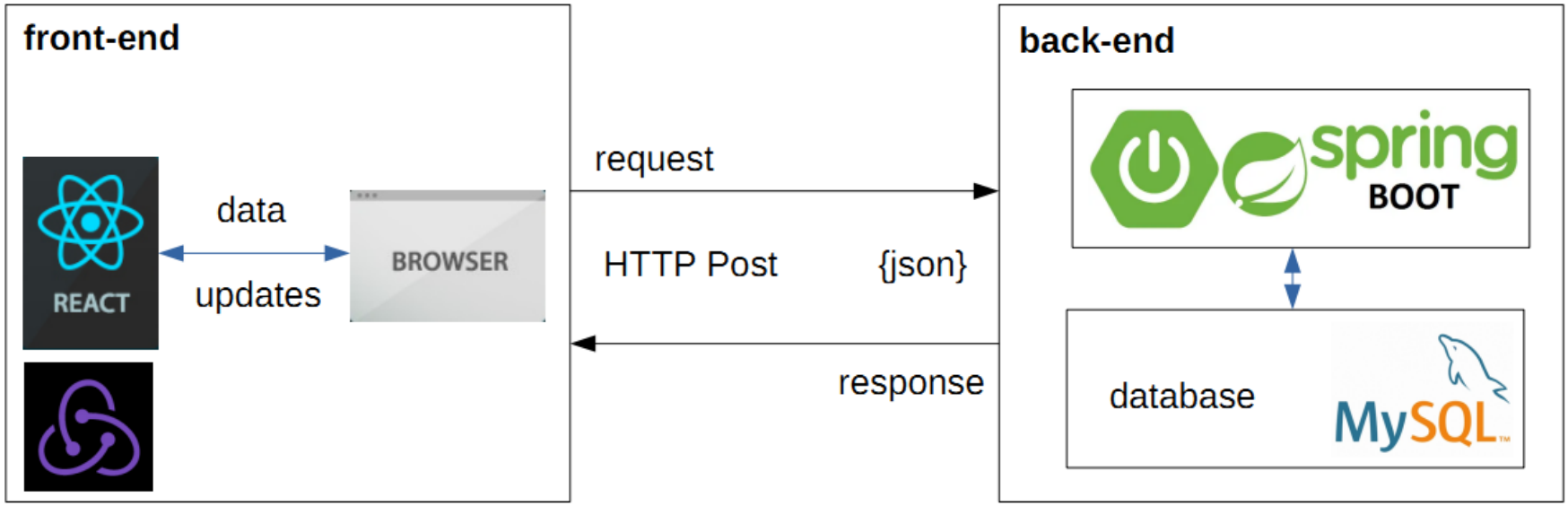
*Student project:*

**Application for displaying MySQL database in the form of unordered tree.**

**Application structure.**




The application consists of a client (front-end) and server (back-end) parts.
Back-end is based on Spring Boot using the REST API in Java. The client part is written using JS, React and Redux framesworks.

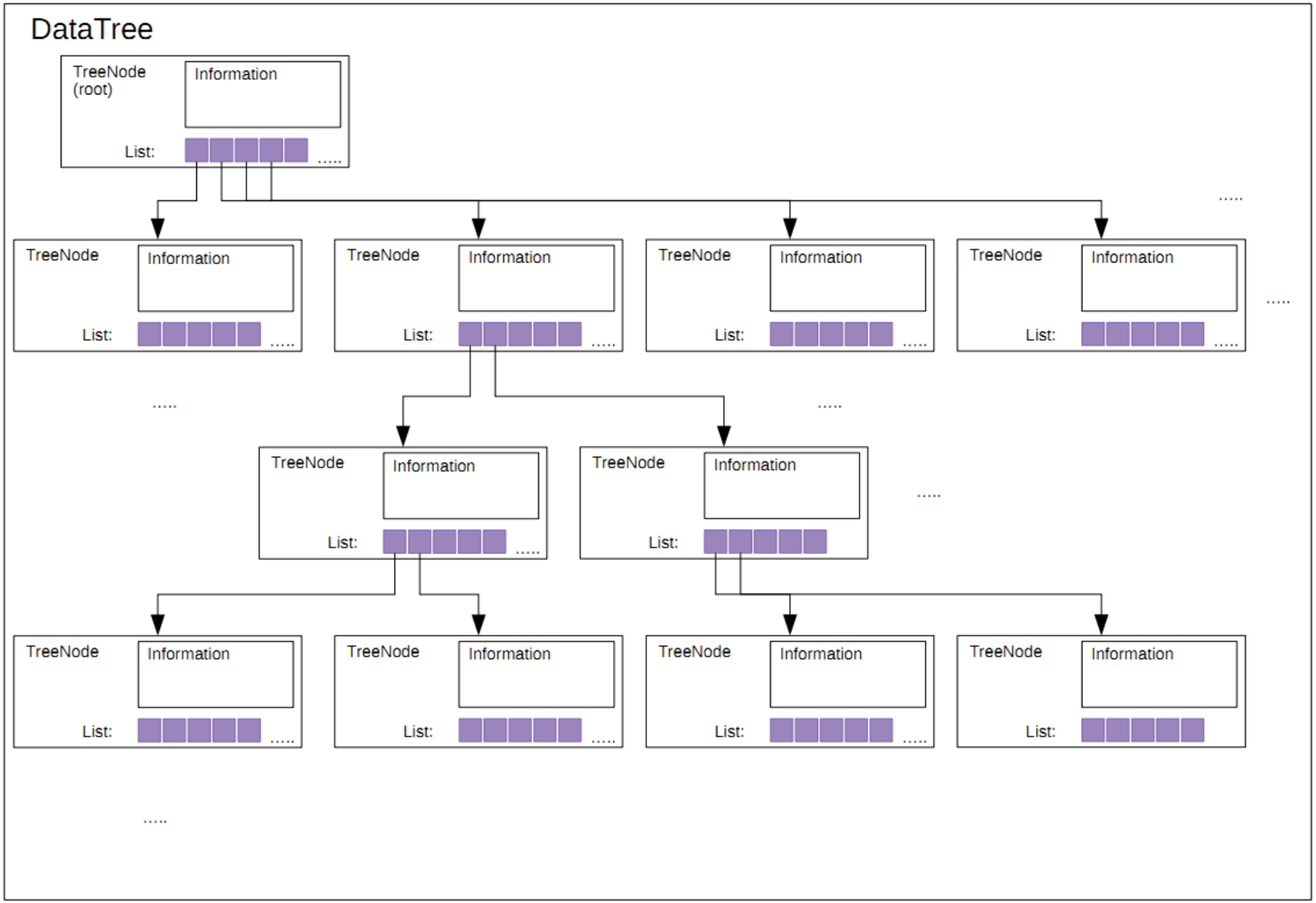
## Back-end.

```
DataTree
DataTree()
DataTree(TreeNode)
addRoot(TreeNode)                                          void
addRoot(Information)                                       void
addNodeWithDataToCurrentNode(Information)                  void
addNodeWithDataToCurrentNode(Information, TreeNode)        void
treeTraversalDFS()                                         List<TreeNode>
traversalDFS(TreeNode, List<TreeNode>)                     void
treeTraversalBFS()                                         List<TreeNode>
findBFS(Predicate<TreeNode>)                               TreeNode
findBFSofList(Predicate<TreeNode>)                         List<TreeNode>
findDFS(Predicate<TreeNode>)                               TreeNode
findNodeDFS(TreeNode, TreeNode[], Predicate<TreeNode>)     void
amountNodes()                                              long
amountOfNodesTreeTraversalOfDFS(TreeNode, long[])          void
currentNode                                                TreeNode
root                                                       TreeNode
```

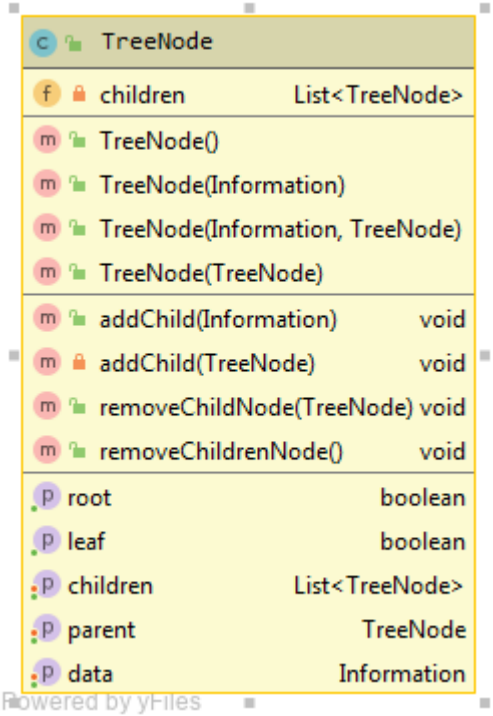


The database model is represented by a data structure in the form of an unordered tree (recursive tree). The tree is implemented using the DataTree class in the program.

# Back-end. Tree scheme.



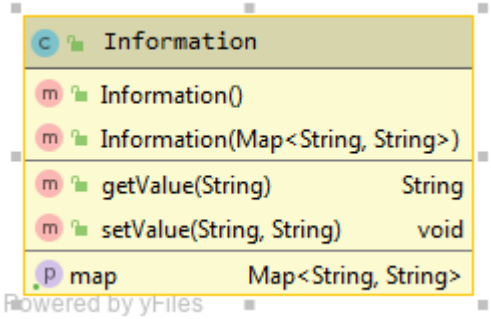
## Back-end.




