

# Enterprise Programming - Completed Sections

6G6Z1103, 6G6Z1903

## Completed Sections

### Completed Sections

	<b>1</b>
Criteria 1	2
Criteria 2	5
Criteria 3	7
Criteria 4	9
Criteria 5	10
Criteria 6	12
Criteria 7	15

## Criteria 1

*Access to the data using simple http web service calls*

Believed class: 1st class, my implementation contains HTTP requests to the backend that as its errors dealt with.

Evidence:

```
// get all films
useEffect(() => {
  async function getFilms() {
    try {
      let url = `format=${format}`;
      if (useREST) {
        url = `${endpoints.restEndpoint}?${url}&getType=all`;
      } else {
        url = `${endpoints.getAllFilmsEndpoint}?${url}`;
      }

      setShowSpinner(true);
      switch (format) {
        case 'xml':
          setFilms(await XMLRequest(url, 'GET'));
          break;
        case 'csv':
          setFilms(await CSVRequest(url, 'GET'));
          break;
        default:
          setFilms(await JSONRequest(url, 'GET'));
      }
    } catch (e) {
      console.error(e);
    }
    setShowSpinner(false);
    setShouldGetAllFilms(false);
  }

  if (shouldGetAllFilms) getFilms();
}, [shouldGetAllFilms]);
```

All HTTP requests are structured identically, with the URL in the array of the second argument of the `useEffect` hook determining the URL and logic to be used, e.g. the create new film HTTP request:

```
// create new film
useEffect(() => {
  async function postFilm() {
    let url = `format=${format}`;
    if (useREST) {
      url = `${endpoints.restEndpoint}?${url}`;
    } else {
      url = `${endpoints.insertFilmEndpoint}?${url}`;
    }

    setShowSpinner(true);
    try {
      switch (format) {
        case 'xml':
          await XMLRequest(url, 'POST', `<Film>${jsontoxml(formData)}</Film>`);
          break;
        case 'csv':
          await CSVRequest(url, 'POST', new json2csv({ header: false, delimiter: ',' }).parse(formData));
          break;
        default:
          await JSONRequest(url, 'POST', formData);
          break;
      }
    } catch (e) {
      console.error(e);
    }
    setShouldPostFilm(false);
    setShowSpinner(false);
  }

  You, 3 weeks ago • WIP pass form as XML
  if (shouldPostFilm) postFilm();
}, [shouldPostFilm]);
```

JSONRequest function is structured as follows:

```
const JSONRequest = async (url, method, body) => {
  let options;

  if (body) {
    options = { method, body: JSON.stringify(body) };
  } else {
    options = { method };
  }

  const response = await fetch(url, options);
  return await response.json();
};

You, 3 weeks ago • Refactor
export default JSONRequest;
```

XMLRequest:

```
const XMLRequest = async (url, method, body) => {
  let response = await fetch(url, {
    method,
    body
  });

  response = await response.text();

  const xml = new DOMParser().parseFromString(response, 'application/xml');
  return new XMLSerializer().serializeToString(xml.documentElement);
};

export default XMLRequest;
```

CSVRequest:

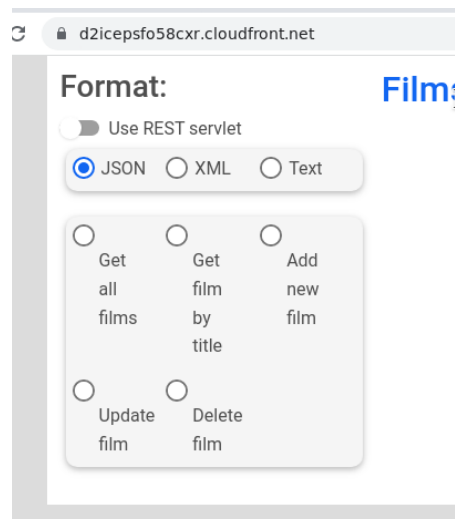
```
const CSVRequest = async (url, method, body) => {  
  let response = await fetch(url, {  
    method,  
    body  
  });  
  
  return await response.text();  
};  
  
export default CSVRequest;
```

## Criteria 2

*Options to return the data in text, json, or xml*

Believed class: 1st Class, the frontend of the project has options to handle the project's functionality in CSV (text), XML and JSON, with the back-end also handling the format and able to return data in the required format. Additionally, where appropriate libraries have been used to convert between data types.

