

Oracle Application Express (APEX)

A Time and Motion Analysis Developing Applications Using APEX versus Traditional Development Approaches

PIQUE SOLUTIONS

December 2020

THE DEVELOPMENT OF THIS WHITE PAPER WAS SPONSORED BY ORACLE. THE UNDERLYING RESEARCH AND ANALYSIS WERE EXECUTED INDEPENDENTLY BY PIQUE SOLUTIONS.

Contents

Executive Summary	1
Introduction	3
Approach and Methodology	6
Study Execution	12
Development in APEX	12
Interactive Grid on APEX.....	12
Faceted Search on APEX	12
WBS	13
Development in ReactJS	13
ReactJS Development Environment Setup	13
JavaScript Express MySQL API Development	14
Interactive Grid on ReactJS	14
Faceted Search on ReactJS.....	15
Study Results	17
Data Collection	17
Analysis.....	18
Research Questions	18
Statistical Analysis	18
Summary.....	20
Conclusions	21
Appendices	22

Oracle Application Express (APEX) is a registered trademark of Oracle and/or its affiliates.

Other names may be trademarks of their respective owners.

Pique Solutions is a competitive research and market analysis firm supporting Fortune-500 technology companies. Pique is based in San Francisco, California.

Executive Summary

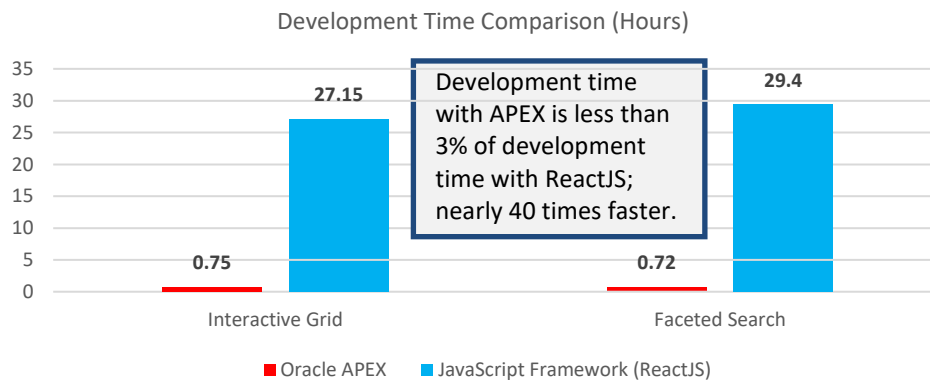
Pique Solutions conducted a study comparing a no-/low-code application development approach with a traditional JavaScript framework approach to seek the quantitative and qualitative differences in developing two real-world application modules—interactive grid and faceted search. Specifically, we compared development on Oracle Cloud Infrastructure using Oracle Application Express (APEX) with a traditional approach using a JavaScript framework (ReactJS) while leveraging existing libraries and writing code to produce comparable deliverables to the greatest extent possible.

Oracle APEX is a popular low-code development platform for rapidly building opportunistic and data-driven enterprise applications. APEX, combined with Oracle Database, provides a fully integrated environment to build, deploy, maintain, and monitor data-driven business applications that display well on mobile and desktop devices.

React is an open-source JavaScript library for building user interfaces or their components. It is maintained by Facebook by a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications.

Using a time and motion methodology, we defined and broke down the development activities for both environments in Oracle Cloud and tracked the time and the lines of code written for both application modules. The developer who designed and executed the study is a professional developer with 25 years of experience using a variety of development tools and languages but, by design, had no prior experience with APEX.

Figure 1. Development Time Comparison between APEX and JavaScript Framework



The key points and findings from this study, as summarized in [Figure 1](#), include the following:

- ⊕ APEX is more than 38 times faster than using a JavaScript framework approach based on the average of our two development projects: an interactive grid and a faceted search. This is largely due to the out-of-the-box capabilities and features that are packaged with the APEX framework versus what needs to be configured and developed by hand using ReactJS.
- ⊕ From a context perspective, the application delivery time for APEX, including setup and module development, took only 44 minutes versus 3.5 days using a JavaScript framework.
- ⊕ Twenty times less code was required, on average, in the APEX applications scenarios versus commensurate applications developed using a JavaScript framework.

- ⊕ In addition to the time and code benefits, APEX also was found to have numerous advanced enterprise capabilities that are taken care of in its platform, compared with what a JavaScript framework offers, such as security, concurrency control, reporting, save as capability, and very large dataset handling.
- ⊕ From a developer skillset perspective, we also found that more sophisticated developer skills are required to architect and develop a code-based solution compared to architecting and developing the same applications using APEX. The additional skills required for code-based development include proficiency in researching code libraries, front-end user interface design elements, manipulating data, and backend integration and APIs. Few developers possess all those skills.
- ⊕ From a business perspective, the results of this study demonstrate that companies using APEX can lower the cost of developing applications, get to market much faster, and leverage a wider pool of developer resources to implement new capabilities.

Introduction

Professional developers rely on the availability of libraries, frameworks, and platforms that help them bridge the gap between the customer's requirements and the implementation specifics. Decisions made by professional developers during conception, architecture, coding, and testing of an enterprise application can have profound impacts on time, resources, and associated cost—otherwise known as the iron triangle.

The time it takes to implement an enterprise application can be hampered or helped, in part, by the libraries, frameworks, and platforms that developers select to complete their solution. As software development has evolved over the past 25 years, it has created a dizzying spectrum of options spread across a variety of categories. Should a project be based on a platform that allows for less coding but has limited customization? Or should a project be designed to run on bare computing resources and allow the ultimate flexibility in implementation? These are just two of the thousands of decisions that face today's project managers, software architects, code developers, testers, and operators.

The complexity of building an enterprise application has many dimensions of considerations that overwhelm the process of prioritization and execution. Fortunately, the advances in areas such as open-source projects and platform richness are making bigger strides to remove some of the limitations and tradeoffs software professionals face. The decision to use no-/low-code versus traditional developer tools is becoming less complicated as a result.

The focus of this study was to delineate the differences between the no-/low-code data analysis tools offered by Oracle APEX and the ReactJS library used to build rich user experiences. At first glance this may not seem like a fair comparison—no-/low-code versus code. Naturally, the ratio of code and time spent in one is not equal to the other; however, the comparison is validated when one listens in on meetings with software professionals debating the merits of these two distinctive paths.

APEX

There are a variety of choices when evaluating database-native extraction and visualization toolsets. According to Oracle's website, APEX is "a low-code development platform that enables you to build scalable, secure enterprise apps, with world-class features, that can be deployed anywhere." Oracle claims developers can build an enterprise application significantly faster with far less code than they can with traditional development practices. The declarative APEX solution coupled with default security and inherent time savings appears to be aligned with the concerns of most product and project managers, and the resulting application should align with customer expectations. A professional developer is naturally tempted to evaluate the potential.

Creating an application is seemingly simple: Create an Oracle Cloud account and then an Oracle APEX account; import the data to be visualized, manipulated, and analyzed; select a template to view the data with; and voila. Without any prior knowledge of the APEX platform, nondevelopers can be up and running with a visualization of their data.

Once the initial APEX application is built, a developer can access the APEX Integrated Development Environment (IDE) to further add and customize the existing data views in a multitude of ways, from selecting colors and fonts to incorporating complex JavaScript functions capable of interacting with every element of the data manipulation and visualization. Additionally, Oracle has built APEX on top of Oracle Cloud, which provides integrated security. APEX is also supported natively on other cloud providers such as Amazon Web Services and can be set up on top of Oracle Database in on-premises environments. This means the solution is native to the cloud and can be ported to other cloud providers and to on-premises environments that support APEX-native solutions.

What about the limitations and tradeoffs that come with any platform? In this study, not many were discovered. Oracle's incorporation of an IDE allows for instant customization using native JavaScript with a rich set of objects to manipulate. Inclusion of default secure access is also provided, which eliminates the mandatory third-party security integration. Some elements of the experience could be improved and are noted later in this white paper.

ReactJS

Ever since JavaScript was released in its native format, open-source communities and private corporations have developed a comprehensive ecosystem capable of integrating disparate database technologies, message queueing, security providers, and every other public and private solution provider available. The introduction of rich libraries—such as Angular, ReactJS, and Vue, to name a few—has transformed the JavaScript language into an agile toolkit capable of achieving great results in a small timeframe. ReactJS is supported by an immense community of contributors, is optimized for performance, and reduces the need to manage bindings between objects by leveraging events and triggers to update virtual DOM elements.

As with any solution, ReactJS does have limitations. There is a large emphasis on user interface navigation. For most backend developers, this can be helpful because this area can be overlooked in the stack if they specialize in data and API access. ReactJS can quickly bring data and API access into a web application that can be further tailored to customer requirements. Once a professional learns how to leverage modules to blend web navigation with programming logic, the simplicity of ReactJS is realized.

Platform versus Native Development

From a project manager's perspective, the effect of reducing the skillset and time needed to deliver a solution to a customer is attractive. From a professional developer's perspective, the possibility of limitations and tradeoffs inherent with any no-/low-code platform could reduce efficiency due to unfamiliarity or a lack of integration with the other resources they intend to include to resolve technical gaps. The two mindsets are diametrically opposed from the start.

Oracle APEX and ReactJS are not necessarily equals in terms of the skillset that can use them. A professional developer, however, can learn APEX and then customize the default application with JavaScript to increase features and functionality. Furthermore, a professional developer can mirror the functionality of an APEX application and then continue to tailor the application. The point is that either entry point, ReactJS or APEX, can be used to build an enterprise-grade application. This white paper helps quantify the time and code savings if APEX is chosen to do the initial data ingestion and visualization bits as opposed to coding this from scratch.

Roles and Responsibilities

A variety of roles are involved when developing enterprise-class software solutions. This study focused on the perspective of the developer. Specifically, this white paper was designed based on the perspective of a professional developer. Although there are other varieties of developers in the software community, such as data scientists (using R to manipulate data sets for the purpose of analysis in the scientific research community) and citizen developers (Microsoft's Visual Basic used to code functionality into Excel spreadsheets), we focused on professional developers with a high degree of coding experience and a history of exposure to many different coding languages.

This white paper provides a quantitative analysis of Oracle APEX and ReactJS and includes a discussion of the approach and methodology used, the execution of the comparison, results, and a conclusion. The following section, on approach and methodology, includes detail regarding the problem statement, the purpose of the study, research questions, the nature of the study, the method, and assumptions. After the approach and methodology are discussed, the section on execution of the comparison includes a

narrative on the implementation of the example applications. The following four high-level applications were implemented:

- ⊕ An interactive grid example using Oracle APEX
- ⊕ An interactive grid example using ReactJS
- ⊕ A faceted search example using Oracle APEX
- ⊕ A faceted search example using ReactJS

The details of the execution of the comparison are discussed in the results of the study section. The last section of this white paper is a conclusion summarizing the results.

Approach and Methodology

Problem Statement

Organizations that rely on software to create solutions make significant investments in new technology. The right design, tools, and language are a few of the elements that can determine the profitability and sustainability of software solutions. Each selection may have a significant impact on the iron triangle: time, cost, and resources. Controlling the time, it takes to complete a task can have an impact on cost and resource allocation. Reducing the time for a task can also lead to market disruption by creating a new platform from which to build more complex and value-added services. The problem is that some professional developers do not have an empirical way to compare the time and amount of code required when using a platform versus a coding language to create software solutions.

Purpose of the Study

Claims have been made about the speed and efficiency of implementing certain features associated with one coding language versus another. Oracle maintains that its low-code platform APEX can enable faster development and deployment with far less hand-coding than traditional development methods require. If such a claim is true, APEX may improve efficiency of labor, which improves reliability and repeatability and, thus, may also improve project success. The purpose of this quantitative, time and motion study was to test and record the time taken to execute the steps necessary to implement two different reporting applications in two different coding environments—Oracle APEX and ReactJS.

Research Questions

Research Question 1: Is there a significant difference in the time required to build an interactive grid application using APEX as compared with ReactJS?

- ⊕ H_01 : There is no significant difference in the time required to build an interactive grid application using APEX as compared with ReactJS.
- ⊕ H_a1 : There is a significant difference in the time required to build an interactive grid application using APEX as compared with ReactJS.

Research Question 2: Is there a significant difference in the time required to build a faceted search application using APEX as compared with ReactJS?

- ⊕ H_02 : There is no significant difference in the time required to build a faceted search application using APEX as compared with ReactJS.
- ⊕ H_a2 : There is a significant difference in the time required to build a faceted search application using APEX as compared with ReactJS.

Nature of the Study

The nature of the study was time and motion, which was used to record the time required to complete each task in a unit of work. A unit of work was represented in this study by each of the hypotheses, each of which was broken down into subtasks using the work breakdown structure (WBS) technique. The data collected on the elapsed time for each subtask was then summed and used to compare the differences between the coding platforms.

Method

The study was designed and conducted to compare the relationship between developing applications in two different ways. The first method of developing applications was with a no-/low-code approach (APEX) from the perspective of a professional developer having no prior experience with APEX. The second method of developing applications was with a traditional development language (ReactJS) using libraries for common functions where feasible.