The tree consists of nodes, which are elements* of the database and are objects of the TreeNode class.

*elements – schemas, schema, tables, table, columns, column, triggers, trigger, functions, function, procedures, procedure, views, view etc.

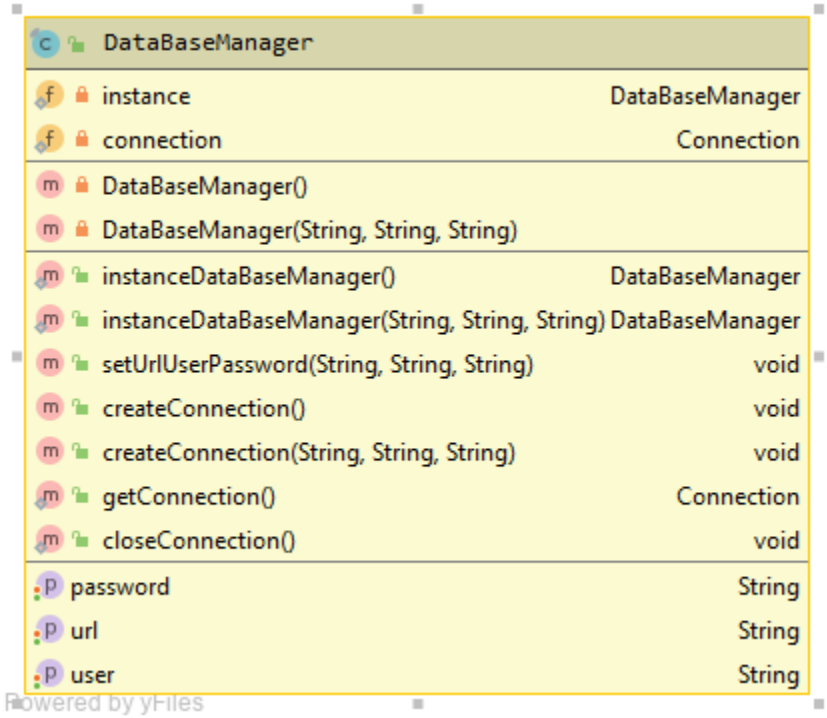
## Back-end.




A database element is described by a special class Information and is aggregately included as an object in a node. Information is implemented using Map <String, String>.

In addition to the keys describing the database elements, there is an "id" field that is necessary for identifying nods. By "id" nodes, searches are made in width and depth.

## Back-end.




To connect and disconnect a database, the project uses the DataBaseManager class (work with java.sql.DriverManager and java.sql.Connection classes).

## Back-end.

```
DataBaseMySQLQueries
DataBaseMySQLQueries()
getTypeOfColumnOfTable(String, String, String) String
getTypeOfColumn(String, String, String)        String
getColumnsOfTable(String, String)              List<String>
getNamesOfColumns(String, String)              List<String>
getFunctionDDL(String, String)                 String
getFunctionsOfSchema(String)                   List<String>
getProcedureDDL(String, String)                String
getProceduresOfSchema(String)                  List<String>
getProductName()                               String
getSchemas()                                   List<String>
getAllSchemas()                                List<String>
getAllSchemasV2()                              List<String>
getTableDDL(String, String)                    String
getTablesOfSchema(String)                      List<String>
getTables(String)                              List<String>
getCreateTimeOfTable(String, String)           String
getTableRowsOfTable(String, String)            String
getTableAvgRowLengthOfTable(String, String)    String
getVersionOfTable(String, String)              String
getTriggerDDL(String, String)                  String
getTriggers(String)                            List<String>
getViewDDL(String, String)                     String
getViews(String)                               List<String>
```

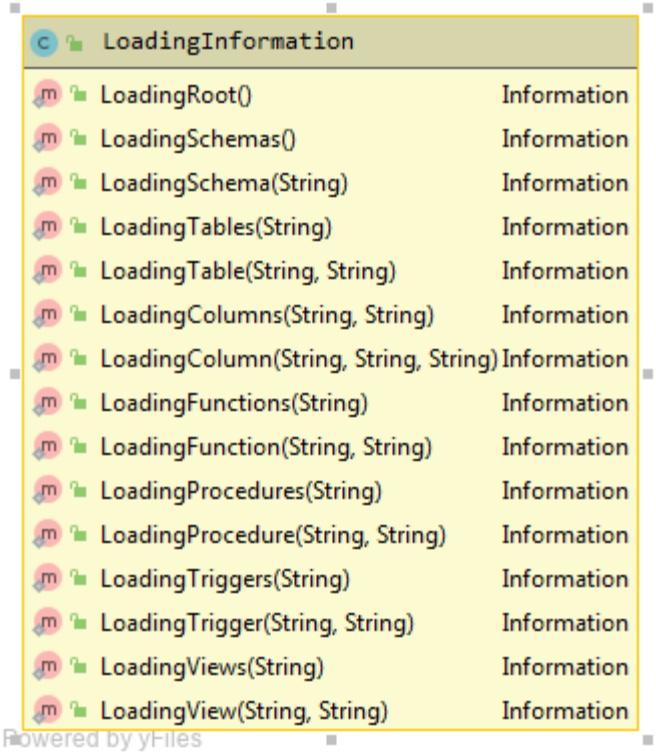


```
Requests
TABLE_NAMES              String
TABLE_CREATE_TIME        String
TABLE_TABLE_ROWS         String
TABLE_AVG_ROW_LENGTH     String
TABLE_VERSION            String
DDL_OF_TABLE             String
VIEWS                    String
DDL_OF_VIEW              String
PROCEDURES               String
DDL_OF_PROCEDURE         String
FUNCTIONS                String
DDL_OF_FUNCTION          String
TRIGGERS                 String
DDL_OF_TRIGGER           String
COLUMN_TYPE              String
COLUMNS_NAME             String
SCHEMAS_ALL              String
Requests()
```



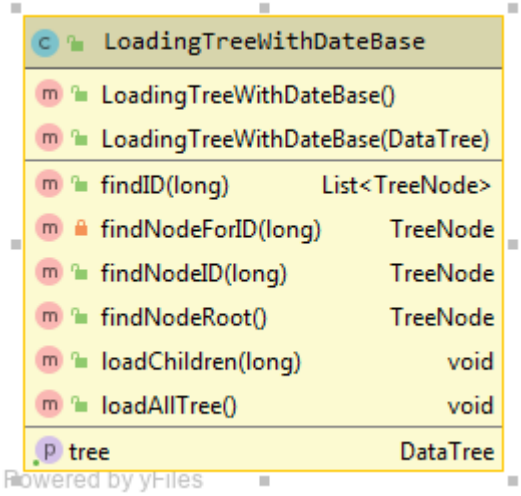
DataBaseMySQLQueries is a class for organizing database queries and getting results.
The Requests class is a constant string variables for SQL queries.

## Back-end.



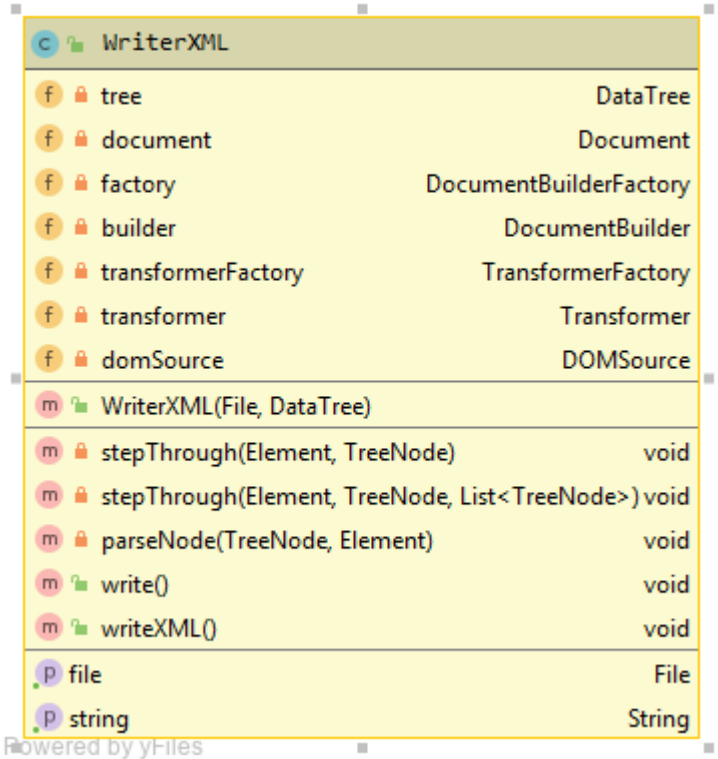

To fill in information about the elements, the LoadingInformation class is used, which includes static methods-loaders for each type of database element.

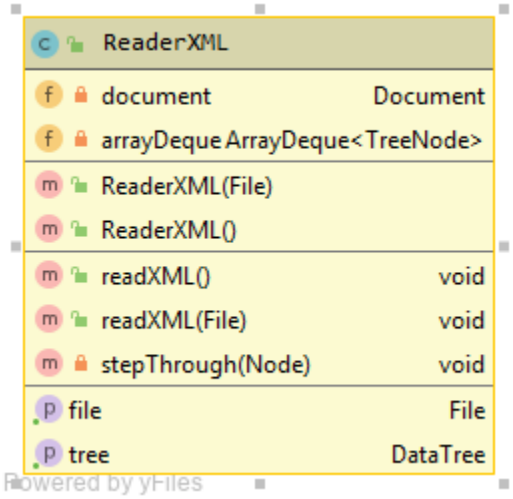
## Back-end.




