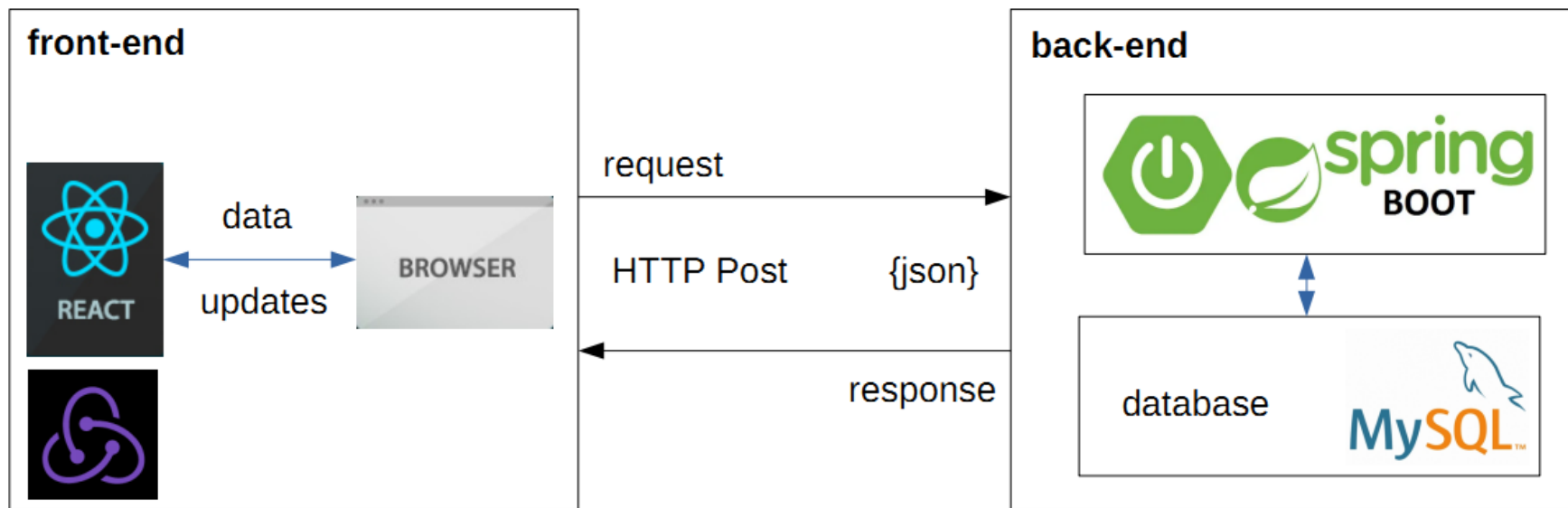


Студенческий проект:

Приложение для отображения базы данных MySQL в виде неупорядоченного дерева.

Структура приложения.



Приложение состоит из клиентской (front-end) и серверной (back-end) частей. Back-end выполнен на базе Spring Boot с использованием REST API на Java. Клиентская часть написана с использованием JS, React и Redux фреймверков.

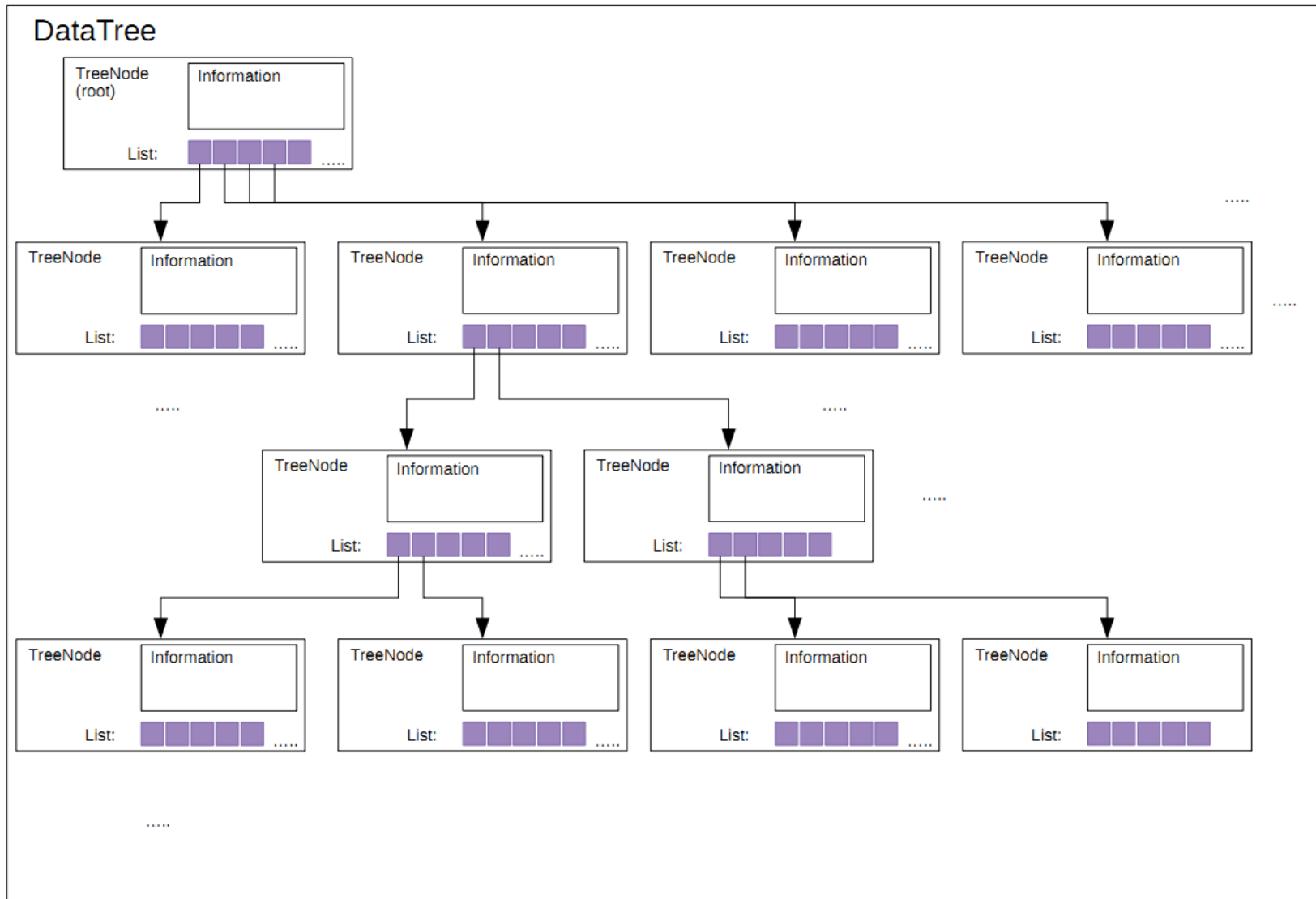
Back-end.

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

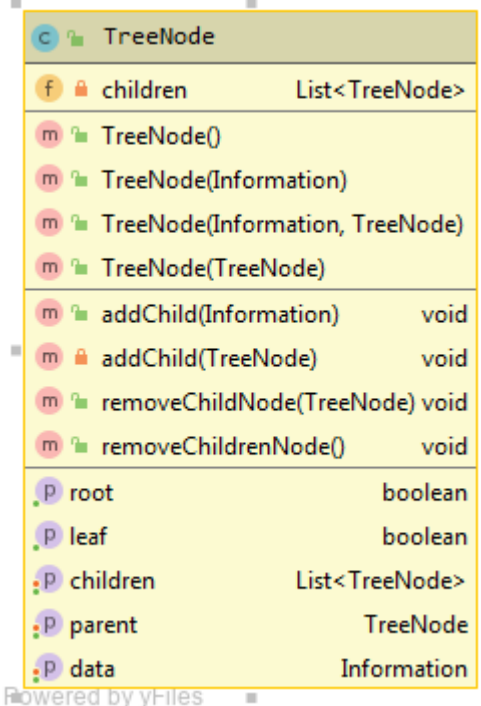
Powered by yFiles

В качестве представления модели базы данных используется структура данных в виде неупорядоченного дерева (рекурсивного дерева). В программе дерево реализуется при помощи класса DataTree.

Back-end. Схема представления дерева.



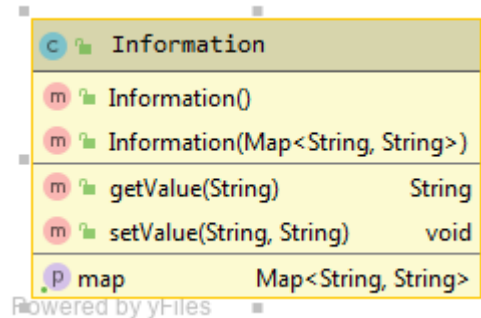
Back-end.



