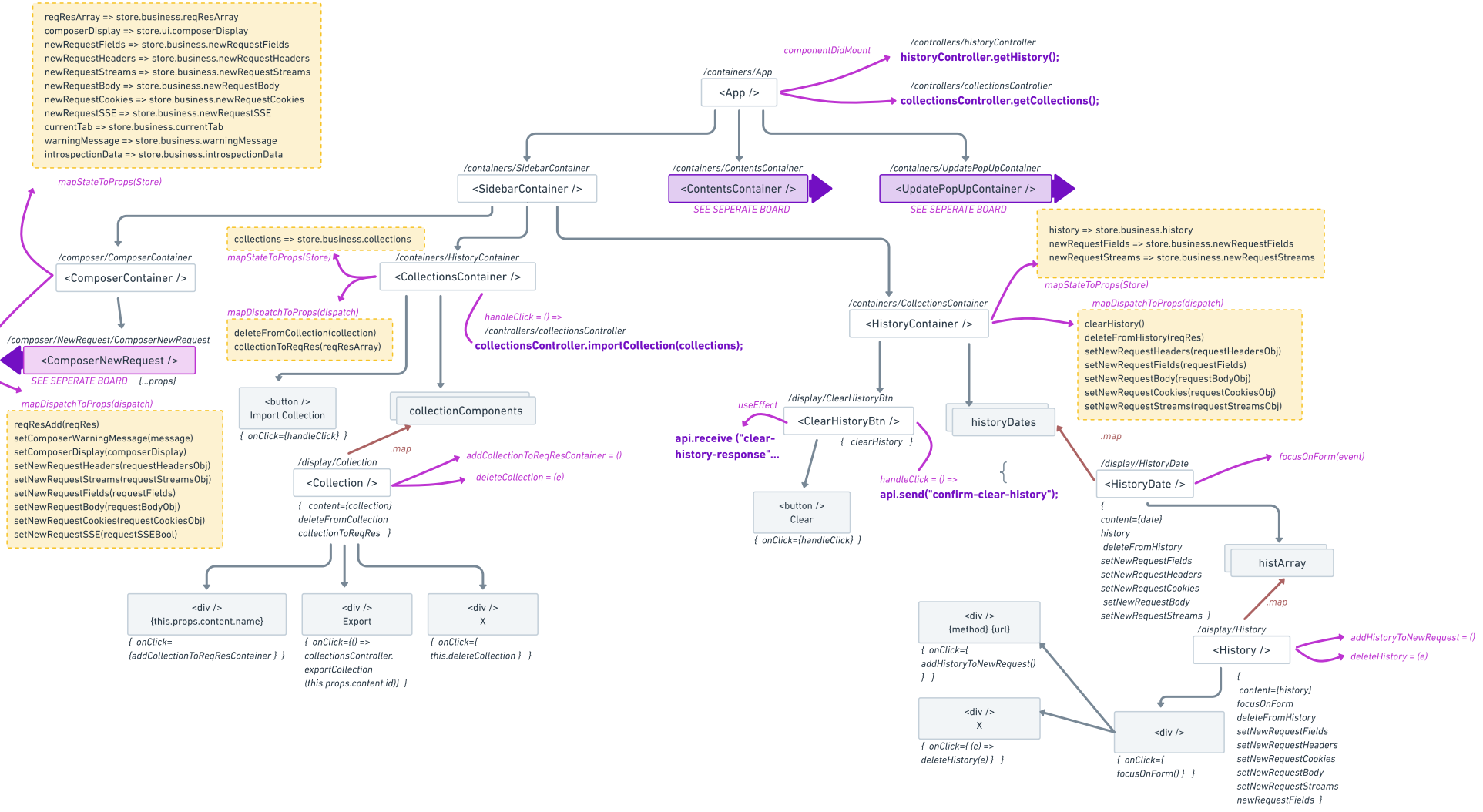

reqResArray => store.business.reqResArray
composerDisplay => store.ui.composerDisplay
newRequestFields => store.business.newRequestFields
newRequestHeaders => store.business.newRequestHeaders
newRequestStreams => store.business.newRequestStreams
newRequestBody => store.business.newRequestBody
newRequestCookies => store.business.newRequestCookies
newRequestSSE => store.business.newRequestSSE
currentTab => store.business.currentTab
warningMessage => store.business.warningMessage
introspectionData => store.business.introspectionData
mapStateToProps(Store)
componentDidMount
/controllers/historyController
historyController.getHistory();
/controllers/collectionsController
collectionsController.getCollections();
/containers/App
<App />
/containers/SidebarContainer
<SidebarContainer />
/containers/ContentsContainer
<ContentsContainer />
SEE SEPERATE BOARD
/containers/UpdatePopUpContainer
<UpdatePopUpContainer />
SEE SEPERATE BOARD
/composer/ComposerContainer
<ComposerContainer />
/composer/NewRequest/ComposerNewRequest
<ComposerNewRequest />
SEE SEPERATE BOARD
{...props}
mapDispatchToProps(dispatch)
reqResAdd(reqRes)
setComposerWarningMessage(message)
setComposerDisplay(composerDisplay)
setNewRequestHeaders(requestHeadersObj)
setNewRequestStreams(requestStreamsObj)
setNewRequestFields(requestFields)
setNewRequestBody(requestBodyObj)
setNewRequestCookies(requestCookiesObj)
setNewRequestSSE(requestSSEBool)
collections => store.business.collections
mapStateToProps(Store)
/containers/HistoryContainer
<CollectionsContainer />
mapDispatchToProps(dispatch)
deleteFromCollection(collection)
collectionToReqRes(reqResArray)
handleClick = () =>
/controllers/collectionsController
collectionsController.importCollection(collections);
<button />
Import Collection
{ onClick={handleClick} }
collectionComponents
.map
/display/Collection
<Collection />
addCollectionToReqResContainer = ()
deleteCollection = (e)
{ content={collection}
deleteFromCollection
collectionToReqRes }

{this.props.content.name}
{ onClick=
{addCollectionToReqResContainer } }

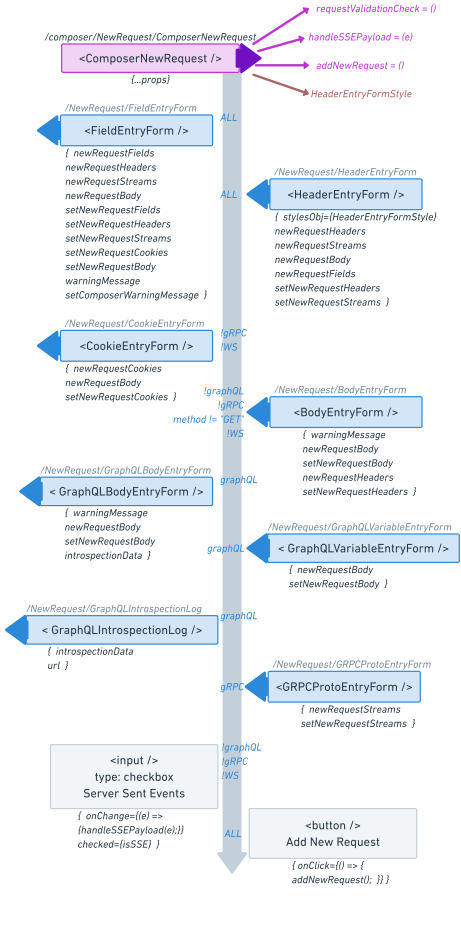
Export
{ onClick={() =>
collectionsController.
exportCollection
(this.props.content.id)} }