To form a tree from a database, the LoadingTreeWithDateBase class is designed, which allows you to search for a node by "id" (implemented search in width and depth), load all nodes (lazy loading), load all levels of the database (forming a full database tree), the class uses the methods of the DataTree class.

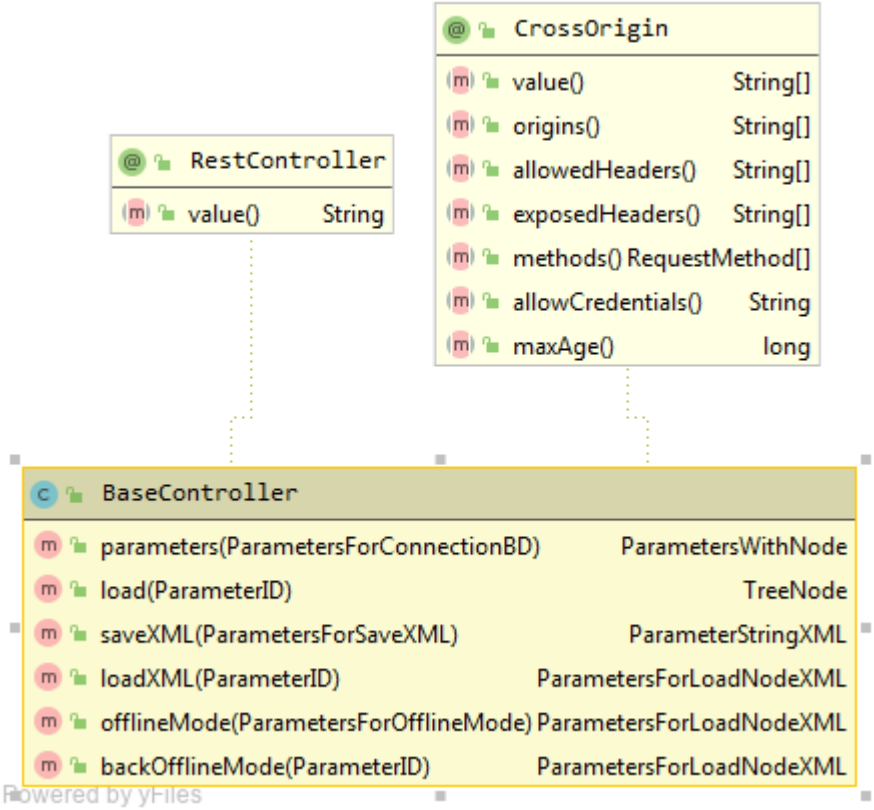
## Back-end.




In the server part of the program, you can save the tree to an XML file and get the tree from an XML file using the WriterXML and ReaderXML classes.

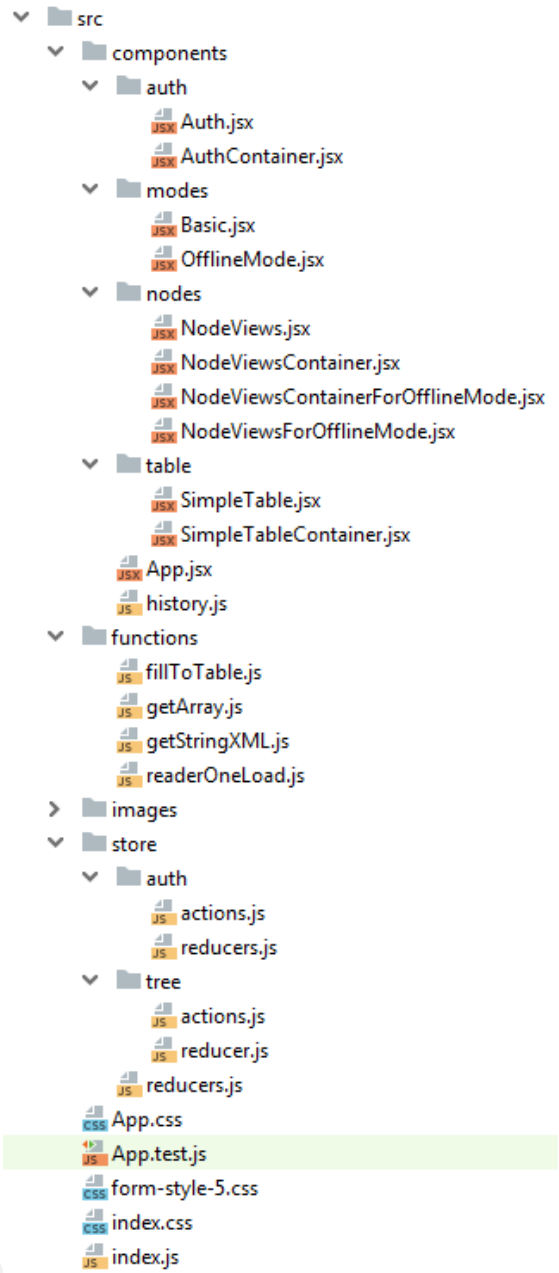
## Back-end.




BaseController REST controller consists of methods processing requests from the client side:

- getting login, password and path to the database;
- lazy loading by "id";
- saving the file in XML format and transferring it as a string to the client;
- transferring of the tree from the saved XML file in main mode;
- switch to offline mode with the transfer of a tree from a saved XML file;
- return to main mode.

## Front-end.

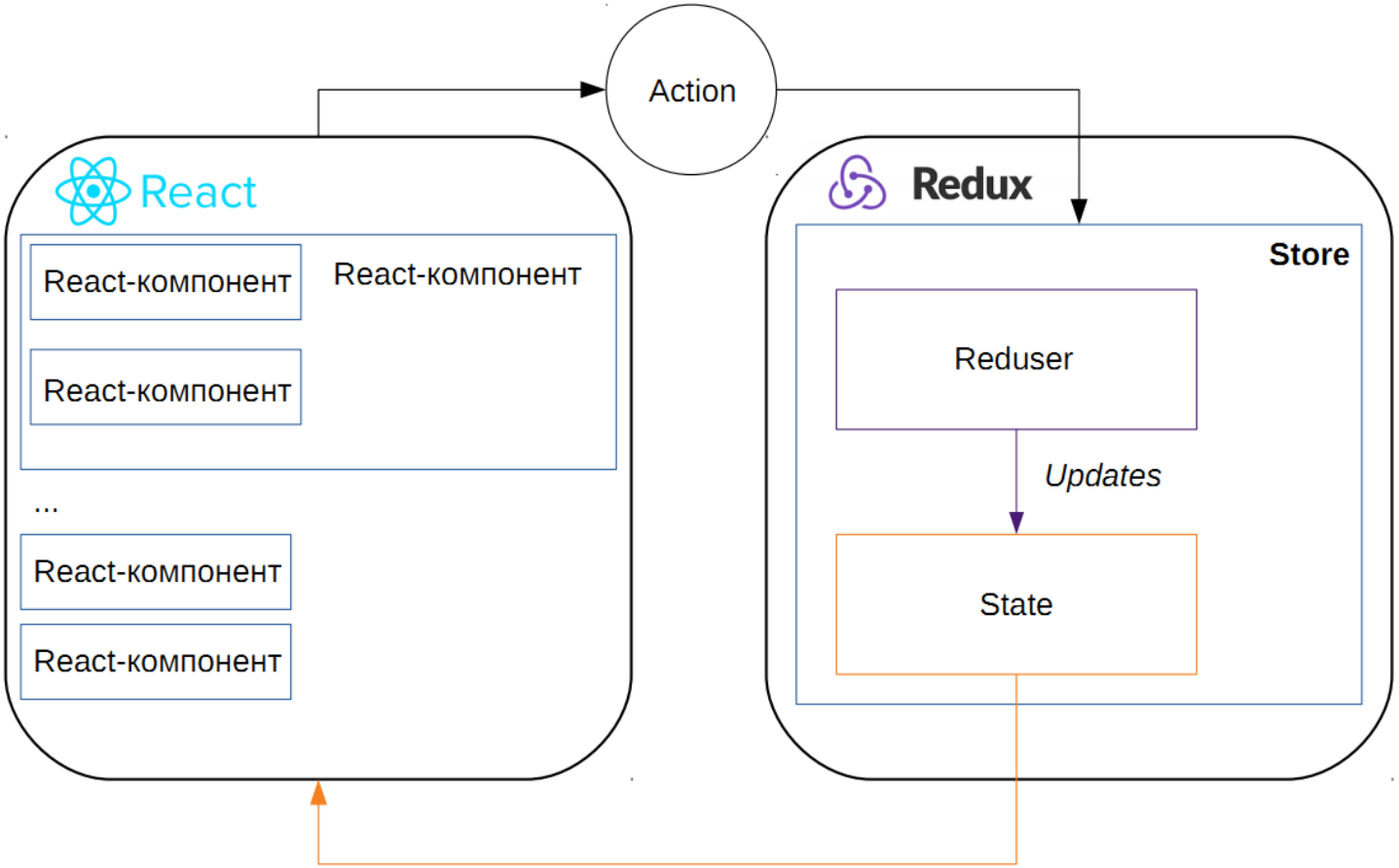



The client part includes:

- store, contains application States, it stores various objects for working with frontend components. (tree, node parameters for the table, arrays for displaying tree buttons, various string variables). The only way to change the state inside it is to send an action to it using dispatchs.
- react-components are used to create elements of web pages (tables, buttons, wrappers for components, smart components, stupid components).
- reducers are functions that accept commands as input and change state. The program is combined using combineReducers into a common reducer.
- dispatches - for passing variables to reducers. dispatch (action), store.dispatch (action) - send commands, and this is the only way to cause the state of the store to change. The program used dispatches with bindActionCreators* and anonymous store.dispatch(action).

