SCHEDULE 2: THE REQUIREMENTS

1. List of requirements

Number	Title	Description
Instance creation		
	Instance creation	It MUST be possible to use the forms to create new instances, where field values are initially unpopulated
	URI Generation	It MUST be possible to define the URI pattern for newly created instances
Validation		
	Validating single field	The system MUST support the validation of single field values
	XML Schema datatypes	Literal values MUST be validated against their datatype
	URI References	The system MUST be able to check that the field value is in a particular set of URIs (expressible in an ASK query)
Multi-linguality		
	Input languages	The input fields MUST support English and Arabic and Russian
	Language tags	It MUST be possible to set / change the language tag of a field value. According to RDF 1.1, language tags are supported for literals of type http://www.w3.org/1999/02/22-rdf-syntax-ns#langString
	Other languages	The input fields SHOULD support other languages.
Field definitions		
	Field labels	The field labels need to be configurable
	Configurable fields	Input fields MUST have some basic configuration functionality for each property, e.g. data/input type, visible, multiple.
	CRM Support	The fields MUST be expressive enough to support authoring instances of the CRM model
	Extensibility	The system SHOULD designed to be configurable (or it be feasible to change the form) to create a different input form based on a different CRM model in the future
	Atomic field values	The system MUST support atomic field values
	Inline field definitions	The developer MUST be able to provide field definitions inline in a form configuration.
	Field definitions as RDF	In addition to the inline field definitions, it SHOULD be possible to specify field definitions as RDF to be read from a configuration storage (for example, database or file).
	Instantiation of field definitions using spreadsheets	The domain expert SHOULD be able to instantiate patterns of field definitions through spreadsheets.
	Spreadsheet transformation	The system MUST provide means (a script) to transform the spreadsheet definition to field definitions that can be used in the form configuration.
Input components		
	Dates	The user MUST be able to specify dates. Dates can be specified as a literal value or using a date picker. Date ranges are treated as separate fields (one field each for start and end).
	Numerical data	The user MUST be able to enter atomic numerical values
	Numerical values with units	The user SHOULD be able to enter atomic numerical values with units (such as kg). The units are fixed for a given field

		definition.
	Text	The user MUST be able to provide free text, the size of the text boxes to be configurable
	URI references	The user MUST be able to select a resource (via its URI)
	a.) URI from Dropdown	The user MUST be able to select a resource from a Dropdown-menu
	b.) URIs with Auto-Suggestions	The system MUST auto-suggest from a configurable set of resources as the user types
Cardinalities		
	Multiple values	It MUST be possible to configure properties with multiple values, e.g. multiple material values, “cotton”, “polyester”, etc
Layout		
	Navigation, Logical grouping	The input form MUST have navigation to allow a user to quickly access a particular part / section of the form. The logical grouping (sections) can be declaratively configured in using markup.
	Order of fields	It MUST be possible to define the order of input fields
Misc		
	Saving state	The input form SHOULD save the data as entered, i.e. maintain state on the client side. The client state is only saved for one current record.
	Saving and updating records	The user MUST be able to save the record and update the record at a later point. During the update, the old version of the record will be deleted entirely and the new version of the record will be stored.
	ResearchSpace platform	The system MUST integrate with the existing ResearchSpace platform
Provenance		
	Basic provenance	Basic provenance of the record SHOULD be saved (inputters name, date)
Security		
	RS Security	The ability to access and use the form SHOULD comply with normal RS security (login, authentication)
Standards compliance		
	RDF 1.1 Support	The input form must be compatible with the RDF 1.1 standard, i.e. they must produce data that conforms to RDF 1.1

2. Field Representations

The goal is to have a reusable, RDF-based scheme for the presentation of patterns, where these pattern definitions could be stored in a database or also provided inline in the configuration of a particular component.

