API for analysis results and figures

The immediate need for this is driven by GradePRO & MAGIC integrations, but longer-term we also need to be able to insert results and figures into reviews for publication, to handle calculations that are not available client side (e.g. network meta-analysis), and to save analysis results as they were at the time in case statistical code changes.

API to provide analysis results for SoF tools

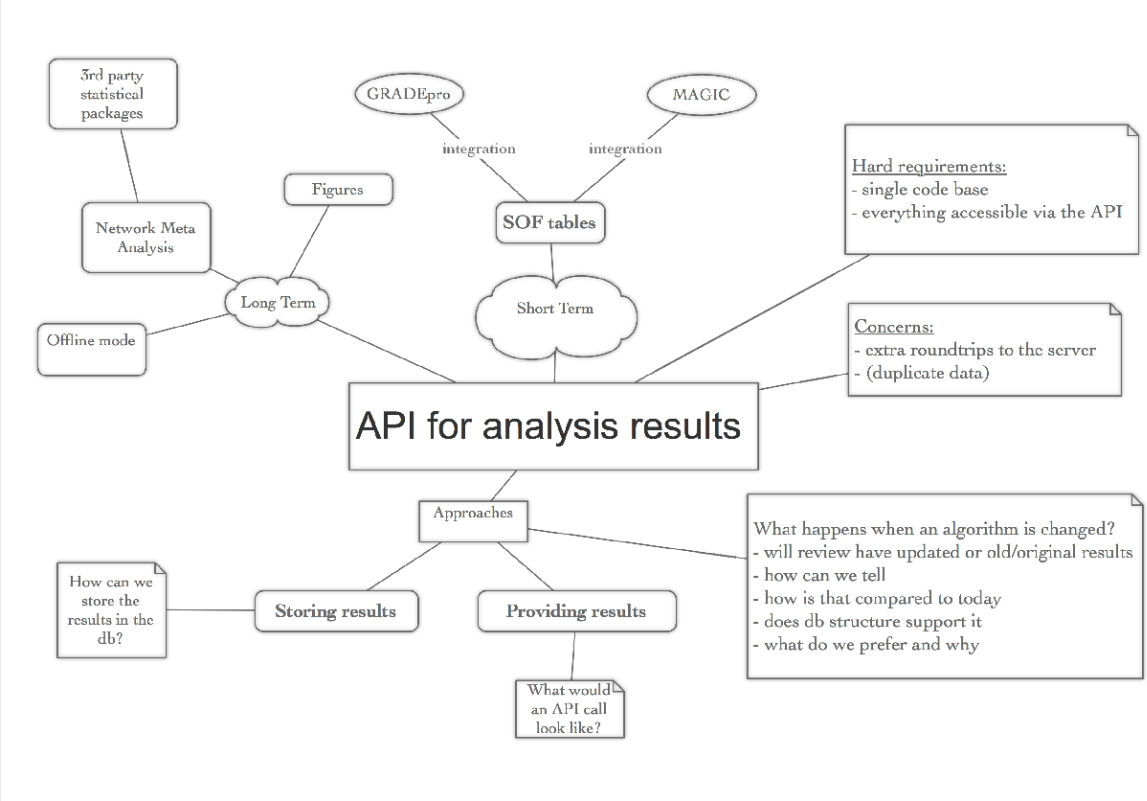
In order to support GRADEPro & Magic integration we need to make results of analysis available outside the revman client.

This page describes some of the the considerations that we've had before undertaking this task.

api_for_results

current

future_step1



Code re-use or serverside only or rewrite in java?

Since our .js code for meta-analysis is fairly well documented and tested, just re-using the javascript code in the backend seems to require the least amount of effort.

Rewriting in java for a sever side only solution doesn't seem worth it since the benefits of simplicity and consistency of having a java only backend would be shortlived as we're most likely going to interoperate with 3rd non-java party libraries for network meta-analysis.

We're also gravitating away from a serverside only solution because some of the javascript that we rely on for calculations are also required by the clientside calculator.

