Array types

Note: we have no production usages of array types at this time.

The final type that can appear in a JSON document is an array. An array can contain simple types, or nested schemas.

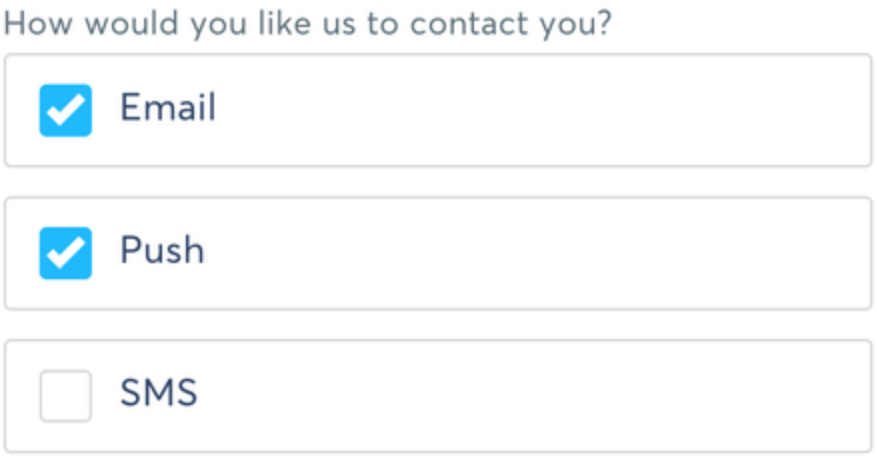
The schema for an array includes a new type “array”, and an items property. The items property contains another JSON schema that tells us what each item in the array should look like

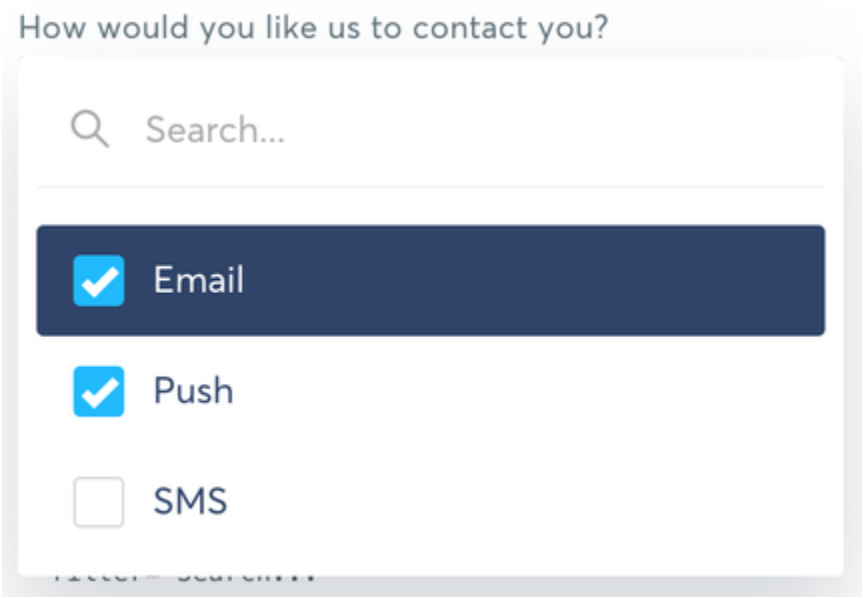
```
{
  type: “array”,
  items: { … }
}
```

We use arrays alongside values collections to indicate that we expect an array, and that the entries in the array must come from that collection. For example, we might use the schema:

```
channels: {
  title: "How would you like us to contact you?",
  type: “array”,
  items: {
    type: "string",
    values: [{
      value: "email",
      label: "Email
    },{
      value: "push",
      label: "Push
    },{
      value: "sms",
      label: "SMS
    }]
  }
}
```

Which our client may render as a set of checkboxes, or perhaps a multi-select box:





This would produce the model:

```
channels: ["email", "push"]
```

Arrays of object schemas

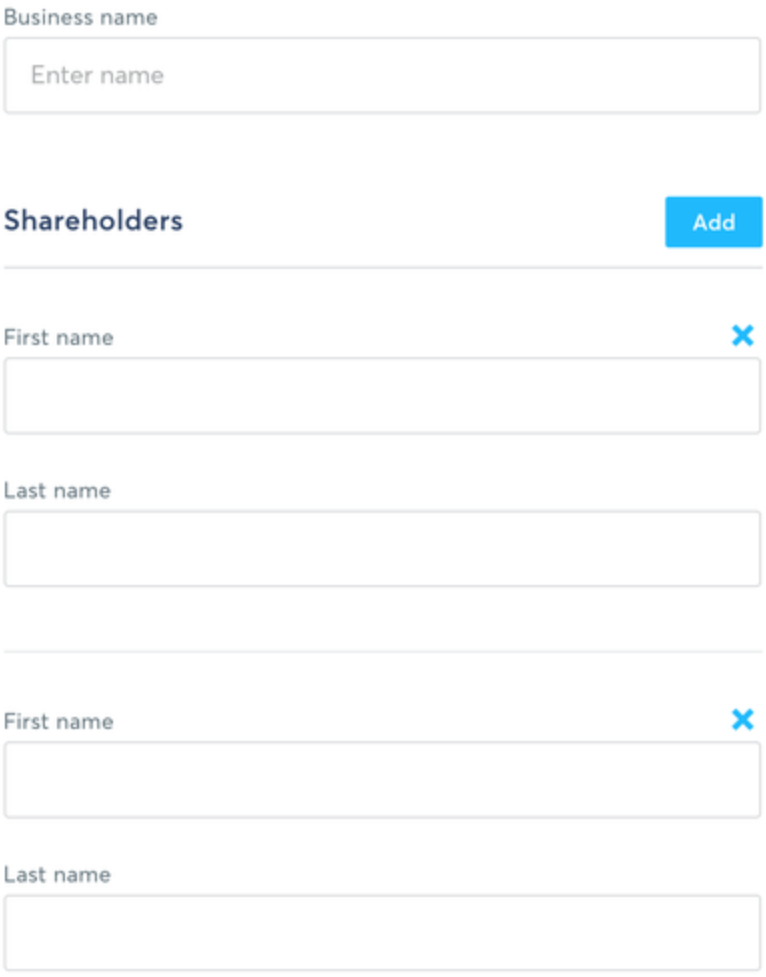
You may also encounter arrays of nested objects. For example, say that we needed to know the major shareholders of a privately held business. In this case we would need an array of nested objects, specifying the information we need for each shareholder, this might look as follows:

```
shareholders: {
  type: “array”,
  items: {
    type: “object”,
    properties: {
      firstName: {
        type: “string”
      },
      lastName: {
        type: “string”
      }
    }
  }
}
```

We would expect a model in the following structure:

```
shareholders: [{
  firstName: "John",
  lastName: "Smith
},{
  firstName: "Jane",
  lastName: "Doe"
}]
```

An example UI would look something like this:



Array size

Say we wish to know all shareholders with at least 25% ownership, there must be at least one, and at most four such shareholders. This places a validation constraint on the schema, which can be specified as follows:

```
{
  type: “array”,
  items: { … }
  minItems: 1,
  maxItems: 4
}
```

We can use this information to help tailor our UX. For example, if minItems is specified we might render that number of nested forms by default. We may also hide the remove buttons from that number of forms. If maxItems is specified we might disabled the 'Add' button once we reach the maximum number of items.