Both no-/low-code and traditional coding were analyzed for the time and lines of code necessary to be feature complete. Feature completeness was determined by identifying all features available in the Oracle APEX version of the example application to be written. For instance, to marshal all the features available in the interactive grid example, a sample interactive grid application was created in APEX and then analyzed for feature inclusion. After the feature set was recorded in a spreadsheet, the task of matching (parity) began by coding the equivalent using ReactJS in combination with MySQL (database) and Okta (access control).

The two application types, faceted search and interactive grid, were decomposed using the WBS technique. Each WBS was loaded into a spreadsheet that was used to record the time required to complete each task. Once the application was completed, the task breakdown structure was reviewed for completeness and augmented to add or remove tasks, as necessary. The goal was to be as close to parity between the two different coding environments as possible to give an accurate real-world depiction of the time associated with each. Some features were excluded for purposes of objectivity.

The following is an example of a WBS for Research Question 2 (faceted search):

Prerequisites

1. Obtain/create/identify data taxonomy.
2. Sign up for APEX service or create ReactJS project.

Initializing Actions

1. Create empty virtual database and table based on identified data taxonomy (APEX or local file).
2. Populate database table contents to database table.
3. Create function(s) to read contents of database table (without filters for facets/constraints).
4. Create base UI (blank web page).
5. Create base UI containers for:
 - a. Page title
 - b. Facet/constraint selection
 - c. Results listing
6. Create function(s) to list contents of database table in UI (results listing container):
 - a. Create sort function.
 - b. Create pagination of results.
7. Render list of facets: Add to facet/constraint selection UI container.
8. Render list of constraints for facets:
 - a. Identify constraint ranges for currency/price, integers representing quantity, etc.
 - b. Add selectable constraint to facet/constraint selection UI container.

Recurring Actions

1. Update results page with constrained data.
2. Update breadcrumb with selected constraints.

Interactive Grid

According to [Oracle](#), “An interactive grid presents users a set of data in a searchable, customizable report. In an editable interactive grid, users can also add to, modify, and refresh the data set directly on the page. Functionally, an interactive grid includes most customization capabilities available in interactive reports plus the ability to rearrange the report interactively using the mouse.”

Creating an interactive grid is essentially componentizing common spreadsheet functionality in a way that can be visualized and manipulated inside a webpage. Features to highlight rows, take action on specific cells, apply filters, and execute mathematical functions are all possible using an interactive grid application. The depth of features is only limited by one’s requirement specifications.

Figure 2. Oracle APEX Interactive Grid Example Application

id 1	Budget	Homepage	Id	Original L...	Original Ti...	Overview	Popularity	Release D...	Revenue	Runtime	Status	Tagline	Title	Vote
16	225000000		2454	en	The Chroni...	One year af...	53.978602	5/15/2008	419651413	150	Released	Hope has a ...	The Chroni...	
17	220000000	http://marv...	24428	en	The Aveng...	When an u...	144.448633	4/25/2012	1519557910	143	Released	Some asse...	The Avenge...	
18	380000000	http://disne...	1865	en	Pirates of T...	Captain Ja...	135.413856	5/14/2011	1045713802	136	Released	Live Foreve...	Pirates of T...	
19	225000000	http://www....	41154	en	Men in Blac...	Agents J (...	52.035179	5/23/2012	624026776	106	Released	They are ba...	Men in Blac...	
20	250000000	http://www....	122917	en	The Hobbit...	Immediatel...	120.965743	12/10/2014	956019788	144	Released	Witness the ...	The Hobbit...	
21	215000000	http://www....	1930	en	The Amazi...	Peter Parke...	89.866276	6/27/2012	752215857	136	Released	The untold ...	The Amazin...	
22	200000000	http://www....	20662	en	Robin Hood	When soldi...	37.668301	5/12/2010	310669540	140	Released	Rise and ris...	Robin Hood	
23	250000000	http://www....	57158	en	The Hobbit...	The Dwarv...	94.370564	12/11/2013	958400000	161	Released	Beyond dar...	The Hobbit...	
24	180000000	http://www....	2268	en	The Golden...	After overh...	42.990906	12/4/2007	372234864	113	Released	There are w...	The Golden...	

Faceted Search

According to [Oracle](#), “Faceted Search (or faceted navigation) is seen pretty often on the internet; typically on shop or sales web sites. The end user can set filters using *Facets* on the left or upper side of the screen. A facet shows possible values together with the occurrence count within the result set. After the end user changed a facet, results, dependent facets and occurrence counts refresh immediately.”

Faceted search is found commonly on e-commerce shopping sites. Amazon, Etsy, and Home Depot routinely use faceted search to help their customers efficiently narrow down selections for criteria such as manufacturer name, price, quality, and many other associated metadata attributes. Faceted search effectively combines multiple search criteria and quickly brings matching data into view.

Figure 3. Oracle APEX Faceted Search Example Application

Budget	Homepage	Original Language	Original Title	Overview	Popularity	Re C
237,000,000	http://www.avatarmovie.com/	en	Avatar	In the 22nd century, a paraplegic Marine is dispatched to the moon Pandora on a unique mission, but becomes torn between following orders and protecting an alien civilization.	150.44	12/11
300,000,000	http://disneygo.com/disneypictures/pirates/	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, has come back to life and is headed to the edge of the Earth with Will Turner and Elizabeth Swann. But nothing is quite as it seems.	139.08	5/16
245,000,000	http://www.sonypictures.com/movies/spectre/	en	Spectre	A cryptic message from Bond's past sends him on a trail to uncover a sinister organization. While M battles political forces to keep the secret service alive, Bond peels back the layers of deceit to reveal the terrible truth behind SPECTRE.	107.38	10/2
250,000,000	http://www.thedarkknighttrilogy.com/	en	The Dark Knight Rises	Following the death of District Attorney Harvey Dent, Batman assumes responsibility for Dent's crimes to protect the late attorney's reputation and is subsequently hunted by the Gotham City Police Department. Eight years later, Batman encounters the mysterious Selina Kyle and the villainous Bane, a new terrorist leader who overthrows Gotham's finest. The Dark Knight resurfaces to protect a city that has branded him an enemy.	112.31	7/16
260,000,000	http://movies.disney.com/john-carter	en	John Carter	John Carter is a war-weary, former military captain who's inexplicably transported to the mysterious and exotic planet of Barsoom (Mars) and reluctantly becomes embroiled in an epic conflict. It's a world on the brink of collapse, and Carter rediscovers his humanity when he realizes the survival of Barsoom and its people rests in his hands.	43.93	3/7

Dataset

This study relied on the existence of a dataset capable of exercising the majority of features and functions associated with an interactive grid and faceted search example application: The Movie Database (TMDb) as represented on [Kaggle](#). Columns included in the dataset are as follows:

- Budget
- Genre
- Homepage
- ID
- Keywords
- Original Language
- Original Title
- Overview
- Popularity
- Production Company
- Release Date
- Revenue
- Runtime
- Status
- Tagline
- Title
- Vote Average
- Vote Count

Some columns contained arrays of data such as Genre, Keywords, and Production Company. These fields were removed from the dataset, as they would add unnecessary complexity to the creation of the example applications on both platforms. Additionally, some records contained null values in certain cells and were excluded from the dataset to accommodate non-null constraints on the database used for the ReactJS example applications. It should be noted that the Oracle Autonomous Database used for this study was capable of ingesting null data cells without errors.

TMDb contained a good mixture of character and numeric data types. The character types ranged from 2 to 1,500 characters. The numeric types ranged from 0 to 2.8 billion. The Oracle Autonomous Database and the MySQL database used in this study were able to handle the character types and ranges associated with each of the non-null cells.

Database

Oracle Autonomous Database is used to power the Oracle APEX product. There is an option to use an external database connection, but that was outside the scope of this study. JavaScript is flexible and can use any database technology provided that either a library exists to connect to it, or an API exists that can be called using TCP or UDP protocols. In this study a MySQL database running inside a docker container, was used to house the dataset and expose it using the Node.JS web application framework, Express, with the MySQL middleware component library. A sample model was adapted to connect to the movie dataset schema (tmdb_5000_movies) defined in MySQL.

Figure 4. movie_model.js—MySQL Database Pool Definition and Query

```
const Pool = require('mysql');
const pool = Pool.createConnection({
  user: 'root',
  // host: 'localhost',
  host: 'mysql-service',
  database: 'movies',
  password: 'admin123',
  port: 3306,
});

const getMovies = () => {
  return new Promise(function(resolve, reject) {
    pool.query('SELECT * FROM tmdb_5000_movies ORDER BY id ASC', (error, results) => {
      // pool.query('SELECT * FROM movies ORDER BY id ASC', (error, results) => {
      if (error) {
        reject(error)
      }
      console.log(results);
      resolve(results);
    })
  })
};
```

Figure 5. index.js—Express Implementation Using MySQL Pooled Connection Component Model as Middleware

```
const express = require('express');
const app = express();
const port = 3001;

const movie_model = require('./movie_model');

app.use(express.json());
app.use(function (req, res, next) {
  res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3000');
  res.setHeader('Access-Control-Allow-Methods', 'GET,POST,PUT,DELETE,OPTIONS');
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Access-Control-Allow-Headers');
  next();
});

app.get('/', (req, res) => {
  movie_model.getMovies()
    .then(response => {
      res.status(200).send(response);
    })
    .catch(error => {
      res.status(500).send(error);
    });
});
```

Containers—Docker and Kubernetes

In keeping with the state-of-the-art development practices available, this study was conducted using Docker containers for all backend components (database: MySQL; database API: Express with MySQL middleware). Docker containers were hosted on [Oracle's Container Engine for Kubernetes](#). A local development environment was used to build the frontend component. The split between hosted off-premises and local on-premises components was used to reduce the deployment time associated with container-based development (e.g., building containers, pushing to a registry, restarting components). Deployment time was deemed as overhead that would skew the results in this study, so it was removed from consideration as a differentiating metric.

JavaScript Grid Library

It is common practice for professional developers to select open-source or licensed libraries for use in their applications. One of the common elements used in creating an interactive grid and faceted search application is the grid component. JavaScript has many grid components to choose from. For this study, and to be at parity with APEX, [ag-Grid](#) was chosen. ag-Grid was one of the top results in a Google search on “react interactive grid” (indication of popularity). Further investigation reflected, based on the thorough documentation available, a rich set of features needed to duplicate all the identified features in the APEX interactive grid and faceted search example applications.

JavaScript Security Library

Oracle APEX includes a secure access feature by default. To replicate this feature in ReactJS, and be at parity with APEX, it was necessary to integrate with a third-party security provider. For the purpose of this study, [Okta](#) was chosen because it is quick to obtain an account, configure, and integrate the necessary components to implement security access for a JavaScript application.

Assumptions

The developer will become more proficient in the use of each coding platform over multiple executions of the tasks performed; therefore, it is assumed that the time needed to learn and become familiar with the nuances of either platform will become less of a factor in the time difference associated with coding the application identified in the research questions. It is also assumed that a JavaScript grid and a third-party security provider's common libraries are available that can be used for both interactive grid and faceted search tasks. Without the use of common libraries, it would be necessary to build them from scratch, but that would not reflect a real-world experience and therefore was not part of this study. It is also assumed that a suitable dataset is available that will exercise the features and functionality of both interactive grid and faceted search tasks.

Data Analysis Plan

Based on the requisite feature set, Pique Solutions tracked the time taken to set up and configure the environment, develop the core modules, and perform any additional customization required. For each identified subtask, the time taken to complete the activity was recorded, as well as the number of lines of code written. Upon completion of each example application, the information on time and lines of code was summed and compared. For example, the interactive grid application was constructed declaratively in Oracle APEX and written programmatically in ReactJS. Once completed, the time and lines of code recorded for each instance of the interactive grid example application were summed. In the case of APEX, because it required no coding, a feature to change the genre of specific rows of data was added to establish a code baseline and some additional time to implement.

Study Execution

For the execution phase of this study, the approach taken was to create the APEX version of one of the examples and then create the equivalent version in ReactJS. The first example to be built in APEX was interactive grid. This was an arbitrary decision and is assumed to have no bearing on the time collection metrics for any of the research questions posed.

Development in APEX

Interactive Grid on APEX

Creating the interactive grid example application in APEX required some setup subtasks such as creating an Oracle Cloud account, creating an APEX account, launching the APEX service, naming a workspace, adding credentials to the newly created workspace, and logging out of the administrative account. The work of creating the interactive grid APEX application could then begin by logging into the newly created workspace, creating an application from file, importing the movies database file, naming the application (IG), and clicking create. This set of subtasks created a base APEX application with the movie database as its data source. The last step in creating the interactive grid application on APEX was to add a report page of type 'interactive grid.'

Viewing the interactive grid application was as easy as selecting the IG page and clicking the Play button in the IDE. After providing the credentials that were established during the setup of the APEX application, the interactive grid application was displayed (see Appendix G).

A small amount of code was added to the APEX interactive grid application to change the Genre of a movie from English to French and back again. The code was 93 lines in length and took 10 minutes to implement. A [detailed explanation](#) of the code is found on the Oracle Apex blog. The equivalent function was implemented in the ReactJS interactive grid example application; however, it took 7 lines of code and 10 minutes to implement. A more detailed explanation of the differences discovered is found in "Results," later in this white paper.