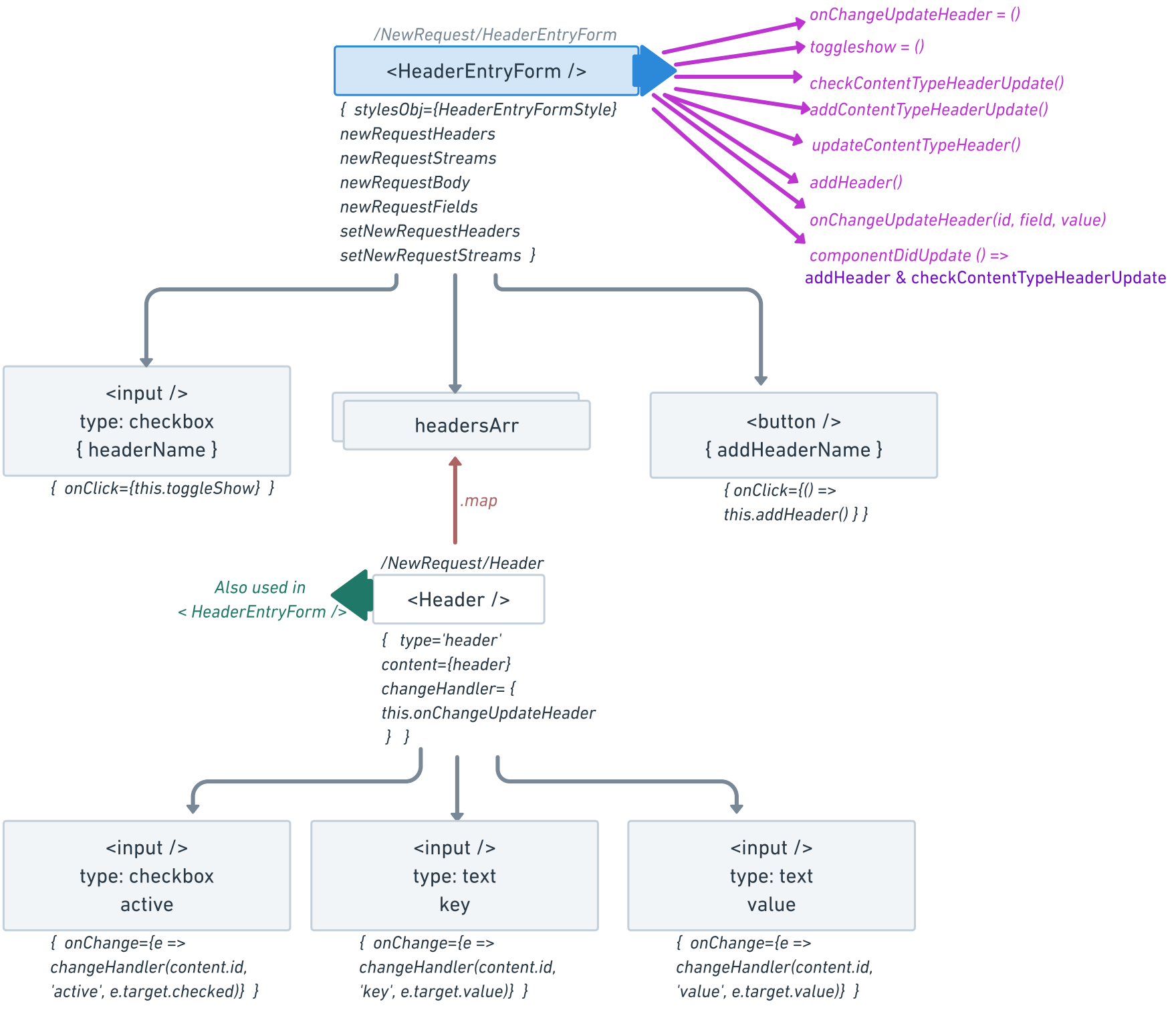
X
{ onClick={
this.deleteCollection } }
history => store.business.history
newRequestFields => store.business.newRequestFields
newRequestStreams => store.business.newRequestStreams
mapStateToProps(Store)
/containers/CollectionsContainer
<HistoryContainer />
mapDispatchToProps(dispatch)
clearHistory()
deleteFromHistory(reqRes)
setNewRequestHeaders(requestHeadersObj)
setNewRequestFields(requestFields)
setNewRequestBody(requestBodyObj)
setNewRequestCookies(requestCookiesObj)
setNewRequestStreams(requestStreamsObj)
useEffect
/display/ClearHistoryBtn
<ClearHistoryBtn />
{ clearHistory }
api.receive ("clear-
history-response"...
handleClick = () =>
api.send("confirm-clear-history");
<button />
Clear
{ onClick={handleClick} }
historyDates
.map
/display/HistoryDate
<HistoryDate />
focusOnForm(event)
{
content={date}
history
deleteFromHistory
setNewRequestFields
setNewRequestHeaders
setNewRequestCookies
setNewRequestBody
setNewRequestStreams }
histArray
.map
/display/History
<History />
addHistoryToNewRequest = ()
deleteHistory = (e)
{
content={history}
focusOnForm
deleteFromHistory
setNewRequestFields
setNewRequestHeaders
setNewRequestCookies
setNewRequestBody
setNewRequestStreams
newRequestFields }

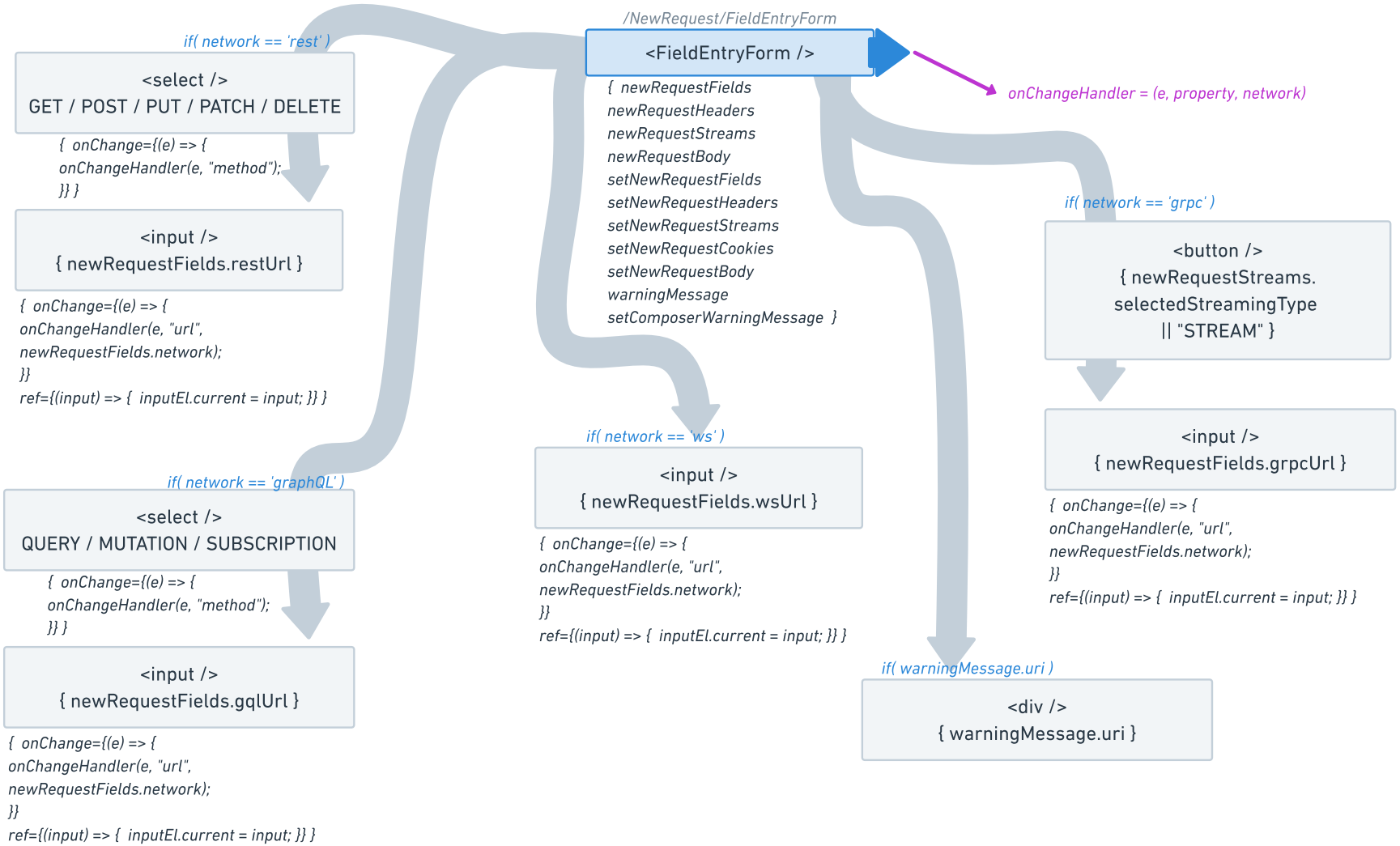
{method} {url}
{ onClick={
addHistoryToNewRequest()
} }

