made it easy to start writing. We also learned the basics of **theming** with Hugo, writing the layout for the blog from scratch.

Ultimately, learning how to use Hugo was a fruitful experience, and it also helped us to polish our writing skills.



Appendix

SPARK?!

Design System

colour palette

typography

01

Montserrat

Bold

a b c d e f g h i j k l m
n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9

Regular

a b c d e f g h i j k l m
n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9

02

Barlow Condensed

Regular

a b c d e f g h i j k l m
n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9

logo + logotype

conversations

purpose
dialogue
reflection

+ interrobang

bold
surprise
meaning

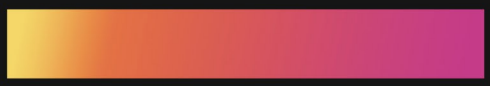
regular

icon

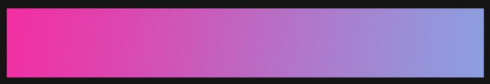
gradient palette

Spark
#d7298b 0%,#59fcf7 50%, #fbd651 74%, #ffffff 120%

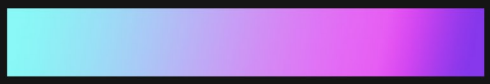
Amber
#f4d688 0%, #e37246 20%, #c53b88 100%



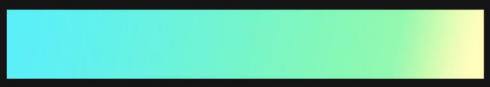
Amethyst
#fe239d 0%, #81ace7 100%

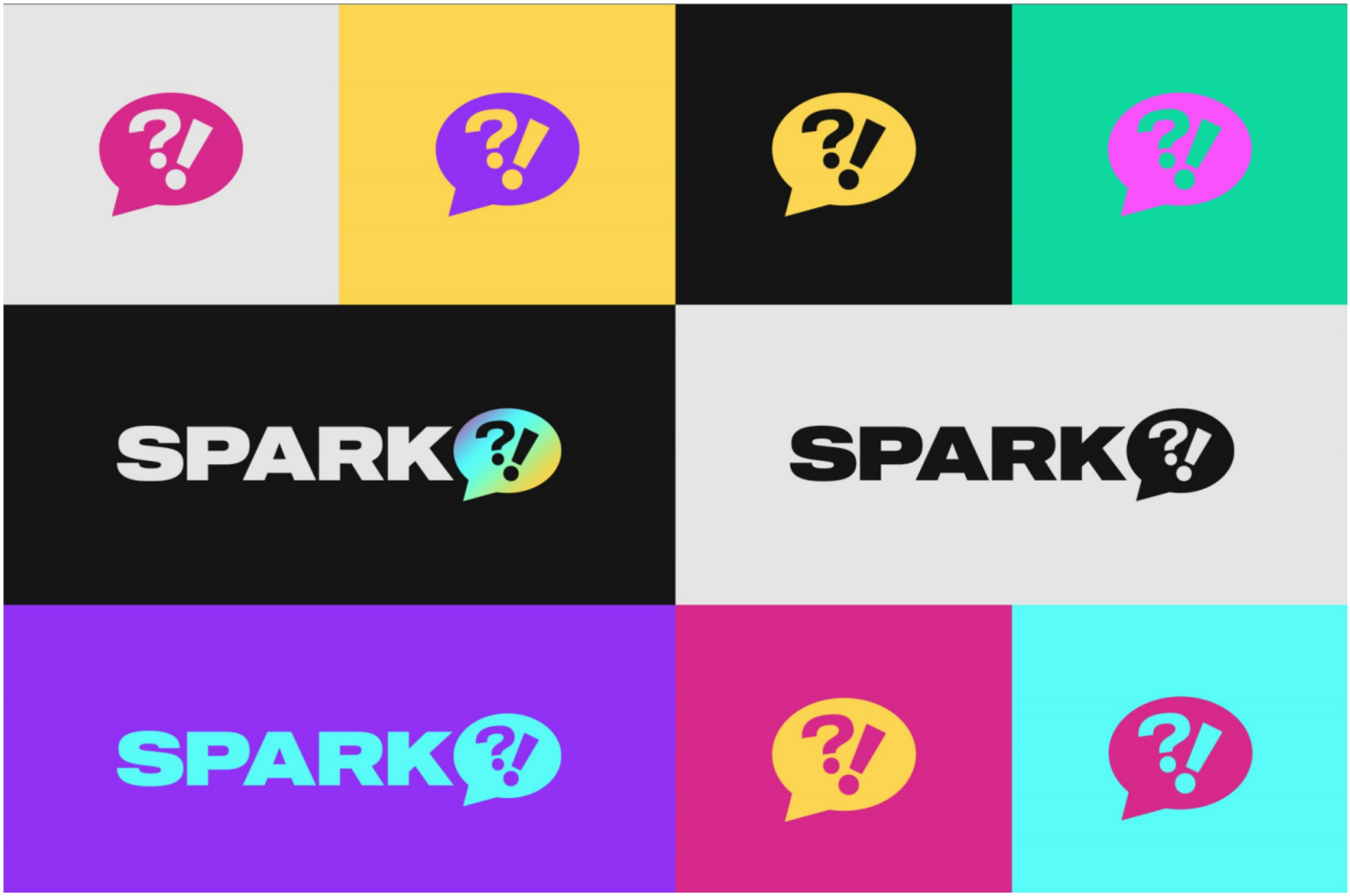


Topaz
#89f8f6 0%, #e65ef4 80%, #8737eb 100%



Jade
#5bf0f7 0%, #93f9b1 83%, #fffebc 100%




SPARK
SPARK
SPARK

Backend Endpoints

This section contains all the backend endpoints, including those not mentioned in the key backend endpoints.

GET /api/questions/{category}/{level}	
Returns questions from the given category and level If no level value given, returns questions from all levels If no level and category values given, returns all questions from database	
Parameters	
Name	Description
category string (path)	Category values that need to be considered for filter Available values: *icebreakers* *this-or-that* *deep* *solo* *online-stranger* *online-room* *story*
level string (path)	Level values to need to be considered for filter Available values: *1, 2, 3*
Responses	
Code	Description
200	Successful operation **Example value** `[ { "category": [ "icebreakers", "online-stranger", "online-room", "story" ], "question": "What's something that you've always wanted to try/test out?", "level": "1", "canPicture": false,`

	<pre>{
	  "id": "60d9d3979b9fc39777971ebe"
	  },
	...other questions
	]</pre>

400	Invalid category or level values supplied

POST /api/questions	
Parameters	
Name	Description

body ***required**
object
(object)
	Question object that needs to be added to the database
	<pre>{
	"category": [
	"icebreakers",
