Technical Documentation for TersectBrowser+

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

May 5, 2025

1 Overview

This Technical Documentation describes the particular details of building upon the original TersectBrowser developed by Tomasz Kurowski into TersectBrowser+. It outlines the integration of multiple extension features, providing the user with additional display and data analysis capabilities.

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. This 'bolt-on' approach was chosen for the extensions in order to seamlessly integrate the new extensions with the original workspace within the project time constrainsts. Additionally, this approach allows the new extnesions to use up-to-date packages, resulting in enhanced functionality and a modern design. To run TersectBrowser+ on different platforms, the new extensions are hosted on NPM and can be linked as dependencies within the project.

Extensions added to TersectBrowser+ include the integration of a genome browser capable of displaying a variety of different tracks, notably variants and gene models, search functionality to identify and display regions containing high impact variants, and a tool to generate accession-specific barcodes. These extensions are hosted in a separate section of the project, but interact with both frontent and backend services to provide a seamless user experience.

3 Deployment

The GitHub project `https://github.com/Tersect-Browser/Tersect-browser` README provides detailed instructions for setup, dependencies, and deployment.

Here, we expand upon these instructions and give a detailed overview of the structure of each extension.

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

3.2 Dataset Upload

The Admin is defined as an individual with bioinformatics and software design knowledge, background comprehension for all sections of this Technical Documentation, and acting to facilitate the use of a deployed TersectBrowser+ by the plant breeder main user of the software. When a new dataset is requested for deployment on TersectBrowser+, the Admin will need to follow the below steps:

1. Collect background context and required data from the user.
    - Reference genome in fasta format (may be compressed). Size and chromosome number will impact the speed of dataset upload and deployment - TersectBrowser+ has been tested with a genome size of 1GB, and 20 chromosome pairs.
    - Resequenced genome dataset in VCF format (may be compressed, as a multi-sample VCF or as a directory of individual files). TersectBrowser+ has been tested with a dataset size of 500 VCF files representing individual accessions.
    - Any metadata associated with different accessions in the dataset, such as wild variety vs domesticated variety. This should be provided in a text file.
    - A GFF file of gene model information for the reference genome, produced using SNPeff software. If not using SNPeff, ensure that variant impact levels are categorised into 'High', 'Medium', and 'Low' impact. If not provided, the gene model track in the Variant Browser will be unavailable.

- Whether there are prior identified introgressions known to be functionally relevant for the species. These can be used to 'check' the ability of TersectBrowser+ to identify introgressed regions for the specified dataset.

2. Clone the GitHub repository of TersectBrowser+ to the local machine. Follow the README from TersectBrowser+ GitHub for installation instructions.

3. Download relevant files from section (1) to a new folder within the root directory of the local Tersect-browser repository.

4. Set environment variables `fasta`, `gff`, and `vcfs` according to their new location relative to the root of Tersect-browser.

5. Run the script `setup_new_tbrowser_dataset.py` using the environment variables of files as input:

   ```
   python setup_new_tbrowser_dataset.py -f ${fasta} -g ${gff} -V ${vcfs}
   ```

   - After running the script, there should be visible a new `config.json` file in the root folder, as well as a new folder `./~/mongo-data/gp_data_copy` containing fasta files, gff files, and VCF files along with their index files. The main Tersect index file should be in the folder above. This is where the server will look to find information on tracks when generating the Variant Browser panel.
   - This script checks for the install of modules such as SAMtools, BCFtools, and HTSlib, and will attempt to download these if not found. If a multi-sample VCF file is provided, it will split this file into individual sample level VCFs before proceeding. It will make sure these files are unzipped before using TERSECT CLI to build a tersect index for the whole dataset (it sources a previously created .tersect virtual environment to run this). After recompressing and indexing the fasta and VCF files, it will edit the `add_example_dataset.sh` script to point to the correct index file, and begin dataset addition to TersectBrowser+. Lastly, it will add the GFF file as a gene models track accessible within the browser.

