# Winton Centre/NHSBT TRAC tool project overview

## Introduction

The Lung and Kidney transplant tools have been developed very much in the style of the Predict websites:

<https://breast.predict.nhs.uk/>

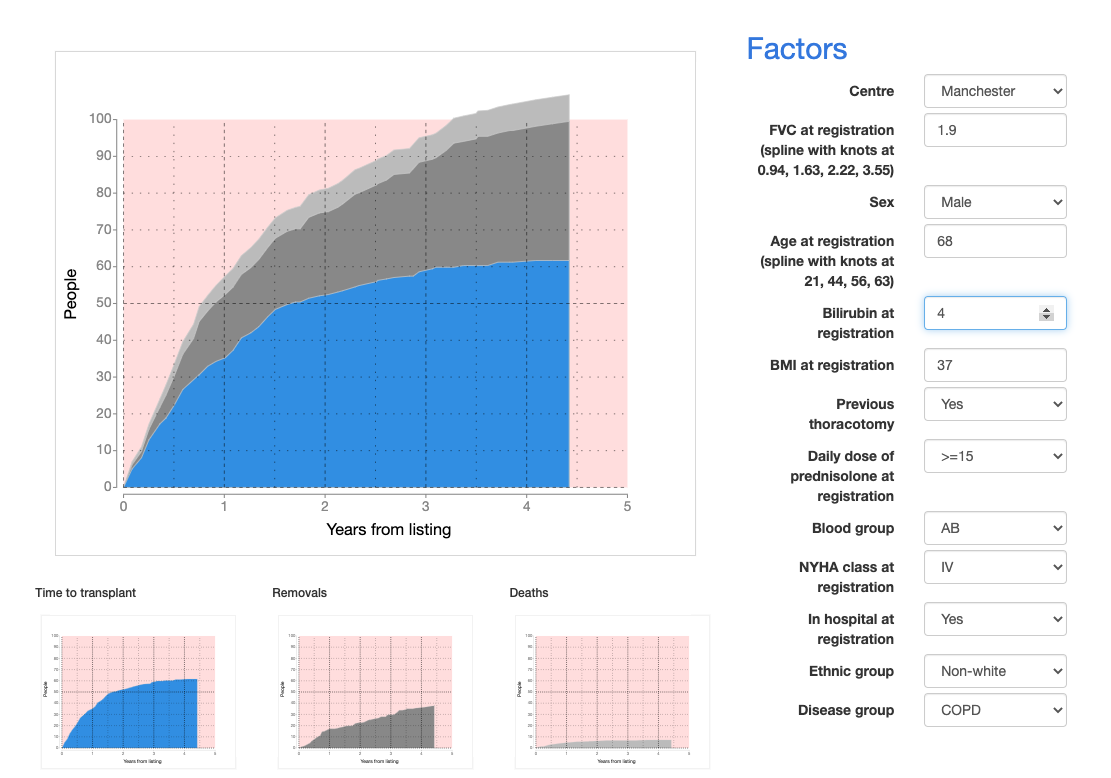
<https://prostate.predict.nhs.uk/>

These are single page applications developed in clojurescript, compiling to javascript. The lung and kidney transplant sites share that language choice and some of the technology stack.

They are also configured as programmable web apps so they can be used offline and on a mobile. Depending on the results of user testing, these may also become requirements in the TRAC tools.

## First lung and kidney prototypes

The initial lung and kidney prototypes highlighted some difficulties in interpreting the statistical model which have since been corrected. For example, 100 people could eventually have more than 100 outcomes:



The visualisations there were for the purpose of understanding the model behaviour and had not been subject to user testing.

## The current development

Is being developed at <https://transplants-dev.wintoncentre.uk/>

Call Mike Pearson on 07426 032555 if you need access.

We currently provide URL routes for

/

/:organ (e.g. /lung)

/:organ/:centre (e.g. /lung/birm)

/:organ/:centre/:tool (e.g. /lung/birm/waiting)

With server redirects to the home page we can convert to hashtag-free URLs in the usual ReactJS manner.

Back button operation is fully supported.

## The new model

The new model is now specified in a couple of spreadsheets – one for kidney and one for lung – plus a text file containing metadata about their structure.

### Understanding the spreadsheets

The **lung** spreadsheet describes three main **tools**,

#### Waiting

The competing risks while waiting for a transplant.

#### Post-transplant

Survival post-transplant

#### From-listing

Survival from time of listing for transplant.

Each tool is then described by 3 spreadsheets. Taking the waiting tool as an example, we have:

#### Waiting-baseline-cifs

* The baseline cumulative incidence frequencies for each outcome. For this tool, the outcomes are *transplanted*, *removed from the list*, or *died*. These baseline cifs are then adjusted according to the individual patient’s parameters.   
    