X
{ onClick={ (e) =>
deleteHistory(e) } }

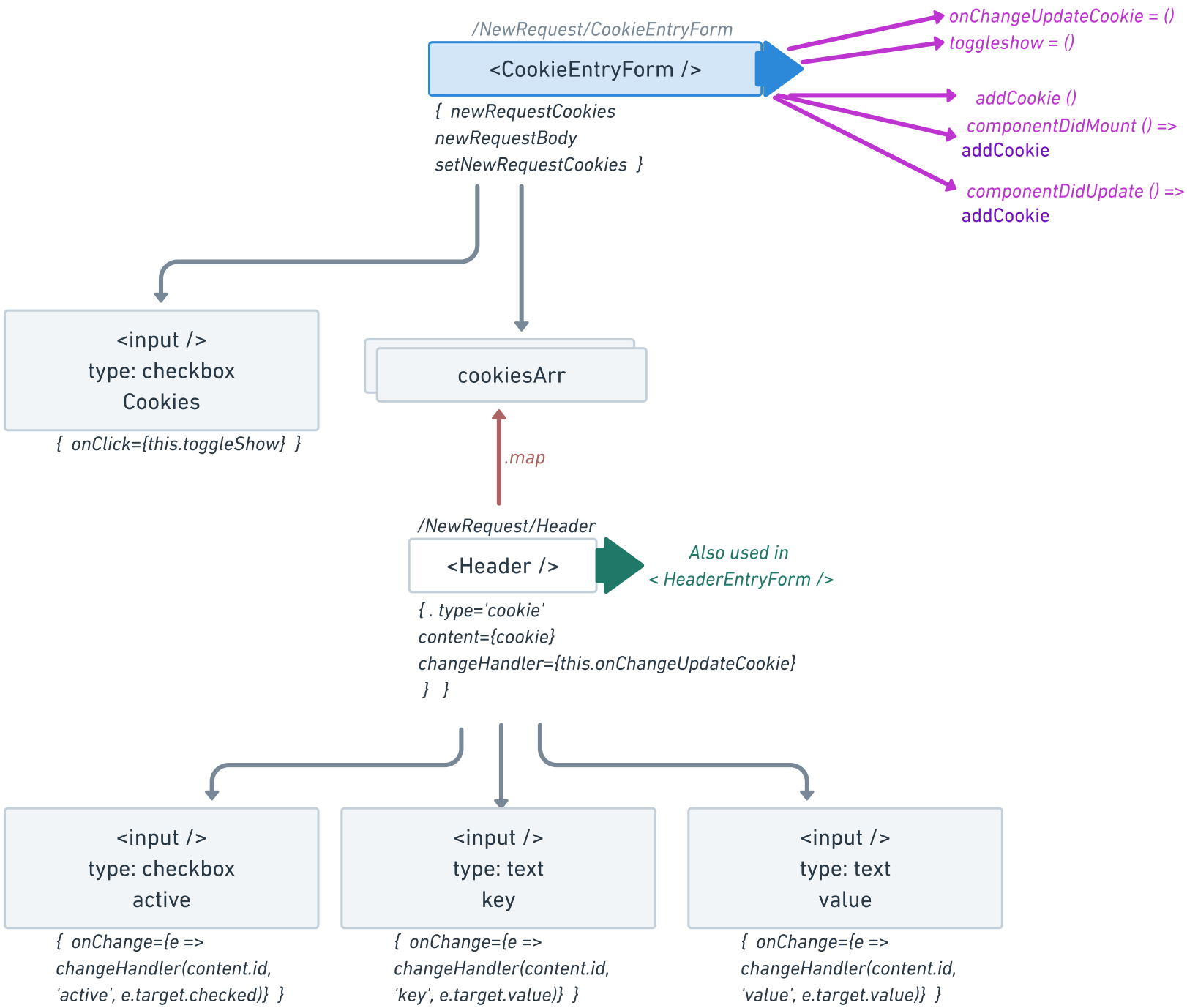
{ onClick={
focusOnForm() } }

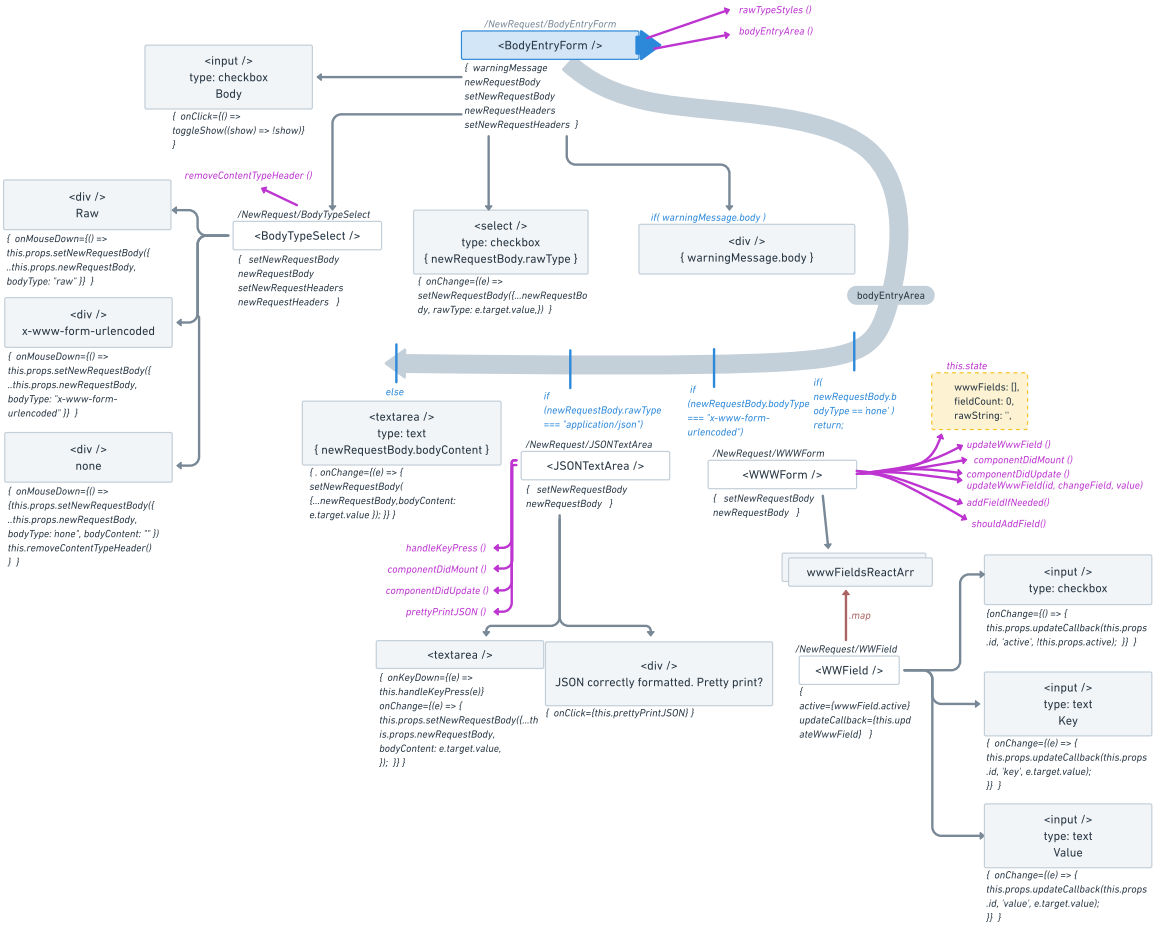

requestValidationCheck = ()
handleSSEPayload = (e)
addNewRequest = ()
HeaderEntryFormStyle
/composer/NewRequest/ComposerNewRequest
<ComposerNewRequest />
{...props}
/NewRequest/FieldEntryForm
ALL
<FieldEntryForm />
{ newRequestFields
newRequestHeaders
newRequestStreams
newRequestBody
setNewRequestFields
setNewRequestHeaders
setNewRequestStreams
setNewRequestCookies
setNewRequestBody
warningMessage
setComposerWarningMessage }
/NewRequest/HeaderEntryForm
ALL
<HeaderEntryForm />
{ stylesObj={HeaderEntryFormStyle}
newRequestHeaders
newRequestStreams
newRequestBody
newRequestFields
setNewRequestHeaders
setNewRequestStreams }
/NewRequest/CookieEntryForm
!gRPC
!WS
<CookieEntryForm />
{ newRequestCookies
newRequestBody
setNewRequestCookies }
!graphQL
!gRPC
method != "GET"
!WS
/NewRequest/BodyEntryForm
<BodyEntryForm />
{ warningMessage
newRequestBody
setNewRequestBody
newRequestHeaders
setNewRequestHeaders }
/NewRequest/GraphQLBodyEntryForm
graphQL
< GraphQLBodyEntryForm />
{ warningMessage
newRequestBody
setNewRequestBody
introspectionData }
/NewRequest/GraphQLVariableEntryForm
graphQL
< GraphQLVariableEntryForm />
{ newRequestBody
setNewRequestBody }
/NewRequest/GraphQLIntrospectionLog
graphQL
< GraphQLIntrospectionLog />
{ introspectionData
url }
/NewRequest/GRPCProtoEntryForm
gRPC
<GRPCProtoEntryForm />
{ newRequestStreams
setNewRequestStreams }
!graphQL
!gRPC
!WS
<input />
type: checkbox
Server Sent Events
{ onChange={(e) =>
{handleSSEPayload(e);}}
checked={isSSE} }
ALL
<button />
Add New Request
{ onClick={() => {
addNewRequest(); }} }


/NewRequest/HeaderEntryForm
<HeaderEntryForm />
{ stylesObj={HeaderEntryFormStyle}
newRequestHeaders
newRequestStreams
newRequestBody
newRequestFields
setNewRequestHeaders
setNewRequestStreams }
onChangeUpdateHeader = ()
toggleshow = ()
checkContentTypeHeaderUpdate()
addContentTypeHeaderUpdate()
updateContentTypeHeader()
addHeader()
onChangeUpdateHeader(id, field, value)
componentDidUpdate () =>
addHeader & checkContentTypeHeaderUpdate
<input />
type: checkbox
{ headerName }
{ onClick={this.toggleShow} }
headersArr
.map
<button />
{ addHeaderName }
{ onClick={() =>
this.addHeader() } }
/NewRequest/Header
Also used in
< HeaderEntryForm />
<Header />
{ type='header'
content={header}
changeHandler= {
this.onChangeUpdateHeader
} }
<input />
type: checkbox
active
{ onChange={e =>
changeHandler(content.id,
'active', e.target.checked)} }
<input />
type: text
key
{ onChange={e =>
changeHandler(content.id,
'key', e.target.value)} }
<input />
type: text
value
{ onChange={e =>
changeHandler(content.id,
