خطوات العمل على فدلي، وكذلك مجموعة كاملة من التوضيحات حول فدلي والزيادات التي عملتها في هذا المشروع من استخدام الـ bootstrap والـ typeahead ونموذج الرسائل، كلُّها موضحة هنا.

The steps:

1. Create new react app called vidly and start it.
2. Install bootstrap and font-awesome libraries and import them in index.js
3. Replace the black panel rendered by <App> with a starter template from getbootstrap.com getting only the main tag (<main class=”container”></main>). This represents the content area of our application.
4. Copy services folder from the Starter Code zip folder into src folder. This folder contains two fake services.
5. Build Movies component, only the table with the delete button and the paragraph at the top (3-17).
6. Build Like component; it is a controlled component to its parent Movies component (4-22).

At this point, end of section 4, we built the basic Movies component. Section 5 is packed with lots of techniques for writing clean code, refactoring, and best practices.

1. Add Pagination component, determine its interface. Whenever we build a reusable component, before implementing it, we should think about interface about that component; what are the inputs that component is going to receive? What are the events is going to raise? the best way to decide the interface of a component is to use it before implementing it. We used a function from lodash library to generate an array of numbers (5-4). Also, we implement paginate() function with the help of lodash functions (5-6).

Note that pagination is implemented here on the client, but when we have large amount of data, we don’t want to load them ahead and paginate on the client, so we should implement it on the server to get a piece of data on each http request.

1. Add ListGroup component to filter movies. We made this component common and flexible (5-10, 11).
2. Refactor Movies to solve the issue of mixed levels of abstraction. So, to have our code symmetric and consistent, we extracted the table to MoviesTable component, and in future God willing we will refactor this code step by step and eventually we extract a reusable Table component.
3. Add sorting feature by raising onClick event of each <th> in MoviesTable component and handle the sorting in Movies.

Refactor Movies and MoviesTable by moving the responsibility of sorting (in fact, determining sortColumn which is the path and order type) to MoviesTable itself through the raiseSort() function .

Note that the sort operation parts, now, is in three places: (1) in raiseSort() of MoviesTable component to indicate the sortColumn (2) in handleSort() in Movies Component to update the state (3) in render() of Movies component too, to apply the ordering via orderBy() lodash function.

1. Refactor MoviesTable: when don’t want to duplicate the logic of raiseSort() that determines path and sort order in another table component like CustomersTable. We need to extract a new component like TableHeader to encapsulate this logic and then we can reuse this TableHeader whenever we have a table.
2. Refactor MoviesTable by extracting the body to TableBody component to solve the issue of mixed levels of abstraction. In (5-21), there is a tricky part to learn.
3. Refactor MoviesTable by extracting a new component call it table so we can reuse it easily whenever we want to render a table of items.
4. In Movies component, we extracted a method to encapsulate all the logic of filtering, sorting, and pagination in that method.

At this point, end of section 5, our Movies component has the features of filtering, sorting, and pagination.

1. Add navbar and routes to the project (6-15, 16, 17, 18). The tricky part is in modifying the table by replacing the title that was a text with a link and when click this link we should go to a new page that is our MovieForm and render the id of the movie that is passed as a url parameter.

At this point, end of section 6, we have the navbar and routes added to our project.

1. Add the LoginForm component. We learned how to prevent the default behavior of submitting a form. Also, we learned the proper way to work with input fields and get their value (7-5). In plain javascript we get the value of input fields like this (document.getElementById(‘username’).value, but this is not how we should access DOM elements in React; in other words, in React applications, we should never work with the document object because the whole point of React is to put an abstraction over DOM, so we don’t work with that directly and with this our applications will be easier to maintain and test. But if we really want to access a DOM element, in React there is a way for that. In order to get access to an element, we need to give it a reference (7-4).

Also, we learned how to have a single source of truth in forms by converting the input elements to controlled elements. And how to handle multiple inputs in the same handler of the onChange event.