The configuration of specific forms and corresponding field definitions themselves are not part of the services provided, they are created by the client / form engineer. The following properties will need to be provided as part of a field definition:

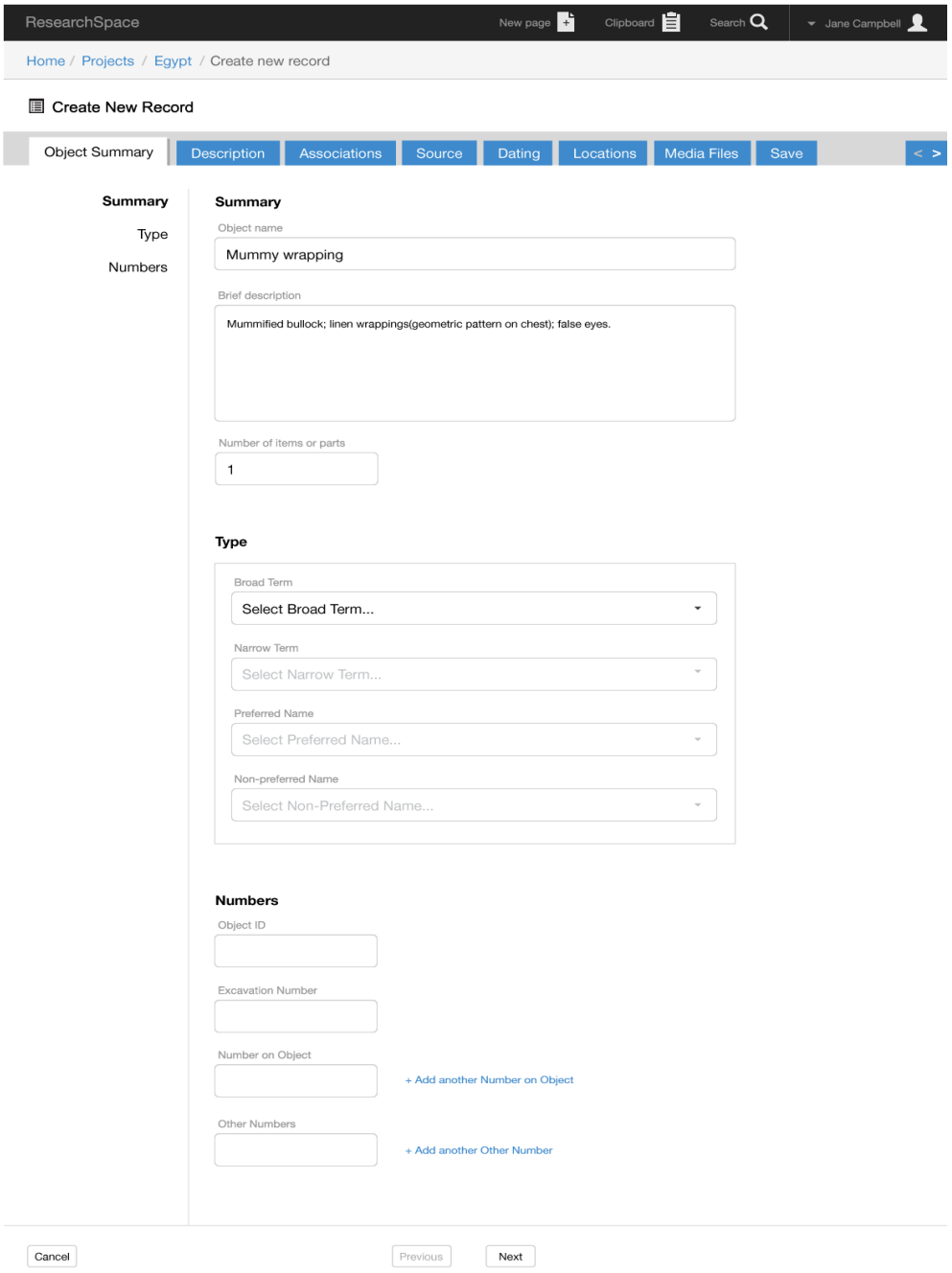
Property	Type	Comment
autosuggestionPattern	string	SPARQL SELECT query string to generate a dynamic suggestion list based on textindex or regex search The following convention apply: • the query **MUST** expose a ?value and ?label variable, otherwise form or field MUST already render error on initalization time