'value', e.target.value)} }


/NewRequest/FieldEntryForm
<FieldEntryForm />
{ newRequestFields
newRequestHeaders
newRequestStreams
newRequestBody
setNewRequestFields
setNewRequestHeaders
setNewRequestStreams
setNewRequestCookies
setNewRequestBody
warningMessage
setComposerWarningMessage }
onChangeHandler = (e, property, network)
if(network == 'rest')
<select />
GET / POST / PUT / PATCH / DELETE
{ onChange={(e) => {
onChangeHandler(e, "method");
}} }
<input />
{ newRequestFields.restUrl }
{ onChange={(e) => {
onChangeHandler(e, "url",
newRequestFields.network);
}}
ref={(input) => { inputEl.current = input; }} }
if(network == 'graphQL')
<select />
QUERY / MUTATION / SUBSCRIPTION
{ onChange={(e) => {
onChangeHandler(e, "method");
}} }
<input />
{ newRequestFields.gqlUrl }
{ onChange={(e) => {
onChangeHandler(e, "url",
newRequestFields.network);
}}
ref={(input) => { inputEl.current = input; }} }
if(network == 'ws')
<input />
{ newRequestFields.wsUrl }
{ onChange={(e) => {
onChangeHandler(e, "url",
newRequestFields.network);
}}
ref={(input) => { inputEl.current = input; }} }
if(network == 'grpc')
<button />
{ newRequestStreams.
selectedStreamingType
|| "STREAM" }
<input />
{ newRequestFields.grpcUrl }
{ onChange={(e) => {
onChangeHandler(e, "url",
newRequestFields.network);
}}
ref={(input) => { inputEl.current = input; }} }
if(warningMessage.uri)

{ warningMessage.uri }


/NewRequest/CookieEntryForm
<CookieEntryForm />
{ newRequestCookies
newRequestBody
setNewRequestCookies }
onChangeUpdateCookie = ()
toggleshow = ()
addCookie ()
componentDidMount () =>
addCookie
componentDidUpdate () =>
addCookie
<input />
type: checkbox
Cookies
{ onClick={this.toggleShow} }
cookiesArr
.map
/NewRequest/Header
<Header />
Also used in
< HeaderEntryForm />
{ . type='cookie'
content={cookie}
changeHandler={this.onChangeUpdateCookie}
} }
<input />
type: checkbox
active
{ onChange={e =>
changeHandler(content.id,
'active', e.target.checked)} }
<input />
type: text
key
{ onChange={e =>
changeHandler(content.id,
'key', e.target.value)} }
<input />
type: text
value
{ onChange={e =>
changeHandler(content.id,
'value', e.target.value)} }


/NewRequest/BodyEntryForm
<BodyEntryForm />
{ warningMessage
newRequestBody
setNewRequestBody
newRequestHeaders
setNewRequestHeaders }
rawTypeStyles ()
bodyEntryArea ()
<input />
type: checkbox
Body
{ onClick={() =>
toggleShow!(show) => !show)}
}
removeContentTypeHeader ()

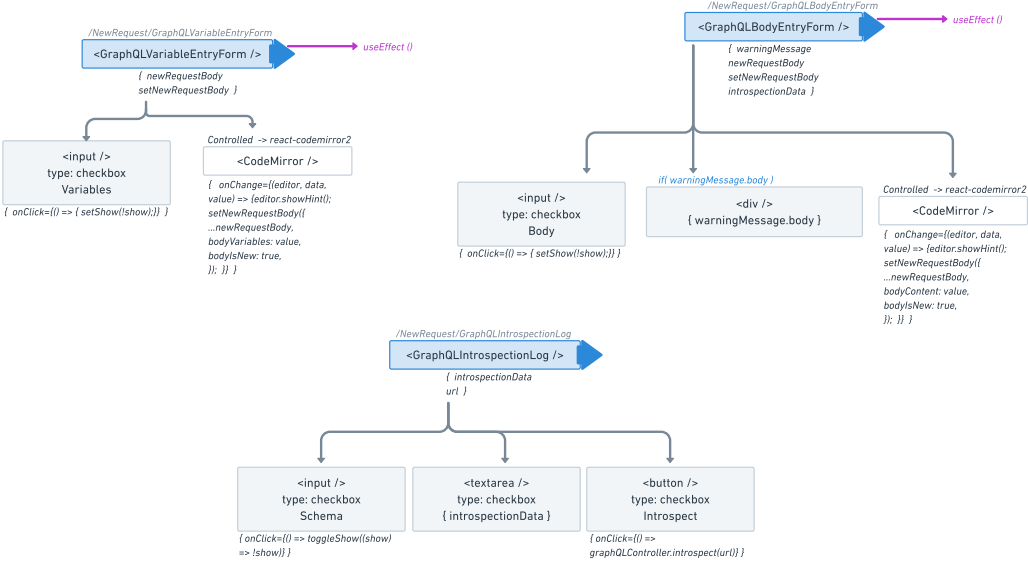
Raw
{ onMouseDown={() =>
this.props.setNewRequestBody({
..this.props.newRequestBody,
bodyType: "raw" }) }

x-www-form-urlencoded
{ onMouseDown={() =>
this.props.setNewRequestBody({
..this.props.newRequestBody,
bodyType: "x-www-form-
urlencoded" }) }