Overall, the process was straightforward and required no coding. It should be noted that logging into Oracle Cloud and Oracle APEX required two different sets of credentials and became confusing at times during the execution of this study. A federated login would be a better solution to avoid confusion. It was also difficult to navigate to a direct link for APEX after subsequently logging into Oracle Cloud. An article about APEX was listed on the homepage, but there was no link to the APEX service. An initial attempt to use the service, without any previous knowledge, was challenging, as it was not clear that the CSV movie metadata file could be used as the basis for an APEX application, instead of building an APEX application shell and then importing a dataset. Only after viewing an Oracle-supplied video tutorial did the Create Application from File option lead to completing the task of building the interactive grid application.

Faceted Search on APEX

Creating the faceted search example application in APEX also required setup subtasks such as creating an Oracle Cloud account, creating an APEX account, launching the APEX service, naming a workspace, adding credentials to the newly created workspace, and logging out of the administrative account. The work of creating the faceted search APEX application could then begin by logging into the newly created workspace, creating an application from file, importing the movies database file, naming the application (FS), and clicking Create. This set of subtasks created a base APEX application with the movie database as its data source. The last step in creating the faceted search application on APEX was to add a report page of type 'faceted search.'

Viewing the faceted search application was just as easy. Simply selecting the FS page and clicking the Play button in the IDE brought up the faceted search page login screen. After being prompted for the credentials established during the setup of the APEX application, the faceted search application was displayed (see Appendix H).

WBS

With the two APEX example applications created, the feature set for each could be collected. Many of the elements in the APEX examples were common, such as filtering, grid layout, menu items, and sorting. The interactive grid example included a multitude of features that allowed for easy viewing and manipulation of data, as well as the ability to chart, save, and export data for external use. Saving a report was a particular highlight, as it allowed for collaboration in a multiuser access scenario. This was not implemented in the ReactJS version of interactive grid because the significant work required was not specific to the interactive grid and was part of a more comprehensive collaboration environment not in scope for this study.

The WBS for an interactive grid informed the subtask execution steps (Appendix D) for the ReactJS-equivalent interactive grid application, and the WBS for faceted search informed the subtask execution steps (Appendix E) for the ReactJS-equivalent faceted search application.

Development in ReactJS

ReactJS Development Environment Setup

Building a state-of-the-art ReactJS application does not require cloud deployment; however, many new development projects in the open-source community and enterprise realm do use Docker containers and Kubernetes clusters. For the purpose of this study, we configured the MySQL database container (version 5.7) from Docker Hub's official Docker images collection. We also built a JavaScript Express application (MySQL API) in a Docker container. Both containers were deployed on the OKE environment. Connecting to the MySQL database, containing the TMDb movies metadata, required accessing the JavaScript Express application on port 3001 using JavaScript's `fetch` function with the post, get, update, and delete methods.

Deploying Docker containers on a Kubernetes cluster required creating an Oracle Cloud account and allocating a Kubernetes cluster in OKE. After the Kubernetes cluster was deployed, we configured our local Kubernetes environment to use the OKE cluster. A few location and account-specific details were marshaled from the OKE and Oracle Cloud administration pages to complete this task.

We pushed the containers to a remote registry so that the Kubernetes definition files could correctly reference the MySQL and MySQL API images. We used the Oracle Cloud Infrastructure Registry (OCIR) for our remote registry. Just as the OKE cluster required location and account-specific details, so did the OCIR connection. A few test pushes of the MySQL container indicated the remote registry was configured and operating properly.

We created config, secrets, storage, deployment, and service definitions for both MySQL and MySQL API containers. After MySQL was deployed, we port-forwarded the service to the local development environment, connected the MySQL client, created an empty database, and imported the TMDb movies metadata into a schema. Some records caused errors because they contained null values in required fields. Those records were removed (about 20), and the data import completed successfully.

The environment setup process required a total of 246 minutes with 365 lines of configuration to complete. Some of the major time usage came from the following: waiting for the Oracle Kubernetes cluster allocation (40 minutes), configuring the MySQL Docker container with an empty database (20 minutes) and importing clean data (30 minutes), learning how to configure access to the OCIR (20 minutes), and creation of the Docker and Kubernetes definition files for MySQL (22 minutes), the MySQL API (26 minutes), and the IG or FS example application (40 minutes).

JavaScript Express MySQL API Development

Building the interactive grid example application using ReactJS was achieved in 1,373 minutes (22.88 hours) with 1,592 lines of code. We created the database model using the JavaScript Express application (MySQL API) to properly access the creation, reading, updating, and deletion (CRUD) of records in the MySQL database. We leveraged a sample model used to access PostgreSQL as a template and adapted it to use MySQL. The process of developing the MySQL API container, deployment, and validating the model took 210 minutes and 136 lines of code.

Interactive Grid on ReactJS

Development of the interactive grid example application using ReactJS included a series of steps within a few high-level coding categories, as follows: creating the ReactJS project, adding and configuring the ag-Grid component, populating the ag-Grid with MySQL data, styling the application, creating menus for feature access, adding dialogs for interactive features, and adding code to support features. Each of the coding categories are detailed in this section.

Creating the ReactJS project involved the use of the [create-react-app](#) (Figure 6) environment. The environment creation process builds a folder structure, installs all base libraries via package.json (npm), and builds base ReactJS component files (index.js, App.js, App.css, and test-specific files). After 3 minutes, we had a working ReactJS application shell that we would extend to build our interactive grid example application.

Figure 6. Using create-react-app to Create a Base ReactJS Environment

```
npx create-react-app my-app
cd my-app
npm start
```

Next, we researched, added, and configured our base grid component. For this study, we chose ag-Grid because of its popularity and thorough documentation of features required to be at parity with the Oracle APEX interactive grid application. We installed (npm), included, and implemented the ag-Grid library in our App.js file in 70 minutes using 115 lines of code. Restarting the application (npm start) showed an empty grid ready for data and configuration. The ag-Grid configuration code section was revisited several times throughout the project development phase to modify and extend its capabilities. Additional time was added to this category of subtasks as the study progressed and refactoring was necessary.

Once the empty ag-Grid was implemented in code, we started the process of ingesting data from our populated MySQL database. A client-side import function was created based on examples found on the ag-Grid documentation site. The creation and validation of the data ingestion took about 15 minutes and 5 lines of code to complete.

The user interface was considerably basic at this point and needed some care and feeding to be equivalent to the APEX interactive grid example application we built. Styling the application took a total of 105 minutes and 532 lines of code. Of course, this was an iterative process relying on the native Safari HTML inspection tools. It was necessary to revisit this category at various points throughout the development process. We did our best to capture these refactor events to keep the time recording accurate.

We created application, grid, header, and row menus based on examples in the ag-Grid component documentation (Appendix K). We mimicked the APEX interactive grid naming convention of each menu item and subitem. The process of developing the functionality for each of the items was completed in subsequent steps.

Some modal dialogs were needed for contextual help (Appendix M), login (Appendix L), and fill. ReactJS makes the process of adding dialogs simple. We were able to reuse the help text from the APEX interactive grid example application as a placeholder because the time it would take to build a help dialogue was not in scope for this study.

Finally, the ag-Grid component was styled and decorated with menu items and could be wired with the features needed to compare equally with the APEX interactive grid example application we built. We started with search, sort, hide, format, copy-to-clipboard, and moving columns. Many of these features required code with accompanying ag-Grid configuration changes. More features were added such as stretching/shrinking columns, data aggregation functions, single-row view, deleting and duplicating rows, clearing and filling cells, and refreshing data and rows. The complete list of features implemented was captured in the React JS Interactive Grid Example Application Task Tracking Worksheet (Appendix D).

Every feature found in the APEX interactive grid example application was matched except for saving reports and multiple highlight policies. The ability to save reports would have required a multiuser collaboration experience that was beyond the scope of this study. Multiple highlight policies should have been achievable, but an unexpected effect of implementing the highlight feature was interference with other features. We suspect the dynamic CSS changes negatively impacted the ag-Grid component styling and, thus, we did not complete the multiple highlight policy development.

The final touch on the ReactJS interactive grid example application was the inclusion of the Genre change buttons that were also included in the APEX interactive grid example application. The two buttons toggled between English and French for selected records in the ag-Grid component. In ReactJS, this code took 10 minutes to implement and only 7 lines of code as compared to 93 lines of code in APEX.

The major difference in lines of code between ReactJS and APEX for nondefault features can possibly be attributed to coding style and decisions made on API access at early stages of development. It is unknown, however, if more coding is necessary in APEX to achieve equivalent results from ReactJS for additional features. Future studies could be developed to investigate research questions surrounding additional feature coding requirements.

Faceted Search on ReactJS

Development of the faceted search example application using ReactJS included a series of steps within a few high-level coding categories, which were as follows: creating the ReactJS project, adding and configuring the ag-Grid component, populating the ag-Grid with MySQL data, styling the application, adding functional code, and adding code to support features. Each of the coding categories is detailed in this section.

Creating the ReactJS project involved the use of the create-react-app (Figure 6) environment. The environment creation process builds a folder structure, installs all base libraries via package.json (npm), and builds base ReactJS component files (index.js, App.js, App.css, and test-specific files). After 3 minutes, we had a working ReactJS application shell that we would extend to build our faceted search example application.

We researched, added, and configured our base grid component. We chose ag-Grid because of its popularity and thorough documentation of features required to be at parity with the APEX interactive grid application. We installed (npm), included, and implemented the ag-Grid library in our App.js file in 70 minutes using 115 lines of code. Restarting the application (npm start) showed an empty grid ready for data and configuration. The ag-Grid configuration code section was revisited several times throughout the project development phase to modify and extend its capabilities. Additional time was added to this category of subtasks as the study progressed and refactoring was necessary.

With a working, but empty, ag-Grid implementation, we started the process of ingesting data from our populated MySQL database. We created a client-side import function based on examples found on the ag-

Grid documentation site. The creation and validation of the data ingestion took about 15 minutes and 5 lines of code to complete.

Next, we needed to style the ReactJS faceted search user interface to match the APEX faceted search example application we built. Styling this application took more than double the interactive grid time with a total of 235 minutes and 539 lines of code. The increase in time for styling was due to the dropdown, hide, and show functionality required by the faceted search features. It was necessary to revisit this category at various points throughout the development process.

Functional code was a major focus for the ReactJS faceted search example application. The functions required to duplicate the APEX faceted search example application were as follows: map-reduce records for collecting unique items (60 minutes and 100 lines of code) and item counts (60 minutes and 11 additional lines of code), distribution of numeric elements (60 minutes and 10 lines of code), filters for column in arrays (30 minutes and 32 lines of code), and a generic filter feature (10 minutes and 20 lines of code). Functional code was exposed to the user in the form of features.

Features for the ReactJS faceted search example application included the following: rendering the unique items and item counts (140 minutes and 85 lines of code), creation of 'show more' and 'show less' based on 5 or fewer items (105 minutes and 53 lines of code), naming and assigning of HTML facet items dynamically (30 minutes and 4 lines of code), applying single text filter for columns from array (30 minutes and 4 lines of code), applying multiple text filters for columns from array (90 minutes and 6 lines of code), clearing filters for columns (75 minutes and 32 lines of code), applying single numerical filter for columns from array (75 minutes and 43 lines of code), formatting for currency and commas (30 minutes and 46 lines of code), and implementing the recursive filtering logic required after an initial filter is selected (90 minutes and 20 lines of code).

It should be noted that we believed faceted search would require less time and code than interactive grid. Many of the functions and features needed to implement faceted search required research and several iterations to achieve. Familiarity with faceted search algorithms may have decreased the research or iteration time; therefore, the resulting time for other developers may vary.

Study Results

The first part of this section includes information on data collection, and the second part contains a narrative to explain the results using a statistical comparison chart organized by research question and hypotheses.

Data Collection

The time and code required to implement two different reporting applications (interactive grid and faceted search) were the focus of this study. For each of the example applications, data (passage of time) was collected in regular intervals throughout the coding effort. Once coding was completed, the lines of code were recorded for each subtask. Care was taken to reduce the number of blank and comment lines in the lines of code count. Time and lines of code were recorded for the following:

- ⊕ Oracle APEX Faceted Search example application (Appendix A)
- ⊕ Oracle APEX Interactive Grid example application (Appendix B)
- ⊕ ReactJS common setup and preparation (Appendix C)
- ⊕ ReactJS Interactive Grid example application (Appendix D)
- ⊕ ReactJS Faceted Search example application (Appendix E)

After collection was completed, a separate and combined summary chart was created to call out the results of the high-level tasks represented by the research questions and hypotheses (Appendix F). **Figure 7** presents the aggregate time and lines of code counts to build a comparison of effort needed to produce APEX-equivalent applications.

