Technical Documentation for TersectBrowser+

David Oluwasusi, Tanya Stead, Gregory Lupton, Gabrielle Baumberg
Supervised by: Dr. Tomasz Kurowski

April 28, 2025

1 Overview

This Technical Documentation describes the particular details of updating the original TersectBrowser developed by Tomasz Kurowski into the TersectBrowser+ by integrating multiple extension features.

2 Architecture

The TersectBrowser+ software is an Angular project, with an original frontend and backend written in Angular 8, and a separate extension space written in Angular 19 using Node 22. It was chosen to use a 'bolt-on' approach for the extensions in order to fit the project delivery within the time requirements, as well as allowing the new extensions to use up-to-date packages. The app is dockerized to allow both versions of node to be used seamlessly in different sections.

For the TersectBrowser+ update to the original TersectBrowser, main changes have been made to the frontend implementation of features added in a separate extension section of the project. The main addition is in a 'genome-browser' extension, which contains elements for each of the specific features mentioned in the project brief.

3 Deployment

The GitHub project `https://github.com/Tersect-Browser/Tersect-browser` README provides detailed instructions for setup, dependencies, and deployment, however the main outline is given here.

3.1 System Requirements

TersectBrowser+ can be setup and deployed from both Mac and Windows machines, with variations in the configuration required.

The main requirements for TersectBrowser+ deployment are:

- nvm (versions 16 and 22 specifically, as these will be deployed separately)
- npm registery `https://registry.npmjs.org/`
- Angular CLI v1.7.1
- Tersect CLI
- JBrowse CLI
- MongoDB
- Python virtual environment
- RapidNJ / Rosetta

4 TersectBrowser+

4.1 Variant Browser Extension

4.1.1 Rationale

4.1.2 User Interface

Feature Description

- Window is located at top of Tersect Browser (TB)- below settings bar but above TB heatmap
- Jbrowse window (JB) layout - shows only sequence tracks and ruler track/linear genome scale. Zoom features/chromosome selection features have been removed from Jbrowse window. Hamburger button opens menu containing many options, notably track selector, where tracks can be added/removed. The chromosome scale for TB has a dynamic offset depending on the accession/tree view and the zoom level. This is synced between JB and TB, so that the chromosome scale lines up.
  - Layout TBC: Set window size
- Tracks:
  - Reference sequence
  - Variant tracks for each loaded accession
  - Gene models FeatureTrack (Ext-GM)
- Variant browser window is preloaded with default tracks.

  - Track TBC: Currently pre-loaded with first 3 tracks from dataset. Should be changed to 1-2 meaningful accessions?

- Syncing of the Browser pane and the Tersect heatmap pane, including the following:
  - Zoom synced, both when the zoom plus/minus buttons are pressed, and also when scrolling the mouse to zoom.
  - Bin size changes synced.
  - Interval synced using an offset of the Browser pane.
  - Chromosome selection changes synced.
  - Horizontal scrolling along the Browser and Tersect panes is synced.

4.1.3 Technical Description

View State The `JBrowseLinearGenomeView` component is imported from @jbrowse/react-linear-genome-view and is wrapped using React. The configurations for the assembly, assembly, and variant tracks, tracks, are imported from the respective assembly.ts and tracks.ts configuration files. The styling configurations, config, are imported from jbrowseConfig.ts. The `JbrowseWrapperProps` are imported from `JbrowseInterface`.

The function `JbrowserWrapper` takes as arguments the props from `JbrowseWrapperProps` and defines the `viewState`. If an accession has been selected and the accession name is stored under `props.location.accession.name`, the `viewState` defined in the constant `JbrowseWithAccessionName` is returned (See below parameters). If not, a conditional check verifies whether the props `defaultInterval` and `offsetCanvas` have been populated. If either of them are empty, a container with the phrase "Loading..." is returned. If all data is present, the `viewState` is defined using the function `createViewState`. This takes as arguments the following variables:

- `assembly`: the assembly track
- `tracks`: the data tracks to display
- `configuration`: the config variable where the theme is defined
- `defaultSession`: an object of type LinearGenomeView, which defines the initial state. This object includes the following configurations:
  - `bpPerPx`: the base pairs per pixel, which defines the bin size
  - `assemblyName`: the corresponding assembly for the track
  - `start`: the track start position
  - `end`: the track end position
  - `refName`: the genomic coordinates of the viewing window

When no accession is selected in the Canvas interface, the first three tracks defined in `tracks.ts` are added to the `viewState`. The dynamic left-hand offset is set by `horizontalScroll()`.