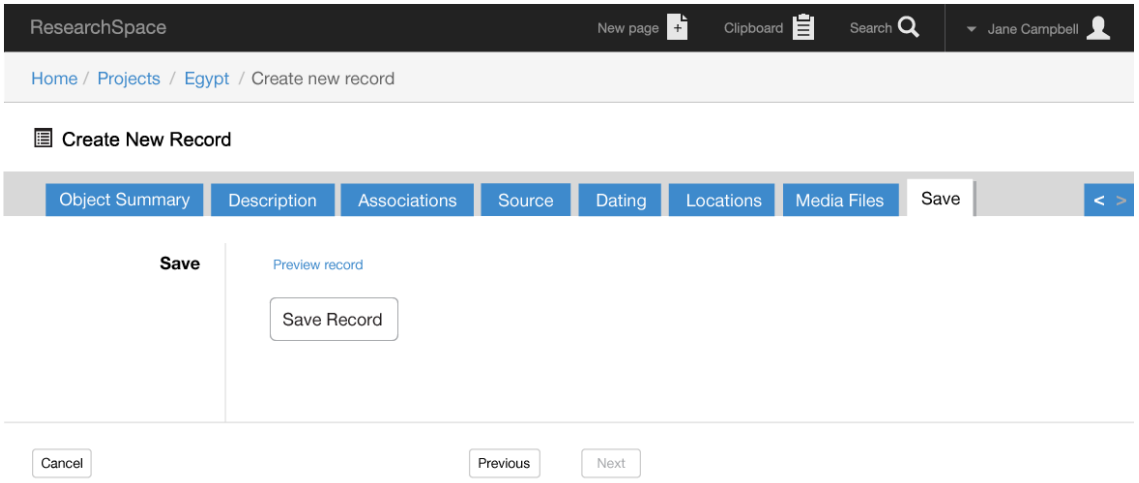
		• $subject refers to the current entity (i.e. if form is embedded on certain instance page, it will automatically be replaced) • optionally, further projection variables **might** be exposed which are will be fed into tupleTemplate to format the rendering within the dropdown or autosuggestion list • usage of $ or ? should not matter, however, $ *indicates for the field engineer* that the variable is supposed to be replaced/injected later • if query contains a ?token variable, this will be replaced by the actual token the user is typing i.e. into the autosuggestion field
deleteInsertPattern	string	SPARQL DELETE / INSERT query string The following conventions apply: • $subject refers to the current entity (i.e. if form is embedded on certain instance page, it will automatically be replaced)
description	string	description of a field, might be rendered e.g. onHover or as an info icon next to the field
id	string	unique identifier of the field definition, in most cases it will be the URI of the field definition
label	string	label used for rendering the field, for example as an HTML input label before the input element
maxOccurs	number|string	xsd schema max cardinality number of 0:N or "unbound"
minOccurs	number|string	xsd schema min cardinality number of 0:N or "unbound"
selectPattern	string	SPARQL SELECT query string to read existing values for a field from the database The following convention apply: • the query **MUST** expose a $value variable which binds the actual value(s), otherwise form or field MUST already render an error on initalization time • $subject refers to the current entity (i.e. if the form is embedded on certain instance page, it will automatically be replaced) • optionally, further projection variables **might** be exposed which are will be fed into tupleTemplate to format the rendering within the input element. • usage of $ or ? should not matter, however, $ *indicates for the field engineer* that the variable is supposed to be replaced/injected later
validationAskPattern	string	SPARQL ASK query to validate values entered by the user against the database The following conventions apply: • $subject refers to the current entity (i.e. if form is embedded on certain instance page, it will automatically be replaced)
valueSetPattern	string	SPARQL SELECT query string to generate a fixed list (choices) of values that the user may choose from. The following conventions apply: • the query **MUST** expose a $value variable which binds the actual value(s), otherwise form or field MUST already on initalization time • $subject refers to the current entity (i.e. if form is embedded on certain instance page, it will automatically be replaced) • optionally, further projection variables **might** be exposed which are will be fed into tupleTemplate to format the rendering