Figure 7. Separate and Combined Summary Comparison of Time and Motion

<i>interactive grid</i>	APEX		JavaScript Framework (ReactJS)		APEX vs. JavaScript Framework			
	Time (min)	Lines of Code	Time (min)	Lines of Code	% faster	% less code	x faster	x less code
Setup/Preparation	21	0	246	365	91%	100%	11.71	
Module Development	14	0	1,373	1,592	99%	100%	98.07	
Additional/Solution Development	10	93	10	7	0%	-1229%	1.00	0.08
Totals	45	93	1,629	1,964	97%	95%	36.20	21.12
Totals (time in hours)	0.75		27.15					

<i>faceted search</i>	APEX		JavaScript Framework (ReactJS)		APEX vs. JavaScript Framework			
	Time (min)	Lines of Code	Time (min)	Lines of Code	% faster	% less code	x faster	x less code
Setup/Preparation	21	0	246	365	91%	100%	11.71	
Module Development	12	0	1,518	1,378	99%	100%	126.50	
Additional/Solution Development	10	93	10	7	0%	-1229%	1.00	0.08
Totals	43	93	1,764	1,743	98%	95%	41.02	18.74
Totals (time in hours)	0.72		29.40					

<i>combined development*</i>	APEX		JavaScript Framework (ReactJS)		APEX vs. JavaScript Framework			
	Time (min)	Lines of Code	Time (min)	Lines of Code	% faster	% less code	x faster	x less code
Setup/Preparation	21	0	246	365	91%	100%	11.71	
Module Development - FS	12	0	1,518	1,378	99%	100%	126.50	
Module Development - IG	14	0	1,373	1,592	99%	100%	98.07	
Additional/Solution Development	10	93	10	7	0%	-1229%	1.00	0.08
Totals	57	93	3,147	3,342	98%	97%	55.21	35.94
Totals (time in hours)	0.95		52.45					

*If both modules would have been built in a single cycle.

Analysis

A time and lines of code summary chart was created using the summary of measurements taken, per subtask, during each implementation exercise. All research questions were tested based on the null and alternative hypotheses.

Research Questions

Research Question 1: Is there a significant difference in the time required to build an interactive grid application using APEX as compared with ReactJS?

- H_01 : There is no significant difference in the time required to build an interactive grid application using APEX as compared with ReactJS.
- H_a1 : There is a significant difference in the time required to build an interactive grid application using APEX as compared with ReactJS.

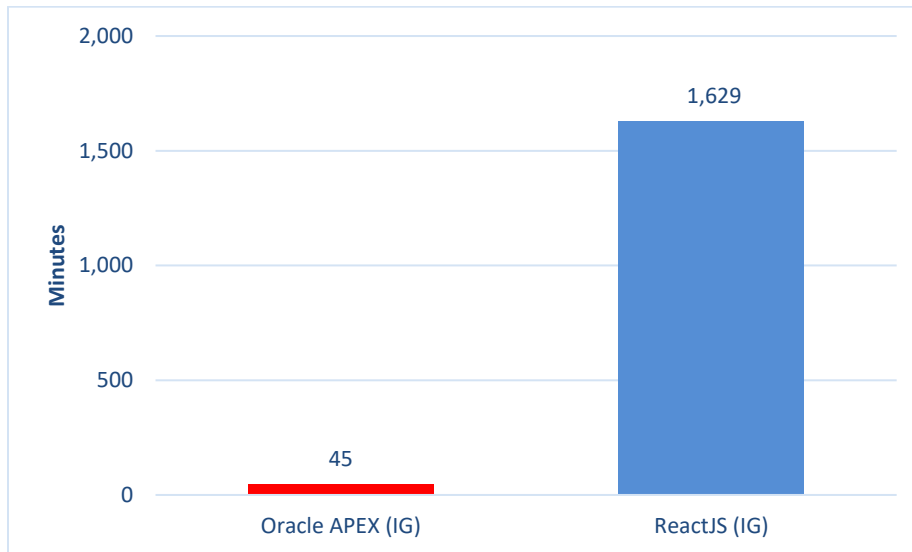
Research Question 2: Is there a significant difference in the time required to build a faceted search application using APEX as compared with ReactJS?

- H_02 : There is no significant difference in the time required to build a faceted search application using APEX as compared with ReactJS.
- H_a2 : There is a significant difference in the time required to build a faceted search application using APEX as compared with ReactJS.

Statistical Analysis

The first question was used to gauge the difference in the time required to build an interactive grid application using APEX as compared with ReactJS. The time recorded to develop an APEX interactive grid example application was 45 minutes (**Figure 8**): Preparing the APEX environment took 21 minutes, creating the application using the APEX automated workflow took 14 minutes, and developing an additional feature to toggle Genre between English and French in selected cells took 10 minutes. Total lines of code implemented for the interactive grid solution on APEX was 93, and lines of code written per minute was 2.06.

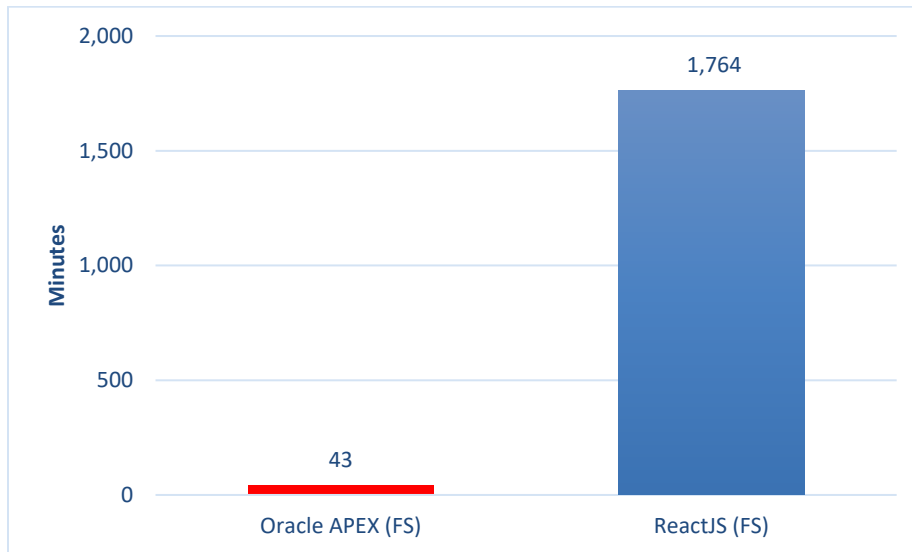
The same interactive grid solution built using ReactJS and MySQL took 1,629 minutes to create (**Figure 8**): Preparing the ReactJS environment (Docker, Kubernetes, etc.) took 246 minutes, developing the main application took 1,592 minutes, and developing an additional feature to toggle Genre between English and French in selected cells took 10 minutes. Total lines of code implemented for the interactive grid solution on ReactJS was 1,964, and lines of code written per minute was 1.21.

Figure 8. Difference in Time between APEX and ReactJS Interactive Grid Example Applications

Overall, the APEX interactive grid example application required 21.12 times less code and was 36.20 times faster to build than the equivalent ReactJS interactive grid example application. Without Additional/Solution Development ([Figure 7](#)), 1,957 more lines of code were required in ReactJS versus in APEX. Based on the results of this experiment, the alternate hypothesis is confirmed. There is a significant difference between the time required to build an interactive grid application using APEX as compared with ReactJS.

The second question was used to gauge the difference in the time required to build a faceted search application using APEX as compared with ReactJS. The time recorded to develop an APEX faceted search example application was 43 minutes ([Figure 9](#)): Preparing the APEX environment took 21 minutes, creating the application using the APEX automated workflow took 12 minutes, and developing an additional feature to toggle Genre between English and French in selected cells took 10 minutes. Total lines of code implemented for the interactive grid solution on ReactJS was 93, and lines of code written per minute was 2.16.

The same faceted search solution built using ReactJS and MySQL took 1,764 minutes to create ([Figure 9](#)): Preparing the ReactJS environment (Docker, Kubernetes, etc.) took 246 minutes, developing the main application took 1,518 minutes, and developing an additional feature to toggle Genre between English and French in selected cells took 10 minutes. Total lines of code implemented for the faceted search solution on ReactJS was 1,743, and lines of code written per minute was 0.99. Without Additional/Solution Development ([Figure 7](#)), 1,736 more lines of code were required in ReactJS versus in APEX.

Figure 9. Difference in Time between APEX and ReactJS Faceted Search Example Applications

Overall, the APEX faceted search example application required 18.74 times less code and was 41.02 times faster to build than the equivalent ReactJS faceted search example application. Based on the results of this experiment, the alternate hypothesis is confirmed. There is a significant difference between the time required to build a faceted search application using APEX as compared with ReactJS.

Summary

On average, the Oracle APEX example applications required 19.93 times less code and were 38.61 times faster to build than their equivalent in ReactJS. It should be noted that features such as saving reports, highlighting rows in interactive grid, and applying multiple numerical filters in faceted search were not implemented in ReactJS due to scope inflation or bugs at the time of the study's execution. APEX also included features such as accessibility, right-to-left text, better performance for large datasets, and multiuser scenarios such as saving and editing reports. The time and lines of code recorded for each example application was deemed sufficient for the purposes of this study.

The following conclusion provides a discussion of the overall development experience for both APEX and ReactJS and suggestions for future versions of this study.

Conclusions

Oracle's claim that creating an application with the Oracle APEX platform dramatically reduces the development time and lines of code was validated in this study. The average time is roughly 40 times less and the average lines of code is 20 times less than when creating an application with a ReactJS framework. This savings is substantial for those wanting to build applications that will support a very mature set of features coupled with performance and multitenant collaboration. The free-tier development model and the expedited application creation workflows (creating an application from the data itself) are powerful reasons to consider the APEX platform for value-added solution development.

It should be noted that ag-Grid's availability, and our decision to select it among its competitors, was a major determiner of our outcome. If ag-Grid were not as feature-rich as it is, contained major defects, or was not cost-effective, the study outcome would have been much different. Additionally, the skillset of our professional developer was well rounded and capable of addressing feature parity in an agile fashion. The skillset required to match APEX's feature set was nontrivial and should not be overlooked as a factor in this study. APEX is clearly suited for a wider range of professional developers and allows for lower cost developer skillset options.

Oracle could improve its APEX federated login to let users remember one set of credentials instead of two. Using one set of credentials to log in to Oracle Cloud and then another set to log in to Oracle APEX was confusing and led to losing access, inadvertently, during the study execution. Federated login is one area where Oracle could improve the APEX experience.

Future versions of this time and motion study could be expanded to take on larger projects (e.g., a decision support system) to qualify the savings of time and lines of code for those applications requiring nondefault features. The addition of nondefault features might expose a narrowing or widening of the gaps discovered for time to develop or lines of code written. The additional code written to support a nondefault function in Oracle APEX was equal in terms of the time to develop (10 minutes) but was 13.29 times more lines of code written when compared with ReactJS. The difference may be explained by the difference in the API exposed by APEX and ag-Grid. Also, some features required fewer lines of code in ReactJS because knowledge about the ag-Grid API was accumulated during the experiment that allowed for reuse of code previously written to accomplish other features.

Appendices

Appendix A. Oracle APEX Faceted Search example application task tracking worksheet

Appendix B. Oracle APEX Interactive Grid example application task tracking worksheet

Appendix C. React JS common setup and preparation task tracking worksheet

Appendix D. React JS Interactive Grid example application task tracking worksheet

Appendix E. React JS Faceted Search example application task tracking worksheet

Appendix F. Comparison table of results for time and code

Appendix G. Oracle APEX Interactive Grid User Interface

Appendix H. Oracle APEX Faceted Search User Interface

Appendix I. ReactJS Interactive Grid User Interface

Appendix J. ReactJS Faceted Search User Interface

Appendix K. Application, grid, header, and row menus

Appendix L. Login Modal Dialog

Appendix M. Help Modal Dialog

Appendix N. Difference in Time for APEX Example Applications

Appendix O. Difference in Time for ReactJS Example Applications

Appendix P. Combined Difference in Time between APEX and ReactJS Example Applications

Appendix Q. Difference in Lines of Code for ReactJS Example Applications

Appendix R. Excerpt of code from Interactive Grid on ReactJS

Appendix S. Excerpt of code from Interactive Grid on APEX

Appendix A. Oracle APEX Faceted Search Example Application Task Tracking Worksheet

Stage	Task	Sub Task	Time to Complete (Minutes)	Lines of Code Written
Setup/preparation	create Oracle cloud account	personal info	2	0
Setup/preparation	create Oracle cloud account	financial info	3	0
Setup/preparation	provision Autonomous Database	configure	2	0
Setup/preparation	provision Autonomous Database	credentials	2	0
Setup/preparation	provision Autonomous Database	provision	5	0
Setup/preparation	launch APEX	launch APEX	3	0
Setup/preparation	login to administration	login to administration	1	0
Setup/preparation	create workspace	name workspace	1	0
Setup/preparation	create workspace	credentials	1	0
Setup/preparation	logout of administration	logout of administration	1	0
		Total	21	0
Module development	login to workspace	login to workspace	1	0
Module development	launch app builder	new application	1	0
Module development	launch app builder	import database	4	0
Module development	launch app builder	name application	1	0
Module development	launch app builder	create application	2	0
Module development	launch app builder	add page - faceted search	3	0
		Total	12	0
Additional/Solution development	original language code	two buttons to toggle language from english to french, and french to english	10	93
		Total	10	93

Appendix B. Oracle APEX Interactive Grid Example Application Task Tracking Worksheet

Stage	Task	Sub Task	Time to Complete (Minutes)	Lines of Code Written
Setup/preparation	create Oracle cloud account	personal info	2	0
Setup/preparation	create Oracle cloud account	financial info	3	0
Setup/preparation	provision Autonomous Database	configure	2	0
Setup/preparation	provision Autonomous Database	credentials	2	0
Setup/preparation	provision Autonomous Database	provision	5	0
Setup/preparation	launch APEX	launch APEX	3	0
Setup/preparation	login to administration	login to administration	1	0
Setup/preparation	create workspace	name workspace	1	0
Setup/preparation	create workspace	credentials	1	0
Setup/preparation	logout of administration	logout of administration	1	0
		Total	21	0
Module development	login to workspace	login to workspace	1	0
Module development	launch app builder	new application	1	0
Module development	launch app builder	import database	4	0
Module development	launch app builder	name application	1	0
Module development	launch app builder	create application	2	0
Module development	launch app builder	add page - interactive grid	5	0
		Total	14	0
Additional/Solution development	original_language code	two buttons to toggle language from english to french, and french to english	10	93
		Total	10	93