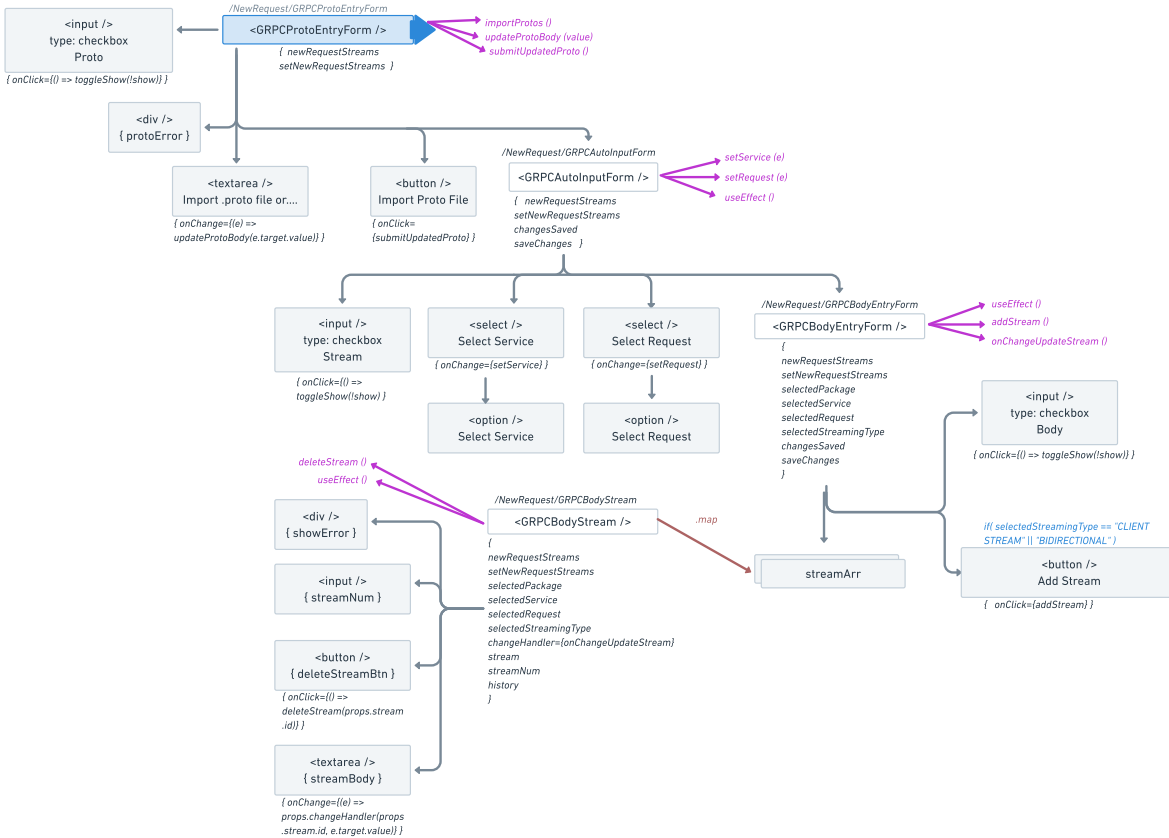
none
{ onMouseDown={() =>
{this.props.setNewRequestBody({
..this.props.newRequestBody,
bodyType: 'none', bodyContent: "" })
this.removeContentTypeHeader()
} }
/NewRequest/BodyTypeSelect
<BodyTypeSelect />
{ setNewRequestBody
newRequestBody
setNewRequestHeaders
newRequestHeaders }
<select />
type: checkbox
{ newRequestBody.rawType }
{ onChange={(e) =>
setNewRequestBody({...newRequestBo
dy, rawType: e.target.value,}) }
if(warningMessage.body)

{ warningMessage.body }
bodyEntryArea
else
<textarea />
type: text
{ newRequestBody.bodyContent }
{ . onChange={(e) => {
setNewRequestBody(
{...newRequestBody,bodyContent:
e.target.value }); }} }
if
(newRequestBody.rawType
=== "application/json")
/NewRequest/JSONTextArea
<JSONTextArea />
{ setNewRequestBody
newRequestBody }
handleKeyPress ()
componentDidMount ()
componentDidUpdate ()
prettyPrintJSON ()
<textarea />
{ onKeyDown={(e) =>
this.handleKeyPress(e)}
onChange={(e) => {
this.props.setNewRequestBody({...th
is.props.newRequestBody,
bodyContent: e.target.value,
}); }} }

JSON correctly formatted. Pretty print?
{ onClick={this.prettyPrintJSON} }
if
(newRequestBody.bodyType
=== "x-www-form-
urlencoded")
if(
newRequestBody.b
odyType == none')
return;
/NewRequest/WWWForm
<WWWForm />
{ setNewRequestBody
newRequestBody }
this.state
wwwFields: [],
fieldCount: 0,
rawString: '',
updateWwwField ()
componentDidMount ()
componentDidUpdate ()
updateWwwField(id, changeField, value)
addFieldIfNeeded()
shouldAddField()
wwwFieldsReactArr
.map
/NewRequest/WWField
<WWField />
{
active={wwwField.active}
updateCallback={this.upd
ateWwwField} }
<input />
type: checkbox
{onChange={() => {
this.props.updateCallback(this.props
.id, 'active', !this.props.active); }} }
<input />
type: text
Key
{ onChange={(e) => {
this.props.updateCallback(this.props
.id, 'key', e.target.value);
}} }
<input />
type: text
Value
{ onChange={(e) => {
this.props.updateCallback(this.props
.id, 'value', e.target.value);
}} }