3.3 Deployment with loaded dataset

1. Run the Mongo Database using:
   ```
   mongod --dbpath <pathto/mongo-data>
   ```

2. Change to the root of the Tersect-browser repo and run:
   ```
   nvm use 16
   npm start
   ```

3. Change to the ersect-browser/extension/genome-browser/ directory and run:
   ```
   nvm use 22
   npm start
   ```

4. Open TersectBrowser+ on `http://localhost:4200/TersectBrowserGP/`

4 TersectBrowser+

The project brief to extend the original TersectBrowser listed four main extension requirements, as well as a stretch extension. These requirements were provided as discrete goals for the project, and informed the main working priorities for the project timeline, however in practice the four extensions do have significant overlap in terms of code structure. For this reason, the accompanying TersectBrowser+ Article includes the extensions Genome Browser and Gene Models in the same section, while this Technical Documentation keeps the two separate according to the chronological timeline of developments to these extensions. Architecture diagrams are used to display the overlap between various extension components, especially relating to how they orchestrate data flow through the entire software.

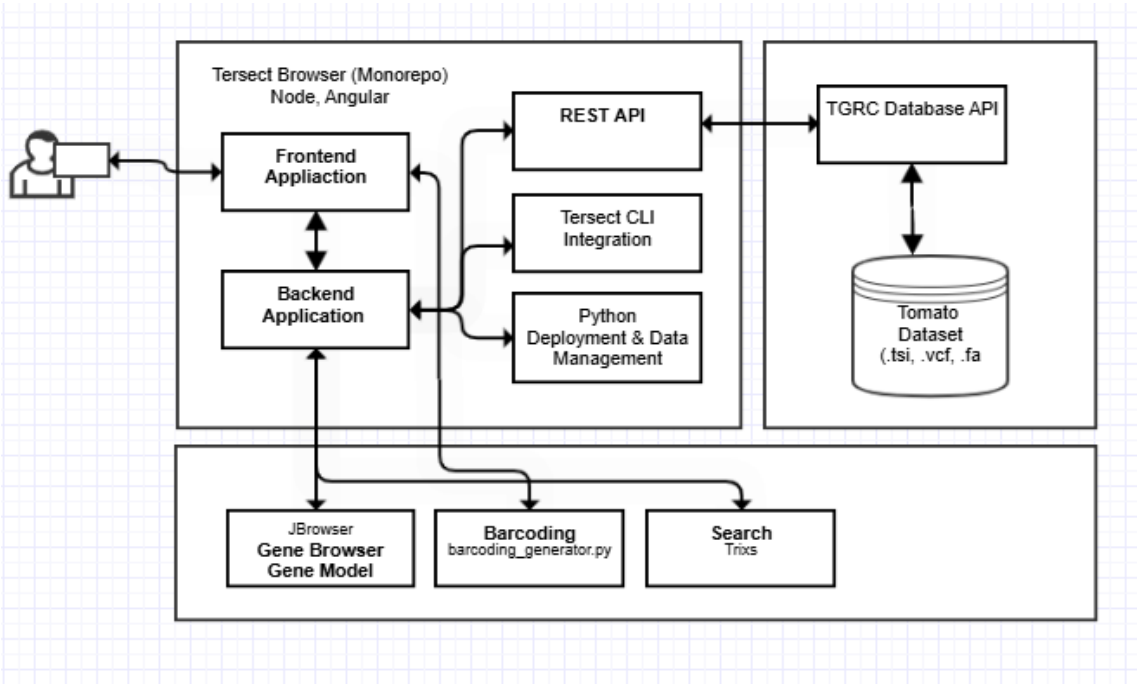

Figure 1: Summary Architecture of TersectBrowser+.

4.1 Genome Browser Extension

4.1.1 Rationale

The main benefit of TersectBrowser relative to similar genome comparison software for displaying key differences within resequenced datasets is its use of Tersect CLI, to index and then quickly query these genomes for investigation at user-defined scale. However, in order for TersectBrowser+ to be competitive as a tool with other browsers, it was necessary to implement features that enhance further investigation within the scope of the same browser. The main view of the original TersectBrowser displayed the density of variants along each chromosome in comparison to the reference genome. To enhance this view, a variant browser with multiple tracks providing extra information at a per-base level was added to enable to user to directly correlate variant density with accession-specific variants.