  In order to correct the interpretation problem seen in the prototype, the waiting (competing risks) tools also have an ‘all-reasons’ outcome – the baseline risk of leaving the list for any reason. The sum of the other outcomes is scaled to this value.

#### Waiting-baseline-vars

* The input factors along with the value or level that the factor would take for it not to change the outcome.

#### Waiting-inputs

* Details about each input factor – their internal identifiers or keywords, their categorical levels or numerical values, ranges, and precisions, their external names, a beta coefficient to indicate influence on each outcome, the associated input widget type, any sub-texts or explanatory texts, and the order the widget should appear within the user interface.

Sub-texts are listed alongside the value that causes the sub-text to appear. There are many examples in Predict where this facility is needed.

In Predict, info-boxes are modals activated by an info button which the user can press for further explanations. Early user testing of the transplants tools with clinicians is suggesting that info-boxes are not needed for these tools.

The **kidney** spreadsheet also describes three main tools, each with a similar structure.

#### Waiting

The competing risks while waiting for a transplant.

#### Graft

Survival of the graft post transplant

#### Survival

Survival of the patient post transplant

Each of these kidney tools is similarly described by sheets for baseline-cifs, baseline-vars, and input factors

In addition to these main data sheets, there are a few sheets containing metadata – the colour coding used in the spreadsheet to indicate editable, fixed and comment areas, and some textual details of tools and centres.

## Technical Architecture

A close up of a sign

Description automatically generated

Green documents are editable in Excel or in a text editor.

Blue components are written in clojure.

Orange components are run-time text files.

The Configure and Test tools run on the JVM (Java Virtual Machine) to produce validated run-time text files which should be collocated with the javascript, CSS and HTML in a static web server.

### Technical Stack

We have been developing in clojure and clojurescript since 2014 and have found it to be an exceptionally stable platform with a comprehensive core library. There have been few, if any breaking changes during this period.

The web app is a clojurescript application based on reframe and reagent. These are highly performant wrappers providing state management and excellent testability over ReactJS. They are described in great detail at <https://github.com/day8/re-frame> and <https://github.com/reagent-project/reagent>.

### Development tools

We use VSCode for development with the Calva plug-in to support both Clojure and Clojurescript. Other viable alternatives are Intellij with Cursive, Atom with proto-repl,

Emacs with Cider.

We use shadow-cljs (<https://shadow-cljs.github.io/docs/UsersGuide.html>) to build the javascript target. Shadow-cljs has good interoperability with npm libraries where we need them – primarily useful for react and react-bootstrap. Npm and webpack support is now also available in the clojurescript core build tool.

### NPM Libraries

This is the current list, but I expect to remove react-bootstrap-range-slider, rc-input-number, and highlight.js before we get to production. Karma provides the unit test framework.

:npm-deps {"highlight.js" "9.18.1"

"react" "16.13.0"

"react-dom" "16.13.0"

"react-bootstrap" "^1.0.1"

"react-bootstrap-range-slider" "1.0.0"

"rc-input-number" "5.0.1"}

:npm-dev-deps {"shadow-cljs" "2.9.3"

"karma" "4.4.1"

"karma-chrome-launcher" "3.1.0"

"karma-cljs-test" "0.1.0"

"karma-junit-reporter" "2.0.1"}

##### Clojure Libraries

Version numbers will be bumped before production.

[org.clojure/clojure "1.10.1"]

[org.clojure/clojurescript "1.10.764"

:exclusions [com.google.javascript/closure-compiler-unshaded

org.clojure/google-closure-library

org.clojure/google-closure-library-third-party]]

[thheller/shadow-cljs "2.9.3"]

[reagent "0.10.0"]

[re-frame "0.12.0"]

[day8.re-frame/tracing "0.5.5"]

[day8.re-frame/http-fx "v0.2.0"]

[metosin/reitit "0.4.2"]

[cljs-ajax "0.7.3"]

[winton-utils "0.2.1"]

Day8.re-frame-tracing will be removed before production. It’s useful in development as it provides an inspector for the in-memory database with event traceback.

Metosin/reitit is an excellent fast router for single page apps.

## Unit test

Tests start life as executable comments developed in the Clojure REPL, and are then transferred to the ‘clojure.test’ core framework and executed in-browser using karma.

## End to End tests

We normally use the clojure etaoin webdriver library for automated browser tests. These run in a JVM.

## Attachments

lung-models-master.xlsx

kidney-models-master.xlsx