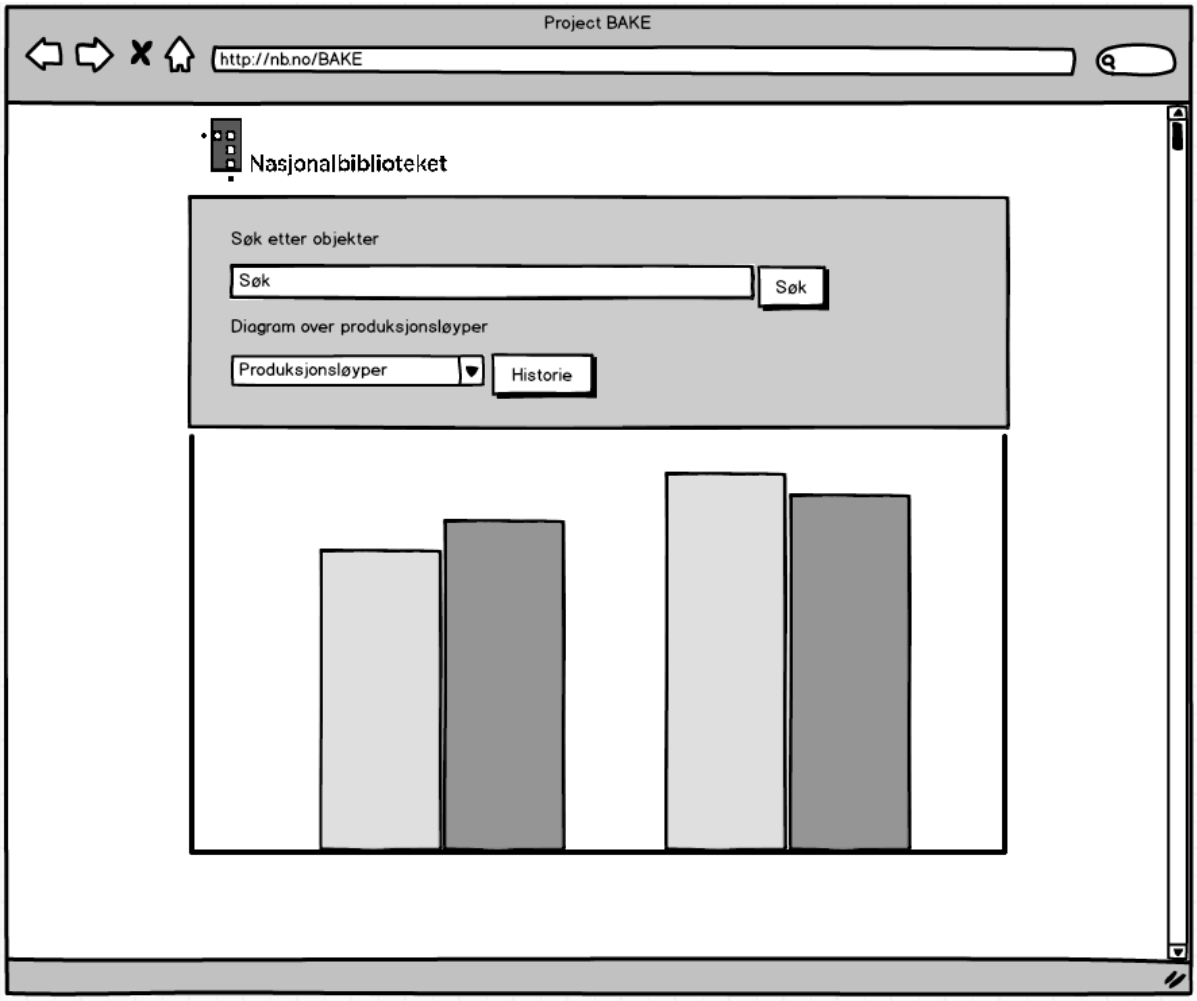
Driftsmanual - BAKE



1 Innledning

Denne applikasjonen er skrevet i *Spring framework* og kjører på *tomcat* 7, og er avhengig av å ha kontakt med *REST-tjenesten* til produksjonsdatabasen. Det er lagt inn en konfigurasjonsfil i applikasjonen der du setter URLen til *REST-tjenesten*.