After a review of node-compatible browser tools such as CRAMER, JBrowse was ultimately chosen to carry out this extension request, due to its high customisability, up-to-date features, and active technical support. The initial build of JBrowse into TersectBrowser+ as an extension focused on retaining features most relevant to the user aims, and bringing the styling of the panel in line with the rest of the page. This extension also involved generating a popup window of the same JBrowse component to be initialised when the user was interested in a specific accession or bin region.

4.1.2 User Interface

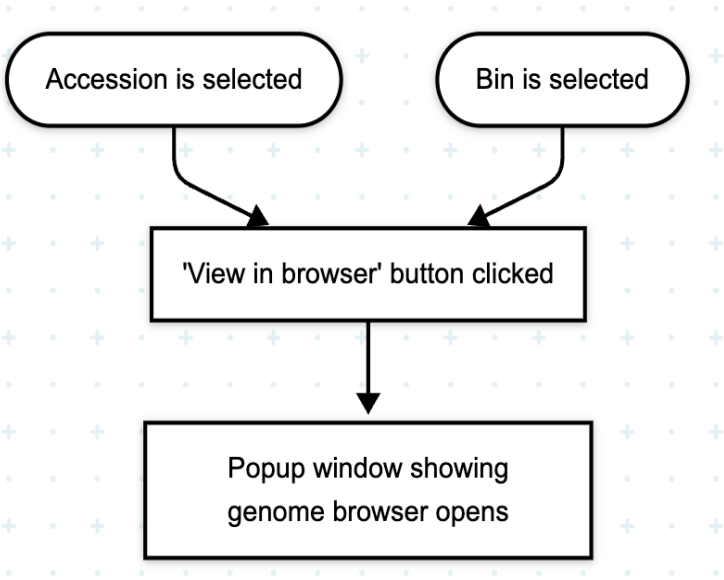

Figure 2: UI diagram for Genome Browser.

Feature Description

- JBrowse window:
  - Main window is located at the top of Tersect Browser, below settings bar but above the heatmap
  - Main window shows only sequence tracks and linear genome scale. Interactive features such as zoom anc chromosome selection have been removed from the main window. The container window has a dynamic offset, which changes according to tree view and zoom level to ensure the chromosome scales between TersectBrowser and JBrowse line up.
  - By default, the main window shows only the gene models track.
  - Popup window provides a more sandbox experience, providing more user interaction. The user can change zoom level and displayed chromosome, and two scale bars enable both precise and rapid navigation. The hamburger menu on the top left provides the capability for the user to upload their own data to visualise unique tracks.

- By default, the popup window shows the reference sequence track, gene models track, and variant track for the specified accession.
- Tracks:
  - Reference sequence
  - Variant tracks for each loaded accession
  - Gene models FeatureTrack (Gene Models Extension)
- Syncing of the JBrowse window and the TersectBrowser heatmap includes the following:
  - Zoom - both by clicking the plus/minus buttons and by scrolling with the mouse wheel.
  - Bin size.
  - Selected interval.
  - Chromosome.
  - Horizontal scrolling along the x-axis.

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 method `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 `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

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:

$$bpPerPx = ((props.location.binSize) * (100/props.location.zoomLevel)) \quad (1)$$

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 left-margin offset requires dynamic syncing to ensure the scale between JBrowse and the heatmap remain aligned. 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.

The position of the JBrowse container is dynamically set in `tersect-browser.component.ts`, with the `style.margin-left.px` set to `offsetCanvas`, which holds thw pixel width of the phenetic tree. As a result, the position of the JBrowse container dynamically adjusts to different canvas states to ensure the scale bar between JBrowse and the heatmap remains aligned.

Horizontal Scroll Horizontal scroll inputs from the user are disabled inside the main JBrowse window through the implementation of a `Plugin` of the `DisableScroll` class in `JbrowserWrapper.tsx`, which overrides the JBrowse configuration to prevent horizontal scrolling

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.