Assembly Config The configurations for the assembly track, which is built upon the reference accession SL2.50, is defined in `assembly.ts` in json format. The name is set to the reference accession name and `trackId` is set to "SL2.50-ReferenceSequenceTrack". The paths to the fasta file, along with its corresponding fasta index, are specified as local server URLs provided by the backend during the Tersect Browser setup.

Tracks Config The configurations for the `VariantTrack` and Gene Models `FeatureTrack` are defined in `tracks.ts` in json format. For the variant tracks, the `name` and `trackId` are set as the accession name, and for the Gene Models track these are set to "ITAG2.4 Gene Models". The paths to the zipped files, along with its corresponding Tabix index files, are specified as local server URLs provided by the backend during the Tersect Browser setup. Each track is separated by a comma.

Styling The container styling is defined in `jbrowseConfig.ts`. The palette theme is set to colour '#459e00' and the `boxShadow` is set to 'none'.

Zoom The zoom is synchronized between Tersect Browser and the JBrowse component. The `zoomLevel` observable is a component of the `PlotStateService` class. Inside `tersect-browser.component.ts`, the subscription `zoomSub` listens to the `zoomLevel` observable and assigns the latest value to the component `zoomLevel`. Inside `tersect-browser.component.html`, the `zoomLevel` is passed to `JbrowseWrapper` as a prop and is used to define the `bpPerPx` in the `viewState`.

The bin size is synchronised in the same way: the `binsize` observable is a component of the `PlotStateService` class. Inside `tersect-browser.component.ts`, the subscription `binSizeSub` listens to the `binsize` observable and assigns the latest value to the component `binSize`. Inside `tersect-browser.component.html`, the `binSize` is passed to `JbrowseWrapper` as a prop. Together, `bpPerPx` is calculated with the following equation:

$$bpPerPx = ((props.location.binSize) * (100/props.location.zoomLevel)) \tag{1}$$

Chromosome selection The displayed chromosome is synced in a similar fashion to the zoom and bin size. The `chromosome` observable is a component of the `PlotStateService` class. Inside `tersect-browser.component.ts`, the subscription `chromosomeSub` listens to the `chromosome` observable and assigns the latest value to the object `selectedChromosomeSub`. Inside `tersect-browser.component.html`, the `selectedChromosomeSub` is passed to `JbrowseWrapper` as the prop chromosome. Inside `JbrowseWrapper`, the chromosome name is called from the chromosome object and used to define the `refName` in the `viewState`.

Additionally, the default chromosome that is pre-selected when Tersect Browser initially loads is passed to `JbrowseWrapper` and used to define the default `viewState`. Inside `tersect-browser.component.ts`, the variable `preselectedChromosome` is defined. On initializing, when the `settings` observer subscribes to the `tersectBackendService`, the current `plotState.chromosome` is saved to the `preselectedChromosome` variable. This is then passed as the prop `preselectedChromosome` to `JBrowseWrapper` inside `tersect-browser.component.html`. Inside `JbrowseWrapper`, the chromosome name is called from the `preselectedChromosome` object and used to define the `refName` in the default `viewState`.

Interval display The displayed interval is synced in a similar fashion to the zoom, bin size, and chromosome selection. The `interval` observable is a component of the `PlotStateService` class. Inside `tersect-browser.component.ts`, the subscription `selectedIntervalSub` listens to the `interval` observable and assigns the latest value to the array `selectedInterval`. Inside `tersect-browser.component.html`, the `selectedInterval` is passed to `JbrowseWrapper` as the prop `selectedInterval`. Inside `JbrowseWrapper`, the first element of the array is used to define the start position in the `viewState`, and the second element is used to define the end position. Additionally, the default interval that is pre-selected when Tersect Browser initially loads is passed to `JbrowseWrapper` and used to define the default `viewState`. Inside `tersect-browser.component.ts`, the variable `defaultInterval` is defined. On initializing, the method `generateMissingSettings` loads the interval based on the size of the selected chromosome, which is obtained from `BrowserSettings`. The interval is saved as an array to `defaultInterval` and inside `tersect-browser.component.html` it is passed as the prop `defaultInterval` to `JbrowseWrapper`. Inside `JbrowseWrapper`, the first element of the array is used to define the start position in the default `viewState`, and the second element is used to define the end position.