DECISION: the existing JS analysis code and test suite will be separated into an independent NPM package. This will be a dependency for RevMan Web, and a (to be developed) Node.js analysis backend that ReviewDB will call to get results.

Client API

Endpoint: /reviews/{reviewid}/comparisons/{comparisonId}/outcomes/{outcomeId}/result

There are several options for client API that closely tied to future developments.

Mixing derived values and actual data is one concern.

Requiring 2 roundtrips also seems suboptimal (fetching "input" and results). It might be non trivial for clients to reassemble inputs and results.

Results endpoint also returning data of its "parent" (ie outcome) also feels wrong.

DECISION: the API will return only results, with IDs of the associated input objects attached. The endpoint may be extended to also provide the inputs

There is no ReviewDB endpoint for DTA reviews - only the backend backend.

JSON model for analysis results

The following describes the structure of the JSON returned by the API:

Analysis results model

```
RESULT: {
    result: <OVERALL_RESULT_DATA>,
    dataRows: <ROW_RESULT_DATA>*,
    subgroups: [{
        result: <SUBGROUP_RESULT_DATA>
        dataRows: <ROW_RESULT_DATA>*
    }])
}


RESULT_DATA: {
    estimable: true|false,
    logScale: true|false,
    mean: 0.0, (NOTE: ON THE SCALE OF ANALYSIS, SO LOG SCALE FOR LOG SCALE OUTCOMES)
    se: 0.0,
    ciStart: 0.0, (NOTE: ON THE SCALE OF ANALYSIS, SO LOG SCALE FOR LOG SCALE OUTCOMES)
    ciEnd: 0.0, (NOTE: ON THE SCALE OF ANALYSIS, SO LOG SCALE FOR LOG SCALE OUTCOMES)
    weight: 0.0
}


ROW_RESULT_DATA extends RESULT_DATA: {
    id: "<studyDataRowId>", (NOTE: this is not included in the meta-analysis API output, but is added by
ReviewDB.)
    studyId,
    applicability: "SUBGROUP_ONLY" | "OVERALL_ONLY" | "SUBGROUP_AND_OVERALL",
    (notIncludedInTotal: true)? (NOTE: used to indicate that a subgroup row is not included in the calculation
of the total)
}

SUBGROUP_RESULT_DATA extends RESULT_DATA: {
    id: "<subgroupAnalysisId>", (NOTE: this is not included in the meta-analysis API output, but is added by
ReviewDB.)
    heterogeneity: {
        chiSquared: 0.0,
        degreesOfFreedom: 0,
        iSquared: 0.0,
        p: 0.0,
        (tauSquared: 0.0)? (NOTE: FOR RANDOM EFFECTS)
    },
    overallEffect: {
        z: 0.0,
        p: 0.0
    },
    (experimental: {
        total: 0
    },
    control: {
        total: 0
    })?
}

OVERALL_RESULT_DATA extends SUBGROUP_RESULT_DATA: {
    (subgroupDifferences: {
        chiSquared: 0.0,
        degreesOfFreedom: 0,
        iSquared: 0.0,
        p: 0.0},)?
    (experimental: {
        total: 0
    },
    control: {
        total: 0
    })?
}
```

DTA Analysis result model

```
RESULT: {
    diagnosticDataRows: [
        <DIAGNOSTIC_RESULT_TEST_DATA_ROW>*
    ] |
    diagnosticSubgroups: [{
        meanD: 0.0,
        a: 0.0,
        b: 0.0,
        diagnosticDataRows: [
            <DIAGNOSTIC_RESULT_TEST_DATA_ROW_FOR_SUBGROUP>*
        ]
    },...]
}

DIAGNOSTIC_RESULT_TEST_DATA_ROW: {
    sensitivity: 0.0,
    sensCiStart: 0.0,
    sensCiEnd: 0.0,
    specificity: 0.0,
    specCiStart: 0.0,
    specCiEnd: 0.0
}

DIAGNOSTIC_RESULT_TEST_DATA_ROW_FOR_SUBGROUP extends DIAGNOSTIC_RESULT_TEST_DATA_ROW: {
    d: 0.0,
    s: 0.0,
    weight: 0.0,
    scaleSpec: 0.0,
    scaleSens: 0.0
}
```

Platforms/protocol for analysis backend

DECISION: The web interface will expose an HTTP API. Other options considered are described below.