When a new dataset is added, the custom addition script automatically updates the names of tracks and assembly in the relevant files within the genome-browser extension. This flexibility allows the Genome Browser extension to be accessible by plant breeders working on many different datasets. TersectBrowser was initially validated with a human genome dataset, so it was important to maintain this flexibility of input in the new extensions.

4.1.5 Future Improvements

In addition to displaying the reference track, gene models track, and variants track, additional feature tracks, such as a multiple sequence alignment view, could be added to the genome browser.

Currently, the genome browser popup window is limited to opening a single window within the TersectBrowser+ webpage. A button to open the popup window in a new tab could be added, which would allow the user to open multiple detatched popup windows and view them simultaneously.

4.2 Gene Models Extension

4.2.1 Rationale

Once the Genome Browser panel was added to the main page of TersectBrowser+, this facilitated the inclusion of a reference gene model track. This allows the plant breeder user to identify an area of high variant density on the heatmap, and then check the gene model track to investigate whether this region occurs near a gene of interest.

4.2.2 User Interface

Feature Description

- The gene models track is displayed in the main JBrowse window.
- In the popup window, the gene models track is displayed as the second track, underneath the reference sequence track and above the variant track.

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 track stored in `tracks.ts` (gene models track) is added to the `viewState`.

For the popup window, the JBrowse `viewState` is configured when an accession is selected in `JbrowseWithAccession.tsx`, as described in Genome Browser Extension. If a track with a `trackId` matching the selected accession is found, the viewer displays the reference sequence track from `assembly.ts`, the first track defined in `tracks.ts` (gene model track) and the selected accession variant track.

4.2.4 Test Results

This extension has been tested with the additional Soybean dataset, with correct functionality. The relevance of the gene models in the GFF file depend on what the user provides in the dataset, and how the GFF file was created.

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 selected intervals and identify which accessions contain high/medium/low impact variants within that region. The bins containing high impact variants for that gene will be highlighted at the accession level, allowing high resolution investigation of variants. The highlighted bins may then be selected and opened in the popup Genome Browser Extension window for further investigation.

4.3.2 User Interface

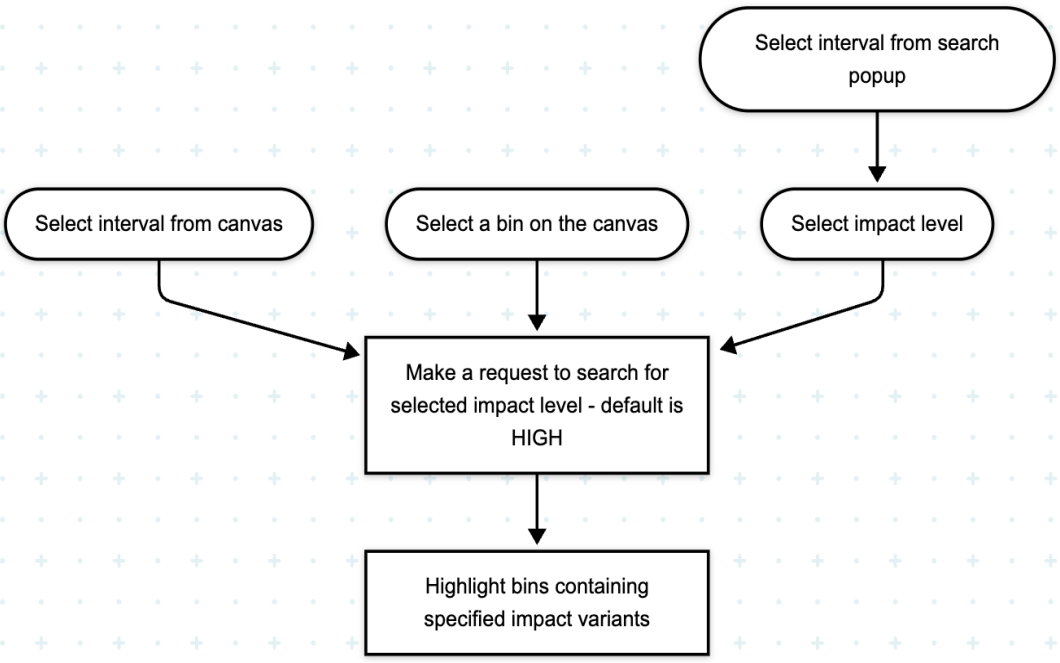

Figure 3: UI digram for Feature Search.