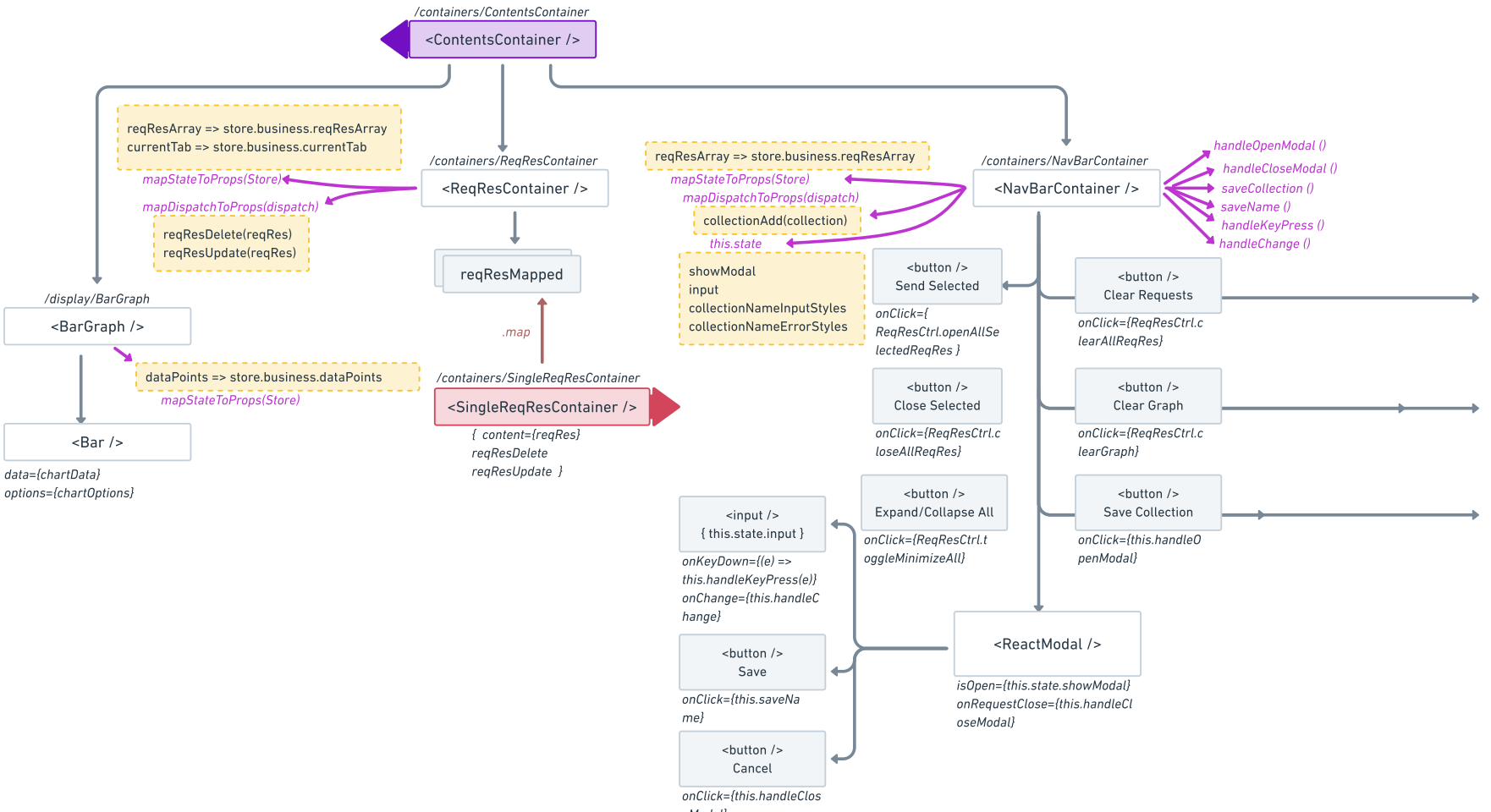

/NewRequest/GraphQLVariableEntryForm
<GraphQLVariableEntryForm />
useEffect ()
{ newRequestBody
setNewRequestBody }
<input />
type: checkbox
Variables
{ onClick={() => { setShow(!show);}} }
Controlled -> react-codemirror2
<CodeMirror />
{ onChange={(editor, data,
value) => {editor.showHint();
setNewRequestBody({
...newRequestBody,
bodyVariables: value,
bodyIsNew: true,
});} } }
/NewRequest/GraphQLBodyEntryForm
<GraphQLBodyEntryForm />
useEffect ()
{ warningMessage
newRequestBody
setNewRequestBody
introspectionData }
<input />
type: checkbox
Body
{ onClick={() => { setShow(!show);}} }
if(warningMessage.body)

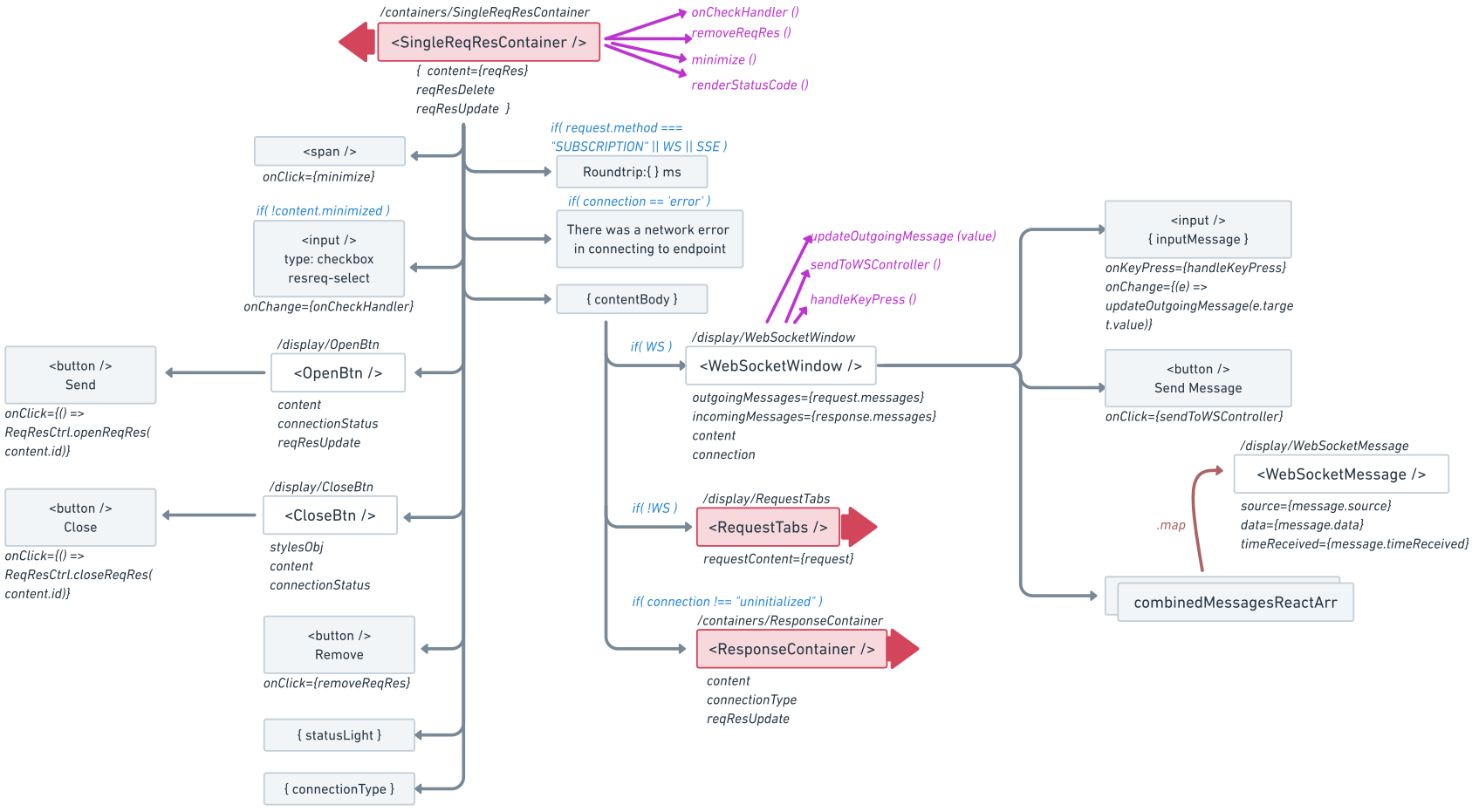
{ warningMessage.body }
Controlled -> react-codemirror2
<CodeMirror />
{ onChange={(editor, data,
value) => {editor.showHint();
setNewRequestBody({
...newRequestBody,
bodyContent: value,
bodyIsNew: true,
});} } }
/NewRequest/GraphQLIntrospectionLog
<GraphQLIntrospectionLog />
{ introspectionData
url }
<input />
type: checkbox
Schema
{ onClick={() => toggleShow((show)
=> !show)} }
<textarea />
type: checkbox
{ introspectionData }
<button />
type: checkbox
Introspect
{ onClick={() =>
graphQLController.introspect(url)} }