Frontend format handling:



Back-end format handling:

All servlets implement a similar switch-case block based upon the format URL parameter.

```
coreservlets > GetAllFilms.java
@WebServlet("/getAllFilms")
public class GetAllFilms extends HttpServlet
    implements IRequestHelpers, interfaces.IPolyObjServletCommon {
    private static final long serialVersionUID = -1809220141023596490L;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) {
        // set relevant headers
        response = IRequestHelpers.setHeaders(response, "GET");
        // get all films from data access object
        ArrayList<Film> films = new FilmDAOSingleton().getAllFilms();
        // get format from url
        String format = IRequestHelpers.getFormat(request);
        Object payload;
        // convert film array list to relevant data type
        // and set appropriate header for HTTP response
        switch (format) {
            case "xml":
                response.setContentType("text/xml");
                payload = IPolyObjServletCommon.filmsToXMLArray(films);
                break;
            case "csv":
                response.setContentType("text/csv");
                payload = IPolyObjServletCommon.filmsToCSVArray(films);
                break;
            default:
                response.setContentType("application/json");
                payload = IPolyObjServletCommon.filmsToJSONArray(films);
        }
        // send response containing formatted list of all films
        IRequestHelpers.sendResponse(response, payload);
    }
}
```

Some servlets implement the “IFormatToPOJO” interface, responsible for containing the methods shared between servlets that handle single objects. Furthermore, a similar implementation exists for servlets capable of handling multiple objects known as IPolyObjServletCommon.

Single JSON objects use the Gson Java JSON serialisation library:

```
public interface IMonoObjServletCommon {

    static Film jsonToFilm(String jsonString, Boolean newFilm) {

        // if result is a new film then create a new film POJO
        // from JSON but generate new ID, otherwise create a
        // new film POJO from JSON using existing ID
        if (newFilm) {
            return new Film.Builder(new Gson().fromJson(jsonString, Film.class)).id(ISQLOperations.generateNewID()).build();
        } else {
            return new Film.Builder(new Gson().fromJson(jsonString, Film.class)).build();
        }
    }
}
```

Single XML objects use the XStream library:

```
static Film xmlToFilm(String xmlString, Boolean newFilm) {
    XStream xstream = new XStream();
    xstream.allowTypes(new Class[] { Film.class });
    xstream.processAnnotations(Film.class);

    try {
        if (newFilm) {
            return new Film.Builder((Film) xstream.fromXML(xmlString)).id(ISQLOperations.generateNewID()).build();
        } else {
            return new Film.Builder((Film) xstream.fromXML(xmlString)).build();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

For multiple objects, JSON again uses the Gson library, whereas XML uses the XStream library:

```
public interface IPolyObjServletCommon {

    static String filmsToJSONArray(ArrayList<Film> data) {
        // return array list of films to JSON
        return new Gson().toJson(data);
    }

    static String filmsToXMLArray(ArrayList<Film> data) {
        // convert films to XML

        XStream xstream = new XStream();
        // change xml tag name from class name to value of
        // first function parameter
        xstream.alias("root", List.class);
        xstream.alias("film", models.Film.class);

        // convert films to XML
        String xmlString = xstream.toXML(data);