Feature Description

- `Open Variant Search` button in TersectBrowser+ header, opens popup window for user to select advanced features.
- Popup window where user can select interval, choose impact level, and search button to initiate search.

- Output: Bins in the canvas are highlighted red at the chromosomal position where the variant is located, and only for accessions containing variants of the specified impact.
- Clear button located in TersectBrowser+ header to clear highlighted bins from canvas and restore original heatmmap.

4.3.3 Technical Description

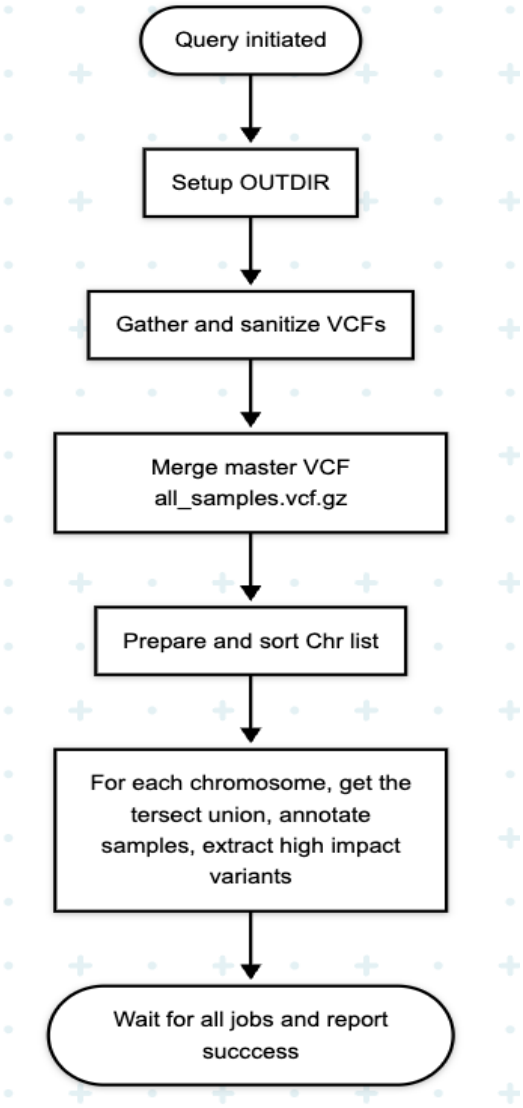

Figure 4: Data flow diagram for Feature Search.

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 The clear button is located to the right of the `Open variant search` button and styled with `fa fa-refresh`. It contains the event handler `(click)="refreshBin()"` that calls the `refreshBin()` method in `BinDrawService`. This method calls `generatePlotArray()`, which re-creates the bin view, and `updateCanvas()`, which redraws the canvas using the re-created bin view.

4.3.4 Test Results

4.3.5 Future Improvements

Currently, only bins on the TersectBrowser+ heatmap are highlighted in red to indicate the positions of high-impact variants. Highlighting could be extended across the Genome Browser and Gene Models extension, so that gene models matching the parameters of the feature search query are also highlighted in red.

4.4 Barcode Generation Extension

4.4.1 Rationale

A key aim for plant breeders when comparing their own plant strains with other cultivars in resequenced datasets is to be able to identify their strain at the genome level. This would allow for protection of intellectual property in the case of an advantageous new introgression that can be uniquely identified. The Barcoding extension allows this unique accession-specific identification by generating a range of barcodes with a user-specified length and maximum number of SNPs, so that the barcodes are easily analysed in wetlab environments. Metrics comparing these putative barcodes are also provided to the user, and available for local download.