1. Extract a reusable Input from the LoginForm component.
2. Add validation to our LoginForm component using joi. For simplicity, I took only the validation on form submission and omitted the validation on input change. In (7-14) we learned how to map an array items into properties of an object using the simplest way using for-of loop.
3. Review code of Login to decide what parts need to extract and put in a reusable component.

* We want every form to have errors property in the state object.
* schema is specific to Login.
* validate method: there is nothing about the concept of logining in it and it is a completely reusable piece of code. The only tricky part is the argument that we pass to the Joi.validate() method, here we’re referencing this.state.account but we aren’t going to have this account object in another form. Here we can come with a convention and assume that in all our forms a property called data that represents the data behind that form. Here the data should be set to account object, but in another form, data will be set to different kind of objects. With this convention, we can easily extract the validate() method and put it in a reusable component.
* handleSubmit method: this one is an interesting one, the first part of this method is reusable because every time we build a form in React, we want to disable the default behavior - we don’t want to post this form to the server. Also, every time the user clicks the submit button, we want to validate the data; and if there are any errors, we want to return - we don’t want to continue; but what happens after is dependent upon the kind of form we’re building.
* handleChange method: Also, a reusable piece of logic because every time the user is typing something in an input field, we want to update the state of the form.
* Render method: Obviously, render method is specific to each form. But we don’t need the render method in our new reusable Form component because it’s not supposed to render anything.

So, we will move the logic of validation, handling the change in input fields, and the first part of submitting a form to a new component called Form and make LoginForm extends this Form component. I omitted (7-19) to focus on big picture. Notes: (1) I didn’t need the state at all in the form module. (2)I added two lines at the beginning of handleChange() method to clear the error message when the user put back the cursor in the field and type anything. I also enabled the button.

1. Add RegisterForm component.
2. Build the MovieForm component (Exercise 7-22). This includes building Select component for the dropdown list.
3. Add searching feature to Movies. We have built SearchBox component which is identical to Input and simpler, but we didn’t use the Input because we want to have a separate search box whenever we need it. In SearchBox we sent (e.currentTarget.value) and we could sent nothing onChange={onChange} and deal with (e) in handleSearch() method but we wanted to simplify this method by getting the query string not the event argument of the input element.
4. Connect our app to its back-end which is vidly-api-node project. Everything is clear in section 8. In lectures 1-16, we learned how to talk to the back-end via the http-app example. From lecture 22 up to the end of the section we connected our vidly app to its back-end project. Everything was clear, so, there is no need to write down the details.
5. Implement authentication and authorization, all lectures are clear for register, login, and display the current user on NavBar, and so on.
6. Deploy the application to Heroku. All details are in deployment.docx file.

Steps in short:

* Build Movies component, only the table with the delete button and the paragraph at the top.
* Build Like component and rendering it inside Movies component.
* Add pagination feature by building Pagination component and rendering it inside Movies component. Implement paginate function in a reusable separate file.
* Add filtering feature by building ListGroup component and rendering it inside Movies component.
* Refactor Movies by extracting the table to MoviesTable component.
* Add sorting feature and refactor Movies and MoviesTable by moving the responsibility of sorting (determining the path and order type) to MoviesTable itself.
* Refactor MoviesTable by extracting the TableHeader component.
* Refactor MoviesTable again by extracting the TableBody component.
* Refactor MoviesTable again by extracting the Table component.
* Encapsulate the logic of filtering, sorting, and pagination in a separate method.
* Add navbar and routes to the project.
* Add the LoginForm component.
* Extract a reusable Input.
* Add validation to LoginForm.
* Extract the logic of validation, handling the change in input fields, and submitting of the form to a new reusable component called Form.
* Add the RegisterForm component.
* Build MovieForm as well as Select component.
* Add search feature to the MovieForm by building the SearchBox component.
* Connect our app to its back-end.
* Implement authentication and authorization.
* Deploy.