Само дерево (tree) состоит из узлов (node), которые представляют собой элементы* базы данных и являются объектами класса `TreeNode`.

*элементы – schemas, schema, tables, table, columns, column, triggers, trigger, functions, function, procedures, procedure, views, view и т. п.

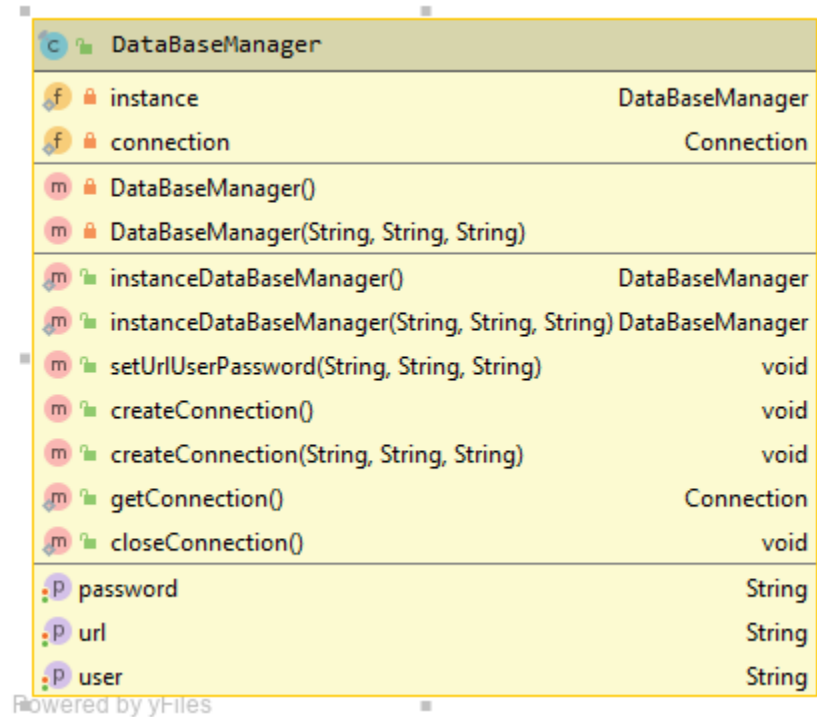
Back-end.



Элемент базы данных описывается специальным классом `Information` и агрегатно включается, как объект в `node` (ноде). `Information` реализуется при помощи `Map<String, String>`.

Кроме ключей описывающих элементы базы данных и значения для этих элементов, существует поля «`id`» которое необходимо для идентификации `node`-ы. По «`id`» ноды производятся поиски в ширину и глубину.

Back-end.



Для подключения и отключения БД в проекте используется класс `DataBaseManager`. (работа с классами `java.sql.DriverManager` и `java.sql.Connection`)

Back-end.

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

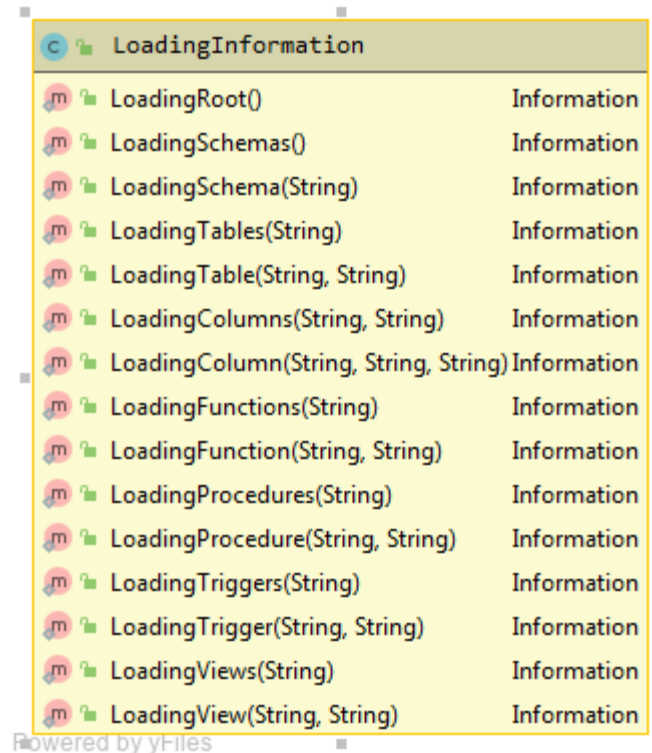
Powered by yFiles

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

Powered by yFiles

DataBaseMySQLQueries - это класс для организации запросов к базе данных и получения результата. Класс Requests — константные стринговые переменные запросы SQL.

Back-end.

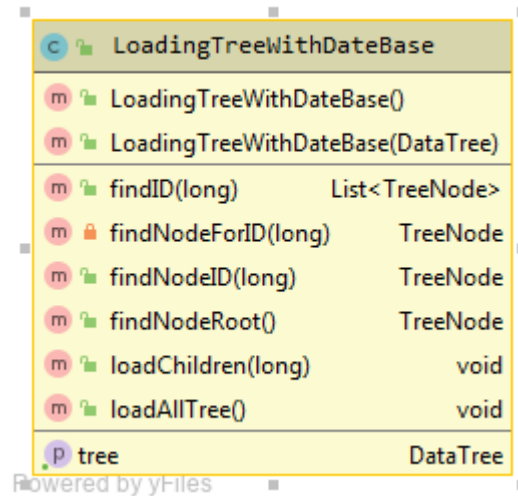


LoadingInformation	
LoadingRoot()	Information
LoadingSchemas()	Information
LoadingSchema(String)	Information
LoadingTables(String)	Information
LoadingTable(String, String)	Information
LoadingColumns(String, String)	Information
LoadingColumn(String, String, String)	Information
LoadingFunctions(String)	Information
LoadingFunction(String, String)	Information
LoadingProcedures(String)	Information
LoadingProcedure(String, String)	Information
LoadingTriggers(String)	Information
LoadingTrigger(String, String)	Information
LoadingViews(String)	Information
LoadingView(String, String)	Information

Powered by yFiles

Для заполнения информации об элементах, используется класс LoadingInformation, который включает в себя статические методы-loader-ы под каждый тип элемента базы данных.

Back-end.



Для формирования дерева из базы данных предназначен класс `LoadingTreeWithDateBase`, который позволяет искать ноду по id (реализован поиск в ширину и глубину), подгружать детей ноды (ленивая загрузка), подгрузить все уровни базы данных (формирование полного дерева базы данных), в классе используются методы класса `DataTree`.

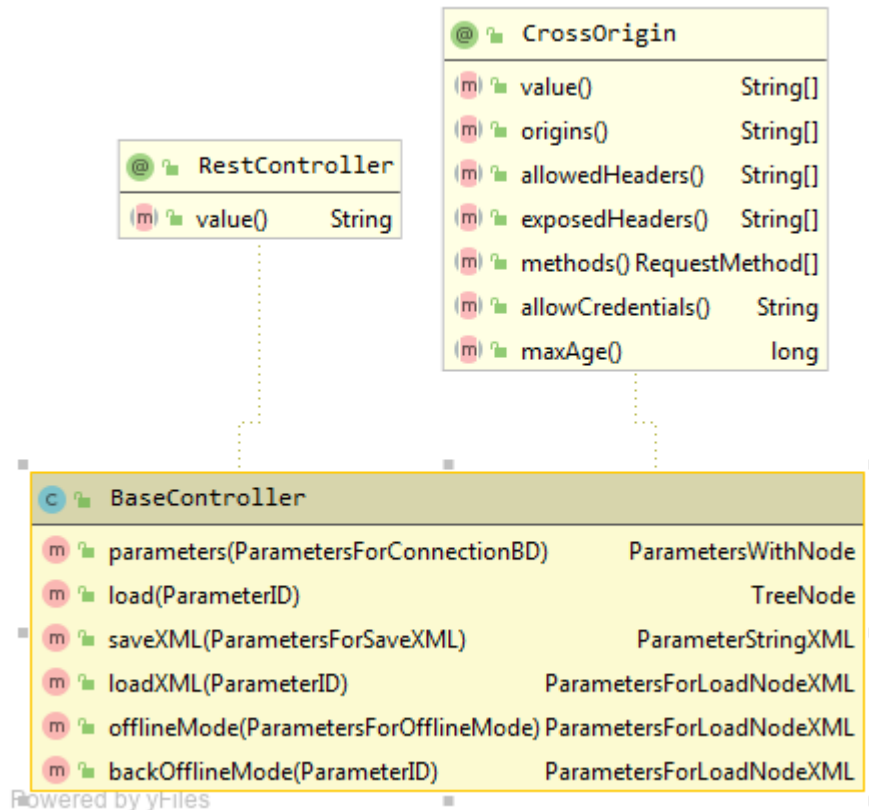
Back-end.