	"this-or-that",
	"online-room",
	"online-stranger",
	"story"
	],
	"question": "Would you rather be too hot or cold?",
	"level": "1",
	"canPicture": false,
	"this": "hot",
	"that": "cold"
	}</pre>

Responses	
Code	Description

200	Successful operation
	Example value
	<pre>{
	"category": [
	"icebreakers",
	"this-or-that",
	"online-room",
	"online-stranger",
	"story"
	],</pre>

	<pre>{
	  "question": "Would you rather be too hot or cold?",
	  "level": "1",
	  "canPicture": false,
	  "this": "hot",
	  "that": "cold",
	  "id": "60e95e74e337c30d5dcd5d2e"
	}</pre>

400	Invalid question

GET /api/pictures/topic/{topics}/{count}/{orientation}	
Returns pictures from the given topics Multiple topic values can be provided with comma separated strings If orientation value is not given, pictures returned can be of any orientation If orientation and count values are not given, returns one picture	
Parameters	
Name	Description
topics ***required** string (path)	Topic values that need to be considered for filter Available values: *animals* *athletics* *architecture* *arts-culture* *business-work* *current-events* *experimental* *fashion* *film* *food-drink* *health* *history* *interiors* *nature* *people* *spirituality* *technology* *textures-patterns* *travel* *wallpapers*
count string (path)	The number of pictures to return Default: 1, Max: 30
orientation	Orientation values that need to be considered for filter

string (path)	Available values: *landscape*, *portrait*, *squarish*
Responses	
Code	Description
200	Successful operation **Example value** `[{ "id": "wQk0pSCyw94", "created_at": "2021-01-18T17:36:36-05:00", "updated_at": "2021-07-08T00:48:05-04:00", "promoted_at": null, "width": 2784, "height": 4176, "color": "#d9d9d9", "blur_hash": "LaOVy1R+oexZ_4oLj[kCAIxZNGNG", "description": "model: @yulyapulya262 ", "alt_description": "woman in gray hoodie standing near body of water during daytime", "urls": { "raw": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1", "full": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=srgb&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=85", "regular": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=1080", "small": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=400", "thumb": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=200" }, ...other informations about the picture }, ...other pictures]`
400	Invalid topic or orientation values supplied