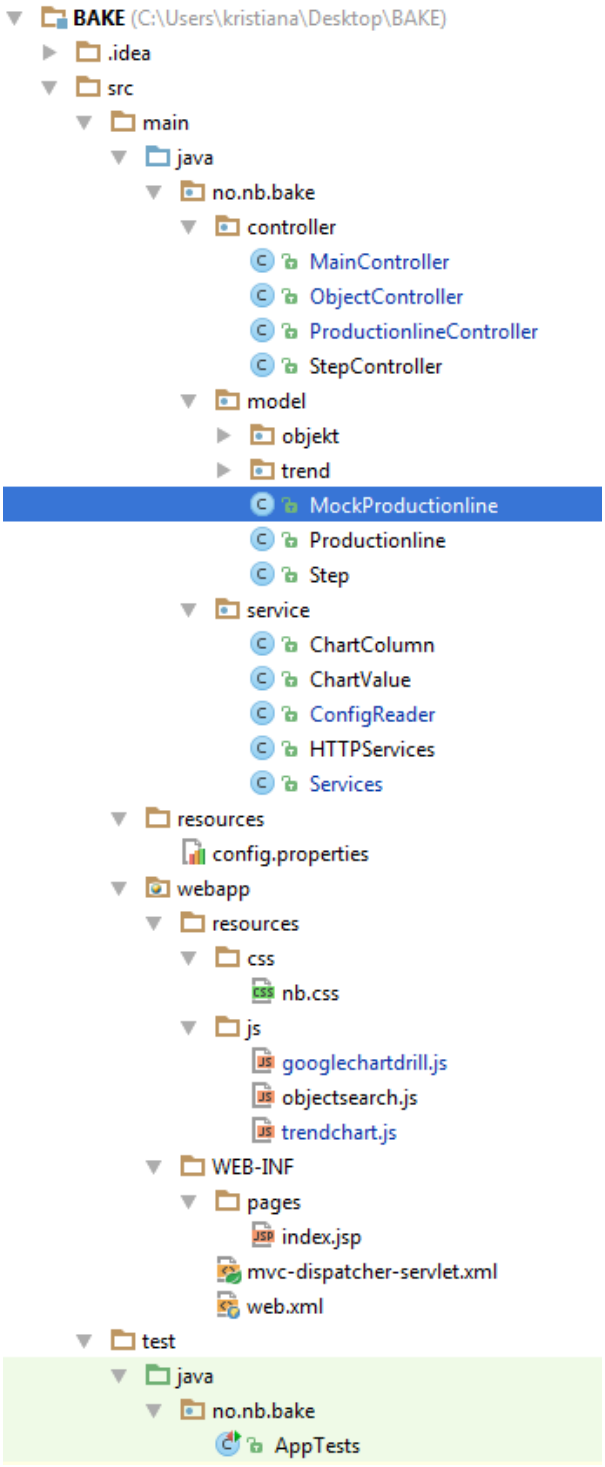
Innhold

2 Kodestruktur

Det er slik koden til applikasjonen er organisert, vi har delt opp koden i tre mapper, disse har vi kalt controller, model og service. Alle kontrollerene er lagt inn i controller mappen. Kontroller er funksjoner som lytter på en bestemt URL og som svarer hvis en bruker spør etter en nettside.