Offset The dynamic offset is synced in a similar fashion to zoom, bin size, chromosome selection, and interval. The observable `offsetCanvas` is defined as a component of the `PlotStateService` class. The public variable `offsetCanvasSource` is defined as an instance of the `BehaviourSubject` class and holds all recorded values of the canvas offset. In the class constructor, `offsetCanvas` is initialised to continuously hold the latest value from `offsetCanvasSource`. The canvas offset is set in the `TreePlotComponent` class, and is passed to `offsetCanvasSource` when the tree is redrawn.

Inside `tersect-browser.component.ts`, the variable `offsetCanvas` is defined, and the subscription `offsetCanvasSub` listens to the `offsetCanvas` observable and assigns the latest value to `offsetCanvas`. Inside `tersect-browser.component.html`, the `offsetCanvas` is passed to `JbrowseWrapper` as the prop `offsetCanvas`. Inside `JbrowseWrapper`, the variable defines the extent of the `horizontalScroll` using the following logic: $horizontalScroll(-(location.offsetCanvas - 4))$.

4.1.4 Test Results

When additional variant tracks are added to the Browser view, the size of the browser panel does not change. The user is able to scroll downwards to view more tracks in the browser.

4.1.5 Future Improvements

(Current) Limitations:

Only SL2.50 loaded as assembly and reference

Can only view variant tracks for 150_VCF_Tomato dataset - this is hard coded, and would need to be changed if we want this to be dynamic (when the user adds their own dataset) / if we as admins want to load soybean dataset!

4.2 Gene Models Extension

4.2.1 Rationale

4.2.2 User Interface

Feature Description

- In the Jbrowse window, the first track shows gene models based on the GFF file provided.
- Gene model annotations are based on the official tomato genome annotation v2.4: Slycopersicum ITAG annotation v2.4
- The popup view of the same Jbrowse component has similar default: the first track is the gene models track, and the second track is the variant track for the selected accession.

4.2.3 Technical Description

The configuration for the Gene Models `FeatureTrack` is defined as the first entry in `tracks.ts`. The paths to the sorted and compressed GFF file, along with its corresponding Tabix index, are specified as local server URLs provided by the backend during the Tersect Browser setup.

The Jbrowse `viewState` is configured in `JbrowseWrapper.tsx`, as described in Extension-JBrowse. If no accession is selected, the first three tracks stored in `tracks.ts` are added to the `viewState`. As the Gene Models `FeatureTrack` is the first track in `tracks.ts`, it is displayed as the first track in the Jbrowse window.

For the popup window, the JBrowse `viewState` is configured when an accession is selected in `JbrowseWithAccession.tsx`, as described in Variant Browser Extension. If a track with a `trackId` matching the selected accession is found, the viewer displays both the first track defined in `tracks.ts` and the matching track.

4.2.4 Test Results

4.2.5 Future Improvements

4.3 Feature Search Extension

4.3.1 Rationale

The aim of this extension is to allow the user to search specific genes and identify which accession contains a high/medium/low impact variant impacting that gene.

4.3.2 User Interface

Feature Description

- Separate search bar in the top right of the tersect browser header where user can input gene name and search button
- TBC: Option for user to select based on region in addition to gene name
- Popup window for user to select advanced features
- Output: Bins in the canvas are highlighted red at the chromosomal position where the gene is located, and only for accessions containing a variant impacting that gene
- Clear button to clear highlighted bins from canvas

4.3.3 Technical Description

Search Bar

Popup window with advanced settings

Highlighting bins Bin highlighting is controlled in the `bin-draw.service` by two functions. The first function, `highlightFeatureBins()`, is defined, taking as arguments a string containing accession names, the bin position along the x-axis, and the `binView`. First, the y-axis bin position for accession names is calculated. Then, the `binView` is redrawn in greyscale, with the bin colour determined by the difference to the reference accession. Using the x-axis `binIndex` position and the y-axis accession bin positions, these bins are coloured red in the `binView`. The modified `binView` is returned as the output. The second function, `highlightBins()`, takes as arguments the start position of the interval and the bin size currently shown in the Tersect pane, a list of ordered accessions shown in the canvas, and the searched accessions. For each searched accession, accession name is reformatted to match the format displayed in the Tersect pane. Then, the bin position along the x-axis is calculated using the `getBinIndexFromPosition()` function, which takes as arguments the feature position along the chromosome, the interval start position, and the bin size. Then, `highlightFeatureBins()` is called for each accession. Lastly, the canvas is redrawn using the modified `binView` output from `highlightFeatureBins()`.