Appendix C. ReactJS Common Setup and Preparation Task Tracking Worksheet

Stage	Activity	Sub Activity	Time to Complete (Minutes)	Lines of Code Written
Setup/preparation	Local environment	install local Docker environment (kubernetes CLI)	10	0
Setup/preparation	create Oracle cloud account	personal info	2	0
Setup/preparation	create Oracle cloud account	financial info	3	0
Setup/preparation	Oracle container cluster provisioned	allocate kubernetes cluster	40	0
Setup/preparation	Oracle container cluster provisioned	configure local kubernetes CLI to connect to Oracle kubernetes cluster	10	0
Setup/preparation	deploy database (MySQL)	create config	2	10
Setup/preparation	deploy database (MySQL)	deploy config	1	0
Setup/preparation	deploy database (MySQL)	create storage	2	30
Setup/preparation	deploy database (MySQL)	deploy storage	1	0
Setup/preparation	deploy database (MySQL)	create deployment	2	30
Setup/preparation	deploy database (MySQL)	deploy deployment	1	0
Setup/preparation	deploy database (MySQL)	create service	2	12
Setup/preparation	deploy database (MySQL)	deploy service	1	0
Setup/preparation	connect to MySQL	port forward	10	1
Setup/preparation	connect to MySQL	install mysql client on local environment to connect to MySQL container	5	0
Setup/preparation	create psql table	Create empty virtual database and table based on identified data taxonomy	15	0
Setup/preparation	data loading	clean data (remove problematic records)	25	0
Setup/preparation	data loading	Populate database table contents to database table	5	0
Setup/preparation	build container for node.js (interactive grid code)	create secrets	2	15
Setup/preparation	build container for node.js (interactive grid code)	deploy secrets	1	0
Setup/preparation	build container for node.js (interactive grid code)	create config	2	13
Setup/preparation	build container for node.js (interactive grid code)	deploy config	1	0
Setup/preparation	build container for node.js (interactive grid code)	create deployment	2	35
Setup/preparation	build container for node.js (interactive grid code)	deploy deployment	1	0
Setup/preparation	build container for node.js (interactive grid code)	create service	2	16
Setup/preparation	build container for node.js (interactive grid code)	deploy service	1	0
Setup/preparation	Docker config	create interactive grid docker file	20	36
Setup/preparation	Docker config	create dockerignore file	5	5
Setup/preparation	Docker config	build docker file	3	0
Setup/preparation	Docker config	connect to Oracle cluster registry	20	0
Setup/preparation	Docker config	push docker build to Oracle cluster registry	4	0
Setup/preparation	Docker config	create build and deploy script for all components (IG App)	15	26
Setup/preparation	Docker config	build and deploy	4	0
Setup/preparation	build container for API (abstract MySQL)	create secrets	2	15
Setup/preparation	build container for API (abstract MySQL)	deploy secrets	1	0
Setup/preparation	build container for API (abstract MySQL)	create config	2	13
Setup/preparation	build container for API (abstract MySQL)	deploy config	1	0
Setup/preparation	build container for API (abstract MySQL)	create deployment	2	35
Setup/preparation	build container for API (abstract MySQL)	deploy deployment	1	0
Setup/preparation	build container for API (abstract MySQL)	create service	2	16
Setup/preparation	build container for API (abstract MySQL)	deploy service	1	0
Setup/preparation	Docker config	createAPI docker file	2	20
Setup/preparation	Docker config	create dockerignore file	2	11
Setup/preparation	Docker config	create build and deploy script for all components (API)	10	26
		Total	246	365

Appendix D. ReactJS Interactive Grid Example Application Task Tracking Worksheet

Highlighted rows show tasks that were not completed due to lack of support in underlying grid framework.

Module development	API application	create table model (service-side CRUD)	30	62
Module development	API application	create express app (expose table model via webservice)	30	74
Module development	API application	build and deploy API application on OKE	20	
Module development	API application	validate API for server-side CRUD (Create, Read, Update, Delete) functions - using Postman	120	
Module development	API Application	fix and redeploy API application	10	
Module development	nodejs application (interactive grid code)	create React application (index.js)	3	1
Module development	nodejs application (interactive grid code)	research and select nodejs grid library	30	
Module development	nodejs application (interactive grid code)	add ag-Grid nodejs library	10	25
Module development	nodejs application (interactive grid code)	create basic grid in React	30	90
Module development	nodejs application (interactive grid code)	create client-side import function to read database content through API application	15	5
Module development	nodejs application (interactive grid code)	create index.css	90	450
Module development	nodejs application (interactive grid code)	create app.scss	15	82
Module development	nodejs application (interactive grid code)	create MenuBar.js (include placeholder for hamburger menu, application name, user profile, sign-in, and sign-out)	60	20
Module development	nodejs application (interactive grid code)	create action sub-menu	90	130
Module development	nodejs application (interactive grid code)	create row context menu	120	100
Module development	nodejs application (interactive grid code)	implement filter feature	10	20
Module development	nodejs application (interactive grid code)	create help dialog placeholder	15	60
Module development	nodejs application (interactive grid code)	create search dialog and search function	30	17
Module development	nodejs application (interactive grid code)	create sort, hide, format, copy-to-clipboard, move columns	90	23
Module development	nodejs application (interactive grid code)	create selection revert changes and context menu revert row function (flashback, history)	30	20
Module development	nodejs application (interactive grid code)	implement format control break feature (pivot groupings)	30	11
Module development	nodejs application (interactive grid code)	implement getNextID (feature needed for adding new rows)	30	31
Module development	nodejs application (interactive grid code)	implement pagination, pinned column, download, and chart feature	30	12
Module development	nodejs application (interactive grid code)	implement columns and filter feature	15	24
Module development	nodejs application (interactive grid code)	create stretch and shrink columns feature	15	25
Module development	nodejs application (interactive grid code)	implement data aggregation feature	15	11
Module development	nodejs application (interactive grid code)	create context menu single row view	30	35
Module development	nodejs application (interactive grid code)	create selection delete row and context menu delete rows function	45	24
Module development	nodejs application (interactive grid code)	create selection add rows function	60	25
Module development	nodejs application (interactive grid code)	create selection duplicate rows and context menu duplicate row	60	25
Module development	nodejs application (interactive grid code)	create selection clear and fill (includes modal dialog for fill)	45	45
Module development	nodejs application (interactive grid code)	implement edit button in action sub-menu	15	10
Module development	nodejs application (interactive grid code)	implement security page placeholder	60	100
Module development	nodejs application (interactive grid code)	implement selection refresh rows, data refresh, and context menu refresh rows	15	10
Module development	nodejs application (interactive grid code)	implement save button (dirty record detection)	60	25
Module development	nodejs application (interactive grid code)	create highlight creation dialog		
Module development	nodejs application (interactive grid code)	create highlight policy client-side functions		
Module development	nodejs application (interactive grid code)	create highlight policy end-to-end feature		
		Total	1373	1592
Additional/Solution development	original_language code	two buttons to toggle language from english to french, and french to english	10	7
		Total	10	7

Appendix E. ReactJS Faceted Search Example Application Task Tracking Worksheet

Stage	Activity		Sub Activity	Time to Complete (Minutes)	Lines of Code Written
Module development	API application		create table model (service-side CRUD)	30	62
Module development	API application		create express app (expose table model via webservice)	30	74
Module development	API application		build and deploy API application on OKE	20	
Module development	API application		validate API for server-side CRUD (Create, Read, Update, Delete) functions - using Postman	120	
Module development	API Application		fix and redeploy API application	10	
Module development	nodejs application (faceted search code)	file	create React application (index.js)	3	1
Module development	nodejs application (faceted search code)	grid	research and select nodejs grid library	30	
Module development	nodejs application (faceted search code)	grid	add ag-Grid nodejs library	10	35
Module development	nodejs application (faceted search code)	grid	create basic grid in React	30	64
Module development	nodejs application (faceted search code)	database	create client-side import function to read database content through API application	15	5
Module development	nodejs application (faceted search code)	style	create index.css	90	450
Module development	nodejs application (faceted search code)	style	create index.css - add side-by-side faceted selection	30	30
Module development	nodejs application (faceted search code)	style	create index.css - add left side menu	40	30
Module development	nodejs application (faceted search code)	style	create index.css - facet and heading format changes	60	15
Module development	nodejs application (faceted search code)	style	create app.scss	15	14
Module development	nodejs application (interactive grid code)	grid	implement pagination, pinned column, download, and chart feature	30	12
Module development	nodejs application (faceted search code)	function	map reduce records for unique items per column	60	100
Module development	nodejs application (faceted search code)	function	map reduce records for unique item counts	60	11
Module development	nodejs application (faceted search code)	feature	render distinct (unique items) dynamically in html	140	85
Module development	nodejs application (faceted search code)	feature	create 'show more' and 'show less' placeholders based on 5 (less/more) items	15	20
Module development	nodejs application (faceted search code)	function	implement distribution of numeric elements	60	10
Module development	nodejs application (faceted search code)	feature	show more items on click of "show more"	90	33
Module development	nodejs application (faceted search code)	feature	naming of html items and assigning values dynamically	30	4
Module development	nodejs application (faceted search code)	function	set filters for columns in array	30	32
Module development	nodejs application (faceted search code)	feature	apply single text filter for columns from array	30	4
Module development	nodejs application (faceted search code)	feature	apply multiple text filters for columns from array	90	6
Module development	nodejs application (faceted search code)	function	implement security page placeholder	60	100
Module development	nodejs application (faceted search code)	function	implement generic filter feature	10	20
Module development	nodejs application (faceted search code)	feature	clear filters for columns	75	32
Module development	nodejs application (faceted search code)	function	implement columns and filter feature	10	20
Module development	nodejs application (faceted search code)	feature	apply single numerical filter for columns from array	75	43
Module development	nodejs application (faceted search code)	feature	format for currency and commas	30	46
				not supported	
Module development	nodejs application (faceted search code)		apply multiple numerical filters for columns from array		
Module development	nodejs application (faceted search code)		implement recursive filtering after initial filter is selected	90	20
				Total	1518
Additional/Solution development	original language code		two buttons to toggle language from english to french, and french to english	10	7
				Total	10
					7

Appendix F. Comparison Table of Results for Time and Code

<i>interactive grid</i>	APEX		JavaScript Framework (ReactJS)		APEX vs. JavaScript Framework			
	Time (min)	Lines of Code	Time (min)	Lines of Code	% faster	% less code	x faster	x less code
Setup/Preparation	21	0	246	365	91%	100%	11.71	
Module Development	14	0	1,373	1,592	99%	100%	98.07	
Additional/Solution Development	10	93	10	7	0%	-1229%	1.00	0.08
Totals	45	93	1,629	1,964	97%	95%	36.20	21.12
Totals (time in hours)	0.75		27.15					

<i>faceted search</i>	APEX		JavaScript Framework (ReactJS)		APEX vs. JavaScript Framework			
	Time (min)	Lines of Code	Time (min)	Lines of Code	% faster	% less code	x faster	x less code
Setup/Preparation	21	0	246	365	91%	100%	11.71	
Module Development	12	0	1,518	1,378	99%	100%	126.50	
Additional/Solution Development	10	93	10	7	0%	-1229%	1.00	0.08
Totals	43	93	1,764	1,743	98%	95%	41.02	18.74
Totals (time in hours)	0.72		29.40					

<i>combined development*</i>	APEX		JavaScript Framework (ReactJS)		APEX vs. JavaScript Framework			
	Time (min)	Lines of Code	Time (min)	Lines of Code	% faster	% less code	x faster	x less code
Setup/Preparation	21	0	246	365	91%	100%	11.71	
Module Development - FS	12	0	1,518	1,378	99%	100%	126.50	
Module Development - IG	14	0	1,373	1,592	99%	100%	98.07	
Additional/Solution Development	10	93	10	7	0%	-1229%	1.00	0.08
Totals	57	93	3,147	3,342	98%	97%	55.21	35.94
Totals (time in hours)	0.95		52.45					

*If both modules would have been built in a single cycle.

Appendix G. Oracle APEX Interactive Grid User Interface

☰ Movies

Search: All Text Columns Actions ▾

Id 1	Budget	Homepage	Id	Original L...	Original TI...	Overview	Popularity	Release D...	Revenue	Runtime	Status	Tagline	Title	Vote
16	225000000		2454	en	The Chroni...	One year af...	53.978602	5/15/2008	419651413	150	Released	Hope has a ...	The Chroni...	
17	220000000	http://marv...	24428	en	The Aveng...	When an u...	144.448633	4/25/2012	1519557910	143	Released	Some asse...	The Avenge...	
18	380000000	http://disne...	1865	en	Pirates of t...	Captain Ja...	135.413856	5/14/2011	1045713802	136	Released	Live Foreve...	Pirates of t...	
19	225000000	http://www....	41154	en	Men in Blac...	Agents J (...	52.035179	5/23/2012	624026776	106	Released	They are ba...	Men in Blac...	
20	250000000	http://www....	122917	en	The Hobbit...	Immediatel...	120.965743	12/10/2014	956019788	144	Released	Witness the...	The Hobbit...	
21	215000000	http://www....	1930	en	The Amazi...	Peter Parke...	89.866276	6/27/2012	752215857	136	Released	The untold ...	The Amazin...	
22	200000000	http://www....	20662	en	Robin Hood	When soldi...	37.668301	5/12/2010	310669540	140	Released	Rise and ris...	Robin Hood	
23	250000000	http://www....	57158	en	The Hobbit...	The Dwarv...	94.370564	12/11/2013	958400000	161	Released	Beyond dar...	The Hobbit...	
24	180000000	http://www....	2268	en	The Golden...	After overh...	42.990906	12/4/2007	372234864	113	Released	There are w...	The Golden...	