        // return XML and XML declaration
        return "<?xml version='1.0' encoding='UTF-8' ?>" + "\n" + xmlString;
    }
}
```

## Criteria 3

*Google App Engine or Microsoft Azure to implement the application on a remote cloud based server*

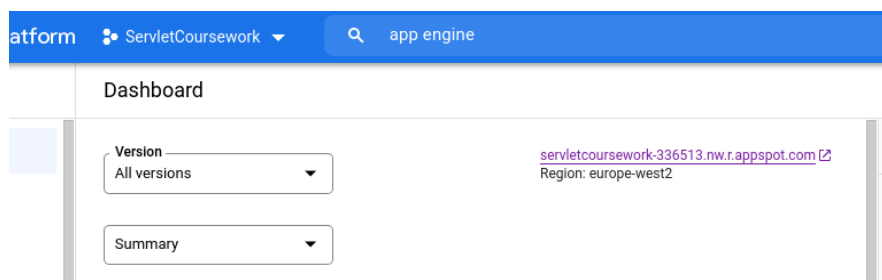
Believed grade: 1st Class: All CRUD operations in addition to RESTful servlet containing CRUD servlet functionality has been implemented both locally and on cloud.

Unfortunately, due to difficult setting up React with Google Cloud Platform, my project in fact uses two separate Cloud platforms, GCP App Engine backend + GCP SQL database, and AWS Cloudfront + S3 for frontend.

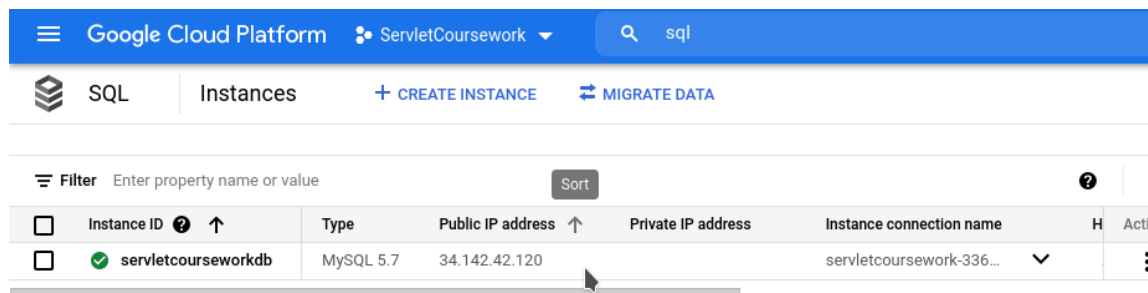
Further proof of CRUD operations being “demo’d” will be found in the “CRUD Operations.pdf” file found in the project root.

GCP:

Backend:

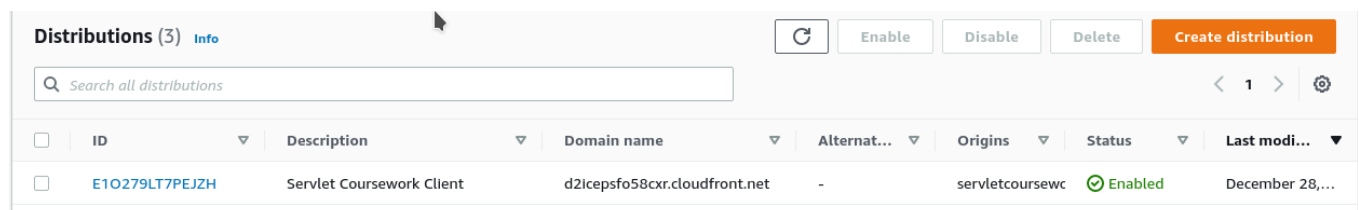


Database:

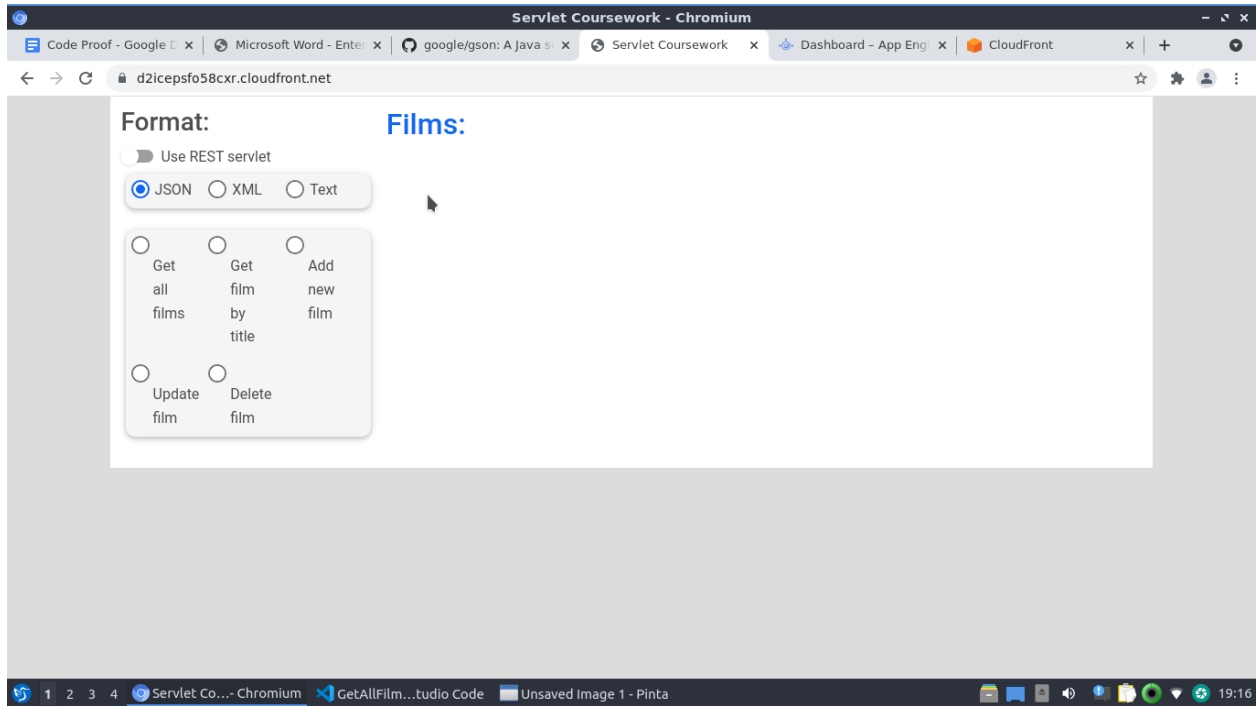


AWS:

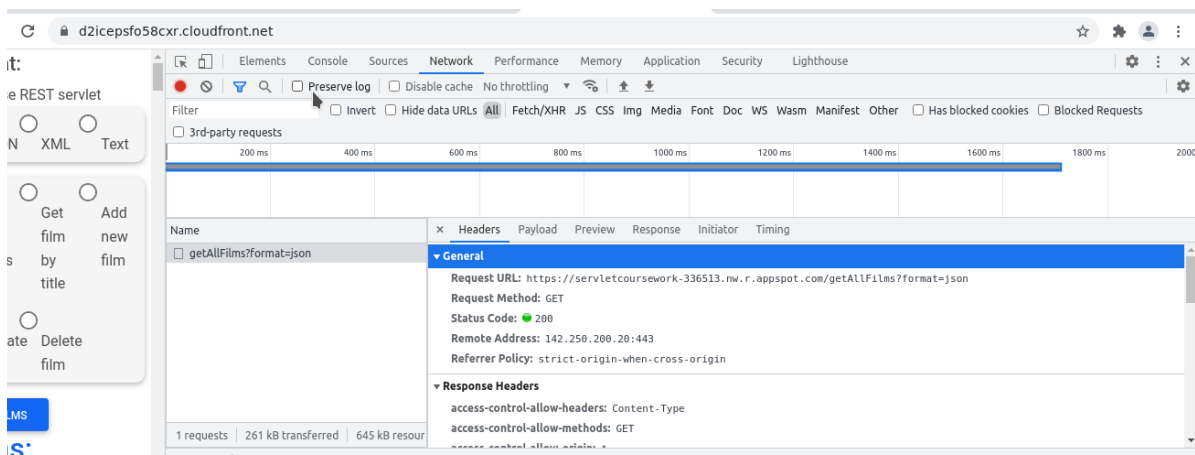
Frontend:



Access the frontend:



### Request to backend:



### Backend code connecting to GCP SQL server:

```
public interface ISQLOperations {

    static Connection loadDriver() {
        // mudfoot details
        // String url = "jdbc:mysql://mudfoot.doc.stu.mmu.ac.uk:6306/sifleetj";
        // String username = "sifleetj";
        // String password = "passwordgoeshere";

        // for GCP SQL:
        String url = "jdbc:mysql://34.142.42.120/servletcoursework?user=root";

        // initialise MySQL driver
        try {
            // for mudfoot:
            // Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
            // return DriverManager.getConnection(url, username, password);

            // for GCP SQL:
            Class.forName("com.mysql.jdbc.GoogleDriver").newInstance();
            return DriverManager.getConnection(url);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return null;
    }
}
```



## Criteria 4

*A WSDL description of the interface to the web service*

Believed grade: 1st Class, all WSDL files for all servlet including the HTTP servlet can be found in project root / WebContent / wsdl

E.g.:

```
Content > wsdl > GetAllFilms.wsdl
You, a week ago | 1 author (You)
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://coreservlets" xmlns:intf="http://coreservlets"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://coreservlets">
  <!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48
    PDT) -->
  <wsdl:portType name="GetAllFilms">

</wsdl:portType>
<wsdl:binding name="GetAllFilmsSoapBinding"
  type="impl:GetAllFilms">
  <wsdlsoap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
</wsdl:binding>
<wsdl:service name="GetAllFilmsService">
  <wsdl:port binding="impl:GetAllFilmsSoapBinding"
    name="GetAllFilms">
    <wsdlsoap:address
      location="http://localhost:8080/ServletCoursework/services/GetAllFilms" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

## Criteria 5

### *Access to the data using REST type interaction*

Believed grade: 1st Class, all servlet functionality is included in a RESTful manner:

REST servlet can be accessed via the URL: <https://servletcoursework-336513.nw.r.appspot.com/REST>  
Get films:

```
eservlets > REST.java

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) {
    // set relevant headers
    response = IHandleHTTP.setHeaders(response, "GET");

    FilmDAOSingleton filmDAO = new FilmDAOSingleton();
    // get format from url
    String format = IGetFormat.getFormat(request);
    Object payload = null;

    // get get type from url
    String getType = request.getParameter("getType");

    // determine which type of get function to use
    // based on get type from url
    switch (getType) {
        case "all":
            payload = getAllFilms(filmDAO, format, response);
            break;
        case "title":
            String title = request.getParameter("title");
            payload = getFilmByTitle(filmDAO, format, title, response);
            break;
        case "id":
            int id = Integer.parseInt(request.getParameter("id"));
            payload = getFilmByID(filmDAO, format, id, response);
    }

    // send response containing film(s)
    IHandleHTTP.sendResponse(response, payload);
}
```

Create film:

```
reservlets > REST.java

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) {
    // set relevant headers
    response = IHandleHTTP.setHeaders(response, "POST");

    // get film from HTTP body
    String requestBodyFilm = IMonoObjServletCommon.getRequestBody(request);
    // get format from url
    String format = IGetFormat.getFormat(request);

    // set film equal to film object converted based on
    // relevant format
    Film film;
    switch (format) {
        case "xml":
            film = IMonoObjServletCommon.xmlToFilm(requestBodyFilm, true);
            break;
        case "csv":
            film = IMonoObjServletCommon.csvToFilm(requestBodyFilm, true);
            break;
        default:
            film = IMonoObjServletCommon.jsonToFilm(requestBodyFilm, true);
    }
};

// send response containing number of rows affected by inserting
// new film
IHandleHTTP.sendResponse(response, new FilmDAOSingleton().insertFilm(film));
}
```

Update film:

```
@Override
protected void doPut(HttpServletRequest request, HttpServletResponse response) {
    // set relevant headers
    response = IHandleHTTP.setHeaders(response, "PUT");

    // get film from HTTP body
    String requestBodyFilm = IMonoObjServletCommon.getRequestBody(request);
    // get format from url
    String format = IGetFormat.getFormat(request);

    // set film equal to film object converted based on
    // relevant format
    Film film;
    switch (format) {
        case "xml":
            film = IMonoObjServletCommon.xmlToFilm(requestBodyFilm, false);
            break;
        case "csv":
            film = IMonoObjServletCommon.csvToFilm(requestBodyFilm, false);
            break;
        default:
            film = IMonoObjServletCommon.jsonToFilm(requestBodyFilm, false);
    };

    // print number of affected rows due to updating film
    IHandleHTTP.sendResponse(response, new FilmDAOSingleton().updateFilm(film));
}
```

Delete film:

```
@Override
protected void doDelete(HttpServletRequest request, HttpServletResponse response) {
    // set relevant headers
    response = IHandleHTTP.setHeaders(response, "DELETE");
    // get id from url
    int id = Integer.parseInt(request.getParameter("id"));

    // print number of affected rows due to deleting film
    IHandleHTTP.sendResponse(response, new FilmDAOSingleton().deleteFilm(id));
}
```

## Criteria 6

*An Ajax based web front end to retrieve the data and display in a suitable format using library based routines for an enhanced user interface*

Believed grade: 1st Class

My frontend consists of the React JavaScript library, developed from scratch. My project is structured in a logic format:

- Client
  - components
  - constants
  - pages
  - styles
  - utils

React components are styled using the CSS modules and SCSS instead of regular CSS. Where possible CSS attributes are sorted alphabetically, and so are many other aspects of the frontend, e.g.