/NewRequest/GRPCProtoEntryForm
<GRPCProtoEntryForm />
{ newRequestStreams
setNewRequestStreams }
importProtos ()
updateProtoBody (value)
submitUpdatedProto ()
<input />
type: checkbox
Proto
{ onClick={() => toggleShow(!show)} }

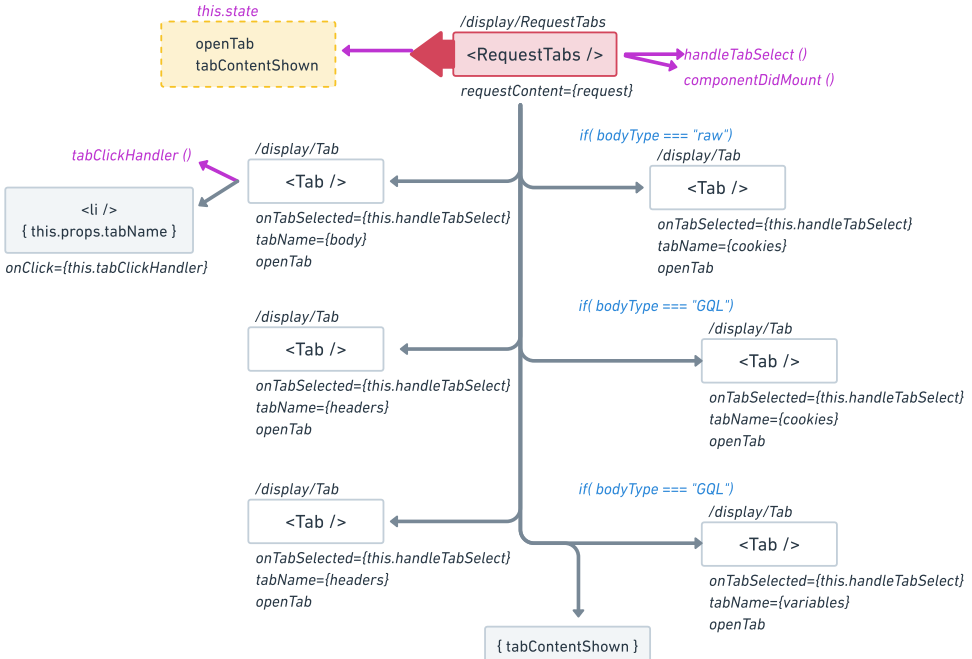
{ protoError }
<textarea />
Import .proto file or....
{ onChange={(e) =>
updateProtoBody(e.target.value)} }
<button />
Import Proto File
{ onClick=
{submitUpdatedProto} }
/NewRequest/GRPCAutoInputForm
<GRPCAutoInputForm />
{ newRequestStreams
setNewRequestStreams
changesSaved
saveChanges }
setService (e)
setRequest (e)
useEffect ()
<input />
type: checkbox
Stream
{ onClick={() =>
toggleShow(!show) }
<select />
Select Service
{ onChange={setService} }
<select />
Select Request
{ onChange={setRequest} }
<option />
Select Service
<option />
Select Request
/NewRequest/GRPCBodyEntryForm
<GRPCBodyEntryForm />
{
newRequestStreams
setNewRequestStreams
selectedPackage
selectedService
selectedRequest
selectedStreamingType
changesSaved
saveChanges
}
useEffect ()
addStream ()
onChangeUpdateStream ()
<input />
type: checkbox
Body
{ onClick={() => toggleShow(!show)} }
if(selectedStreamingType == "CLIENT
STREAM" || "BIDIRECTIONAL")
<button />
Add Stream
{ onClick={addStream} }
deleteStream ()
useEffect ()
/NewRequest/GRPCBodyStream
<GRPCBodyStream />
{
newRequestStreams
setNewRequestStreams
selectedPackage
selectedService
selectedRequest
selectedStreamingType
changeHandler={onChangeUpdateStream}
stream
streamNum
history
}
.map
streamArr

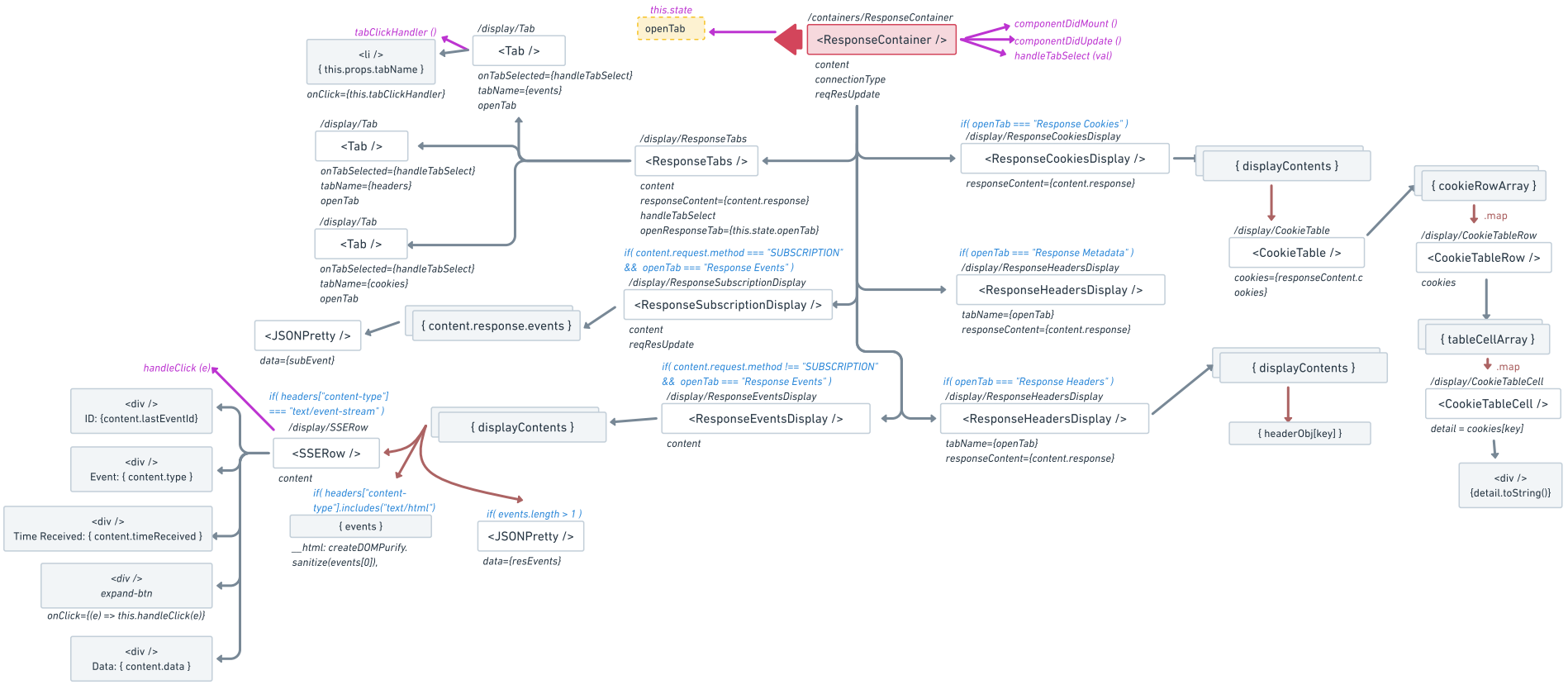
{ showError }
<input />
{ streamNum }
<button />
{ deleteStreamBtn }
{ onClick={() =>
deleteStream(props.stream
.id)} }
<textarea />
{ streamBody }
{ onChange={(e) =>
props.changeHandler(props
.stream.id, e.target.value)} }