Appendix H. Oracle APEX Faceted Search User Interface

☰ Movies

Total Row Count 4,799

	Budget	Homepage	Original Language	Original Title	Overview	Popularity	Re d
<input checked="" type="checkbox"/> Original Language <input type="checkbox"/> en (4,501) <input type="checkbox"/> fr (70) <input type="checkbox"/> es (32) <input type="checkbox"/> de (27) <input type="checkbox"/> zh (27) Show More	237,000,000	http://www.avatarmovie.com/	en	Avatar	In the 22nd century, a paraplegic Marine is dispatched to the moon Pandora on a unique mission, but becomes torn between following orders and protecting an alien civilization.	150.44	12/11
<input checked="" type="checkbox"/> Status <input type="checkbox"/> Released (4,791) <input type="checkbox"/> Rumored (5) <input type="checkbox"/> Post Production (3)	300,000,000	http://disney.go.com/disneypictures/pirates/	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, has come back to life and is headed to the edge of the Earth with Will Turner and Elizabeth Swann. But nothing is quite as it seems.	139.08	5/15
<input checked="" type="checkbox"/> Budget <input type="checkbox"/> <5,000,000 (1,660) <input type="checkbox"/> 5,000,000 - 20,000,000 (1,043) <input type="checkbox"/> 20,000,000 - 40,000,000 (858) <input type="checkbox"/> >=40,000,000 (1,238)	245,000,000	http://www.sonypictures.com/movies/spectre/	en	Spectre	A cryptic message from Bond's past sends him on a trail to uncover a sinister organization. While M battles political forces to keep the secret service alive, Bond peels back the layers of deceit to reveal the terrible truth behind SPECTRE.	107.38	10/2
	250,000,000	http://www.thedarkknightriserises.com/	en	The Dark Knight Rises	Following the death of District Attorney Harvey Dent, Batman assumes responsibility for Dent's crimes to protect the late attorney's reputation and is subsequently hunted by the Gotham City Police Department. Eight years later, Batman encounters the mysterious Selina Kyle and the villainous Bane, a new terrorist leader who overthrows Gotham's finest. The Dark Knight resurfaces to protect a city that has branded him an enemy.	112.31	7/16
	260,000,000	http://movies.disney.com/john-carter	en	John Carter	John Carter is a war-weary, former military captain who's inexplicably transported to the mysterious and exotic planet of Barsoom (Mars) and reluctantly becomes embroiled in an epic conflict. It's a world on the brink of collapse, and Carter rediscovers his humanity when he realizes the survival of Barsoom and its people rests in his hands.	43.93	3/7

Appendix I. ReactJS Interactive Grid User Interface

The screenshot displays the Oracle APEX Interactive Grid interface for a 'Movies' table. The interface includes a search bar, navigation buttons (Go, Actions, Edit, Save, Add Row), and a table with columns: ID, Title, Lang..., Overview, Popul..., Release Date, Revenue, and Runtl... The 'Finding Nemo' row (ID 12) is selected, and an actions menu is open over it, showing options like Columns, Filter, Data, Format, Selection, Chart, Report, Download, and Help. The table footer shows '1 to 100 of 4,785' and 'Page 1 of 48'.

ID	Title	Lang...	Overview	Popul...	Release Date	Revenue	Runtl...
1	Finding Nemo	en	Nemo, an adventurous yo...	85.688789	2003-05-30T00:00:00.000Z	940335536	100
5	Four Rooms	en	It's Ted the Bellhop's first ...	22.87623	1995-12-09T00:00:00.000Z	4300000	98
11	Star Wars	en	Princess Leia is captured ...	126.393695	1977-05-25T00:00:00.000Z	775398007	121
12	Finding Nemo	en	Nemo, an adventurous yo...	85.688789	2003-05-30T00:00:00.000Z	940335536	100
13	Forrest Gump	en	A man with a low IQ has ...	138.133331	1994-07-06T00:00:00.000Z	677945399	142
14	American Beauty	en	Lester Burnham, a depres...	80.878605	1999-09-15T00:00:00.000Z	356296601	122
16	Dancer in the Dark	en	Selma, a Czech immigran...	22.022228	2000-05-17T00:00:00.000Z	40031879	140
18	The Fifth Element	en	In 2257, a taxi driver is un...	109.528572	1997-05-07T00:00:00.000Z	263920180	126
19	Metropolis	de	In a futuristic city sharply ...	32.351527	2027-01-10T00:00:00.000Z	650422	153
20	My Life Without Me	en	A Pedro Almodovar produ...	7.958831	2003-03-07T00:00:00.000Z	9728954	106
22	Pirates of the Caribbean: ...	en	Jack Sparrow, a free whee...	271.972889	2003-07-09T00:00:00.000Z	655011224	143

Appendix J. ReactJS Faceted Search User Interface

☰ Movies (fs)
👤

Language

- af (1)
- ar (1)
- cs (1)
- da (6)
- de (24)
- [Show More](#)

Status

- Post Production (3)
- Released (4401)
- Rumored (5)

Budget

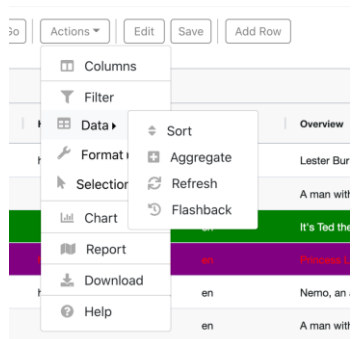
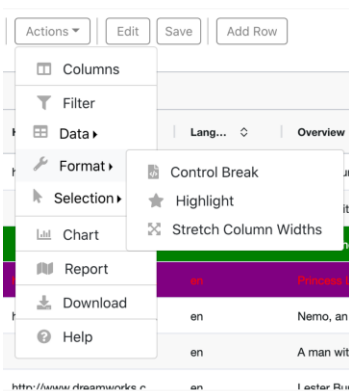
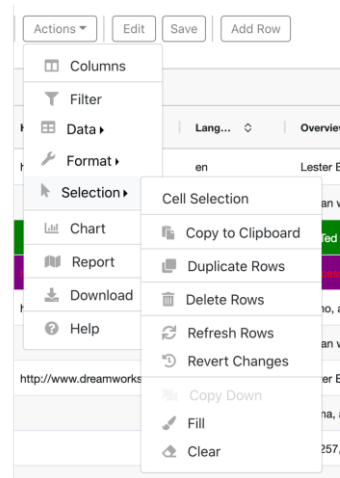
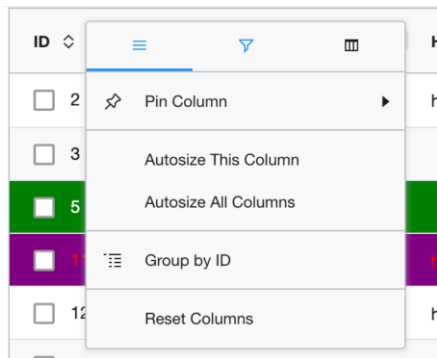
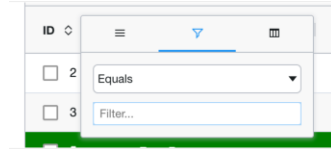
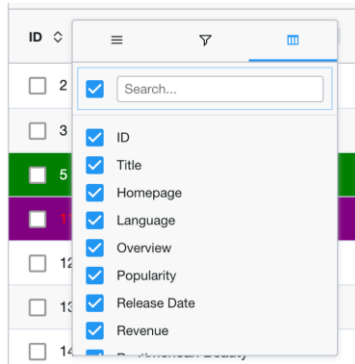
- <11000000 (1958)
- 11000000 - 22000000 (660)
- 22000000 - 201000000 (1765)
- >=201000000 (25)

Popularity

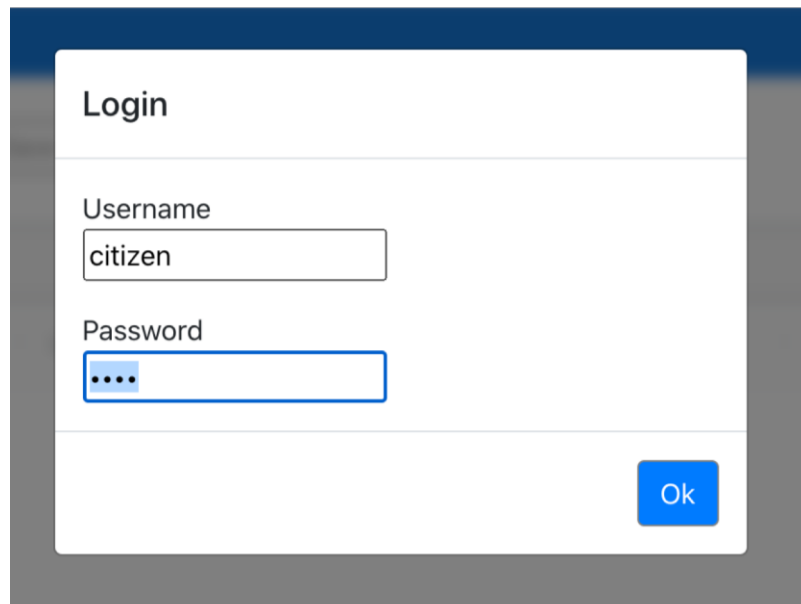
- <13 (2179)
- 13 - 26 (998)
- 26 - 451 (1228)

Title	Homepage	Lang...	Overview	Popul...	Release Date	Revenue
Four Rooms		en	It's Ted the Bellhop's first ...	22.87623	12/9/95	4300000
Star Wars	http://www.starwars.com/...	en	Princess Leia is captured ...	126.393695	5/25/77	775398007
Finding Nemo	http://movies.disney.com/...	en	Nemo, an adventurous yo...	85.688789	5/30/03	940335536
Forrest Gump		en	A man with a low IQ has ...	138.133331	7/6/94	677945399
American Beauty	http://www.dreamworks.c...	en	Lester Burnham, a depres...	80.878605	9/15/99	356296601
Dancer in the Dark		en	Selma, a Czech immigran...	22.022228	5/17/00	40031879
The Fifth Element		en	In 2257, a taxi driver is un...	109.528572	5/7/97	263920180
Metropolis		de	In a futuristic city sharply ...	32.351527	1/10/27	650422
My Life Without Me	http://www.clubcultura.co...	en	A Pedro Almodovar produ...	7.958831	3/7/03	9726954
Pirates of the Caribbean: ...	http://disney.go.com/disn...	en	Jack Sparrow, a freewhee...	271.972889	7/9/03	655011224
Kill Bill: Vol. 1	http://www.miramax.com...	en	An assassin is shot at the...	79.754966	10/10/03	180949000
Jarhead		en	Jarhead is a film about a ...	32.227223	11/4/05	96889998
Unforgiven		en	William Munny is a retred...	37.380435	8/7/92	159157447
The Simpsons Movie	http://www.simpsonsmov...	en	After Homer accidentally ...	46.875375	7/25/07	527068851

Appendix K. Application, Grid, Header, and Row Menus

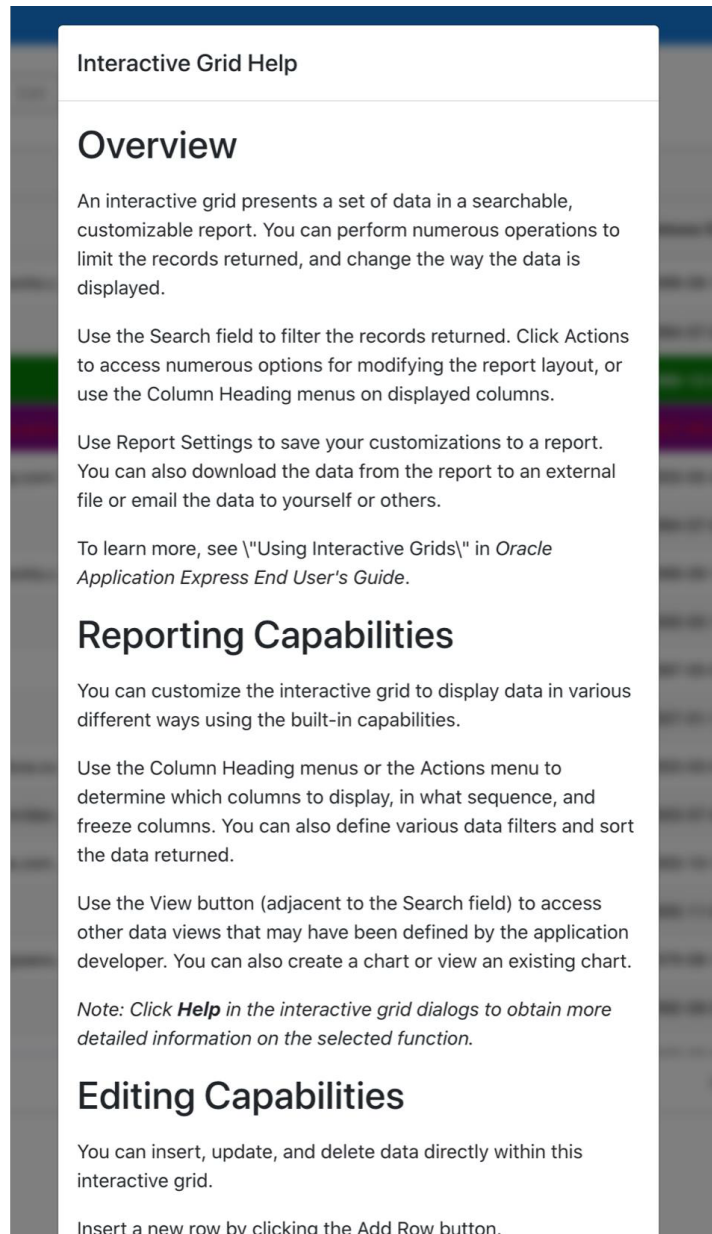


Appendix L. Login Modal Dialog



The image shows a login modal dialog box with a dark blue header and a white body. The title "Login" is positioned at the top left. Below the title, there are two input fields: "Username" with the text "citizen" and "Password" with three dots indicating a masked field. A blue "Ok" button is located in the bottom right corner of the dialog.