- highlightFeatureBins(accessions: string[], binIndex: number, binView: DistanceBinView) - takes a string of accession names, a binIndex (corresponding to the bin position along the x-axis matching the gene position on the chromosome), and binView. The y-axis index for bins matching accession names in the accessions strings is calculated and combined with the binIndex to colour these specific bins red. The rest of the bins are coloured in greyscale, with saturation depending on binDistance (calculated from tersect on the backend
  - Bin-draw.service.ts
  - Called by highlightBins() in bin-draw.service.ts
- highlightBins(intervalStart, binsize, orderedAccessions, searchedAccessions) - takes selected interval start position, selected binSize, list of ordered accessions shown in the canvas, and searched accessions (passed from callback function?).

Accession names in Jbrowse are in a different format to what is stored in tersect browser, so accession names are reformatted to match tersect browser bins. binIndex is calculated for the selected gene. These two are passed to highlightFeatureBins() - along with this.bins - to highlight the bins. The canvas is then redrawn using the binView calculated in highlightFeatureBins().

  - Bin-draw.service.ts
  - Called by callHighlightBins() in tersect-browser.component.ts

- callHighlightBins(searchedAcessions) - takes searchedAccessions. Calls highlightBins(), passing along selected interval start position, selected binSize, list of ordered accessions shown in the canvas, and searched accessions (passed from callback function?).

  - Tersect-browser.component.ts

Calling Highlighting Bins
TBC: Mechanism of searching VCFs to identify variants

Finally, callHighlightBins() is called, which itself calls highlightBins() from the bin-draw.service, passing as arguments the current interval start position, binsize, displayed accessions in an ordered format, and the searched accessions.

Clear Button

4.3.4 Test Results

4.3.5 Future Improvements

4.4 Barcode Generation Extension

4.4.1 Rationale

4.4.2 User Interface

Feature Description Given input parameters specified by the user, barcodes are generated via the backend and automatically downloaded for the user in txt format. The downloaded file is titled in the following format for easy identification and to prevent files being overwritten: *SystemDateAndTime_TB_Barcode_Gen_AccessionName.txt*.

The file contains the following information:

- **Barcode sequence** - The base with the accession-specific SNP is enclosed in square brackets
- **Barcode Start & Barcode End** - The absolute position of the barcode in the chromosome
- **Variant Count** - The number of accession-specific SNPs that are present in the barcode
- **Variant Position** - The relative position of the accession-specific SNPs in the barcode

- **Repeat Sequence** - The sequence of the 2-bp repeating region. A repeating region is defined as 2 base pairs repeating 3+ times
- **Repeat Multiplier** - The number of times the repeat sequence is repeated
- **Repeat Start-End** - the relative start and end position of the repeat region within the barcode sequence
- **GC Content** - The GC content of the barcode

4.4.3 Technical Description

Barcode generation is controlled by two scripts: `barcode_finder.sh` and `find_barcode.py`.

Calling Tersect CLI to extract variants : `Barcode_finder.sh` is run on the command line, and takes as input accession name, chromosome, interval start position, interval end position, barcode size, maximum variant number, reference fasta, and tersect TSI index. The chromosome and interval start and end positions are used to define the searchable `REGION`, and the accession name is used to define `SAFE_ACC` which is used to save files to a temporary file.

The tersect CLI command `tersect view` is called to extract all variants within the specified region for the specified accession. The output is saved as a temporary TSV file as: `${SAFE_ACC}_acc_unique.tsv`. The tersect CLI command tersect view is again used to extract all variants within the specified region for all accessions except the specified accession. This output is saved as a temporary TSV file as: `${SAFE_ACC}_union_vars.tsv`.

Lastly, the `find_barcode.py` file is called, passing as arguments the accession name, reference fasta, chromosome, interval start position, interval end position, barcode size, maximum variant number, and the tersect output files `${SAFE_ACC}_acc_unique.tsv` and `${SAFE_ACC}_union_vars.tsv`.

Generating barcodes : `Find_barcode.py` requires the following dependencies: argparse, SeqIO, and datetime.

The reference fasta is parsed using `SeqIO.parse`, yielding sequence records that are converted to a dictionary using `SeqIO.dict`. Using the user-inputted chromosome name, `[args.chrom].seq` extracts the chromosome sequence from the dictionary and saves it to the variable ref. The user-defined interval start and end positions are used to extract the `ref_window` from ref.