ملاحظات:

**أولاً.** هذه الخطوات بدأت ببناء الـ Movie component لأن الكورس تعليمي، لكن اذا أردتُ أن أعمل مشروع جديد مشابه فاني أبدأ بعمل تخطيط للموقع وصفحاته وما الذي أريده أن يظهر في كل صفحة، أحدد الـ header والـ footer وأحدد الـ navbar وقوائمه، والـ sidebar اذا كان مطلوباً وقوائمه، وأحدد الـ content area التي تكون في الـ main tag في الـ App component.

أول شيء أضع الـ routes في الـ App component. الـ components بدايةً ليس فيها سوى h1 element فيه اسم الـ component، وبذلك أضع الهيكل العام للموقع، ثم بعد ذلك أبدأ بالعمل على الـ components واحداً بعد الآخر.

**ثانياً.** في كل component أقوم ببناءه، أعمل refactoring له باستخلاص أي جزء ممكن أن يكون reusable component أو reusable logic.

أقوم بعمل أول component، بدايةً أوفر له الـ data التي يحتاجها من fake service ثم اذا اشتغل بصورة صحيحة أربطه بالـ backend وأوفر له البيانات من الـ server database. بعد ذلك بقية الـ components مباشرةً أبنيها وأعطيها البيانات من الـ database.

بالنسبة للـ authentication and authorization سوف أشرحه هنا ان شاء الله وما المطلوب عمله في الـ backend والـ frontend.

ثانياً. بالنسبة للفصل الخامس الذي يقوم فيه باستخلاص الـ TableHeader والـ TableBody ثم أخيراً يستخلص الـ Table ليكون reusable component متى احتجنا الى بناء جدول، أرى أنه لا ضير أن لا نعمل كل هذا ونكتفي بالـ MoviesTable كما هو بعد أن عملنا الـ sorting وجعلنا فيه مسؤولية تحديد الـ sortColumn object، فالـ MoviesTable يعتبر template جيد لأي جدول نريد بناءه، فقط في كل جدول سوف يتكرر لدينا الكود الخاص بتحديد الـ sortColumn. طبعاً، لا بد من الاستفادة من هذه الـ techniques التي يستخدمها في عمل الـ refactoring عند استخلاص هذه الأجزاء.

**ثالثاً.** يمكن تبسيط بعض الأكواد، مثل:

في الـ movies.jsx، يمكن كتابة السطر التالي:

const sorted = \_.orderBy(filtered, [sortColumn.path], [sortColumn.order]);

بدون استخدام الـ arrays، هكذا:

const sorted = \_.orderBy(filtered, sortColumn.path, sortColumn.order);

في الـ paginate.js، يمكن استخدام دالة slice الموجودة أصلاً للـ arrays دون الحاجة الى الـ lodash، هكذا:

return items.slice(startIndex, startIndex + pageSize);

**رابعاً.** توضيح فيما يخص فيديو 9-12 واستخدام window.location = “/” بدلاً من عمل history.push(‘/’) في الـ loginForm من أجل الحصول على full reload للـ home page الذي هو الـ App component حتى تظهر التغييرات في الـ NavBar، السبب في ذلك هو أن حصول الـ re-render للـ App component سوف لن يحصل إلا إذا تغيّرت الـ state بعبارة this.setState({user}) من خلال event handler أو دالة غير الـ ComponentDidMount() فالـ ComponentDidMount لا تُستدعى إلا مرة واحدة خلال دورة حياة التطبيق called only once during the lifecycle of our application.

**خامساً.** فيديو 9-16 لم تحصل عندي مشكلة الـ bi-directional dependencies لأنني لم أجعل الـ axios methods مجموعة في الـ httpService module لذلك لم أعمل import لهذا الـ module في الـ authService module. أي أن الاعتمادية باتجاه واحد وهو أن الـ httpService يعتمد على authService والعكس غير موجود.