State objects:

```
const ControlPanel = () => {
  const [endpoint, setEndpoint] = useState('');
  const [films, setFilms] = useState(null);
  const [formData, setFormData] = useState({});
  const [format, setFormat] = useState('json');
  const [formatChanged, setFormatChanged] = useState(false);
  const [selectedAttributeVal, setSelectedAttributeVal] = useState(null);
  const [selectedFilm, setSelectedFilm] = useState(null);
  const [selectedFilmID, setSelectedFilmID] = useState(null);
  const [selectedLabel, setSelectedLabel] = useState('Title');
  const [shouldDeleteFilm, setShouldDeleteFilm] = useState(false);
  const [shouldGetAllFilms, setShouldGetAllFilms] = useState(false);
  const [shouldGetFilmByID, setShouldGetFilmByID] = useState(false);
  const [shouldGetFilmByTitle, setShouldGetFilmByTitle] = useState(false);
  const [shouldPostFilm, setShouldPostFilm] = useState(false);
  const [shouldUpdateFilm, setShouldUpdateFilm] = useState(false);
  const [showSpinner, setShowSpinner] = useState(false);
  const [updateFormData, setUpdateFormData] = useState(false);
  const [useREST, setUseREST] = useState(false);
}
```

Props:

```
<MDBSwitch
  className={classes.RESTToggle}
  label="Use REST servlet"
  onChange={() => toggleHandler()}
  checked={useREST}
/>

<MDBBtnGroup className={classes.FormatRadioGroup}>
  <Radio
    defaultChecked
    label="JSON"
    name="formatGroup"
    onClick={() => {
      setFormatChanged(true);
      setFormat('json');
    }}
  />
```

The frontend also uses asynchronous HTTP requests:

```
switch (format) {
  case 'xml':
    setFilms(await XMLRequest(url, 'GET'));
    break;
  case 'csv':
    setFilms(await CSVRequest(url, 'GET'));
    break;
  default:
    setFilms(await JSONRequest(url, 'GET'));
}
```

Expanded pictures of the various request functions can be found under criteria 1.

In regards to the logic of the frontend, it is also structured logically:

Based upon the operation selected (get all films, create new film, etc.), a switch is responsible for determining with JSX (HTML) to render:

```
const renderSwitch = () => {
  switch (endpoint) {
    case endpoints.getAllFilmsEndpoint:
      return <MDBBtn onClick={() => setShouldGetAllFilms(true)}>Get films</MDBBtn>;
    case endpoints.getFilmByTitleEndpoint:
      return (
        <>
          <Input
            label="Title"
            onChange={(event) => {
              formChangedHandler(event, 'title', 'filmForm');
            }}
          />
          <MDBBtn onClick={() => setShouldGetFilmByTitle(true)}>Get film(s)</MDBBtn>
        </>
      );
    case endpoints.insertFilmEndpoint:
      return (
        <>
          <h3>Film attributes:</h3>

          <form
            onSubmit={(event) => {
              event.preventDefault();
            }}
          >
            <Input
              label="Title"
              onChange={(event) => formChangedHandler(event, 'title', 'filmForm')}
            />
          </form>
        </>
      );
  }
}
```

Each case in the render switch contains a button which once clicked, sets an appropriate state.

Due to the nature of the `useEffect` hook in React, the code inside the `useEffect` hook will run if the value of the variable inside the array of the hook's second parameter is changed. Therefore, if for example, "shouldGetAllFilms" is set to true, the code inside the hook will run. Each different operation has its own `useEffect` hook:

```
// get all films
useEffect(() => { ...
}, [shouldGetAllFilms]);

// get film by title
useEffect(() => { ...
}, [shouldGetFilmByTitle]);

// get film by ID
useEffect(() => { ...
}, [shouldGetFilmByID]);

// create new film
useEffect(() => { ...
}, [shouldPostFilm]);

// update film
useEffect(() => { ...
}, [shouldUpdateFilm]);

// delete film
useEffect(() => { ...
}, [shouldDeleteFilm]);
```

## Criteria 7

*Critical analysis of your work and the techniques you have used*

Critical analysis of my work can be found in the "Critical analysis.pdf" found in the project's root.