4.4.2 User Interface

Frontend Display Feature Description When the user clicks on a bin, the popup menu appears and the button "Create barcode" is visible, styled with a barcode icon. When the user clicks on this button, a popup window appears in the centre of the screen. The window title is the corresponding accession name. The window contains the header "Barcode Generator" and contains a description of what is being generated. There is a box for the user to input barcode size, with default set to 150, and the maximum number of variants allowed per barcode (this is an optional parameter. Leaving this blank means all possible barcodes will be returned).

At the bottom is a green "Generate" button. Hovering over the button turns it dark green, and clicking on this button will call the backend scripts to generate the barcodes output. Whilst scripts are running and barcodes are being generated, a spinner is added to the button to indicate loading. After the script has been generated, the spinner disappears and another green button "Download" appears

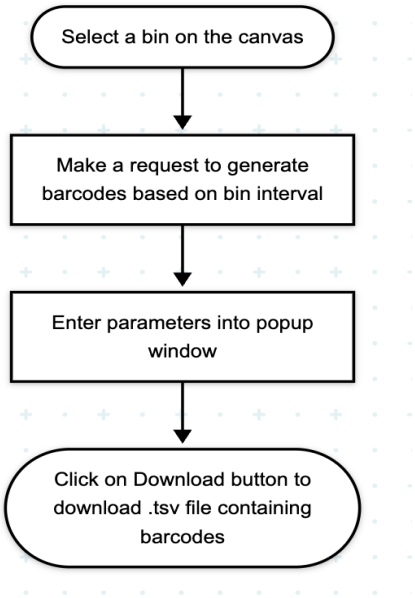

Figure 5: UI diagram for Barcode Generation.

at the bottom left of the window. Clicking on the button downloads the tsv file containing the barcodes.

Backend Script Feature Description Given input parameters specified by the user, barcodes are generated via the backend and can be downloaded in tsv 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.
- **Chromosome** - The chromosome on which the barcode is located.
- **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

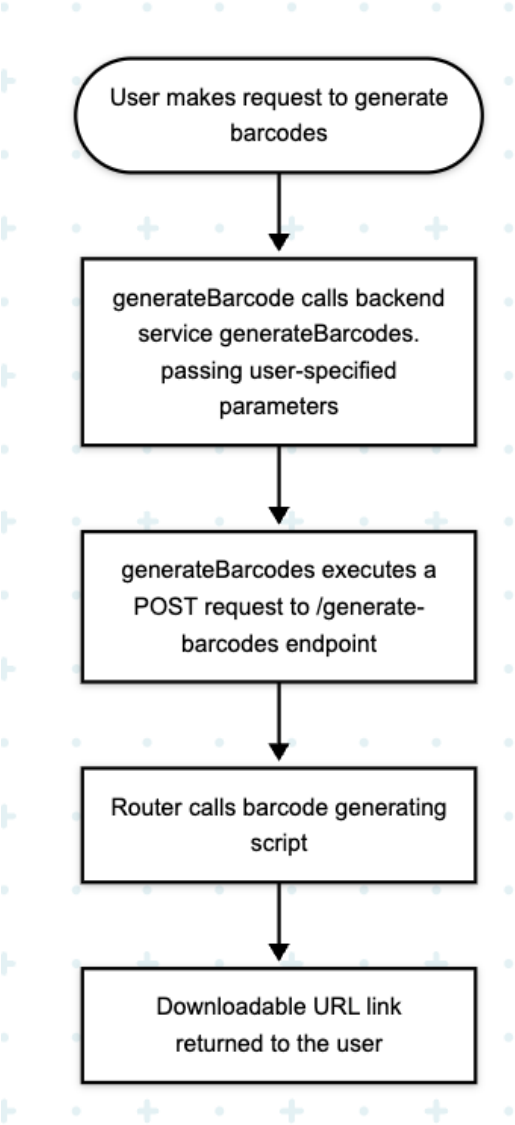

Figure 6: Data flow diagram for Barcode Generation.