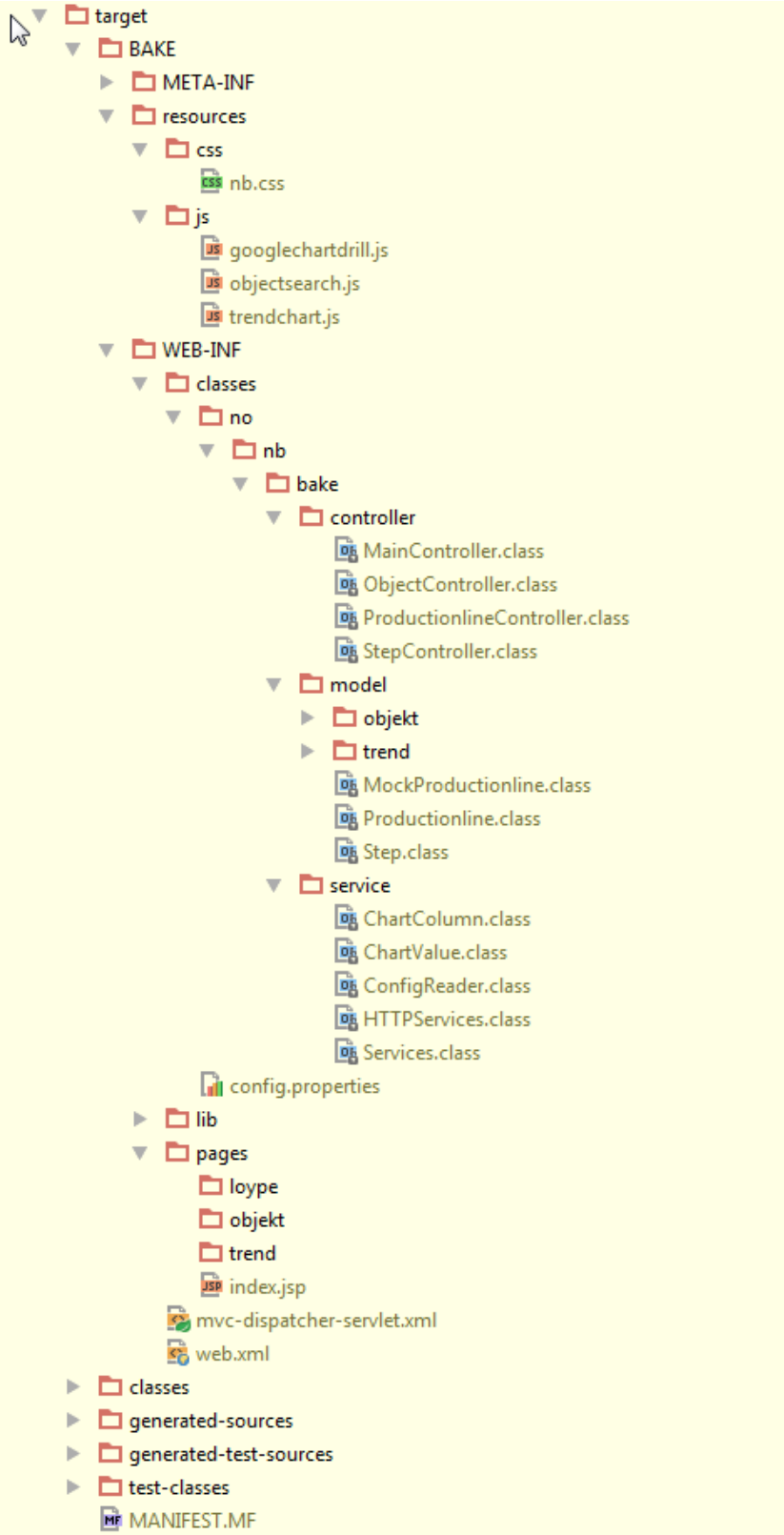
I model mappen har vi lagt alle datamodellene, dette er klassene som beskriver objektene applikasjonen bruker.

I service mappen er de fleste hjelpefunksjoner lagt, disse benyttes for å forkorte koden andre steder.



3 Filstruktur

Slik ser filstrukturen til applikasjonen ut når den er lagt ut på server eller når den er deployet.



4 MainController

Nås på root, og kjører index.jsp

Henter navnet på de forskjellige produksjonsløypene i dropdownmenyen.

```
@Controller
@RequestMapping("/")
public class MainController {
    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView getIndex() {
        ModelAndView model = new ModelAndView("/index");
        HTTPServices httpServices = new HTTPServices();
        Productionline[] productionlines = httpServices.getProductionlineHTTP(ConfigReader.getValue("productionlines"));
        model.addObject("productionlines", productionlines);
        return model;
    }
}
```

5 ObjectController

Er satt opp slik at den må ha med en søkestreng i url'en.

Nås på for eksempel /object?searchstring=digibok_2014081906069 og returnerer informasjon om et objekt i json format.

```
@Controller
public class ObjectController {

    @RequestMapping(value = "/object", method = RequestMethod.GET)
    public
    @ResponseBody
    String object(@RequestParam("searchstring") String id) {...}
}
```

6 ProductionLineController

Dette er kontrolleren for det som har med løypene å gjøre, og kjøres når man velger en produksjonsløype i dropdownmenyen.

Denne har to sider, /loype og /loype/trend. Den førstnevnte er satt opp slik at de må ha en søkestreng med i URL'en.

For eksempel /loype?id=1 returnerer antall objekter i en produksjonsløype i json format.

/loype/trend kjøres når man trykker historieknappen, og her må det også sendes med en søkestreng på samme måte som over. For eksempel: loype/trend?id=1, og her returneres et år med produksjonstall fordelt på måneder i json format.

```
@Controller
public class ProductionlineController {

    @RequestMapping(value = "/loype", method = RequestMethod.GET)
    public
    @ResponseBody
    String drawProductionlineChart(@RequestParam("id") int id) {...}

    @RequestMapping(value = "/loype/trend", method = RequestMethod.GET)
    public
    @ResponseBody
    String drawProductionlineHistoryChart(@RequestParam("id") int id) {...}
}
```

7 StepController

Dette er controlleren som kjøres hvis man klikker på de ulike stegene i grafen etter å ha valgt produksjonsløype i dropdownmenyen. Den fungerer på samme måte som de andre Controller-klassene, men med flere variabler i søkestrengen. Derfor er ikke denne tatt med i konfigurasjonsfilen enda. Hvis man har klikket seg ned til dag-visningen i grafene kan søkestrengen bli for eksempel: /step/week?id=1&stepid=2&monthid=3&weekid=4, og her returneres produksjonstall for hver dag i en uke.

