V3 - Asynchronous tasks

efresh requirements on change

netimes we need to update the requirements of the schema, based on data that the user already entered. A good example would be an address n. If the user picked their country as the US (or Canada, Australia etc), then we would want to ask them which state they live in, and we would nt to customise the list of states to the country. Many other countries don't have states, so we wouldn't want to gather a state at all.

his case, the country property will have an additional value 'refreshRequirementsOnChange'. When this value is set the client should POST the rent data model back to the endpoint that generated the requirements. The endpoint will respond with an updated set of requirements that are propriate to the new value.

```
country: {
  type: "string",
  values: [ ... ],
  refreshRequirementsOnChange: true
}
```

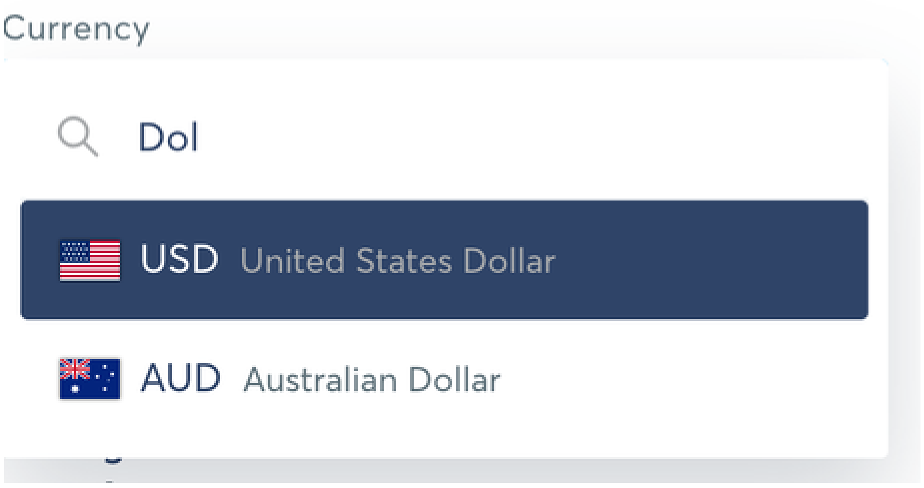
reshRequirementsOnChange is a required feature.

luesAsync

netimes the list of possible values for a field is finite, but large. In that case we will ask you to fetch the list of values asynchronously. You should w the user to enter a search term, then query for values based on that search string. The endpoint will respond with a new array of values.

```
{
  type: “string”,
  valuesAsync: {
    method: “GET”,
    url: “https://…”,
    param: “search”
  }
}
```

ke a request to the specified url using the specified method, and include the users search term using the specified search parameter.



lue models - DRAFT

ynchronous values may be used alongside an additional extension to our values array. Alongside the value we may include a model property. On ection, the values in this model should be applied to the model schema we are currently within.

example, say are trying to complete an address in the UK, we may have a schema:

```
{
  type: “object”,
  title: “Address details”
  properties: {
    postCode: {type: “string”, valuesAsync: {... }},
    address1: {type: “string”}
    city: {type: “string”},
    country: {type: “string”}
  }
}
```

 the user has entered “E1 6JJ”, the asynchronous response may contain a list of addresses in that postcode. The response values will contain a odel’ that contains the address information

```
[{
  value: “E1 6JJ”, - PROBLEM: they will all have same value
  label: “The Tea Building, 56 Shoreditch High Street”,
  model: {
    address1: “The Tea Building, 56 Shoreditch High Street”,
    city: “London”,
    country: “UK”
  }
},{
  …
}]
```

en selected it applies those model values to the current model, in this case preselecting the most relevant value in the next control. This will ne in useful when we encounter asynchronous operations.

oblem, twField only knows about current property will need to somehow broadcast this model out to be applied.

lidationAsync - DRAFT

st basic validation can be achieved with the validation properties, but sometimes we need to go back to the server to validate something. haps we're validating an IBAN, we can check it's structure using a regular expression pattern, but to know if it's an IBAN that is genuinely used, may have to do a lookup against an API. The validationAsync option enables us to do that in real-time, without waiting for the form to be omitted.

s is an optional feature, all data will be validated on form submission as well, but validating immediately may help you to provide a better stomer experience.

```
iban: {
  type: "string",
  pattern: "...}
  validationAsync: {
    method: "POST",
    url: “https://…”,
    param: "iban"
  }
}
```

y 200 response should be considered valid. If the value is invalid, it will return a XXX, along with an error message:

DO

rsistAsync - DRAFT

metimes, when we're within the context of a form, we'd like to upload a file immediately, and only submit the newly created id of that file in our del. We have a special option to enable this: 'persistAsync'.

take advantage of the control property to ask the client to render an upload, and then we persist the file as soon as it is provided.

```
firstName: {
  type: "string"
},
passportId: {
  type: "number",
  control: "upload",
  persistAsync: {
    method: "POST",
    url: “https://…”,
    param: "document",
    idProperty: "id"
  }
}
```

e control should upload the file to the given URL using the supplied method and parameter name. On a successful response, it should access the roperty of the response, and assign that to it's model, for submission with the rest of the form

```
{
  firstName: "John",
  documentId: 123
}
```