WriterXML		
f	tree	DataTree
f	document	Document
f	factory	DocumentBuilderFactory
f	builder	DocumentBuilder
f	transformerFactory	TransformerFactory
f	transformer	Transformer
f	domSource	DOMSource
m	WriterXML(File, DataTree)	
m	stepThrough(Element, TreeNode)	void
m	stepThrough(Element, TreeNode, List<TreeNode>)	void
m	parseNode(TreeNode, Element)	void
m	write()	void
m	writeXML()	void
p	file	File
p	string	String

ReaderXML		
f	document	Document
f	arrayDeque	ArrayDeque<TreeNode>
m	ReaderXML(File)	
m	ReaderXML()	
m	readXML()	void
m	readXML(File)	void
m	stepThrough(Node)	void
p	file	File
p	tree	DataTree

В серверной части программы реализована возможность сохранения дерева в XML файл и получения дерева из XML файла при помощи классов WriterXML и ReaderXML.

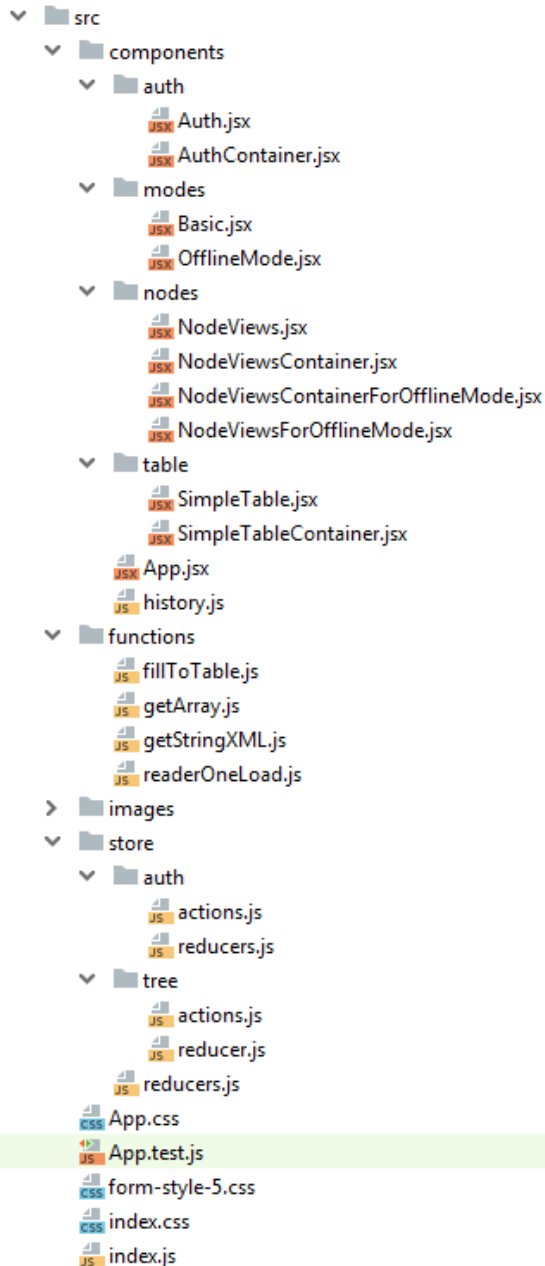
Back-end.



REST контроллер BaseController состоит из методов обрабатывающих запросы с клиентской части:

- получение логина, пароля и пути к БД;
- ленивая загрузка по «id»;
- сохранение файла в XML-формате и передача в виде строки на клиента;
- передача дерева из сохраненного XML-файла в основном режиме;
- переход в режим offline с передачей дерева из сохраненного XML-файла;
- возврат в основной режим.

Front-end.



Клиентская часть включает в себя:

- store, содержит состояния приложения, в нем хранятся различные объекты для работы с компонентами front-end-а. (дерево, параметры ноды для таблицы, массивы для отображения кнопок дерева, различные стринговые переменные). Единственный способ изменить состояние внутри него - отправить на него action используя dispatch-и.

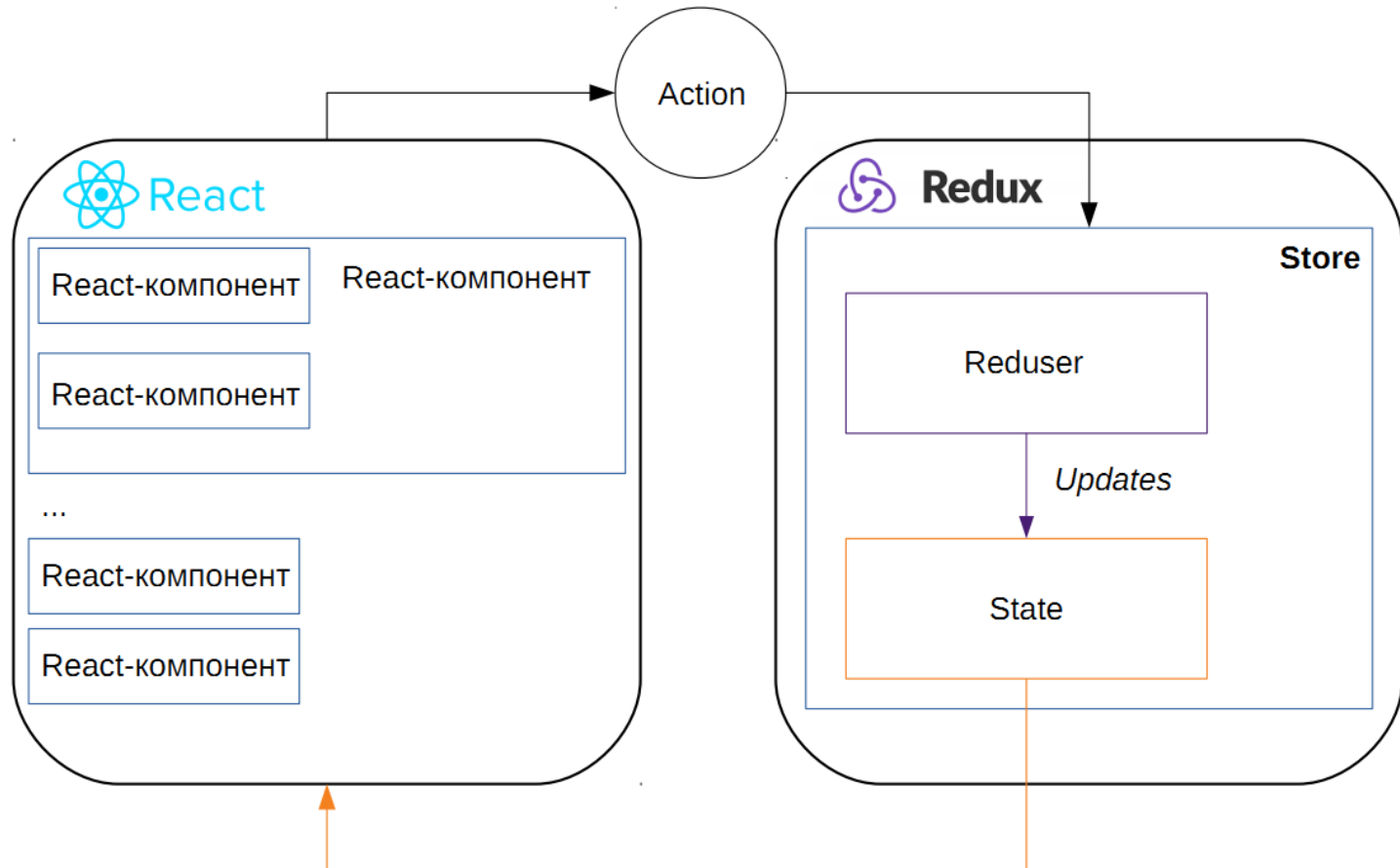
- react-компоненты, используются для создания элементов веб-страниц (таблицы, кнопки, обертки для компонент, умные компоненты - для связи со store).

- reducer-ы - функции, которые принимают на вход команды и изменяет state. В программе объединены при помощи combineReducers в общий reducer.

- dispatch-и - для передачи переменных в reducer-ы. dispatch(action), store.dispatch(action) - отправляют команды, и это единственный способ вызвать изменение состояния store. В программе использовались как dispatch-и с bindActionCreators* так и анонимные store.dispatch(action).