Appendix M. Help Modal Dialog



Interactive Grid Help

Overview

An interactive grid presents a set of data in a searchable, customizable report. You can perform numerous operations to limit the records returned, and change the way the data is displayed.

Use the Search field to filter the records returned. Click Actions to access numerous options for modifying the report layout, or use the Column Heading menus on displayed columns.

Use Report Settings to save your customizations to a report. You can also download the data from the report to an external file or email the data to yourself or others.

To learn more, see *"Using Interactive Grids"* in *Oracle Application Express End User's Guide*.

Reporting Capabilities

You can customize the interactive grid to display data in various different ways using the built-in capabilities.

Use the Column Heading menus or the Actions menu to determine which columns to display, in what sequence, and freeze columns. You can also define various data filters and sort the data returned.

Use the View button (adjacent to the Search field) to access other data views that may have been defined by the application developer. You can also create a chart or view an existing chart.

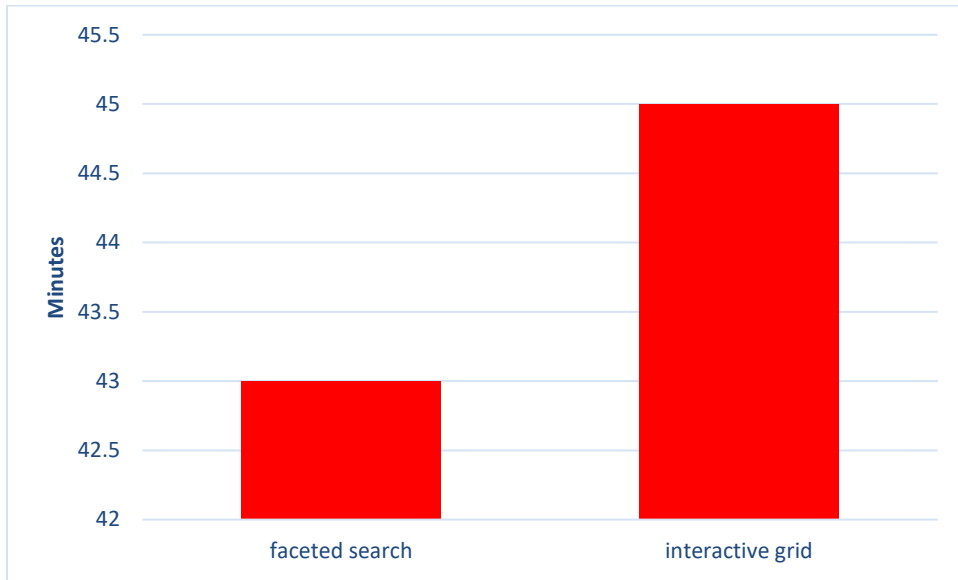
*Note: Click **Help** in the interactive grid dialogs to obtain more detailed information on the selected function.*

Editing Capabilities

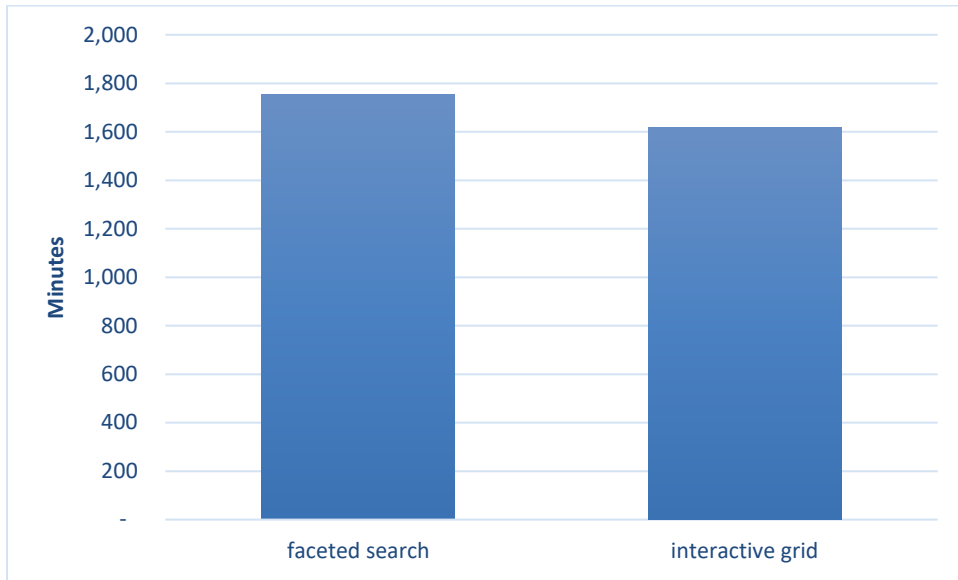
You can insert, update, and delete data directly within this interactive grid.

Insert a new row by clicking the Add Row button.

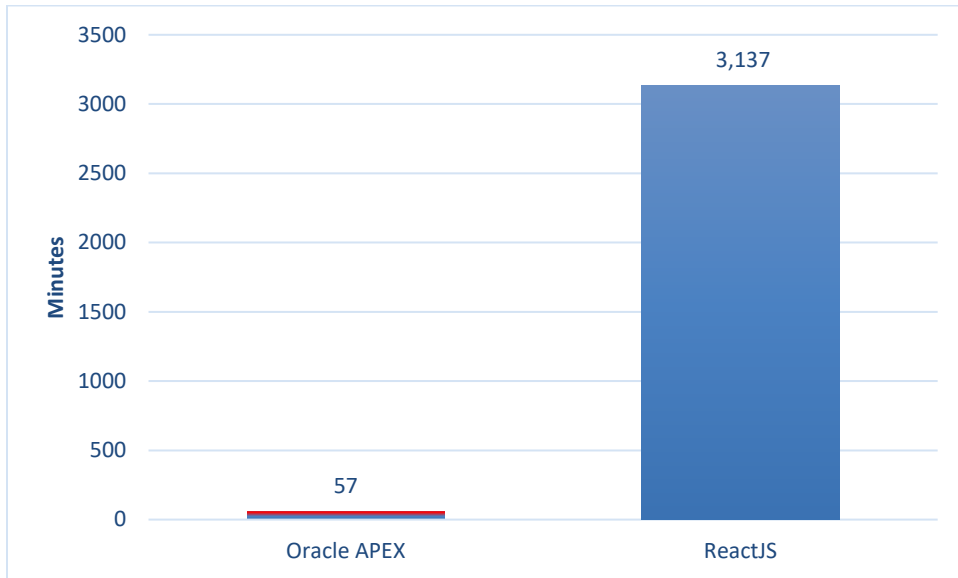
Appendix N. Difference in Time for Oracle APEX Example Applications



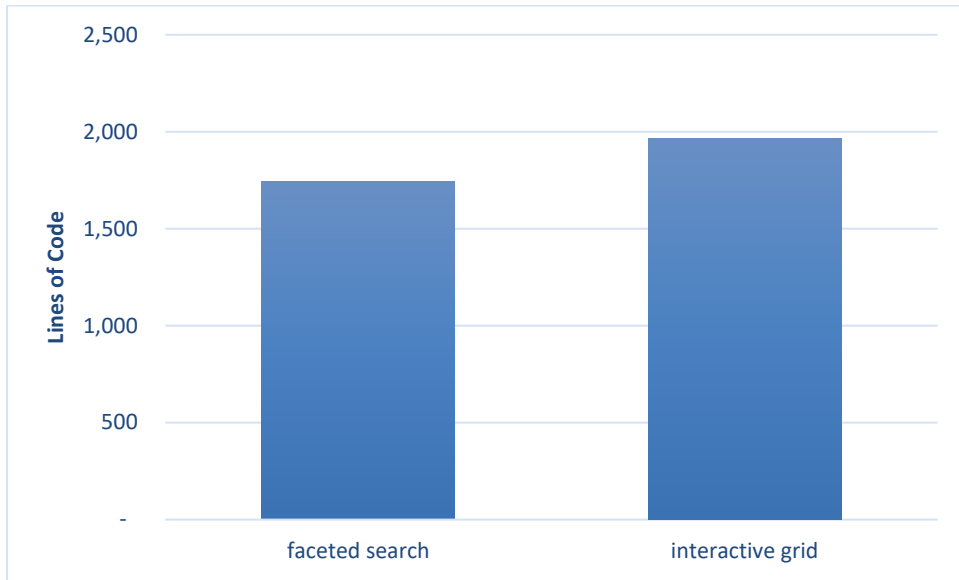
Appendix O. Difference in Time for ReactJS Example Applications



Appendix P. Combined Difference in Time between APEX and ReactJS Example Applications



Appendix Q. Difference in Lines of Code for ReactJS Example Applications



Appendix R. Excerpt of code from Interactive Grid on ReactJS

```
import React, {Component} from 'react';

import MenuBar from './MenuBar.js';
import './App.scss';
import { AgGridReact } from 'ag-grid-react';
// import './ag-grid-enterprise.min';
// import 'ag-grid-community/dist/styles/ag-grid.css';
// import 'ag-grid-community/dist/styles/ag-theme-alpine.css';
import 'ag-grid-enterprise';
import { ModuleRegistry, AllModules } from '@ag-grid-enterprise/all-modules';
import {Button} from "reactstrap";
import {Dropdown} from "react-bootstrap";
import ModalHelp from "./functions/ModalHelp";
import ModalSecurity from "./functions/ModalSecurity";
import ModalHighlight from "./functions/ModalHighlight";
import ModalFill from "./functions/ModalFill";
import ModalSingleRowView from "./functions/ModalSingleRowView";
import fontawesome from '@fortawesome/fontawesome'
import faFreeSolid from '@fortawesome/fontawesome-free-solid'

ModuleRegistry.registerModules(AllModules);

const host = 'localhost';
const api_port = 30001;
// const host = 'np-api.default.svc.cluster.local';
// const api_port = 3001;
console.log(`host:port = ${host}:${api_port}`);

class App extends Component {

  constructor(props) {
    super(props);
    this.getContextMenuItems = this.getContextMenuItems.bind(this);
    this.toggleModalHelp = this.toggleModalHelp.bind(this);
    this.toggleModalSecurity = this.toggleModalSecurity.bind(this);
    this.toggleModalHighlight =
this.toggleModalHighlight.bind(this);
    this.toggleModalSingleRowView =
this.toggleModalSingleRowView.bind(this);
    this.toggleModalFill = this.toggleModalFill.bind(this);
    this.state = {
      searchStr: '',
      stretch: false,
      sort: true,
      username: '',
      password: '',
      isModalSingleRowViewOpen: false,
      isModalHelpOpen: false,
      isModalSecurityOpen: false,

