❖ **To create a simple To-Do List Application with a React.js frontend and a Laravel backend, please follow the below mentioned steps**:

- **Setup Laravel Backend:**

- Install Laravel using Composer if you haven't already.
- Create a database and configure the .env file with database credentials.
- Create a migration for the tasks table and run it to create the database schema.
- Create a Task model and a Tasks Controller to handle CRUD operations for tasks.
- Implement API endpoints for creating, reading, updating, and deleting tasks.

Setup React Frontend:

- Set up a new React.js project using Create React App or any other preferred method.
- Create components for the To-Do list, task item, and any other necessary components.
- Use React state to manage the list of tasks and update the UI accordingly.
- Implement functionalities to add new tasks, mark tasks as completed, and delete tasks using Axios or Fetch API to communicate with the backend.

Integration:

- Use Axios or Fetch API to make HTTP requests from the React frontend to the Laravel backend's API endpoints.
- Test the integration by performing CRUD operations from the React frontend and verifying the changes in the database through the Laravel backend.
- Handle loading and error states in the React frontend to provide feedback to users during API requests.

Styling:

- Style the application using CSS or a CSS framework like Bootstrap to enhance the user experience.

- Ensure responsiveness so that the application works well on different screen sizes.
- Here's a basic example of how your React component for the To-Do list might look like:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const ToDoList = () => {
  const [tasks, setTasks] = useState([]);
  const [newTask, setNewTask] = useState('');

  useEffect(() => {
    fetchTasks();
  }, []);

  const fetchTasks = async () => {
    try {
      const response = await axios.get('/api/tasks');
      setTasks(response.data);
    } catch (error) {
      console.error('Error fetching tasks:', error);
    }
  };

  const addTask = async () => {
    try {
      const response = await axios.post('/api/tasks', { title: newTask });
```

```
    setTasks([...tasks, response.data]);
    setNewTask('');
  } catch (error) {
    console.error('Error adding task:', error);
  }
};

const deleteTask = async (taskId) => {
  try {
    await axios.delete(`/api/tasks/${taskId}`);
    setTasks(tasks.filter(task => task.id !== taskId));
  } catch (error) {
    console.error('Error deleting task:', error);
  }
};

return (
  <div>
    <h1>To-Do List</h1>
    <input
      type="text"
      value={newTask}
      onChange={(e) => setNewTask(e.target.value)}
      placeholder="Enter task..."
    />
```

```
    <button onClick={addTask}>Add Task</button>
    <ul>
     {tasks.map (task => (
       <li key={task.id}>
         <span>{task.title}</span>
         <button onClick={() => deleteTask(task.id)}>Delete</button>
       </li>
     ))}
    </ul>
   </div>
  );
};

export default ToDoList;
```

Here's how you can set up the backend using Laravel and integrate it with the React frontend:

Setup Laravel Project:

- Install Laravel using Composer if you haven't already: composer create-project --prefer-dist laravel/laravel todo-backend.
- Configure your .env file with your database credentials.
- Create a MySQL database for your application.
- Database Setup:

- Create a migration for the tasks table: php artisan make:migration create_tasks_table.
- Define the schema for the tasks table in the migration file.
- Run the migration to create the tasks table: php artisan migrate.

- Task Model and Controller:

- Create a Task model: php artisan make:model Task.
- Create a TasksController: php artisan make:controller TasksController.
- Implement CRUD operations in the TasksController using Eloquent ORM.

API Endpoints:

- Define API routes for CRUD operations in routes/api.php.

For example:

```
Route::get('/tasks', 'TasksController@index');

Route::post('/tasks', 'TasksController@store');

Route::put('/tasks/{id}', 'TasksController@update');

Route::delete('/tasks/{id}', 'TasksController@destroy');
```

Validation:

- Implement validation rules in the store method of TasksController to ensure task titles are not empty.
- Integration with React Frontend:

- Use Axios or Fetch API in your React frontend to make HTTP requests to the Laravel backend's API endpoints.

For example, to fetch tasks:

```
axios.get('/api/tasks')
  .then(response => {
    // Handle successful response
  })
  .catch(error => {
```

```
  // Handle error
});
```

- Similarly, implement methods to add, update, and delete tasks.

Testing and Error Handling:

- Test the integration by adding, updating, and deleting tasks from the React frontend and verifying the changes in the database through the Laravel backend.
- Implement loading and error states in the React frontend to provide feedback to users during API requests.

Here's a simplified example of how your TasksController might look

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Task;

class TasksController extends Controller
{
  public function index()
  {
    $tasks = Task::all();
    return response()->json($tasks);
  }

  public function store(Request $request)
  {
```

```
    $request->validate([
        'title' => 'required|string|max:255',
        'description' => 'nullable|string',
        'completed' => 'nullable|boolean',
    ]);

    $task = Task::create([
        'title' => $request->input('title'),
        'description' => $request->input('description'),
        'completed' => $request->input('completed', false),
    ]);

    return response()->json($task, 201);
}

public function update(Request $request, $id)
{
    $task = Task::findOrFail($id);

    $request->validate([
        'title' => 'required|string|max:255',
        'description' => 'nullable|string',
        'completed' => 'nullable|boolean',
    ]);
```

```
        $task->update([
            'title' => $request->input('title'),
            'description' => $request->input('description'),
            'completed' => $request->input('completed', false),
        ]);

        return response()->json($task);
    }

    public function destroy($id)
    {
        $task = Task::findOrFail($id);
        $task->delete();

        return response()->json(null, 204);
    }
}
```