The `load_variant_file()` method imports the tersect output files and creates a dictionary, with variant position being stored as the key and a tuple containing the original base and the alternate base being stored as the dictionary value. Then, the `remove_overlapping_variants()` method compares the dictionaries and creates a new dictionary `new_unique_vars` with the same format, containing variants that are only present in the specified accession and not also present in any other accession. The method `apply_variants_to_sequence()` then uses the `ref_window` and `new_unique_vars` to generate an accession-specific sequence, `unique_seq`, containing accession-specific SNPs. Lastly, the `find_barcode_windows()` method compares `unique_seq` against `ref_window` and using a sliding window of 1 base pair, identifies regions where the two sequences differ. The sequence, along with the absolute start and end position, is saved to the variable barcodes.

Output file and barcode stats : Statistical metrics are calculated for each barcode, using custom methods. The number of accession-specific variants is calculated using `count_variant_number()`, which also records variant position within the barcode. The variant position is used to highlight the variants within the barcode using the custom `highlight_ref_alt_positions()` method, which encloses the variant in the following format: `[original base/alternate base]`. Repeat content is calculated using `find_dinucleotide_repeats_custom()`, with repeats being defined as regions where a dinucleotide (2-base pair) sequence repeats consecutively three or more times. The repeating dinucleotide, number of times the dinucleotide repeats, and the start and end positions of the repeat region within the barcode are returned. GC content is calculated as a percentage to six decimal places using `calculate_gc_content()`.

The barcodes and respective metrics are written to a tsv file. The file name is formatted to include system date and time, `'TB_Barcode_Gen'`, and the specified accession.

4.4.4 Test Results

4.4.5 Future Improvements

4.5 Future Work and Extension Design

4.5.1 Automated Introgression Search Extension

5 References

5.1 Purpose

Identify the purpose of this SDD and its intended audience. (e.g. This software design document describes the architecture and system design of XX..).

5.2 Scope

Provide a description and scope of the software and explain the goals, objectives and benefits of your project. This will provide the basis for the brief description of your product.

5.3 Overview

Provide an overview of this document and its organization.

5.4 Reference Material

This section is optional. List any documents, if any, which were used as sources of information for the test plan.

5.5 Definitions and Acronyms

This section is optional. Provide definitions of all terms, acronyms, and abbreviations that might exist to properly interpret the SDD. These definitions should be items used in the SDD that are most likely not known to the audience.

Term	Definition
Software Design Document (SDD)	Used as the primary medium for communicating software design information.
Design Entity	An element of a design that is structurally and functionally distinct from other elements.

6 System Overview

Give a general description of the functionality, context and design of your project. Provide any background information if necessary.

7 System Architecture

7.1 Architectural Design

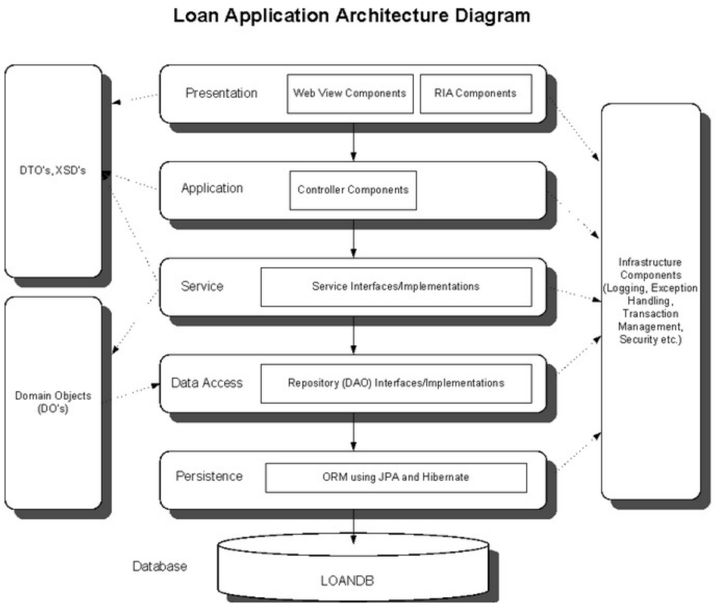
Loan Application Architecture Diagram
DTO's, XSD's
Domain Objects (DO's)
Presentation
Web View Components
RIA Components
Application
Controller Components
Service
Service Interfaces/Implementations
Data Access
Repository (DAO) Interfaces/Implementations
Persistence
ORM using JPA and Hibernate
Database
LOANDB
Infrastructure Components (Logging, Exception Handling, Transaction Management, Security etc.)

Figure 1: Architectural Design

7.2 Decomposition Description

7.3 Design Rationale

8 Data Design

8.1 Data Description

8.2 Data Dictionary

9 Component Design

10 Humnan Interface Design

10.1 Overview of User Interface

10.2 Screen Images

10.3 Screen Objects and Actions

11 APPENDICES

12 References