# Kingsley Speed Dating Application T661

### What technological obstacles did you have to overcome? 350 words max – 50 lines of 78 characters

I designed and developed a web application for speed dating. I researched the market and saw that the application was marginally represented, but there was nothing really available for Toronto. These applications are specific to locals. The main technical challenge was finding an application configuration that could register users, give them sound and video, and recombine them in new pairs every five minutes. I had a number of open-source APIs to work with, and a number of backend options for the core application. I had to experiment with many possibilities and configure my application multiple times to find the best working front and backends, both as individual performers — front and back — and cooperation and collaboration. The application uses real time information. I also had to configure an administrator overlay, so that the organizer of the speed dating session could see all conversations from a management console. I had to try various combinations of APIs to see what went well together and what was easiest to configure, scale, and maintain. Then there was ease of coding and resource usage. I had to configure privacy so that user details were private until two people agreed to meet off the app. I had to attempt numerous WebSocket’s configurations as the application and its APIs evolved. The entire journey of developing the application was a series of experimental developments that refined the process until I arrived at what I deemed to be the best configuration, accounting for speed, ease of use, security, user administration, and most functional and flexible APIs for future changes.

### What work did you perform in the tax year to overcome those technological obstacles?   700 words max - 100 lines of 78

I selected the WebSocket’s protocol standardized by the Internet Engineering Task Force (IETF) for real time information, so that the backend of the application handles timing, and creates new rooms. WebSocket’s sends the person to a new date meeting room — which is a private video call between speed daters timed to five minutes — by pushing the timing information from the backend to the front end. My first API integration was Twilio ®, which I wanted to use for its Video chat integration, and I liked this option because it was peer to peer video calling, which worked quite well when a call was between two people, but when a call got to four or five participants the video quality deteriorated significantly. There was no information I could find that warned of this limitation. I started extending the Daily API ®. Twilio is an open API, and you have to do a lot of the configuration yourself, playing around with different coding, changing the codecs, optimizing for different things like devices — Apple and Android — and other complexities derived from the open nature of the Twilio API. By extending Daily I automated base configuration. It makes some assumptions that worked well with my application. Again, there was no way to investigate this. I was not able to find any commentary on chat forums or Slack channels or anywhere else. I needed to experimentally determine the best approach. I realized the video quality was a lot better, both two party calls and larger group calls. I had to abandon Twilio because the group call was integral to the application. The speed dating events start with a lobby, which includes group calls where everyone in the event is on the call, and it could be twenty or thirty people. The Daily API required a lot of experimental development and testing to overcome its limitations, switching rooms, for example. The Daily API only allows you to stream video calls. Everything else is custom coding. The Daily API has events. An event starts and stops a call. A call is on a channel. The backend starts and stops a channel the speed dating event begins with everyone on one channel. This is the lobby for the event, and integral to the design of the application. To increase functionality, I had to create the perception of a separate channel for the date callers. This was achieved not by creating new channels, but, rather, by developing a process that would mute the call and video to all but the two people on a digital date. This design was implemented to avoid lags on the calls. Some users have better Internet than others. Each date call is five minutes. Due to lags, some of the dates were far less than five minutes. I spun the API to deliver the appearance of a private call while the two callers on the video date are still in the lobby channel with the original group. Each participant entering the lobby is assigned a token, then that token is used as the unique identifier to separate dates into pairs for the five-minute speed dates, before being moved to a new date. I created an admin login that can view info in real time, see all the matches, see who is speaking with who, with a management console where the admin can see information being inputted, the matches, and the rooms and date pairs changing. The backend creates the room and moves the participants around. The strategy and design for this was complex. The algorithm had to be reworked numerous times. For example, you have five rooms and five men and five women, plus the lobby where they all start. The process required extensive experimentation for handling unanticipated events, for example, an extra person or more joining the lobby room after it has been created. The algorithm had to be redone a number of times to accommodate edge cases. I had to develop a chat service. I designed and coded a chat widget at the bottom right of the screen. I had to build from scratch for this application. For the real time I used Firebase ©, arriving at this after my investigations revealed the open-source chat tools were either built for specific frameworks or were linked to services run by another admin. I coded my reusable chat module with React ®.

### What technological advancements were you trying to achieve? 350 words max - 50 lines of 78 characters

I was trying to achieve a robust speed dating tool for the web, that did not require downloading any software, and could operate within any browser. I wanted excellent video quality, which I knew would require extensive experimental development. I wanted to create a tool that could facilitate a group video call with as many as forty participants, which could also break off into private video calls for timed sessions of five minutes. I wanted to ensure security for users, especially where their personal details related to their phones were concerned. Phone numbers for off app calling and texting required consent. I wanted the participants to be able to text to each other through SMS messaging. And I wanted an admin console that could oversee the lobby and the private conversations. I wanted the best combination of group and one on one calling with excellent video quality and no lag. I wanted excellent security so that participants did not feel their privacy would be compromised and they would leave themselves open to unwanted communication. It was my goal to develop our solution by extending the best of class tool at every level, by contributing to open-source market and open standards and APIs, to create a high performing multi-device application in the speed dating market.

.. https://www.twilio.com/

.. https://firebase.google.com/

.. https://www.daily.co/

.. <https://reactjs.org/>

.. https://rubyonrails.org/