Denne funksjonaliteten er dårlig imitert, og bør sees nærmere på i framtiden.

```
@Controller
public class StepController {


    @RequestMapping(value = "/step/week", method = RequestMethod.GET)
    public
    @ResponseBody

    String drawWeek(@RequestParam("id") String id, @RequestParam("monthid") String monthid, @RequestParam("stepid") String stepid, @RequestParam("weekid") String weekid) {...}
    @RequestMapping(value = "/step/month", method = RequestMethod.GET)
    public
    @ResponseBody
    String drawMonth(@RequestParam("id") String id, @RequestParam("monthid") String monthid, @RequestParam("stepid") String stepid) {...}
    @RequestMapping(value = "/step/year", method = RequestMethod.GET)
    public
    @ResponseBody
    String drawYear(@RequestParam("id") String id, @RequestParam("stepid") String stepid) {...}
}
```

8 Services

I denne klassen har vi mange ToJson funksjoner som tar inn et objekt, og konverterer dette til datastrukturen Google charts bruker og returnerer dette som en string.

I tillegg til dette, er det en unixToDate funksjon som tar inn en rekke datoer i form av stringer og gjør dette om til et fullverdig date objekt.

```
public class Services {

    public static String mockProductionlineToJson(MockProductionline productionlines) {
        ArrayList<ChartColumn> array = new ArrayList<~>();
        for (Step l : productionlines.getSteps()) {
            ArrayList<ChartValue> values = new ArrayList<~>();
            ChartColumn objectArray = new ChartColumn(values);
            values.add(new ChartValue(l.getStep()));
            values.add(new ChartValue(l.getReady()));
            values.add(new ChartValue(l.getTerminated()));
            values.add(new ChartValue(l.getFailed()));
            array.add(objectArray);
        }
        String jsonstring = new Gson().toJson(array);
        return jsonstring;
    }
```

9 HTTPServices

Dette er klassen som tar seg av alt av HTTP, den henter ned JSON fra nettet og gjør det om til et java objekt.

```
public class HTTPServices {
    RestTemplate restTemplate;
    HttpEntity<?> requestEntity;

    public HTTPServices() {
        HttpHeaders requestHeaders = new HttpHeaders();
        requestHeaders.setAccept(Collections.singletonList(new MediaType("application", "json")));
        this.requestEntity = new HttpEntity<Object>(requestHeaders);
        this.restTemplate = new RestTemplate();
        restTemplate.getMessageConverters().add(new GsonHttpMessageConverter());
    }

    public Productionline[] getProductionlineHTTP(String uri) {
        ResponseEntity<Productionline[]> responseEntity = this.restTemplate.exchange(uri, HttpMethod.GET, this.requestEntity, Productionline[].class);
        Productionline[] productionlines = responseEntity.getBody();

        return productionlines;
    }
```

10 Konfigurasjonsfil

Konfigurasjonsfilen leses når en side blir besøk. Det gjør at man kan endre konfigurasjonsfilen uten å måtte restarte serveren.

Slik ser konfigurasjonsfilen ut per dags dato. Det er noen URI's som ikke er lagt til enda, på grunn av for dårlig imitering fra vår side.

```
productionlines=http://tctest.nb.no:8110/rest/v1/productionlines.json
object=http://tctest.nb.no:8110/rest/v1/items/name/
loype=http://localhost:8080/rest/loype/
prod=http://localhost:8080/rest/prod/
```

I konfigurasjonsfilen setter man variabelnavn og hva den skal være. I eksemplet under ser vi at variablen productionlines er lik http://tctest.nb.no:8110/rest/v1/productionlines.json.
Om man skal legge til eller endre noen legges dette bare inn i konfigurasjonsfilen på samme måte som vist under.

```
productionlines=http://tctest.nb.no:8110/rest/v1/productionlines.json
object=http://tctest.nb.no:8110/rest/v1/items/name/
loype=http://localhost:8080/rest/loype/
prod=http://localhost:8080/rest/prod/
```

For å bruke variablene fra konfigurasjonsfilen i applikasjonen må man kjøre ConfigReader klassen og metoden getValue.
I Eksemplet under ser vi at vi i HTTPServices kjører ConfigReader.getValue("productionlines"), der "productionlines" henviser til variabelen productionlines i konfigurasjonsfilen.

```
@Controller
@RequestMapping("/")
public class MainController {
    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView getIndex() {
        ModelAndView model = new ModelAndView("/index");
        HTTPServices httpServices = new HTTPServices();
        Productionline[] productionlines = httpServices.getProductionlineHTTP(ConfigReader.getValue("productionlines"));
        model.addObject("productionlines", productionlines);
        return model;
    }
}
```