* bindActionCreators - используются когда передаются некоторые генераторы экшенов (action creators) вниз к компоненту, который ничего не знает о Redux и нет желание передавать компоненту dispatch или Redux-store. (bindActionCreators оборачивает каждый экшн в dispatch).

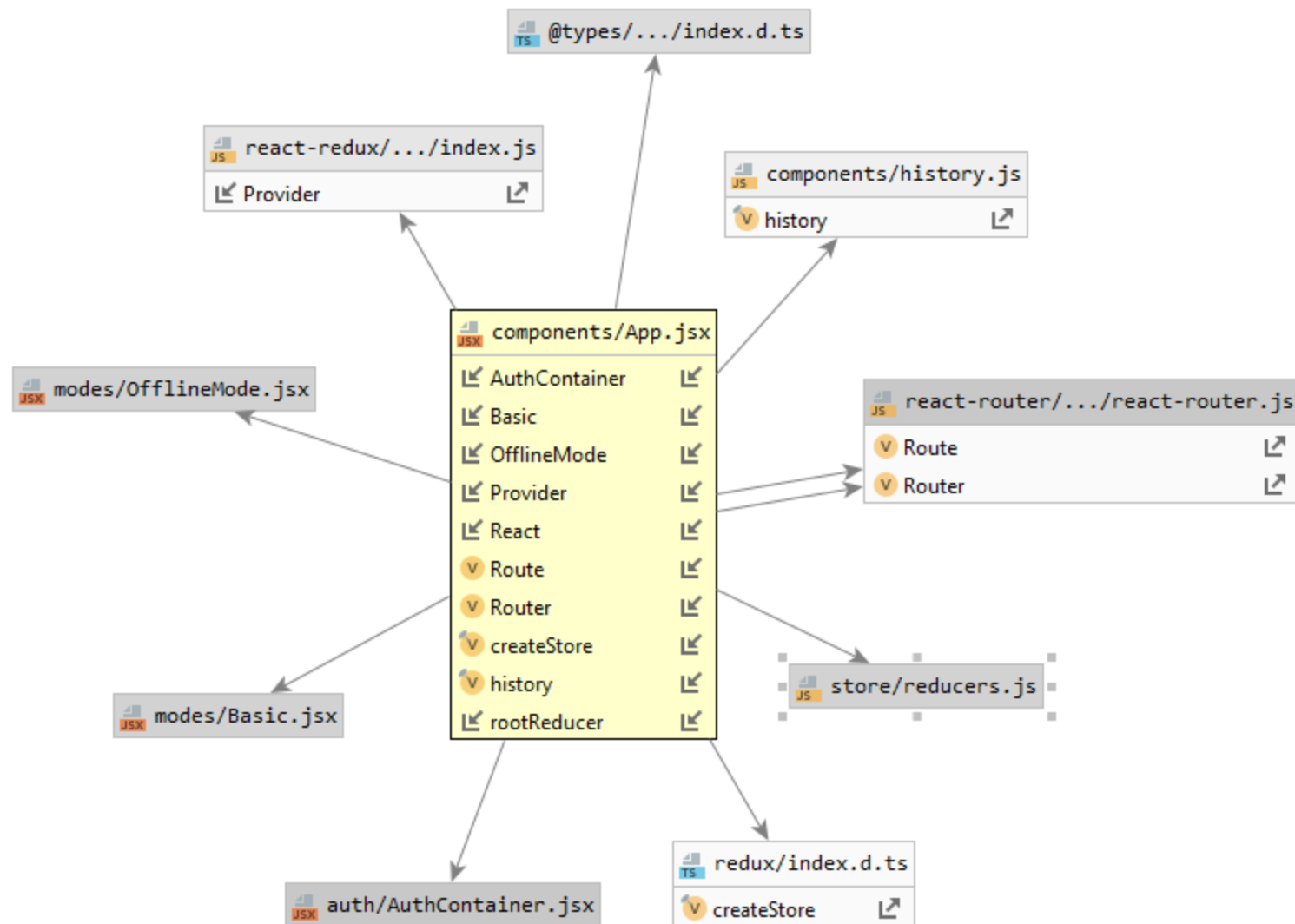
Front-end. Схема взаимодействия React-компонентов и Redux.



Redux решает проблему управления состоянием в приложении, храня глобальные данные в глобальном State и централизованно изменять ее.

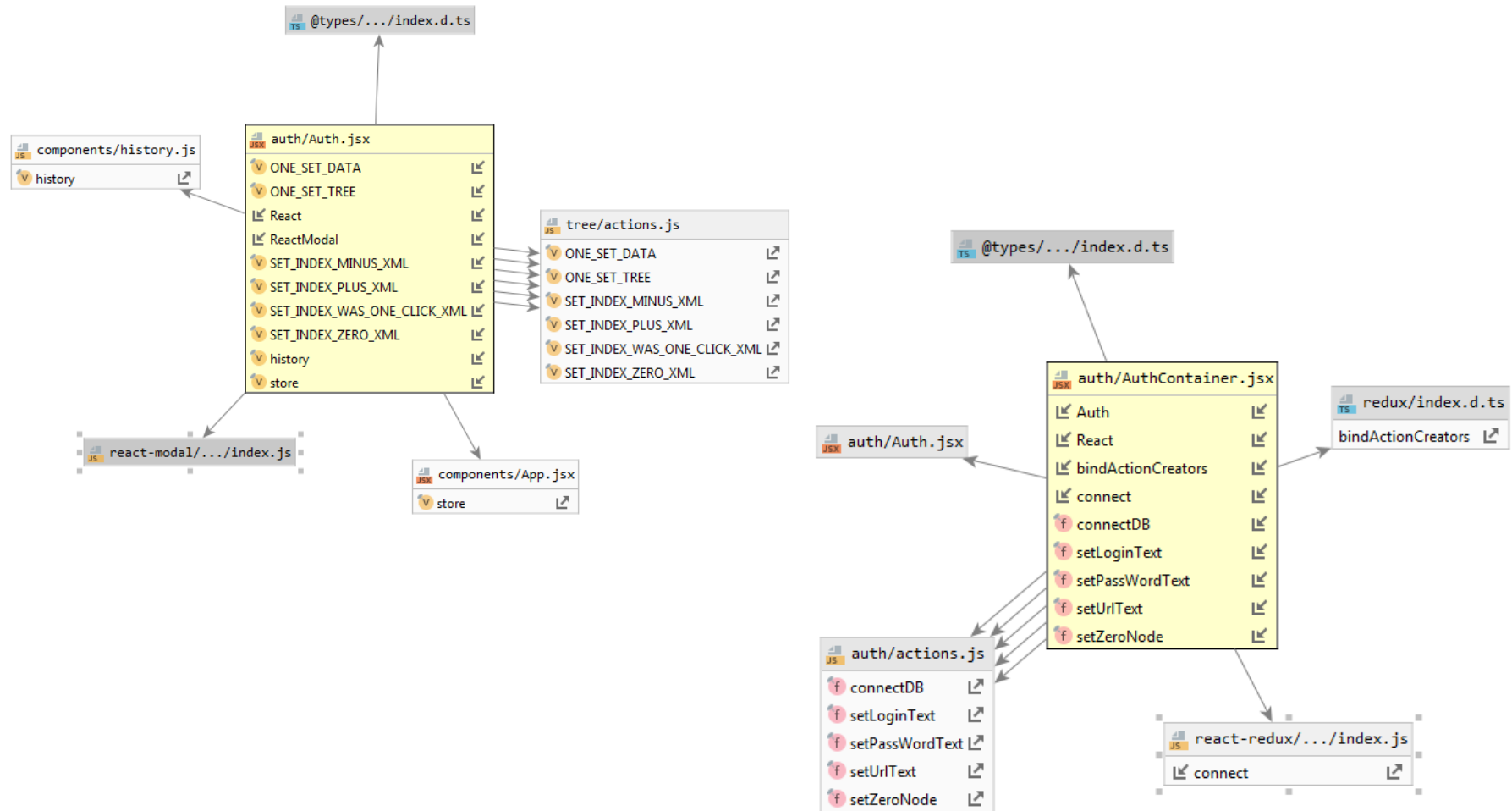
- Компоненты формируют события (actions).
- Reducer – модуль логики, который обрабатывает listeners и изменяет state.
- State общий для всех компонентов.
- Reducer + State = Store.
- Компоненты обновляются при изменении state.

Front-end.



App.jsx — внешняя обертка, корневой компонент.