GET /api/pictures/query/{query}/{count}/{orientation}

Returns pictures with the given search query
If orientation value is not given, pictures returned can be of any orientation
If orientation and count values are not given, returns one picture

Parameters

Name	Description
query ***required** string (path)	Filter pictures matching a search term
count string (path)	The number of pictures to return Default: 1, Max: 30
orientation string (path)	Orientation values that need to be considered for filter Available values: *landscape, portrait, squarish*

Responses

Code	Description

200	Successful operation
	Example value
	
	[
	  {
	    "id": "wQk0pSCyw94",
	    "created_at": "2021-01-18T17:36:36-05:00",
	    "updated_at": "2021-07-08T00:48:05-04:00",
	    "promoted_at": null,
	    "width": 2784,
	    "height": 4176,
	    "color": "#d9d9d9",
	    "blur_hash": "LaOVy1R+oexZ_4oLj[kCAIxZNGNG",
	    "description": "model: @yulyapulya262 ",
	    "alt_description": "woman in gray hoodie standing near body of water during daytime",
	    "urls": {
	
	      "raw": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1",
	      "full": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=srgb&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=85",

<table>
<tr><td></td><td><pre>
      "regular":
"https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=1080",
      "small":
"https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=400",
      "thumb":
"https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=200"
    },
  ...other informations about the picture
  },
...other pictures
]
</pre></td></tr>
<tr><td>400</td><td>Invalid orientation value supplied</td></tr>
</table>

<table>
<tr><th colspan="2">GET /api/pictures/random/{count}/{orientation}</th></tr>
<tr><td colspan="2">Returns random pictures
If orientation value is not given, pictures returned can be of any orientation
If orientation and count values are not given, returns one picture</td></tr>
<tr><td colspan="2">Parameters</td></tr>
<tr><td>Name</td><td>Description</td></tr>
<tr><td>count
string
(path)</td><td>The number of pictures to return

Default: 1, Max: 30</td></tr>
<tr><td>orientation
string
(path)</td><td>Orientation values that need to be considered for filter

Available values: landscape, portrait, squarish</td></tr>
<tr><td colspan="2">Responses</td></tr>
<tr><td>Code</td><td>Description</td></tr>
<tr><td>200</td><td>Successful operation
Example value
<pre>
[
  {
    "id": "wQk0pSCyw94",
    "created_at": "2021-01-18T17:36:36-05:00",
    "updated_at": "2021-07-08T00:48:05-04:00",
</pre></td></tr>
</table>

	"promoted_at": null, "width": 2784, "height": 4176, "color": "#d9d9d9", "blur_hash": "LaOVy1R+oexZ_4oLj[kCAIxZNGNG", "description": "model: @yulyapulya262 ", "alt_description": "woman in gray hoodie standing near body of water during daytime", "urls": { "raw": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1", "full": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=srgb&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=85", "regular": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=1080", "small": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=400", "thumb": "https://images.unsplash.com/photo-1611009382913-a50d7ff0b061?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyMzQxMTd8MHwxfHJhbmRvbXx8fHx8fHx8fDE2MjU4MTczNTQ&ixlib=rb-1.2.1&q=80&w=200" }, ...other informations about the picture }, ...other pictures]
400	Invalid orientation value supplied

GET /api/pictures/topics	
Returns all topics	
Parameters	
No parameters	
Responses	
Code	Description
200	Successful operation **Example value**

	[
	  {
	  "id": "J9yrPaHXRQY",
	  "slug": "technology",
	  "title": "Technology",
	  ...other informations about the topic
	  },
	  {
	  "id": "LeuyY5Vto0w",
	  "slug": "summeronfilm",
	  "title": "Summer on Film",
	  ...other informations about the topic
	  },
	  ...other topics
	]