* bindActionCreators - used when some action creators are passed down to a component that knows nothing about Redux and there is no desire to pass a dispatch or Redux-store component to it. (bindActionCreators wraps each action in dispatch).

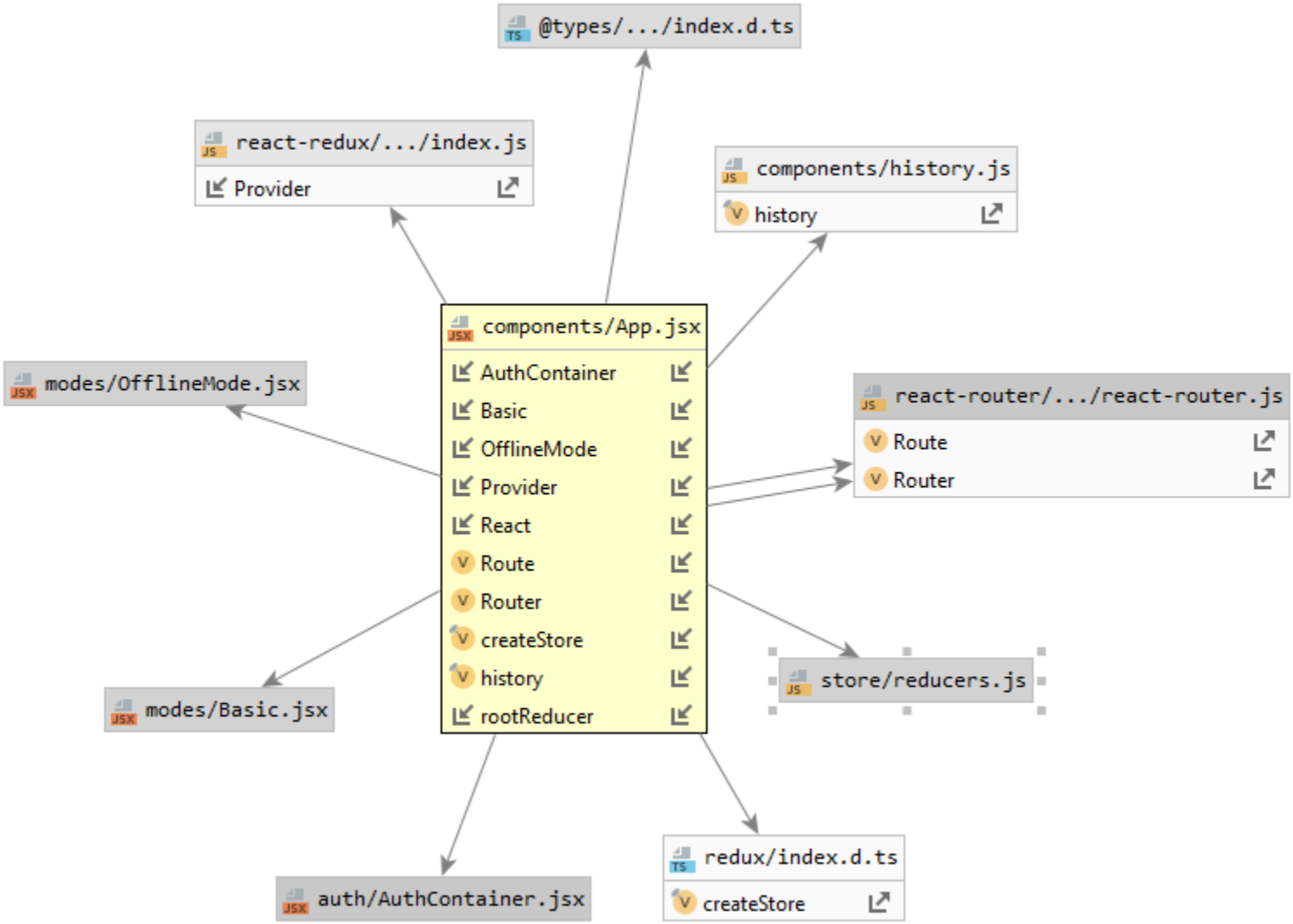
**Front-end. Scheme of interaction between React components and Redux.**




Redux solves the problem of managing state in an application by storing global data in a global State and centrally changing it.

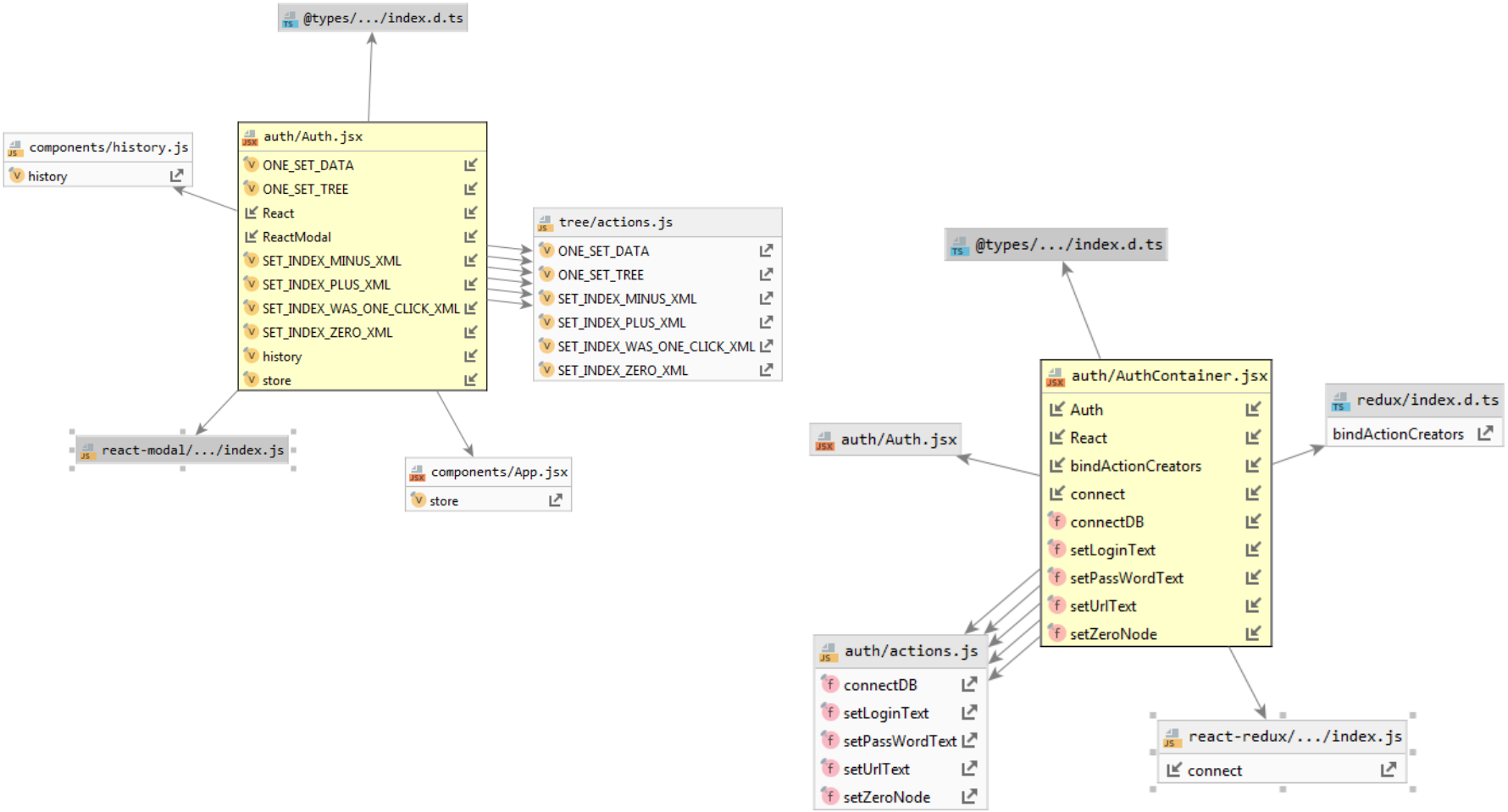
- Components generate events (actions).
- Reducer is a logic module that modifies state.
- State is common to all components.
- Reducer + State = Store.
- Components are updated when state changes.

# Front-end.




App.jsx - overwrap, root component.