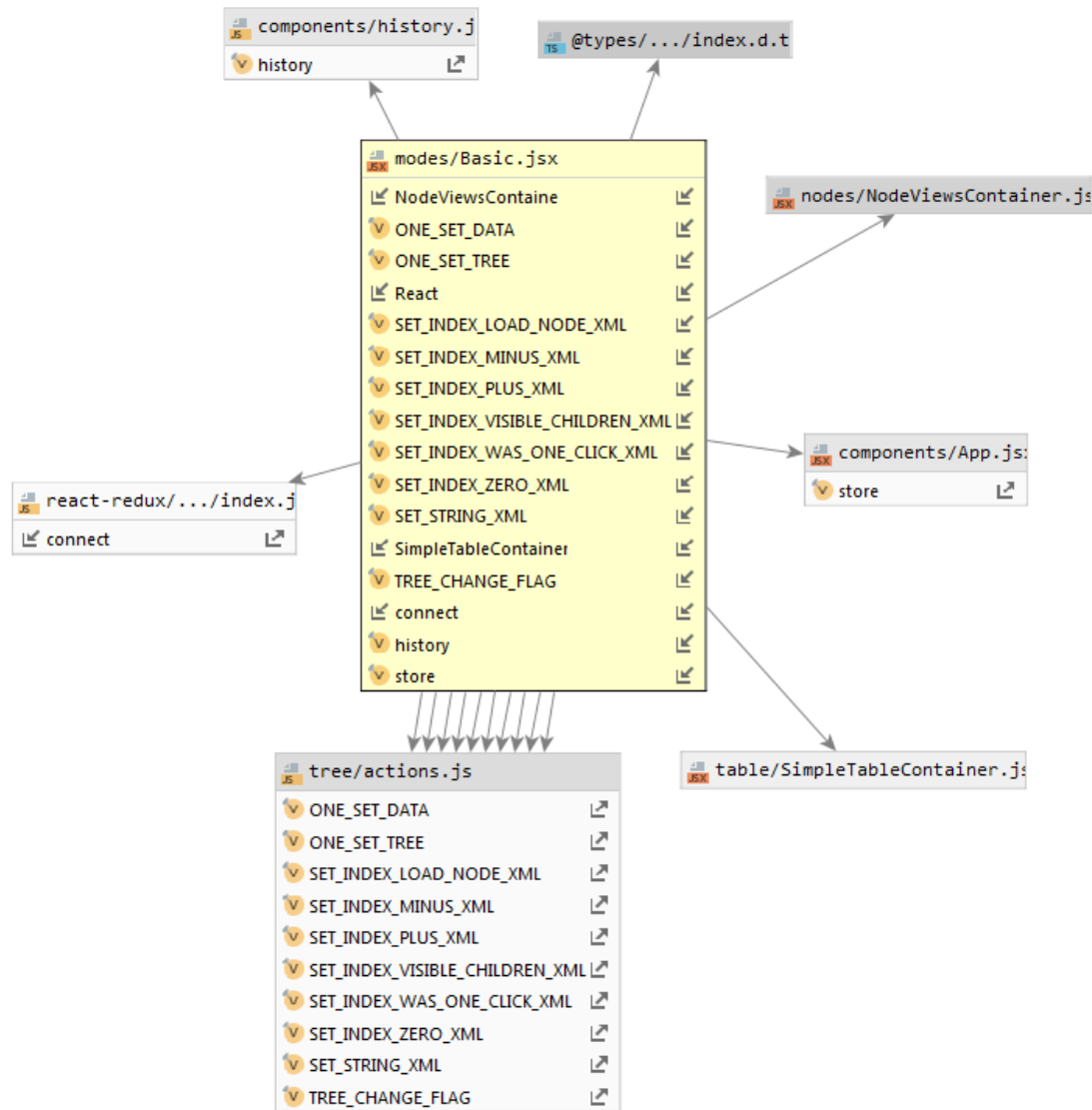
Front-end.



Auth.jsx и AuthContainer.jsx — компоненты для авторизации.

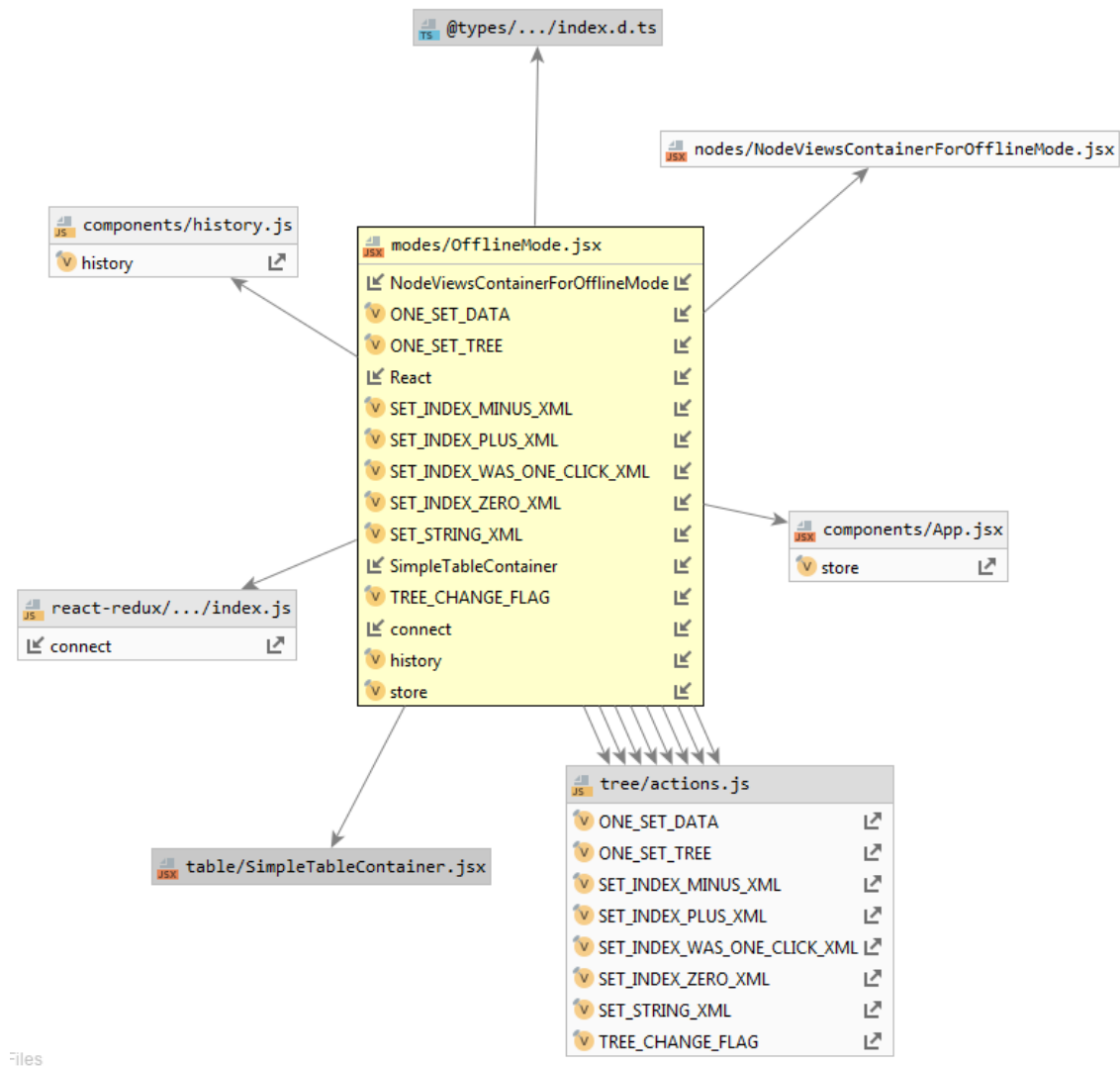
Примечание: для компонент ...Container.jsx определяем функции **mapStateToProps()** для чтения состояния и **mapDispatchToProps()** для передачи события. Генерируем компонент путем передачи созданных функций в **connect()** (connect – подключение компонента React к store Redux).

Front-end.



