Software Engineering 2

Status Update - Report

Work Group 304

01504922 SIGMUND Daniel
01426012 NISSLMÜLLER Utz
01363460 RAMHARTER Alexander

Design Drafts

Design 1 - State-Based Design

Description

The first design we came up with is state-based, meaning that the server stores the state of each canvas locally and lets users access it via an ID.

The approach that our first prototype for this design used was quite basic compared to our current implementation. We started by creating the classes that we thought we were definitely going to need, like `Canvas`, `Layer` and `Shape`. Then we started abstracting our thought process more and more.

We defined what we believe are essential requirements for each shape in the abstract class `GraphicElement` (formerly `Shape`). At that point, we also added the class `Point`, so that we would not have to split up coordinates into their components every time. From there on, we tried to come up with ways to divide the concrete shapes into groups extending `GraphicElement`.

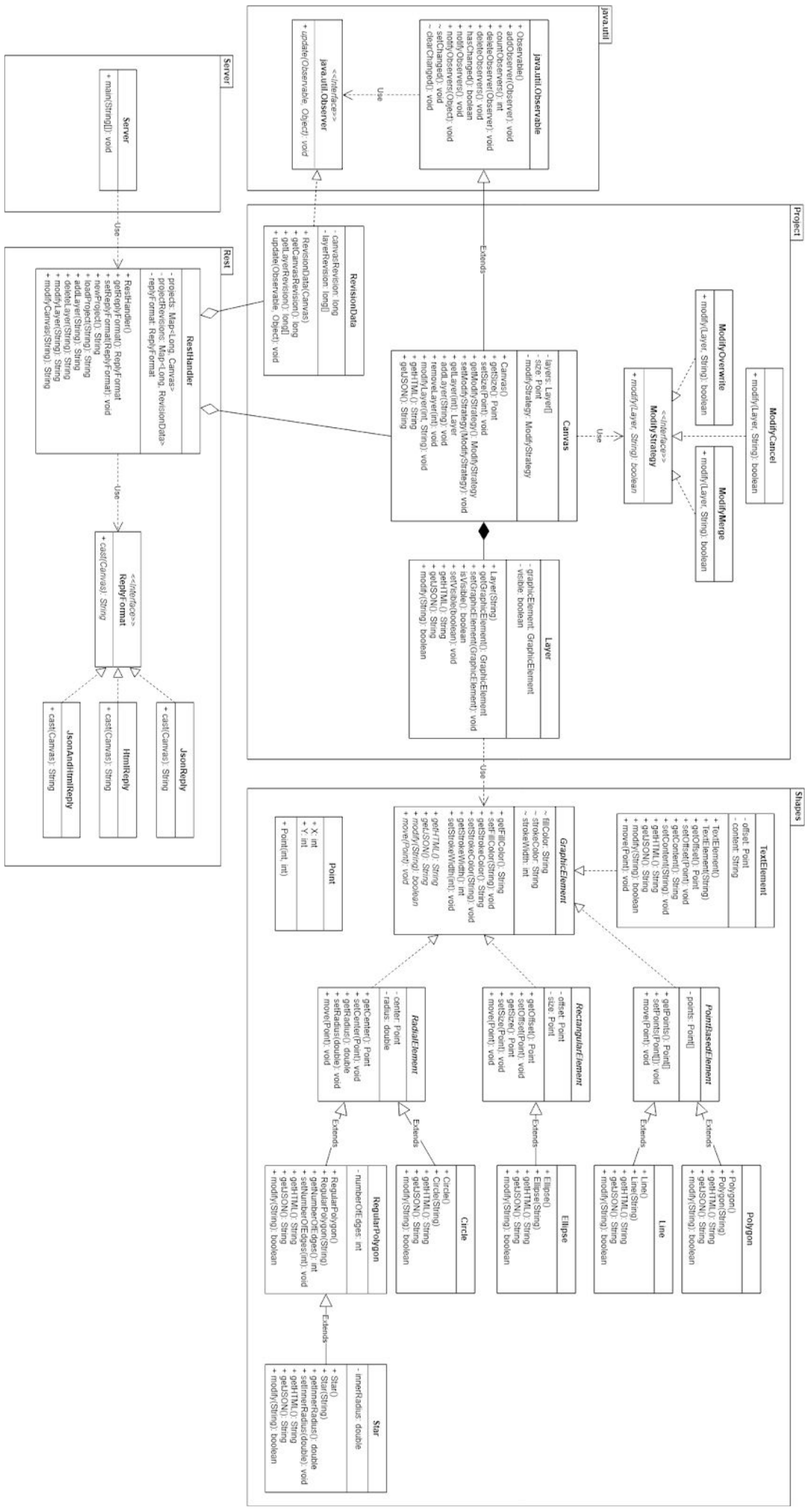
First, we grouped shapes that do not necessarily need to have a class for themselves into a single concrete shape respectively. As a concrete example, we decided that for triangles, quadrangles and n-gons, only a single class `Polygon` would be needed. We also added a `RegularPolygon` class for a different kind of n-gon. We then looked at the least number of attributes needed to define each shape and so we came up with the idea to divide them into `PointBasedElement`, `RectangularElement` and `RadialElement`.

`Canvas` itself is observable. It is observed by `RevisionData`, which keeps track of how often the canvas as a whole and each layer individually have been modified. This enables version checking and finding modification conflicts when multiple users are working on the same canvas. Our implementation of the strategy pattern in `ModifyStrategy` then tries to resolve modification conflicts in different ways (the strategy can be different for each `Canvas` instance).