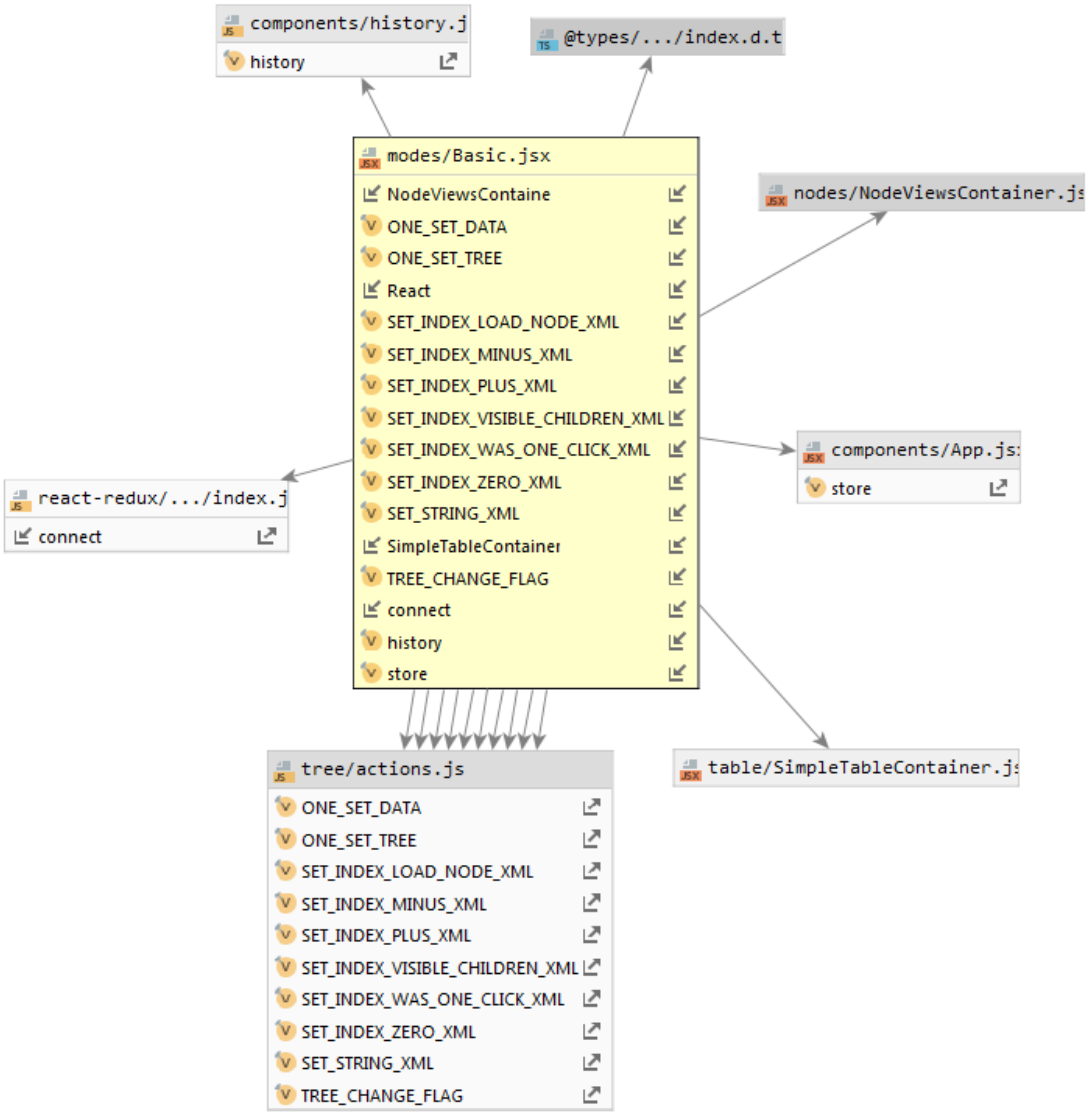
# Front-end.




Auth.jsx and AuthContainer.jsx are components for authorization.

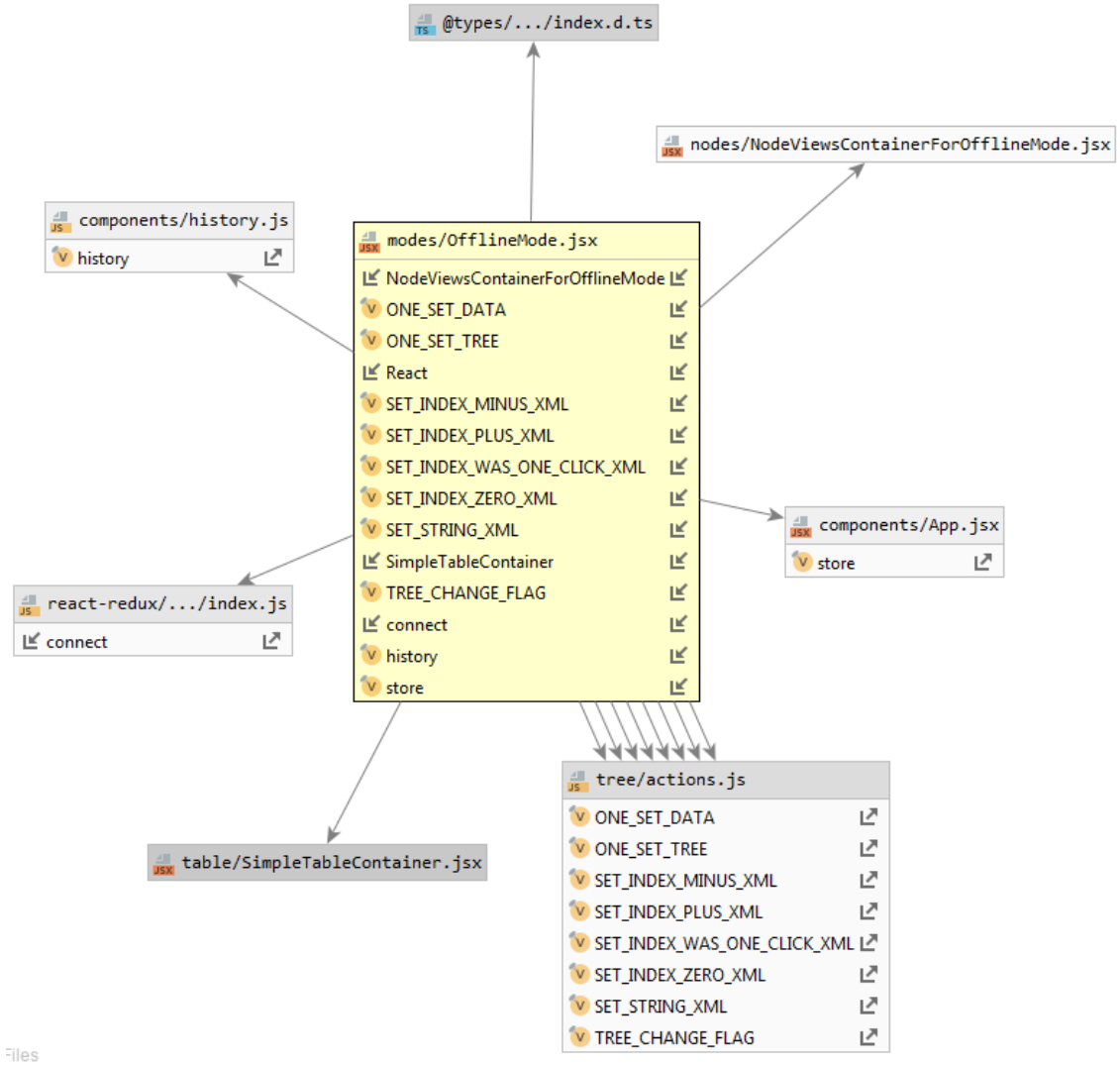
Note: for the ... Container.jsx components, we define mapStateToProps() functions to read the state and mapDispatchToProps() to send the event. We generate the component by passing the created functions to connect() (connect - connecting the React component to store Redux).