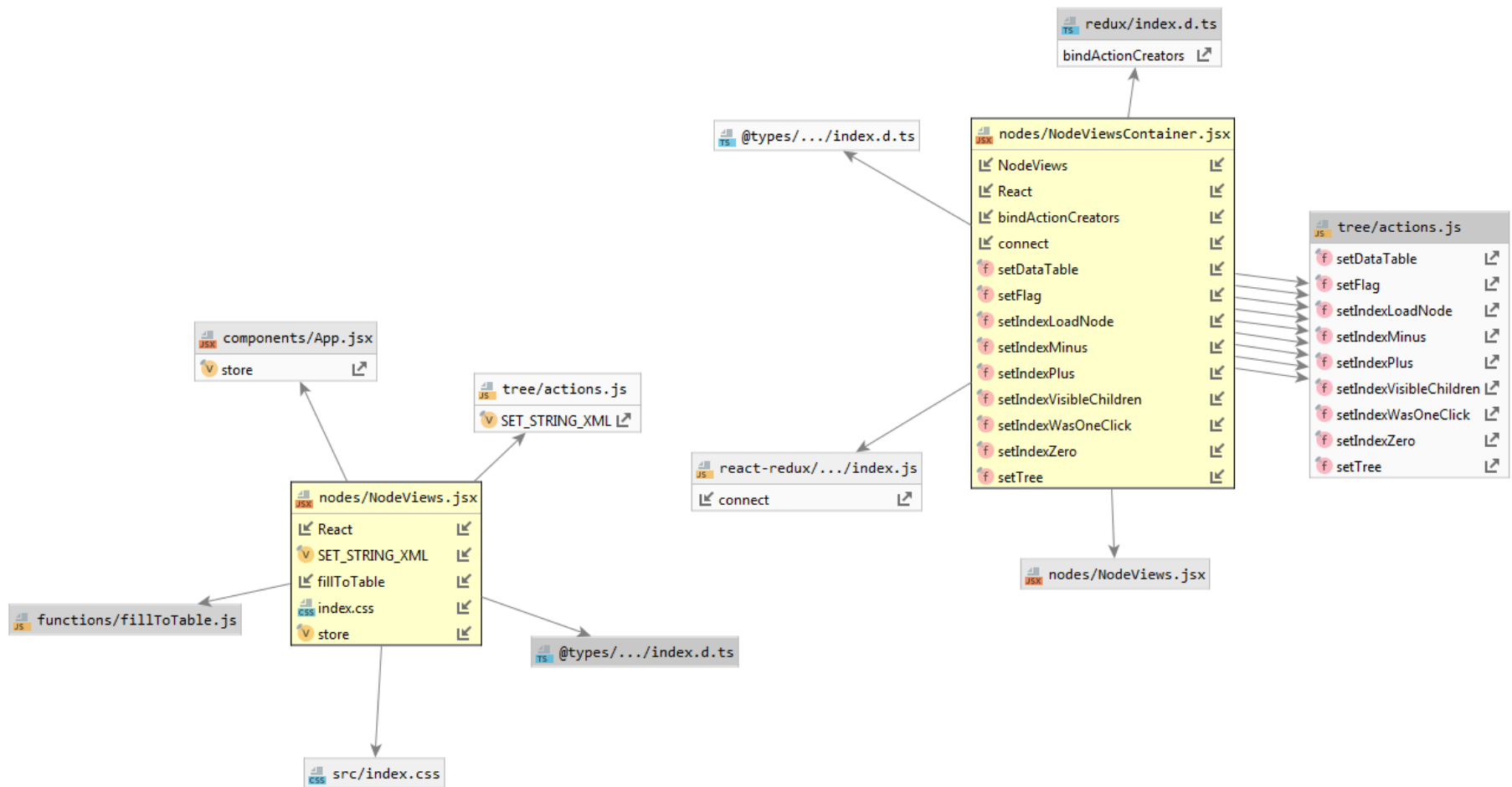
Basic.jsx — обертка, для основного режима.

Front-end.



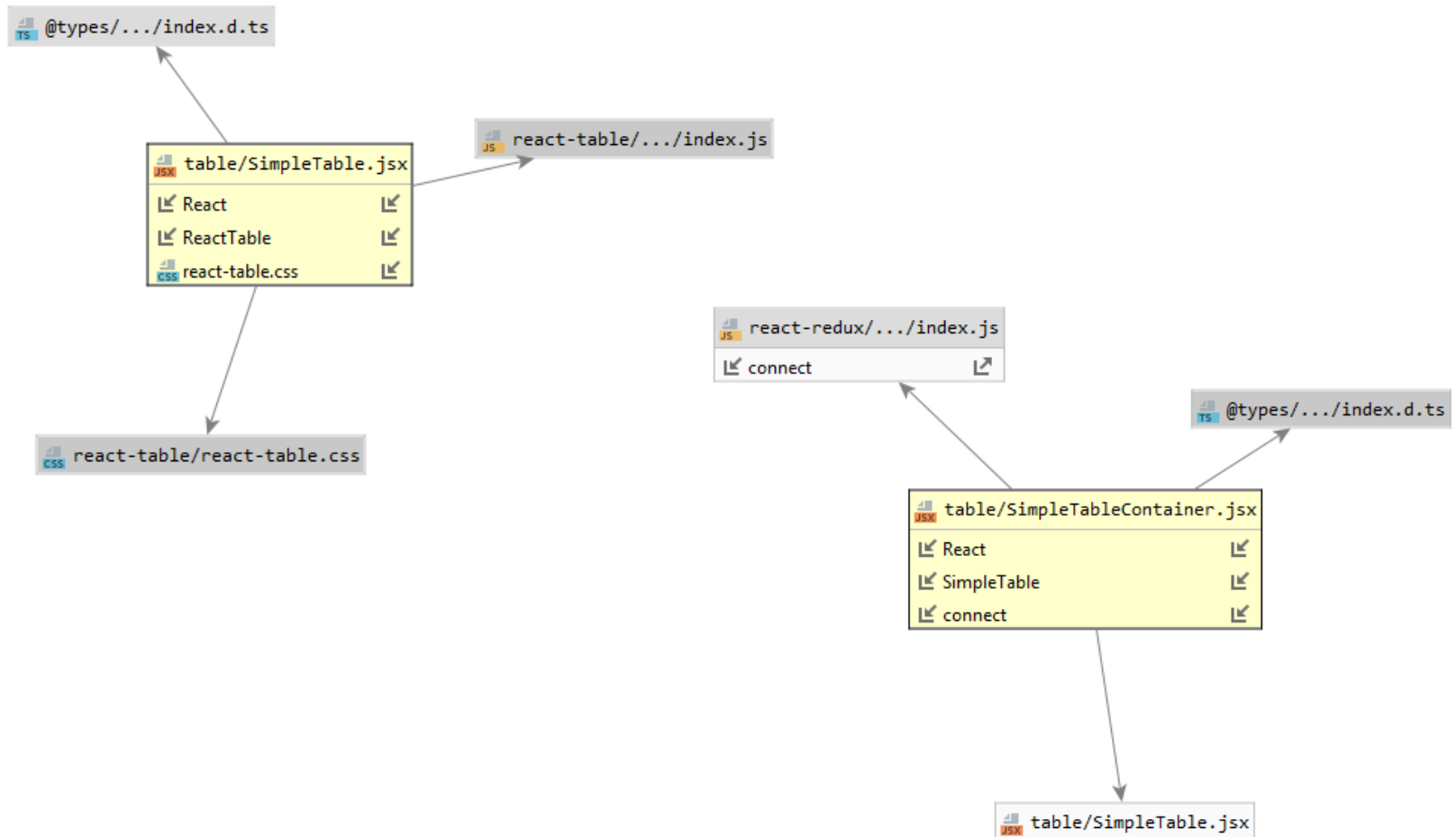
OfflineMode.jsx — обертка, для режима offline.

Front-end.