WAMP RPC

While we do have a proof of concept, this solution would increase coupling between backend and backend backend, since an RPC binding is quite strong.

It would also make us more dependent on crossbar.io which is currently "only" being used for concurrent editing - this is only a minor concern.

RabbitMQ

RabbitMQ (or any other messaging queue) might seem more robust since an asyncronous solution can sometimes continue to work despite outages.

In this use case however, we won't be able leverage this advantage since we're calling backend in a syncronous context (we're in the process of serving an http-request).

This option may become more appealing if we later decide not only to perform calculations on demand.

HTTP/REST

We're using REST-like apis in most other scenarios.

It's a weaker dependency than RPC.

JSON model for analysis backend

The output of the analysis backend will be identical to the output of the client API (shown above), with the IDs omitted. The input data structure is as follows:

Input structure

```
(NOTE: both dataRows and subgroups should be provided when the overall results should be based on a separate
```

```
analysis, e.g. subgroups with studies split by intervention.)

INPUT: [{
    options: {
        dataType: "DICHOTOMOUS"|"CONTINUOUS"|"CONTRAST"|"OBSERVED_EXPECTED", (NOTE: CONTRAST maps to
INVERSE_VARIANCE in RM5 and CONTRAST_LEVEL in Review DB)
        method: "MH"|"GIV"|"PETO",
        effectMeasure: "MD"|"SMD"|"LOR"|"LRR"|"RD"|"PetoLOR"|"Generic",
        model: "FIXED"|"RANDOM",
        ciLevelRows: 0.90|0.95|0.99,
        ciLevelTotals: 0.90|0.95|0.99,
        swapEvents: true|false,
        logData: true|false,
        totals: "YES"|"SUB"|"NO"
    },
    dataRows: [
        <DICHOTOMOUS_DATA>*|<CONTINUOUS_DATA>*|<CONTRAST_DATA>*|<OBSERVED_EXPECTED_DATA>*
    ],
    subgroups: [{
        dataRows: [
                <DICHOTOMOUS_DATA>*|<CONTINUOUS_DATA>*|<CONTRAST_DATA>*|<OBSERVED_EXPECTED_DATA>*
        ]
    },...]
}]


DICHOTOMOUS_DATA: {
    studyId,
    experimental: {
        events: 0,
        total: 0
    },
    control: {
        events: 0,
        total: 0
    }
}

CONTINUOUS_DATA: {
    studyId,
    experimental: {
        total: 0,
        mean: 0.0,
        sd: 0.0
    },
    control: {
        total: 0,
        mean: 0.0,
        sd: 0.0
    }
}

CONTRAST_DATA: {
    studyId,
    estimate: 0.0,
    se: 0.0
    (experimental: {
        total: 0
    },
    control: {
        total: 0
    })?
}

OBSERVED_EXPECTED_DATA: {
    studyId,
    oe: 0.0,
    variance: 0.0,
    (experimental: {
        events: 0,
        total: 0
```

```
    },
    control: {
        events: 0,
        total: 0
    })?
}
```

DTA input structure

```
INPUT: [{
    diagnosticOptions: {
        scale: "EQUAL" | "SAMPLE_SIZE" | "INVERSE_SE" | "COVARIATE",
        weight: "EQUAL" | "SAMPLE_SIZE" | "INVERSE_V",
        ciLevel: 0.90 | 0.95 | 0.99
    },
    diagnosticDataRows: [
        <DIAGNOSTIC_TEST_DATA_ROW>*
    ] |
        diagnosticSubgroups: [{
        diagnosticDataRows: [
            <DIAGNOSTIC_TEST_DATA_ROW>*
        ]
    },...]
}]

DIAGNOSTIC_TEST_DATA_ROW: {
    tp: 0,
    fp: 0,
    fn: 0,
    tn: 0,
    covariateScaleValue?: 0.0
}
```