# Front-end.



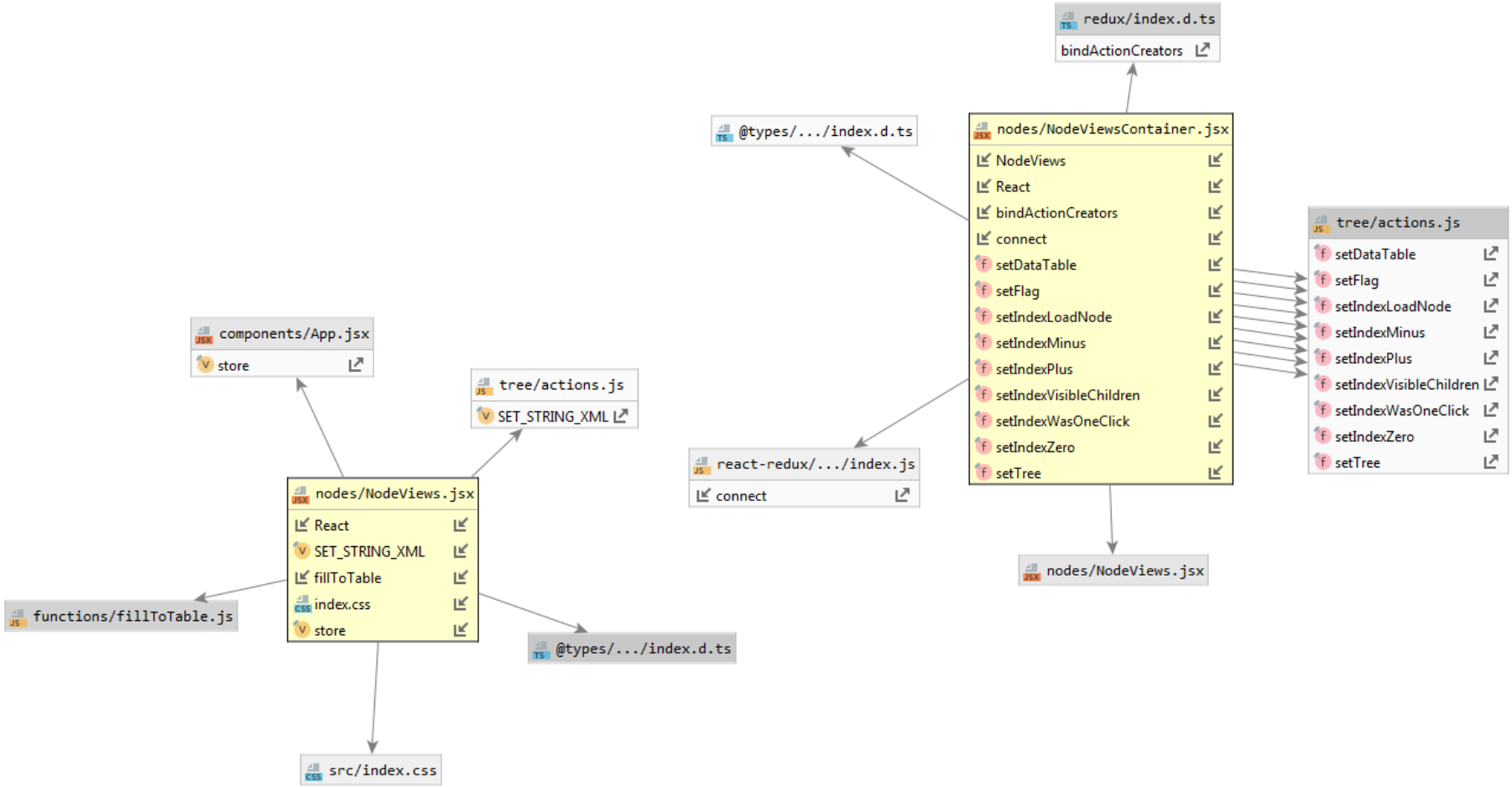

Basic.jsx - a wrapper for the main mode.

# Front-end.



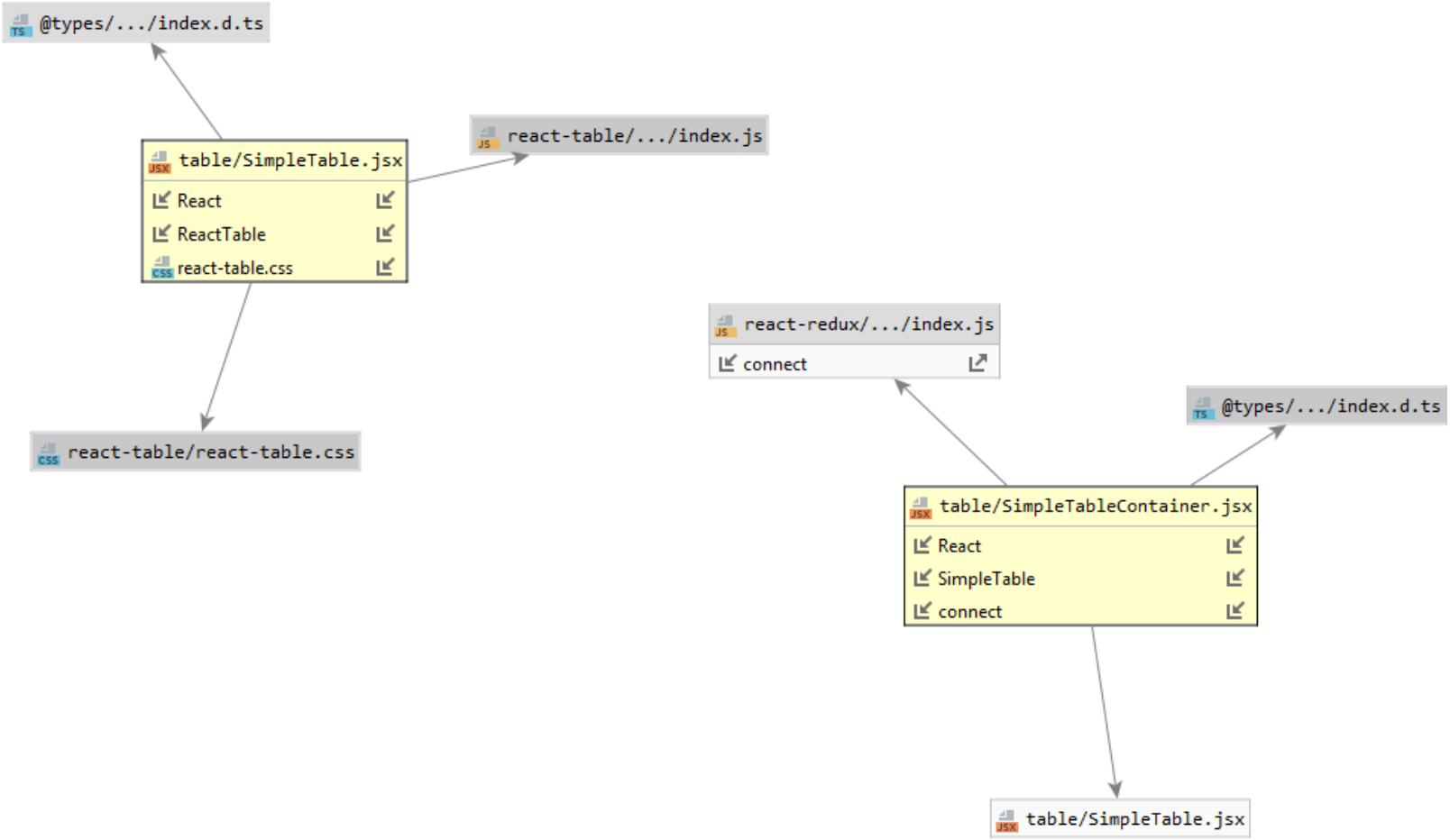

OfflineMode.jsx - a wrapper for offline mode.

# Front-end.



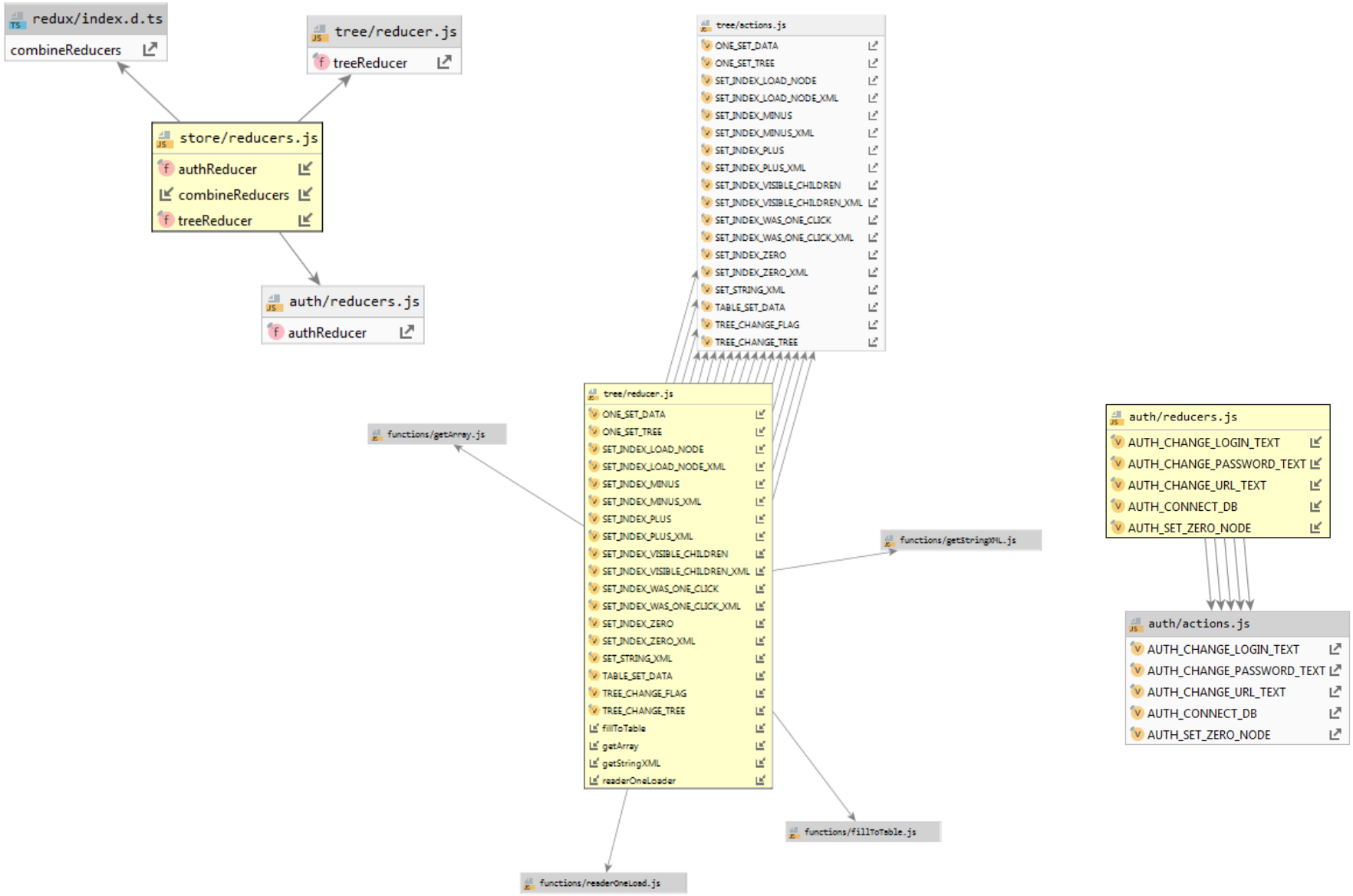

NodeViews.jsx and NodeViewsContainer.jsx are components for displaying a tree.