```

```

        isModalHighlightOpen: false,
        isModalFillOpen: false,
        showColumnPanel: true,
        showFilterPanel: false,
        showPivotPanel: false,
        modalContent: "hello world",
        modalTitle: "",
        cellSelection: true,
        cellSelectionText: "Cell Selection",
        dirtyRecords: [],
        dirtySave: false,
        highlightRecords: [{column: "title", text: "Four Rooms",
operator: "==", color: "red", backgroundcolor: "yellow"},{column:
"title", text: "Star Wars", operator: "==", color: "blue",
backgroundcolor: "yellow"},{column: "runtime", text: "140", operator:
">=", color: "pink", backgroundcolor: "yellow"}],
        highlightPolicy: [],
        highlightPolicy2: { 'rag-back-green rag-fore-white':
'data.title == "Four Rooms"', 'rag-fore-red rag-back-purple':
'data.title == "Star Wars"'},
        columnDefs: [{
            headerName: "ID", editable: false, filter:
'agNumberColumnFilter', width: 100, field: "id", sortable: true,
checkboxSelection: function(params) {
                return true;
            }, resizable: true, enablePivot: true, enableRowGroup:
true,
        }, {
            headerName: "Title", editable: true, field: "title",
sortable: true, filter: true, resizable: true, enableRowGroup: true,
enablePivot: true,
        }, {
            headerName: "Homepage", editable: true, field:
"homepage", sortable: true, filter: true, resizable: true,
enableRowGroup: true, enablePivot: true,
        }, {
            headerName: "Language", editable: true, width: 120,
field: "original_language", sortable: true, filter: true, resizable:
true, enableRowGroup: true, enablePivot: true,
        }, {
            headerName: "Overview", editable: true, field:
"overview", sortable: true, filter: true, resizable: true,
enableRowGroup: true, enablePivot: true,
        }, {
            headerName: "Popularity", filter:
'agNumberColumnFilter', editable: true, width: 120, field:
"popularity", sortable: true, resizable: true, enableValue: true,
enableRowGroup: true, enablePivot: true,
        }, {
            headerName: "Release Date", editable: true, field:
"release_date", sortable: true, filter: true, resizable: true,
enableRowGroup: true, enablePivot: true,
        }, {

```

```

                headerName: "Revenue", editable: true, width: 140,
field: "revenue", sortable: true, filter: 'agNumberColumnFilter',
resizable: true, enableValue: true, enableRowGroup: true, enablePivot:
true,
            }, {
                headerName: "Runtime", editable: true, width: 120,
field: "runtime", sortable: true, filter: 'agNumberColumnFilter',
resizable: true, enableValue: true, enableRowGroup: true, enablePivot:
true,
            }, {
                headerName: "Status", editable: true, width: 120,
field: "status", sortable: true, filter: true, resizable: true,
enableRowGroup: true, enablePivot: true,
            }, {
                headerName: "Tagline", editable: true, field:
"tagline", sortable: true, filter: true, resizable: true,
enableRowGroup: true, enablePivot: true,
            }, {
                headerName: "Budget", editable: true, width: 120,
field: "budget", sortable: true, filter: 'agNumberColumnFilter',
resizable: true, enableValue: true, enableRowGroup: true, enablePivot:
true,
            }, {
                headerName: "Vote Average", editable: true, width: 140,
field: "vote_average", sortable: true, filter: 'agNumberColumnFilter',
resizable: true, enableValue: true, enableRowGroup: true, enablePivot:
true,
            }, {
                headerName: "Vote Count", editable: true, width: 140,
field: "vote_count", sortable: true, filter: 'agNumberColumnFilter',
resizable: true, enableValue: true, enableRowGroup: true, enablePivot:
true,
            }
        ]],
    }

    getContextMenuItems(params) {
        const obj = this;
        var result = [
            {
                name: 'Single Row View',
                action: function() {
                    var rowData = JSON.stringify(params.node.data);
                    console.log('Single Row Selected: ' + rowData);
                    obj.setState({modalTitle : "Single Row Details"});
                    obj.setState({modalContent : rowData});
                    obj.toggleModalSingleRowView();
                },
                icon: '',
            },
            'separator',
            {
                name: 'Add Row',
                action: function() {

```

```
        console.log('Add Row Selected');
        obj.actionAddRow();
    },
    icon: '',
},
{
    name: 'Duplicate Row',
    action: function() {
        console.log('Duplicate Row Selected');
        obj.actionDuplicateRow(params.node.data);
    },
    icon: '',
},
'separator',
{
    name: 'Delete Row',
    action: function() {
        console.log('Delete Row Selected');
        obj.actionDeleteRow();
    },
    icon: '',
},
'separator',
{
    name: 'Refresh Row',
    action: function() {
        console.log('Refresh Row Selected');
        obj.actionRefresh();
    },
    icon: '',
},
{
    name: 'Revert Changes',
    disable: true,
    action: function() {
        console.log('Revert Changes Selected');
        obj.actionFlashback();
    },
    icon: '',
},
'separator',
'copy',
'separator',
'chartRange',
];

return result;
}

toggleSave(position) {
    this.setState({dirtySave : position});
};

toggleModalHelp() {
```

```
        this.setState({ isModalHelpOpen: !this.state.isModalHelpOpen
    });
    }
    toggleModalSecurity() {
        this.setState({ isModalSecurityOpen:
!this.state.isModalSecurityOpen });
    }
    toggleModalHighlight() {
        this.setState({ isModalHighlightOpen:
!this.state.isModalHighlightOpen });
    }
    toggleModalFill() {
        this.setState({ isModalFillOpen: !this.state.isModalFillOpen
    });
    }
    toggleModalSingleRowView() {
        this.setState({ isModalSingleRowViewOpen:
!this.state.isModalSingleRowViewOpen });
    }

    // onClick = e => {
    //     this.gridApi.sizeColumnsToFit();
    //     const selectedNodes = this.gridApi.getSelectedNodes();
    //     const selectedData = selectedNodes.map( node => node.data );
    //     const selectedDataStringPresentation = selectedData.map(
node => node.title + ' ' + node.revenue).join(', ')
    //     alert(`Selected nodes: ${selectedDataStringPresentation}`)
    // };

    onLogin(username,password) {
        this.toggleModalSecurity();
        this.setState({username : username});
        this.setState({password : password});
        this.createHighlightPolicy();
        // setTimeout(() => {
        this.fetchData();
        // }, 1000);

    }

    fetchData() {
        fetch(`http://${host}:${api_port}`)
            .then(result => result.json())
            .then(rowData => this.setState({rowData}))
            .catch(err => console.error(err));
    }

    onFilterChanged(event) {
        this.setState({searchStr : event.target.value})
    }
    actionGo() {
        this.refs.agGrid.api.setQuickFilter(this.state.searchStr);
    }
    actionEdit() {
```

```
        console.log('action: "edit" attempted');
        const focusedCell =
this.refs.agGrid.gridOptions.api.getFocusedCell();
        this.refs.agGrid.gridOptions.api.startEditingCell({
            rowIndex: focusedCell.rowIndex,
            colKey: focusedCell.column,
        });
    }
    actionCheckBox() {
        return this.state.cellSelection;
    }
    actionToggleCellSelection() {
        if (this.state.cellSelection === true) {
            console.log('action: "toggle cell/row selection: row"
attempted');
            // this.columnApi.checkboxSelection(false);
            this.setState({cellSelection: false});
            this.setState({cellSelectionText: "Row Selection"});
        } else {
            console.log('action: "toggle cell/row selection: cell"
attempted');
            //
this.refs.agGrid.gridOptions.suppressCellSelection=false;
            this.setState({cellSelection: true});
            this.setState({cellSelectionText: "Cell Selection"});
        }
    }
    actionCopyDown() {
        console.log('action: "copy down" attempted');
    }

    actionFill() {
        console.log('action: "fill" attempted');
        this.setState({modalTitle : "Fill Selection"});
        this.setState({modalContent : "Fill selection with"});
        this.toggleModalFill();
    }
    actionExecuteFill(fillStr) {
        console.log('action: "execute fill" attempted: ' + fillStr);
        this.toggleModalFill();
        const focusedCell =
this.refs.agGrid.gridOptions.api.getFocusedCell();
        var rowNode =
this.refs.agGrid.gridOptions.api.getRowNode(focusedCell.rowIndex);
        rowNode.setDataValue(focusedCell.column, fillStr);
    }

    actionClear(){
        console.log('action: "clear" attempted');
        const selectedNodes = this.gridApi.getSelectedNodes();
        const selectedData = selectedNodes.map( node => node.rowIndex
);
        for (let rowIndex of selectedData) {
```

```
        this.refs.agGrid.gridOptions.rowData.fill(rowIndex-
1, rowIndex, rowIndex+1);
this.gridApi.setRowData(this.refs.agGrid.gridOptions.rowData);
    }
}
actionColumns() {
    if (this.state.showColumnPanel === false) {
        console.log('action: "columns - true" attempted');
        this.refs.agGrid.api.setSideBar('columns');
        this.refs.agGrid.api.openToolPanel('columns');
        this.setState({showFilterPanel: false});
        this.setState({showColumnPanel: true});
    } else {
        console.log('action: "columns - false" attempted');
        this.setState({showColumnPanel: false});
    }
}
actionFilter() {
    if (this.state.showFilterPanel === false) {
        console.log('action: "filter - true" attempted');
        this.refs.agGrid.api.setSideBar('filters');
        this.refs.agGrid.api.openToolPanel('filters');
        this.setState({showColumnPanel: false});
        this.setState({showFilterPanel: true});
    } else {
        console.log('action: "filter - false" attempted');
        this.setState({showFilterPanel: false});
    }
}
actionSort() {
    var sort = [];
    if (this.state.sort === true) {
        console.log('action: "data:sort - false" attempted');
        this.setState({sort: false});
        sort = [
            { colId: 'title', sort: 'asc' }
        ];
        this.refs.agGrid.api.setSortModel(sort);
    } else {
        console.log('action: "data:sort - true" attempted');
        this.setState({sort: true});
        sort = [
            { colId: 'id', sort: 'asc' }
        ];
        this.refs.agGrid.api.setSortModel(sort);
    }
}
actionAggregate() {
    if (this.state.showColumnPanel === false) {
        console.log('action: "data:aggregate - true" attempted');
        this.refs.agGrid.api.setSideBar('columns');
        this.refs.agGrid.api.openToolPanel('columns');
        this.setState({showFilterPanel: false});
    }
}
```



```
        this.setState({showColumnPanel: true});
    } else {
        console.log('action: "data:aggregate - false" attempted');
        this.setState({showColumnPanel: false});
    }
}
actionRefresh() {
    console.log('action: "data:refresh" attempted');
    var params = {
        force: true,
        suppressFlash: false,
    };
    this.refs.agGrid.api.refreshCells(params);
}
actionFlashback() {
    console.log('action: "data:flashback" attempted');
    this.gridApi.undoCellEditing();
}
actionFlashbacks() {
    console.log('action: "data:flashback" attempted');
    const selectedNodes = this.gridApi.getSelectedNodes();
    const selectedData = selectedNodes.map( node => node.data );
    for (let rowData of selectedData) {
        this.gridApi.undoCellEditing();
    }
}

actionControlBreak() {
    if (this.state.showColumnPanel === false) {
        console.log('action: "columns - true" attempted');
        this.refs.agGrid.api.setSideBar('columns');
        this.refs.agGrid.api.openToolPanel('columns');
        this.setState({showFilterPanel: false});
        this.setState({showColumnPanel: true});
    } else {
        console.log('action: "columns - false" attempted');
        this.setState({showColumnPanel: false});
    }
}
actionStretchColumn() {
    if (this.state.stretch === true) {
        console.log('action: "format:stretchcolumn - false"
attempted');
        this.setState({stretch: false});
        this.refs.agGrid.api.sizeColumnsToFit();
    } else {
        console.log('action: "format:stretchcolumn - true"
attempted');
        this.setState({stretch: true});
        this.refs.agGrid.columnApi.autoSizeColumns([
```

Appendix S. Excerpt of code from Interactive Grid on APEX

```

function (config)
{
    //No selected any row when the page is rendered
    config.initialSelection = false;
    //Begin - Creating two buttons, Inactivate and Activate
    var $ = apex.jquery,
        toolbarData = $.apex.interactiveGrid.copyDefaultToolbar(),
        toolbarGroup = toolbarData.toolbarFind("actions3");

    toolbarGroup.controls.push(
    {
        type: "BUTTON",
        action: "activate",
        icon: "fa fa-thumbs-up fam-check fam-is-success",
        iconBeforeLabel: true,
        hot: true,
    });
    toolbarGroup.controls.push(
    {
        type: "BUTTON",
        action: "inactivate",
        icon: "fa fa-thumbs-down fam-x fam-is-danger",
        iconBeforeLabel: true,
        hot: true,
    });
    config.toolbarData = toolbarData;
    //End - Creating two buttons, Inactivate and Activate
    config.initActions = function (actions)
    {
        // Defining the action for activate button
        actions.add(
        {
            name: "activate",
            label: "Activate",
            action: activate
        });
        // Defining the action for inactivate button
        actions.add(
        {
            name: "inactivate",
            label: "Inactivate",
            action: inactivate
        });
    }

    function activate(event, focusElement)
    {
        var i, records, model, record,
            view =
        apex.region("ig_emp").widget().interactiveGrid("getCurrentView");

        if (view.supports.edit)
        {

```

```
model = view.model;
records = view.getSelectedRecords();
if (records.length > 0)
{
    for (i = 0; i < records.length; i++)
    {
        record = records[i];
        //Set the value for the checked rows to A (Activate)
        model.setValue(record, "STATUS", 'A');
    }
    //Save the changes

apex.region("ig_emp").widget().interactiveGrid("getActions").invoke("save")
;
    }
}

function inactivate(event, focusElement)
{
    var i, records, model, record,
        view =
apex.region("ig_emp").widget().interactiveGrid("getCurrentView");

    if (view.supports.edit)
    {
        model = view.model;
        records = view.getSelectedRecords();
        if (records.length > 0)
        {
            for (i = 0; i < records.length; i++)
            {
                record = records[i];
                //Set the value for the checked rows to I (Inactivate)
                model.setValue(record, "STATUS", 'I');
            }
            //Save the changes

apex.region("ig_emp").widget().interactiveGrid("getActions").invoke("save")
;
            }
        }

    }
    return config;
}
```