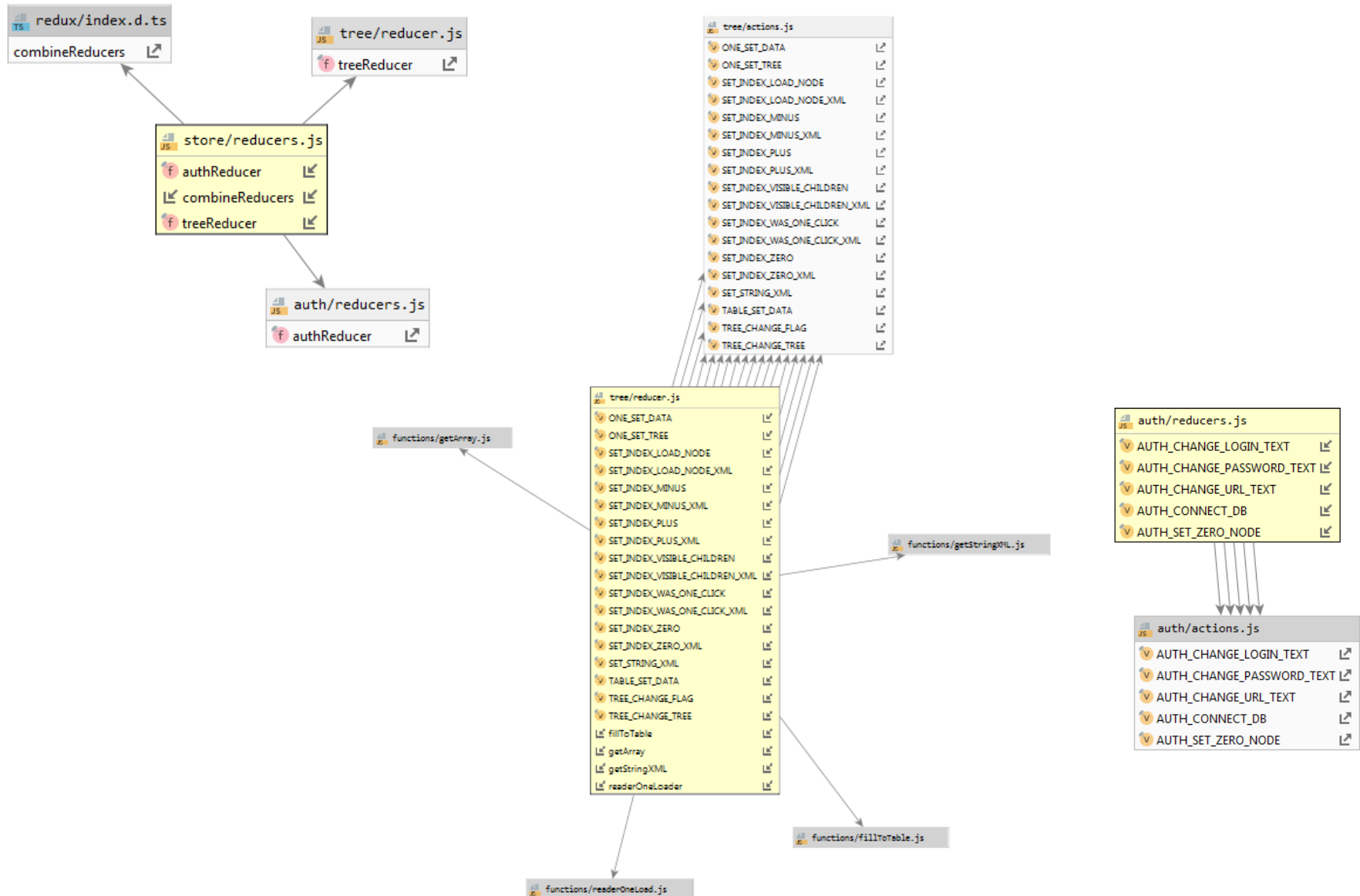
NodeViews.jsx и NodeViewsContainer.jsx — компоненты для отображения дерева.

Front-end.



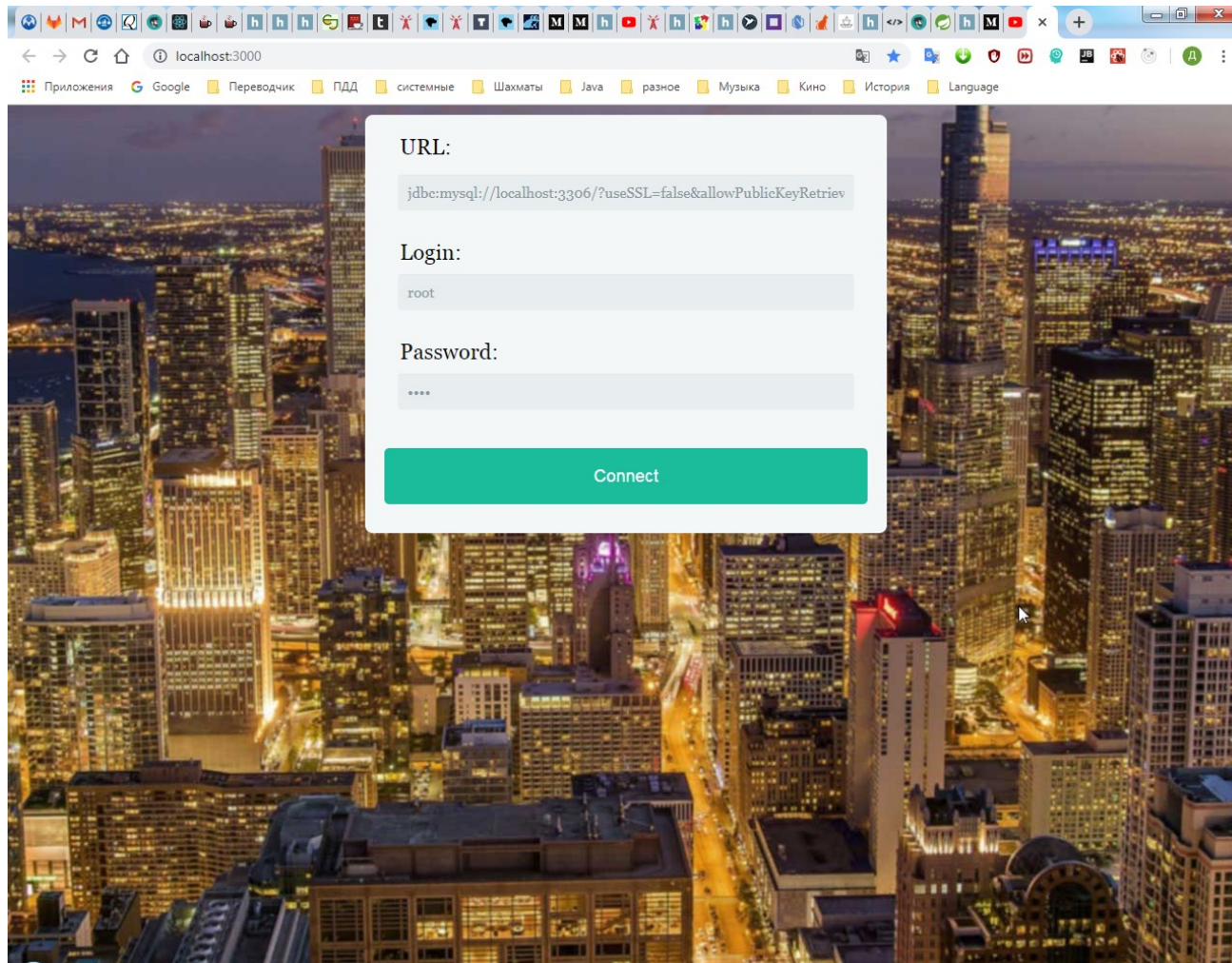
`SimpleTable.jsx` и `SimpleTableContainer.jsx` — компоненты для таблицы.

Front-end.



Редьюсеры.

Описание работы приложения.



Начальная страница приложения имеет поля ввода для выбора базы данных, пользователя и пароля. При правильном вводе пользователь попадает на основную страницу (/tree_node).

Описание работы приложения.

The screenshot shows a web application running in a browser at localhost:3000/tree_node. The interface has three buttons at the top: 'Save XML-file', 'Load XML-file', and 'Connect offline'. On the left, there is a tree view of a database structure. The tree is expanded to show the 'unlock_request' table under the 'b2' schema. The table has two columns: 'id' and 'name_unlock_request'. On the right, there is a table showing the characteristics of the selected node. The table has two columns: 'key' and 'properties'. The 'key' column lists various attributes, and the 'properties' column shows their values. The 'DDL' key shows the SQL statement for creating the table.

key	properties
name	unlock_request
avg row length	8192
version	10
node	table
create time	2018-11-10 02:13:45.0
rows	2
DDL	CREATE TABLE `unlock_request` (`id` int(11) NOT NULL, `name_unlock_request` varchar(20) NOT NULL, PRIMARY KEY (`id`), UNIQUE KEY `name_unlock_request` (`name_unlock_request`)) ENGINE=InnoDB DEFAULT CHARSET=utf8

На основной странице слева располагается дерево базы данных, а справа таблица с характеристиками ноды на которую нажал пользователь.

Описание работы приложения.

Кнопки описывающие ноды имеют следующие значения:

“#” - нода еще не прогружена;

при нажатии на ноду “#”, могут появиться

“-” - у ноды есть потомки(дети), и нода развернута

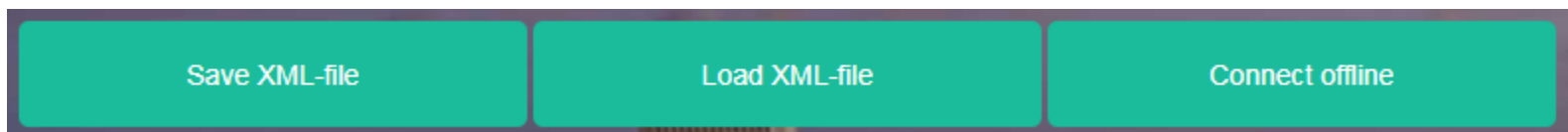
“+” - у ноды есть потомки(дети), но нода не развернута

“0” - у ноды детей нет.

На странице не отключается connect к базе данных и подгрузка новых нод осуществляется при помощи “ленивой” загрузки.