# Front-end.



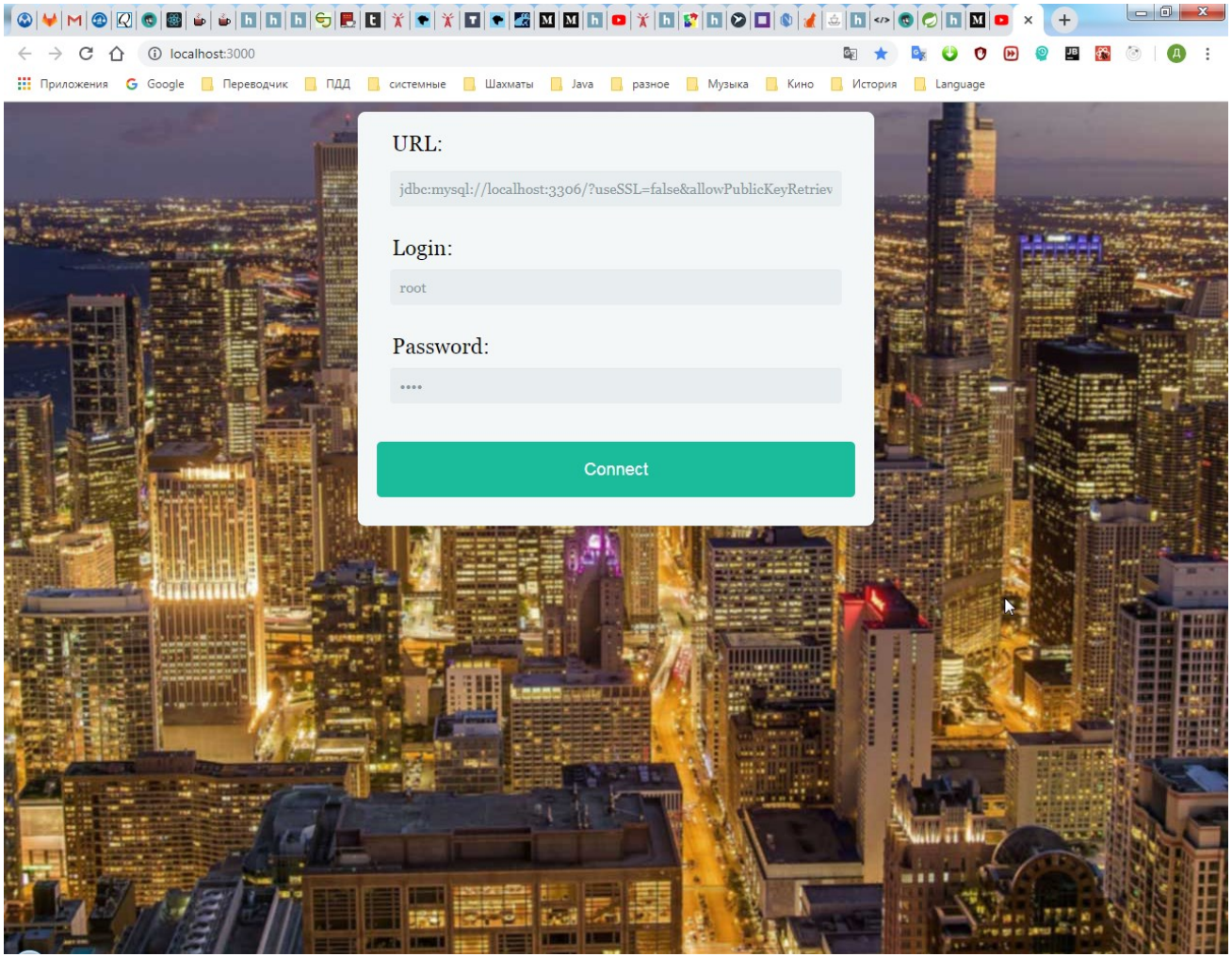

SimpleTable.jsx and SimpleTableContainer.jsx are components for the table.

# Front-end.



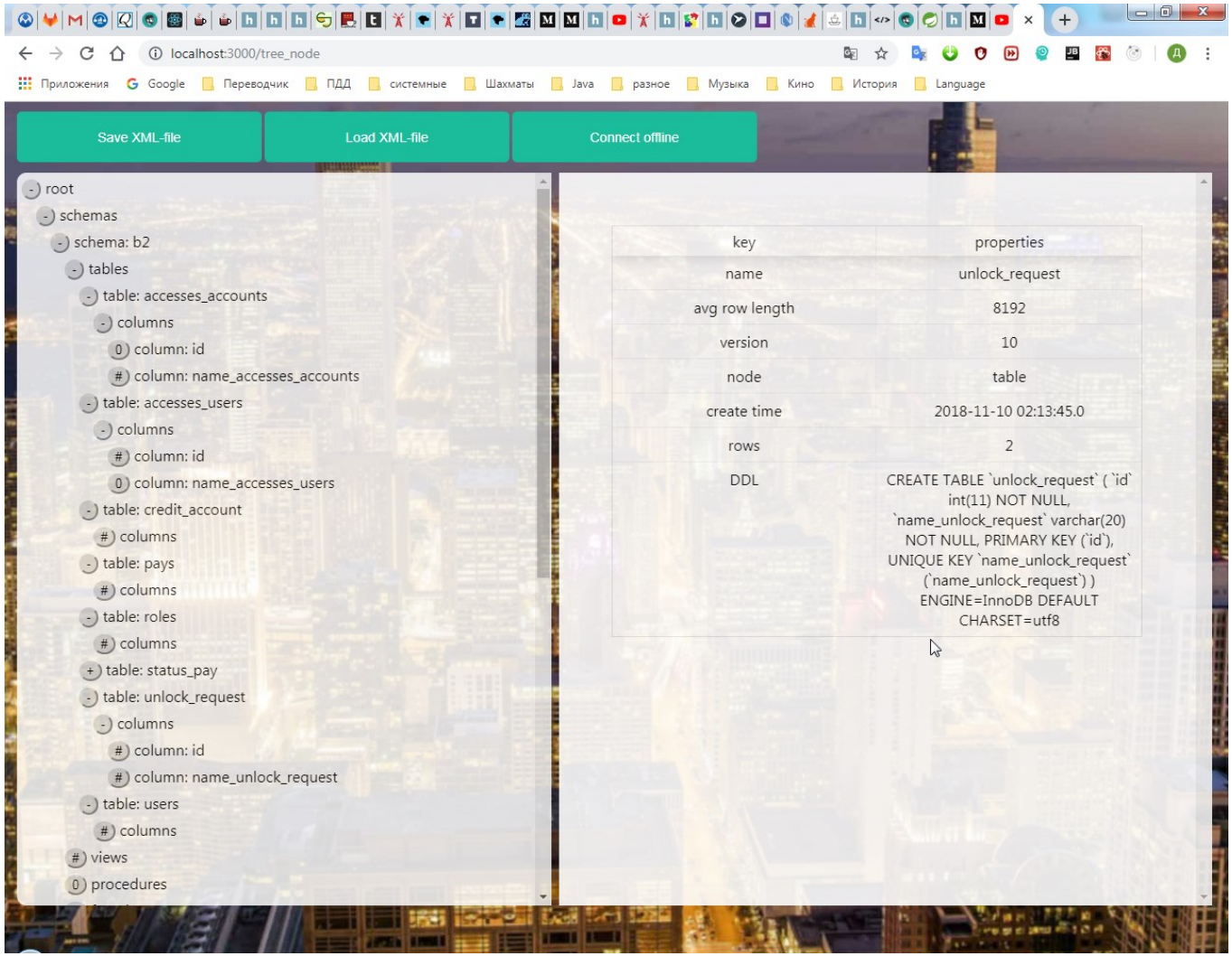

Reducers.

## Description of the application.



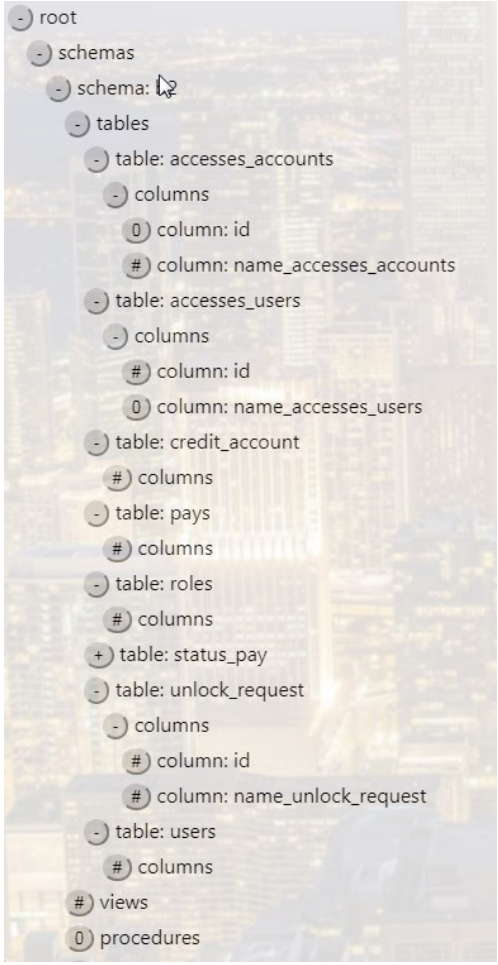

The application start page has input fields for choosing a database, user and password. If successful, the user is taken to the main page. (/tree_node).

# Description of the application.




On the main page on the left is the database tree, and on the right is a table with the characteristics of the node that the user clicked on.