		• within the dropdown or autosuggestion list • usage of $ or ? should not matter, however, $ *indicates for the field engineer* that the variable is supposed to be replaced/injected later
xsdDatatype	string| URI	a full or prefix XSD URI datatype identifier as specified in RDF 1.1 https://www.w3.org/TR/rdf11-concepts/#xsd-datatypes

3. Mockups

The following mockups exemplify how an input form may look like:





---

4. Additional Notes on the Development Approach

ResearchSpace development is split into front-end and back-end development. The interaction between the two is governed by the metaphacts platform and specifically its APIs and Widget Architecture.

4.1 Back-End Development

Back-end development will be carried out in the Java programming language, according to the official Oracle Java 8 release. All code will be tested using Oracle Java 8 on the Ubuntu 14.04 LTS release.

The deliverables will be provided with:

- Javadoc-based basic documentation
- Automated build scripts (e.g. Maven, sbt), including test scripting;
- Junit tests

Communication with front-end functionality will be via stateless and HTTP-based RESTful APIs, exchanging either JSON and/or RDF data. Ideally JSON-LD will be used. All APIs will be documented with reference made to test cases and subject to documentation requirements below.

4.2 Front-End Development

Front-end development will be carried out in HTML-based Javascript. All code will be tested using the latest version of Chrome and Mozilla Firefox officially released at the delivery date on Windows 7+, Mac OSX 10 and Ubuntu 14.04.

Javascript is statically typed through the use of TypeScript.

Front-end code is built using React. Front-end developments are packaged using Webpack. Front-end development uses HTML5 features as preference where needed, be tested for validity with the W3C tools.

4.3 Coding Standards

All development will be liberally commented, indented according to best practices, and with readability required over brevity in variable naming. Best practices are documented collaboratively, for Java and JavaScript, on the ResearchSpace Confluence.

4.4 Version Control

All code will be committed to the relevant component's git repository; either that provided by ResearchSpace (stash.researchspace.org) or provided by the supplier and mirrored by ResearchSpace.

4.5 Continuous Integration

Completed iterations will be capable of inclusion in the continuous integration server at jenkins.researchspace.org for extension by the British Museum or third parties. Metaphacts will provide support in setting up automated builds.

4.6 Documentation

All software products, designs and APIs will be documented either:

- At the ResearchSpace confluence, or
- Using the supplier's Web-based documentation solution, with PDF-based archives of the documentation provided at the end of each iteration.

4.7 Tracking issues by JIRA

All the requirements and possible bugs will be tracked in JIRA (jira.researchspace.org).

Before starting to work on an issue, the developer (internal or external) will break it down into sub tasks. The developer will update the status of issue/sub task to report on progress and the assignee field.

4.8 Database backend

It is assumed that an RDF database is in place and managed by the British Museum. metaphacts can offer support for the setup, deployment and operations of the Blazegraph triple store. Further, metaphacts can provide support for the migration, conversion and data preparation of RDF data.

These services are not covered by this proposal. Quotes are available on request.

4.9 Intellectual Property

As requested, all Intellectual Property created under the contract will be assigned to the British Museum. All development will be subject to a liberal - open source, free, and without restrictions beyond attribution, on commercial exploitation – license policy.

As extensions or replacements for core metaphacts platform functionalities there is no requirement for the copyright to be assigned to the British Museum, but the British Museum will be allowed to fulfil

its obligation to release all functionality for free and/or commercial use and extension, subject to being made available as open source, by third parties.

5. Timeframe

- The development will be completed with the following target date:
  (Assuming a start date of May 1, 2016)
    - June 30, 2016