**سادساً.** وهذه ملاحظة مهمة، الـ configurations على الـ axios التي عزلناها في module واحد الذي هو httpService تعمل حين عمل import له في أي module فيه http requests مثل genreService و movieService، ورغم أنها تظهر grayed في أعلى الفايل إلا أنها تعمل ووجودها ضروري، فبدونها سيكون أي request خالي من الـ token في جزء الـ request header ، فأي route في الـ backend يحتاج الى authentication فسوف لن يعمل ونحصل على 401 response أي unauthorized.

الـ configurations التي عملناها هي اثنان: الأول لتضمين الـ token في كل الـ request headers، والثاني هو لتشغيل الـ interceptor على الـ response في حالة الـ error حيث استخدمناه للـ unexpected errors.

**سابعاً.** فيديو 9-18، في الحقيقة ليس هناك حاجة في الـ Route أن أستبدل الـ component attribute بالـ render attribute من أجل أن أحقن الـ current user مع الـ props لتمريره الى الـ Movies component لأنه ببساطة في أي component أستطيع أن أحصل على الـ current user باستدعاء دالة getCurrentUser() في الـ authService module. وقد نفذتُ طريقتي. لكن طبعاً يجب معرفة هذه الطريقة عند الحاجة لحقن attribute مع الـ props الى component وهذا موضَّح في الفصل السادس فيديو 6-7.

**ثامناً.** فيديو 9-20، أجد أن عملية استخلاص الـ protected route معقدة بعض الشيء ولا أجد ضيراً في تكرار كود الحماية في كل Route في الـ App.js.

**تاسعاً.** للسهولة تركتُ تطبيق فيديو 9-21، فهو ليس مهم جداً، فلا بأس أن يعود المستخدم الى الصفحة الرئيسية بعد أن تم تحويله الى الـ login page أي أن لا يعود من المكان الذي أتى منه. وكذلك لا بأس أن يتمكن المستخدم الـ logged in من الوصول الى الـ login page بكتابة العنوان في شريط العنوان.

**عاشراً.** في vidly استوردنا الـ bootstrap css library من أجل الـ classes التي نحتاجها للـ html elements للحصول على الـ modern look and feel الذي تعطيه مكتبة bootstrap، وكذلك نسخنا أكواد لبعض الـ bootstrap components مثل الـ ListGroup والـ NavBar لكن الـ actions التي تتضمنها هذه الـ components مثل زر الـ collapse في الـ NavBar على سبيل المثال سوف لن تعمل بالتأكيد لأنها تعتمد على مكتبة الـ jQuery التي يجب أن تُستورد داخل المشروع لأجل أن تعمل هذه الـ actions. (ملاحظة: bootstrap 5 الآن لم يعد يعتمد على jquery بل على popper فقط).

لكن هذا لا يصح في مشروع react لأن الـ jQuery يتعامل مع الـ DOM بصورة مباشرة وهذا بالطبع يتعارض مع طريقة عمل react لذلك لابد من استخدام third party library توفر الـ bootstrap components لاغيةً الاعتمادية على الـ jquery مثل react-bootstrap و reactstrap. وهذا الرابط فيه شرح وافي عن هذا الأمر

<https://www.creative-tim.com/blog/web-design/how-to-use-bootstrap-with-reactjs/>

وللتمرين على الـ reactstrap فقد قمتُ باستخدامه بكل بساطة في المشروع الذي إسمه vidly-till end of section 7 with reactstarp فشغلتُ بعض الـ components ذات الـ actions المهم الحصول عليها مثل الـ Modal والـ NavBar. طبعاً هناك بعض التغييرات البسيطة التي يجب أن تعملها عند استيراد reactstrap component، وأدناه بعض التفاصيل:

مثلاً، عندما استخدمت الـ Modal فقد نسختُ كود الـ component من موقع reactstrap وهو <https://reactstrap.github.io/components> وجعلتُه في jsx module أسميته confirmDeletModal ثم استوردتُه أي الـ default object له وهو ConfirmModal في الـ movies module الحاوي على الـ delete handler، ثم بعد ذلك جعلت الـ handlers الخاصة بزرّي الـ yes والـ no في الـ movies وليس كما كان موجود عند النسخ في الـ Modal component نفسه وذلك لأن الـ component استخدمتُه كـ controlled component أي أن ألـ data تأتيه من الـ parent عبر الـ props والـ handlers تكون في الـ parent وهو فقط يُشعِر الـ parent بتغيّر الـ data عن طريق raising the event.

المثال الآخر هو الـ NavBar component الذي استخدمتُه من أجل تشغيل زر الـ collapse عند تصغير الشاشة فعند الضغط عليه تظهر القائمة عمودياً، والتغيير كان قليلاً جداً وهو استبدال الـ NavLink الخاص بالـ reactstrap NavBar component بالـ NavLink الخاص react-router-dom.

المثال الآخر هو الـ reactstrap Alert component حيث استخدمتُه لأتمكن من إضافة قابلية الـ autoClose حيث قمت ببرمجة زر Show Alert بحيث يجعل isOpen = true ثم مباشرة يجعلها fasle لكن بعد مرور 2000ms عن طريق دالة setTimeout().

تعلمتُ استخدام الـ Alert لأني أفضلُه على الـ toast.

الـ reactstrap جميل جداً وسهل

**حادي عشر.** في صفحة الـ Rentals استخدمتُ الـ Typeahead component من مكتبة react-bootstrap-typeahead وتفاصيل استخدام هذا الـ component والـ APIs الخاصة به موجودة طبعاً في الـ documentation الخاص به <https://www.npmjs.com/package/react-bootstrap-typeahead>

ملاحظات عنه:

* هذا الـ package فيه الكثير من التفاصيل، لكننا نتعلم ما نحتاجه لإنجاز العمل وهو جلب اسم بسرعة عن طريق الكيبورد.
* الـ rentals.jsx module استخدمتُ فيه الـ functional component (كورس الـ react الفصل الأخير) لغرض التنويع والتعلّم وكودها أقل، لكن استخدام الـ class قد يُعطيك مرونة أكثر.
* هذا الـ component يمكن أن يكون controlled أو uncontrolled، طبعاً في تطبيقاتنا مثل جلب اسم منتج أو زبون بسرعة عن طريق كتابة حرف أو أكثر فإننا نستخدمه كـ uncontrolled لأننا لا نريد أن يظهر الاختيار مسبقاً بقيمة تأتي عن طريق الـ props من الـ parent وإنما نحن الذين نختار المنتج او الزبون في الـ form مثل الـ rentals أو new order form.
* الـ options prop لهذا الـ component ممكن أن تكون مصفوفة أسماء أو مصفوفة objects وليس غير ذلك، إذا كانت objects فيجب تحديد الـ labelKey prop ليعرف الـ property التي يعرضها في القائمة.
* الـ id prop نحتاجها اذا كان لدينا أكثر من Typeahead component في نفس الصفحة.
* الـ minLength اذا لم تضعها فان القائمة تُفتح كاملةً بمجرد أخذ الـ component للـ focus وتعمل على حرف واحد فأكثر.
* إنتبه الى طريقة التعامل مع مصفوفة objects، فان الـ selected يكون [{}] وليس {}، وقد وضحتُ ذلك عن طريق عبارات console.log() في الـ second typeahead.
* كما يوجد الـ multi selection وهو رائع جداً، أنظر هذا المثال <https://ericgio.github.io/react-bootstrap-typeahead>
* البحث غالباً في تطبيقاتنا يكون على العميل بجلب قائمة المنتجات أو الزبائن، لكن إذا كانت القائمة هائلة أو كبيرة جداً فهنا يكون الـ search على الـ server أفضل وليس على جانب العميل فهنا لدينا الـ AsyncTypeahead، أنظر هذا المثال الرائع في مقطع Asynchronous searching حيث يجلب إسم user مع صورته من الـ github ومعلوم هي تحوي ملايين الـ users <https://ericgio.github.io/react-bootstrap-typeahead/#asynchronous-searching>