## Description of the application.

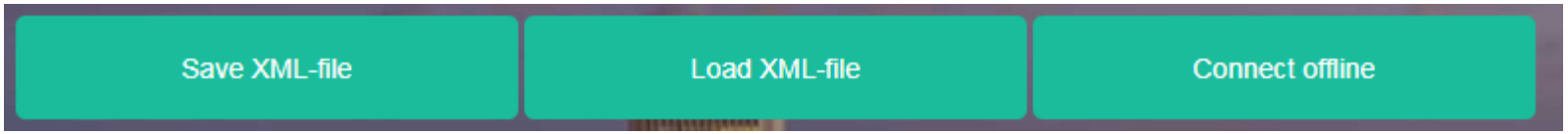



Buttons describing nodes have the following meanings:

“#” - the node is not loaded yet,
when you click on the “ # " button, you may see;
“-” - the node has children, and the node is deployed;
“+” - the node has children, but the node is not deployed;
“0” - the node has no children.

The page does not disconnect connect to the database and the loading of new nodes is carried out using the "lazy" download.

## Description of the application.



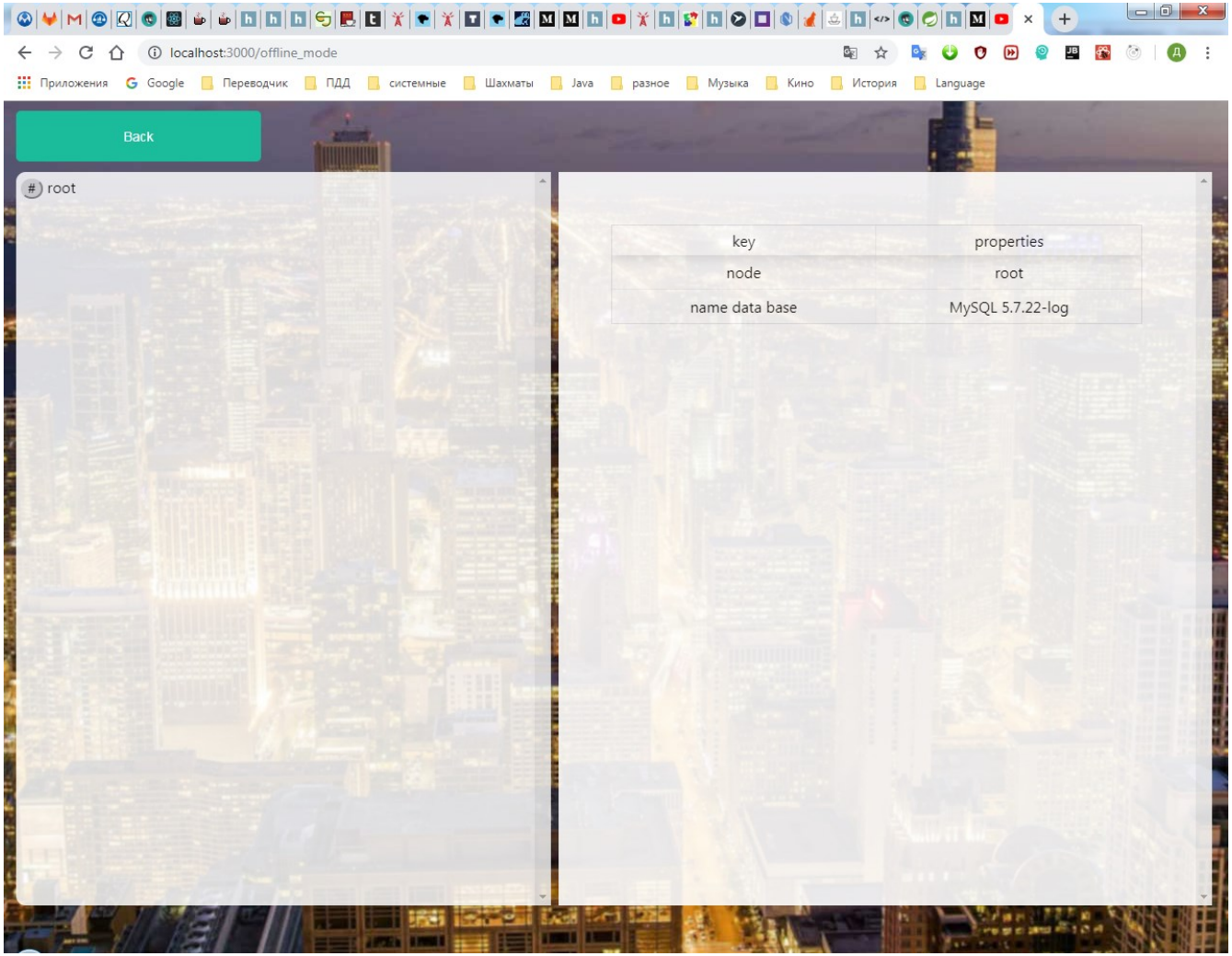

There are three buttons on the page.

"Save XML-file" - save the tree in XML, the current state of the tree is saved in the XML file on the server, if successfully saved, the file itself is displayed in text form on the right side of the table below.

"Load XML-file" - loading a tree from XML - the tree is downloaded from the XML file and the downloaded tree is reloaded in the left part. This mode remains with connect to the database enabled; when the tree is rebooted, you can update it through a lazy boot.

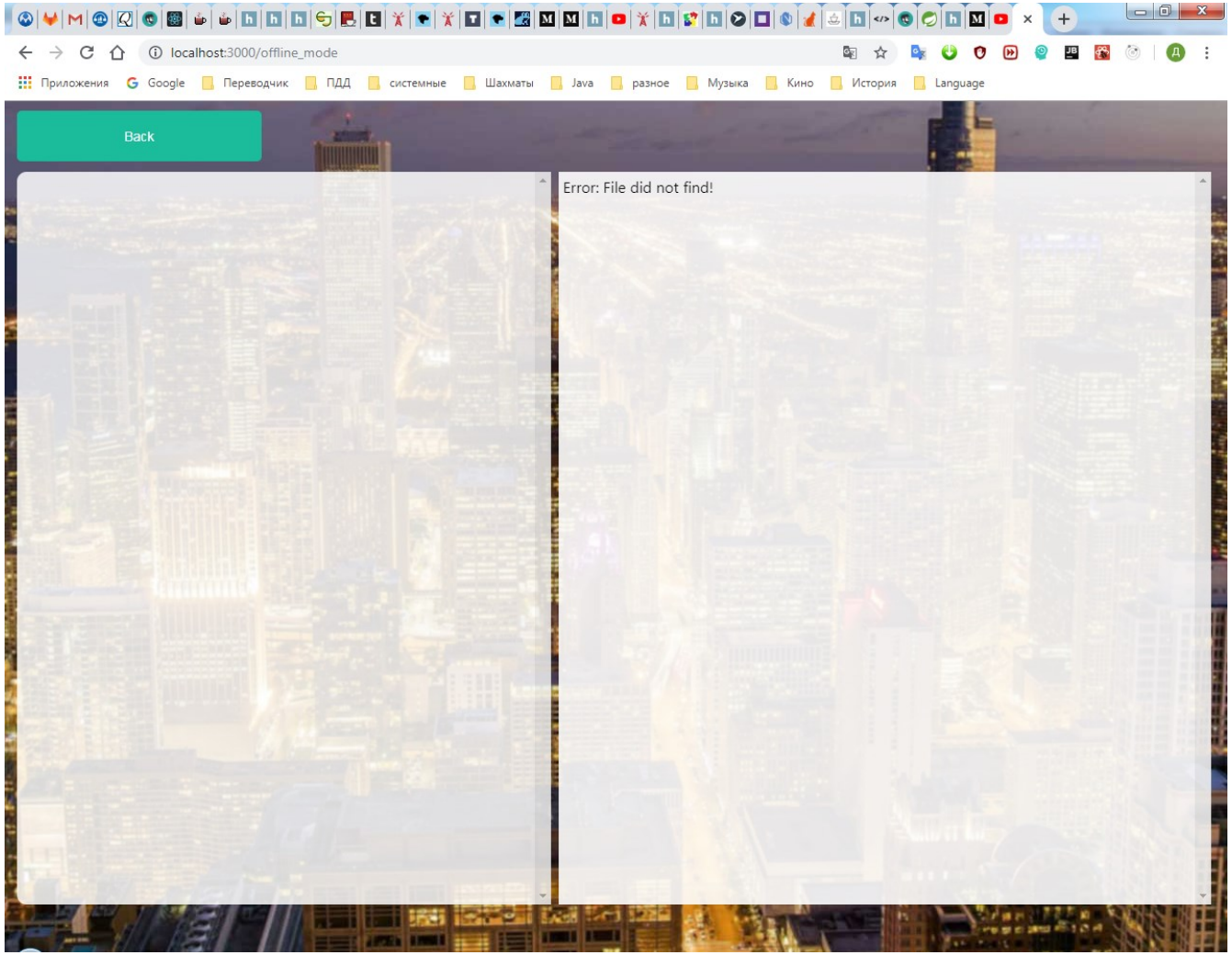
"Connect offline" - switch to offline mode. Switch to a new window with disconnected connect mode to the database. In this situation, the tree is loaded from the XML file without the possibility of loading child nodes from the database. If there is no saved file, the message "Error: File did not find!" Is displayed.

## Description of the application.




Offline mode. Return to the main mode is carried out by the “Back” button, while the original tree is restored.

# Description of the application.




The case when the XML file was not saved.