Unit Tests (Backend)

```
PASS tests/picture_api.test.js (5.907 s)
  fetching random pictures
    √ succeed with one picture (384 ms)
    √ succeed with count (305 ms)
    √ succeed with orientation (545 ms)
  fetching pictures with query
    √ succeed with single query (338 ms)
    √ succeed with count (331 ms)
    √ succeed with orientation (349 ms)
  fetching with topics
    √ succeed fetching list of topics (316 ms)
    √ succeed with random topic (436 ms)
    √ fails with invalid case sensitivity (29 ms)
    √ succeeds with multiple topics (470 ms)
    √ succeed with count (462 ms)
    √ succeed with orientation (459 ms)

PASS tests/question_api.test.js
  with the questions currently saved in the database
    √ questions are returned as json (258 ms)
    √ all questions are returned (79 ms)
    √ unique identifier property of the questions is named id (70 ms)
  fetching an array of questions based on category
    √ succeeds with solo mode (99 ms)
    √ succeeds with story mode (92 ms)
    √ succeeds with icebreakers mode (80 ms)
    √ succeeds with deep questions mode (66 ms)
    √ succeeds with this or that mode (79 ms)
    √ succeeds with online mode (88 ms)
    √ fails with statuscode 400 if category does not exist (49 ms)
  fetching an array of questions based on category and level
    √ succeed with solo mode at level 1 (70 ms)
    √ fails with statuscode 400 if level is incorrect (51 ms)

PASS socket.io/tests/socket.test.js
  In online mode, user
    √ should be able to create room (32 ms)
    √ should be able to join room when room code exists (12 ms)
    √ should not be be able to join room when room code don't exist (9 ms)
    √ should stay in the wating room if there is no available waiting room (16 ms)
    √ should enter the chat room when there is an available room (10 ms)
  When in the chat room
    √ server can receive questions from client (11 ms)
    √ client can recieve questions from server (10 ms)
    √ server can receive message from client (10 ms)
    √ client can receive message from server (11 ms)
    √ server can receive next question request from client (16 ms)
    √ client can receive next question request from server (18 ms)
    √ user can leave the room (9 ms)

Test Suites: 3 passed, 3 total
```

Unit Tests (Frontend)

```
 PASS  src/components/cards/__tests__/SoloInput.test.js
  SoloInput Component
    √ renders question with default props (153 ms)
    √ interaction with buttons and reflection textarea (69 ms)
    √ renders picture card (10 ms)
    √ interactions with picture card and reflection textarea (78 ms)

 PASS  src/components/cards/__tests__/QuestionStack.test.js
  QuestionStack component
    √ renders correctly only 5 questions with default props (127 ms)
    √ renders picture card when its in front (53 ms)

 PASS  src/components/cards/__tests__/QuestionCard.test.js
  QuestionCard component
    √ renders with default props (24 ms)
    √ renders placeholder div without data (7 ms)
    √ correctly doesn't render favourites when flag is set (6 ms)
    √ correctly shows alternate phrasing in story mode (6 ms)
    √ correctly uses this/that field in this-or-that mode (11 ms)
    √ click on fav functionality (22 ms)

 PASS  src/components/cards/__tests__/PictureGallery.test.js
  PictureGallery Component
    √ renders 3 pictures properly with default props (41 ms)
    √ supports user selecting pictures (16 ms)
    √ doesn't allow submit button to be pressed when picture not selected (19 ms)

Test Suites: 4 passed, 4 total
Tests:       15 passed, 15 total
Snapshots:   0 total
Time:        4.266 s, estimated 10 s
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
src/components/cards	68.75	70.93	66.67	69.43	
PictureCard.js	76	86.67	100	76	19,22-23,29-30,34
PictureGallery.js	84.21	80	85.71	84.21	20,28,37
ProgressBar.js	25	0	0	25	5-18
QuestionCard.js	87.8	85.71	66.67	87.5	48-55
QuestionStack.js	100	66.67	100	100	22-36
SoloInput.js	100	100	100	100	

Test coverage as reported by Jest