Описание работы приложения.



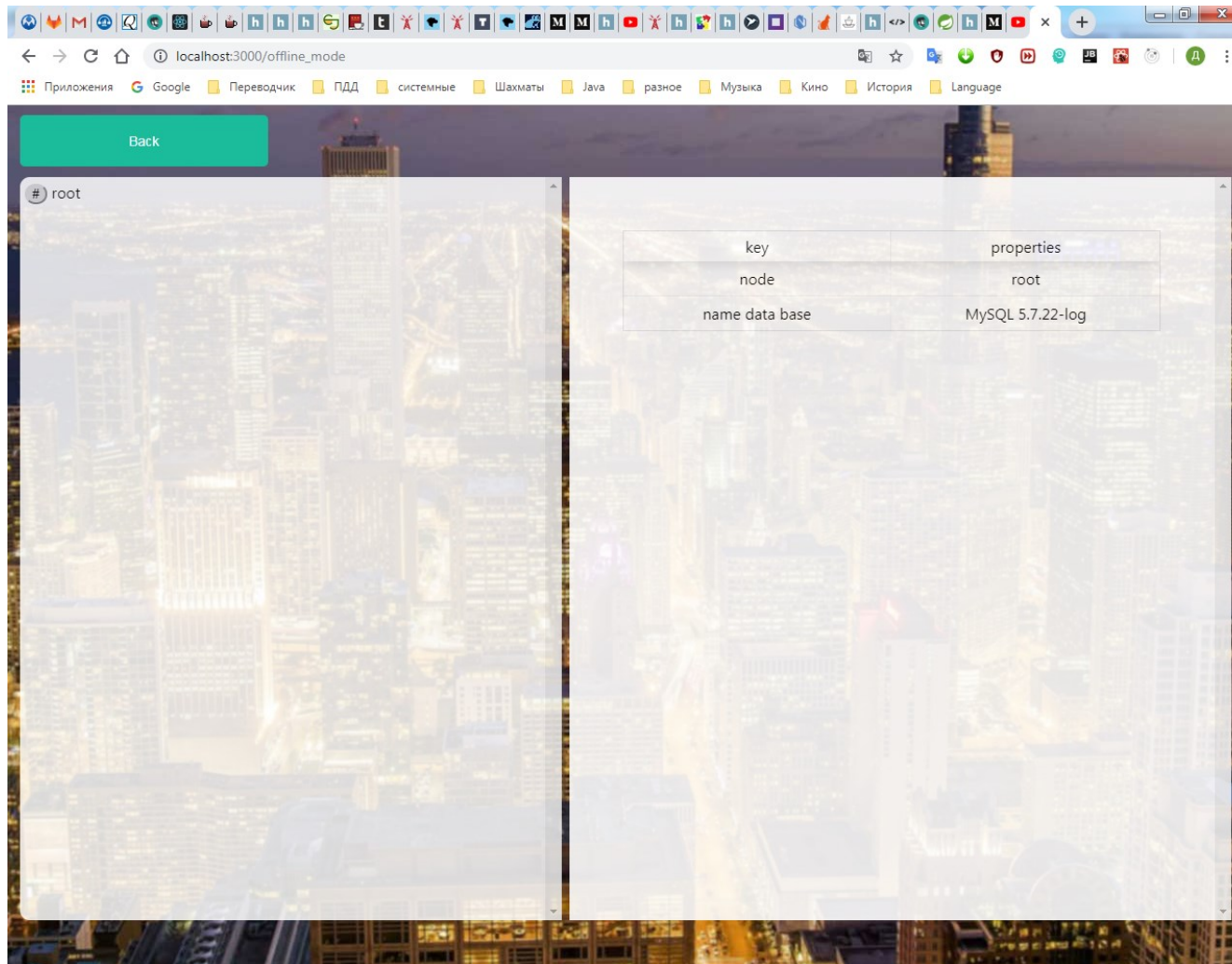
Также на странице имеются три кнопки.

Save XML-file - сохранения дерева в XML, текущее состояние дерева сохраняется в файле XML на сервере, при удачном сохранении в правой части ниже таблицы выводится сам файл в текстовом виде.

Load XML-file - загрузка дерева из XML - происходит скачивание дерева из файла XML и перезагрузка скачанного дерева в левой части. Данный режим остается с включенным connect к базе данных, при перезагрузки дерева можно обновлять его через “ленивую” загрузку.

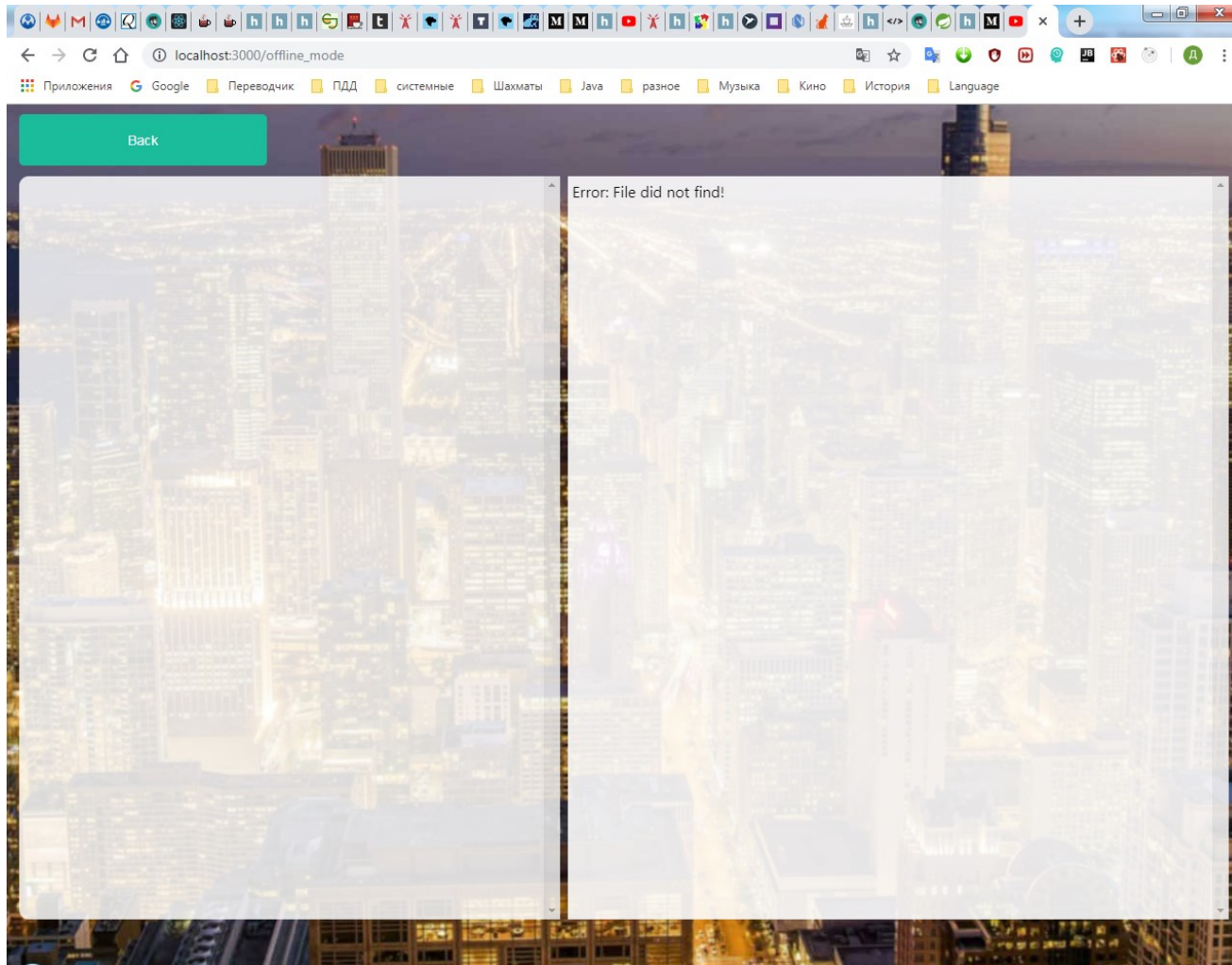
Connect offline - переход к режиму offline. Переход в новое окно с режимом отключенного connect к БД. В данной ситуации происходит прогрузка дерева из XML-файла без возможности догрузки детей-нод из базы данных. В случае если нет сохраненного файла выводится сообщение “Error: File did not find!”.

Описание работы приложения.



Режиму offline. Возврат в основной режим осуществляется кнопкой “Back”, при этом восстанавливается исходное дерево.

Описание работы приложения.



Случай, когда XML-файл, не был сохранен.