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

Bin menu barcode button :
A third menu item with the label `Create barcode` is added to the `getAccessionItem()` method in `plot-click-menu.component`. The item is styled with the `fa fa-barcode` icon, and on command opens the `ModalService openBarcodeModal()` method, passing as arguments the plot state chromosome name and the bin accession label and start and end positions.

Modal Service Component :
The observables `barcodeVisible$`, `barcodeTitle$`, `barcodeChromo$`, `barcodeStart$` and `barcodeEnd$` are added to `modal.service`. The method

`openBarcodeModal()` is added, accepting as arguments `accessionName`, `chrom`, `startPosition`, and `endPosition`. These arguments are saved to their corresponding observables.

A new modal, `global-barcode-modal`, is created. The modal is imported to app.module and added to app.component. The `global-barcode.component.ts` imports `ModalService` and `TersectBackendService`. Variables to hold the modal state, modal loading state, modal title, barcode size, chromosome, interval start position, interval end position, maximum variants per barcode, the downloadable URL and filename are defined. On initialisation, modal state, modal title, chromosome, interval start position and interval end position are saved to observables in modal.service.

The method `generateBarcode()` calls and subscribes to the `tersectBackendService generateBarcodes()` method, setting the `isLoading` constant to "True" and passing as arguments `modalTitle`, `chromosome`, `startPosition`, `endPosition`, `barcodeSize`, and `maxVariants`. The returned URL and filename are saved to the variables `downloadUrl` and `downloadFileName`. Once returned, `isLoading` is set to "False" and `barcodeSize` is set back to "150".

The popup window is reset when closed by `resetModal()` on closing, which sets the `barcodeSize` to "150", `maxVariants` and `downloadUrl` to null, and `downloadFileName` to an empty string, removing the "Download" button from the popup.

Modal Service Styling :
The modal template is defined in `global-barcode.component.html`. The popup window is visible depending on the modal state isVisible, and is centrally position with `modalTitle` set as the header. Explanation about the barcode service is described in the `modal-title` and `modal-subtitle` divs, and the `input-container` div contains input boxes for barcode size and maximum number of variants, the button "Generate" which calls `generateBarcode()` and simultaneously shows a `progressSpinner` whilst the barcodes are being generated. An anchor displays a clickable "Download" p-button that points towards the generated barcode file. The filename is set to `downloadFileName`, and "Download" is only visible once a `downloadUrl` has been generated for the barcodes file.

Styling is defined in the `global-barcode.component.css`. Button background colour is set to `#459e00`, background hover colour is set to `#33b357`, and text colour is set to white. The input boxes and labels are given a set width of 200px.

Router and Backend service :
A new route, `/generate-barcodes`, is added to `tersect-router`. This router receives in the request the following parameters: `accessionName`, `chrom`, `start`, `end`, `size` and `maxVar`. Paths to the tsi index and fasta file are defined. The backend script, `barcode_finder.sh`, is called, with the above parameters passed as arguments. A `downloadableURL` is created, pointing to the created file, and returned in json format.

The method `generateBarcodes()`, is added to `tersect-backend.service`. It sends a HTTP Post request to the endpoint `/generate-barcodes`, triggering barcode generation. The method returns an observable containing the

`downloadableURL` pointing to the generated barcode file. The method passes the following parameters:

- **accessionName** - a string containing the specified accession name for which to generate unique barcodes
- **Chrom** - a string containing the chromosome identifier for the chromosome where the barcodes will be generated
- **Start** - a number specifying the start position of the interval in which to generate barcodes
- **End** - a number specifying the end position of the interval in which to generate barcodes
- **Size** - a number specifying the length of the barcodes
- **maxVar** - an optional parameter, specifying the maximum permissible number of variants located within the barcode. Either a number or null, indicating all barcodes with all numbers of variants are generated

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

Currently, barcode viewing is limited to the downloaded file. It can be extended to display barcodes in the popup window, with the variant highlighted in red.

4.5 Future Work and Extension Design

4.5.1 Automated Introgression Search Extension

5 APPENDICES

6 References