/containers/ContentsContainer
<ContentsContainer />
reqResArray => store.business.reqResArray
currentTab => store.business.currentTab
mapStateToProps(Store)
mapDispatchToProps(dispatch)
reqResDelete(reqRes)
reqResUpdate(reqRes)
/containers/ReqResContainer
<ReqResContainer />
reqResMapped
.map
/containers/SingleReqResContainer
<SingleReqResContainer />
{ content={reqRes}
reqResDelete
reqResUpdate }
/display/BarGraph
<BarGraph />
dataPoints => store.business.dataPoints
mapStateToProps(Store)
<Bar />
data={chartData}
options={chartOptions}
reqResArray => store.business.reqResArray
mapStateToProps(Store)
mapDispatchToProps(dispatch)
collectionAdd(collection)
this.state
showModal
input
collectionNameInputStyles
collectionNameErrorStyles
/containers/NavBarContainer
<NavBarContainer />
handleOpenModal ()
handleCloseModal ()
saveCollection ()
saveName ()
handleKeyPress ()
handleChange ()
<button />
Send Selected
onClick={
ReqResCtrl.openAllSe
lectedReqRes }
<button />
Clear Requests
onClick={ReqResCtrl.c
learAllReqRes}
<button />
Close Selected
onClick={ReqResCtrl.c
loseAllReqRes}
<button />
Clear Graph
onClick={ReqResCtrl.c
learGraph}
<button />
Expand/Collapse All
onClick={ReqResCtrl.t
oggleMinimizeAll}
<button />
Save Collection
onClick={this.handleO
penModal}
<input />
{ this.state.input }
onKeyDown={(e) =>
this.handleKeyPress(e)}
onChange={this.handleC
hange}
<button />
Save
onClick={this.saveNa
me}
<button />
Cancel
onClick={this.handleClos
eModal}
<ReactModal />
isOpen={this.state.showModal}
onRequestClose={this.handleCl
oseModal}


/containers/SingleReqResContainer
<SingleReqResContainer />
{ content={reqRes}
reqResDelete
reqResUpdate }
onCheckHandler ()
removeReqRes ()
minimize ()
renderStatusCode ()

onClick={minimize}
if(request.method ===
"SUBSCRIPTION" || WS || SSE)
Roundtrip:{ } ms
if(!content.minimized)
<input />
type: checkbox
resreq-select
onChange={onCheckHandler}
if(connection == 'error')
There was a network error
in connecting to endpoint
{ contentBody }
updateOutgoingMessage (value)
sendToWSController ()
handleKeyPress ()
/display/OpenBtn
<OpenBtn />
content
connectionStatus
reqResUpdate
<button />
Send
onClick={() =>
ReqResCtrl.openReqRes(
content.id)}
if(WS)
/display/WebSocketWindow
<WebSocketWindow />
outgoingMessages={request.messages}
incomingMessages={response.messages}
content
connection
<input />
{ inputMessage }
onKeyPress={handleKeyPress}
onChange={(e) =>
updateOutgoingMessage(e.targe
t.value)}
<button />
Send Message
onClick={sendToWSController}
/display/WebSocketMessage
<WebSocketMessage />
source={message.source}
data={message.data}
timeReceived={message.timeReceived}
.map
combinedMessagesReactArr
/display/CloseBtn
<CloseBtn />
stylesObj
content
connectionStatus
<button />
Close
onClick={() =>
ReqResCtrl.closeReqRes(
content.id)}
if(!WS)
/display/RequestTabs
<RequestTabs />
requestContent={request}
<button />
Remove
onClick={removeReqRes}
if(connection !== "uninitialized")
/containers/ResponseContainer
<ResponseContainer />
content
connectionType
reqResUpdate
{ statusLight }
{ connectionType }


this.state
openTab
tabContentShown
/display/RequestTabs
<RequestTabs />
handleTabSelect ()
componentDidMount ()
requestContent={request}
tabClickHandler ()
<li />
{ this.props.tabName }
onClick={this.tabClickHandler}
/display/Tab
<Tab />
onTabSelected={this.handleTabSelect}
tabName={body}
openTab
if(bodyType === "raw")
/display/Tab
<Tab />
onTabSelected={this.handleTabSelect}
tabName={cookies}
openTab
/display/Tab
<Tab />
onTabSelected={this.handleTabSelect}
tabName={headers}
openTab
if(bodyType === "GQL")
/display/Tab
<Tab />
onTabSelected={this.handleTabSelect}
tabName={cookies}
openTab
/display/Tab
<Tab />
onTabSelected={this.handleTabSelect}
tabName={headers}
openTab
if(bodyType === "GQL")
/display/Tab
<Tab />
onTabSelected={this.handleTabSelect}
tabName={variables}
openTab
{ tabContentShown }


this.state
openTab
/containers/ResponseContainer
<ResponseContainer />
content
connectionType
reqResUpdate
componentDidMount ()
componentDidUpdate ()
handleTabSelect (val)
tabClickHandler ()
/display/Tab
<Tab />
onTabSelected={handleTabSelect}
tabName={events}
openTab
<li />
{ this.props.tabName }
onClick={this.tabClickHandler}
/display/Tab
<Tab />
onTabSelected={handleTabSelect}
tabName={headers}
openTab
/display/Tab
<Tab />
onTabSelected={handleTabSelect}
tabName={cookies}
openTab
/display/ResponseTabs
<ResponseTabs />
content
responseContent={content.response}
handleTabSelect
openResponseTab={this.state.openTab}
if(openTab === "Response Cookies")
/display/ResponseCookiesDisplay
<ResponseCookiesDisplay />
responseContent={content.response}
{ displayContents }
/display/CookieTable
<CookieTable />
cookies={responseContent.cookies}
{ cookieRowArray }
.map
/display/CookieTableRow
<CookieTableRow />
cookies
{ tableCellArray }
.map
/display/CookieTableCell
<CookieTableCell />
detail = cookies[key]

{detail.toString()}
if(content.request.method === "SUBSCRIPTION"
&& openTab === "Response Events")
/display/ResponseSubscriptionDisplay
<ResponseSubscriptionDisplay />
content
reqResUpdate
{ content.response.events }
<JSONPretty />
data={subEvent}
if(openTab === "Response Metadata")
/display/ResponseHeadersDisplay
<ResponseHeadersDisplay />
tabName={openTab}
responseContent={content.response}
if(content.request.method !== "SUBSCRIPTION"
&& openTab === "Response Events")
/display/ResponseEventsDisplay
<ResponseEventsDisplay />
content
{ displayContents }
if(openTab === "Response Headers")
/display/ResponseHeadersDisplay
<ResponseHeadersDisplay />
tabName={openTab}
responseContent={content.response}
{ displayContents }
{ headerObj[key] }
handleClick (e)
if(headers["content-type"]
=== "text/event-stream")
/display/SSERow
<SSERow />
content

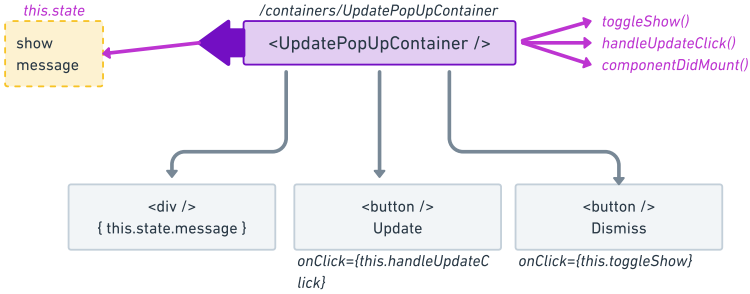
ID: {content.lastEventId}

Event: { content.type }

Time Received: { content.timeReceived }

expand-btn
onClick={(e) => this.handleClick(e)}

Data: { content.data }
if(headers["content-type"].includes("text/html")
{ events }
__html: createDOMPurify.sanitize(events[0]),
if(events.length > 1)
<JSONPretty />
data={resEvents}


this.state
show
message
/containers/UpdatePopUpContainer
<UpdatePopUpContainer />
toggleShow()
handleUpdateClick()
componentDidMount()

{ this.state.message }
<button />
Update
onClick={this.handleUpdateClick}
<button />
Dismiss
onClick={this.toggleShow}