**ثاني عشر.** عملتُ reusable tooltip باستخدام الـ Tooltip component من مكتبة reactstrap. سهلة جداً.

ثالث عشر. إضافة فيديو من موقع غير الـ youtube يكون باستخدام الـ video tag أما من الـ youtube فيكون باستخدام الـ iframe ويكون ذلك بنسخ الـ embedded code من صفحة الفيديو بعمل share وأخذه ولصقه في صفحة الـ html الخاصة بنا أو داخل أي component في jsx module في تطبيق react.

**رابع عشر.** إعطاء قابلية الـ download لملف عملية سهلة جداً وتكون فقط بإضافة الـ download attribute الى الـ anchor tag وفي الـ href نحدد المسار للملف. بالنسبة للـ react فإنه يجب وضع كل الـ asset files -مثل هذه التي نريدها أن تكون قابلة للتحميل- في الـ public folder. في حالة عدم وضع الـ download attribute فان الملف يُفتح في المتصفح في تبويب جديد.

**خامس عشر.** بفضل الله أضفتُ قابلية إرسال email، مثلاً االموقع يرسل رسالة الى الزبون بتفاصيل أمر الشراء الذي تم عبر الموقع، كما هو الحال في موقع موش حيث تُرسل الى المشترك الذي انخرط في أحد الكورسات رسالة بتفاصيل الـ order، وهكذا. أو في صفحة الـ Contact us يكون هناك form لرسالة يبعث بها الزائر أو المشترك برسالة للموقع. هذا رابط يشرح <https://medium.com/javascript-in-plain-english/email-from-your-react-project-w-emailjs-d6af68fa5c17>

وهذا فيديو يشرح نفس الموضوع باستخدام نفس الـ service وهي EmailJs <https://www.youtube.com/watch?v=NgWGllOjkbs>

والعملية سهلة جداً جداً.. عملتُ بفضل الله contact us form وفيه يرسل المشترك أو الزائر رسالة الى الموقع (الخدمة هي EmailJs وهي مسجلة ببريدي أي أن بريدي سيكون بريد vidly). فقط هناك مشكلة صغيرة في حساب الـ Personal على EmailJs هو أن بريد المرسل حين يظهر لك عند استلام الرسالة يكون نفس بريد المستلم وهذا شيء متعمّد منهم مذكور عندما تقف على حقل From Email في الـ template ويمكن تجاوز هذه الحالة بوضع قيمة {{user\_email}} في حقل الـ Reply To حتى نتمكن من إجابة المرسل.

ملاحظات عن هذه الخطوة:

* الخطوات سهلة، تعمل حساب على EmailJs، تختار service ولتكن Gmail، تُنشيء New Service و New Template، ستحتاج من هذه الservice التي أنشأتها الى ثلاث قيم هي الـ service id والـ template id والـ user id لتضعها في دالة sendEmail في تطبيقك.
* في التطبيق، إعمل install لـ emailjs-com package، إعمل صفحة الـ contact us مثلاً وفيها form يحتوي على input elements من أجل الرسالة (أسماء الـ input elements مهمة لأنها مرتبطة بالـ template).
* صممتُ form يرث الـ reusable form الموجود عندي لجعل الكود more clean and descriptive.
* الـ input elements في الـ form هي uncontrolled وليست controlled فقمتُ بعمل reusable uncontrolled input لأجل ذلك.
* وضعت القيم الثلاث في ملف config.json لتجنب الـ hard coding لها في الـ form.
* الـ template لا تحتاج أن يكون أكثر من هذا:

Graphical user interface, application

Description automatically generated