In the last step we added the `Server` and `RESTHandler` classes, which launch the Spring Boot application and deal with requests. We added a second strategy pattern, `ReplyFormat`, which specifies which of the `getJSON()` and `getHTML()` functions of the `Canvas` should be included in the server’s reply.

UML Class Diagram

Note: PNG file of this diagram can be found at /designs/DesignStateBased/DesignStateBased.png



Class, Interface and Method Signatures

Note: Javadoc HTML file with the descriptions of the classes/interfaces and methods can be found at designs/DesignStateBased/javadoc/index.html

```
public class Canvas extends java.util.Observable {
  public Canvas();
  public Point getSize();
  public void setSize(Point size);
  public ModifyStrategy getModifyStrategy();
  public void setModifyStrategy(ModifyStrategy modifyStrategy);
  public Layer getLayer(int layerIndex);
  public void addLayer(String shapeType);
  public void removeLayer(int layerIndex);
  public void modifyLayer(int layerIndex, String attributes);
  public void modifyCanvas(String attributes);
  public String getHTML();
  public String getJSON();
}

public class Layer {
  public Layer(String shapeType) ;
  public GraphicElement getGraphicElement();
  public void setGraphicElement(GraphicElement graphicElement);
  public boolean isVisible();
  public void setVisible(boolean visible);
  public String getHTML();
  public String getJSON();
  public boolean modify(String attributes);
}

public class RevisionData implements java.util.Observer {
  public RevisionData(Canvas canvas);
  public long getCanvasRevision();
  public List<Long> getLayerRevision();
  public void update(Observable canvas, Object layer);
}

public interface ModifyStrategy {
  boolean modify(Layer layer, String attributes);
}

public class ModifyCancel implements ModifyStrategy {
  public boolean modify(Layer layer, String attributes);
}

public class ModifyOverwrite implements ModifyStrategy {
  public boolean modify(Layer layer, String attributes);
}

public class ModifyMerge implements ModifyStrategy {
  public boolean modify(Layer layer, String attributes);
}
```

```java
public class Point {
  public final int X;
  public final int Y;
  public Point(int x, int y);
}

public abstract class GraphicElement {
  public String getFillColor();
  public void setFillColor(String fillColor);
  public String getStrokeColor();
  public void setStrokeColor(String strokeColor);
  public int getStrokeWidth();
  public void setStrokeWidth(int strokeWidth);
  public abstract String getHTML();
  public abstract String getJSON();
  public abstract boolean modify(String attributes);
  public abstract void move(Point vector);
}

public abstract class PointBasedElement extends GraphicElement {
  public List<Point> getPoints();
  public void setPoints(List<Point> points);
  public void move(Point vector);
}

public class Line extends PointBasedElement {
  public Line();
  public Line(String attributes);
  public String getHTML();
  public String getJSON();
  public boolean modify(String attributes);
}

public class Polygon extends PointBasedElement {
  public Polygon();
  public Polygon(String attributes);
  public String getHTML();
  public String getJSON();
  public boolean modify(String attributes);
}

public abstract class RadialElement extends GraphicElement {
  public Point getCenter();
  public void setCenter(Point center);
  public double getRadius();
  public void setRadius(double radius);
  public void move(Point vector);
}
```

```java
public class Circle extends RadialElement {
  public Circle();
  public Circle(String attributes);
  public String getHTML();
  public String getJSON();
  public boolean modify(String attributes);
}

public class RegularPolygon extends RadialElement {
  public RegularPolygon();
  public RegularPolygon(String attributes);
  public int getNumberOfEdges();
  public void setNumberOfEdges(int numberOfEdges);
  public String getHTML();
  public String getJSON();
  public boolean modify(String attributes);
}

public class Star extends RegularPolygon {
  public Star();
  public Star(String attributes);
  public double getInnerRadius();
  public void setInnerRadius(double innerRadius);
  public String getHTML();
  public String getJSON();
  public boolean modify(String attributes);
}

public abstract class RectangularElement extends GraphicElement {
  public Point getOffset();
  public void setOffset(Point offset);
  public Point getSize();
  public void setSize(Point size);
  public void move(Point vector);
}

public class Ellipse extends RectangularElement {
  public Ellipse();
  public Ellipse(String attributes);
  public String getHTML();
  public String getJSON();
  public boolean modify(String attributes);
}
```

```
public class TextElement extends GraphicElement {
  public TextElement();
  public TextElement(String attributes);
  public Point getOffset();
  public void setOffset(Point offset);
  public String getContent();
  public void setContent(String content);
  public String getHTML();
  public String getJSON();
  public boolean modify(String attributes);
  public void move(Point vector);
}

public interface ReplyFormat {
  String cast(Canvas canvas);
}

public class HtmlReply implements ReplyFormat {
  public String cast(Canvas canvas);
}

public class JsonAndHtmlReply implements ReplyFormat {
  public String cast(Canvas canvas);
}

public class JsonReply implements ReplyFormat {
  public String cast(Canvas canvas);
}

public class RestHandler {
  public RestHandler();
  public ReplyFormat getReplyFormat();
  public void setReplyFormat(ReplyFormat replyFormat);
  public String newProject();
  public String loadProject(String json);
  public String addLayer(String json);
  public String deleteLayer(String json);
  public String modifyLayer(String json);
  public String modifyCanvas(String json);
}

public class Server {
  public static void main(String[] args);
}
```

Design 2 - Stateless Design

Description

For the second design, we took a slightly different approach. We tried to build our backend around both the observer and strategy patterns. The basic classes for the design are identical: `Canvas`, `Layer` and `Shape`. However, this design is stateless, meaning that the server does not store any information locally. This means that every request must include the full canvas state, as well as the parameters for the desired operation.

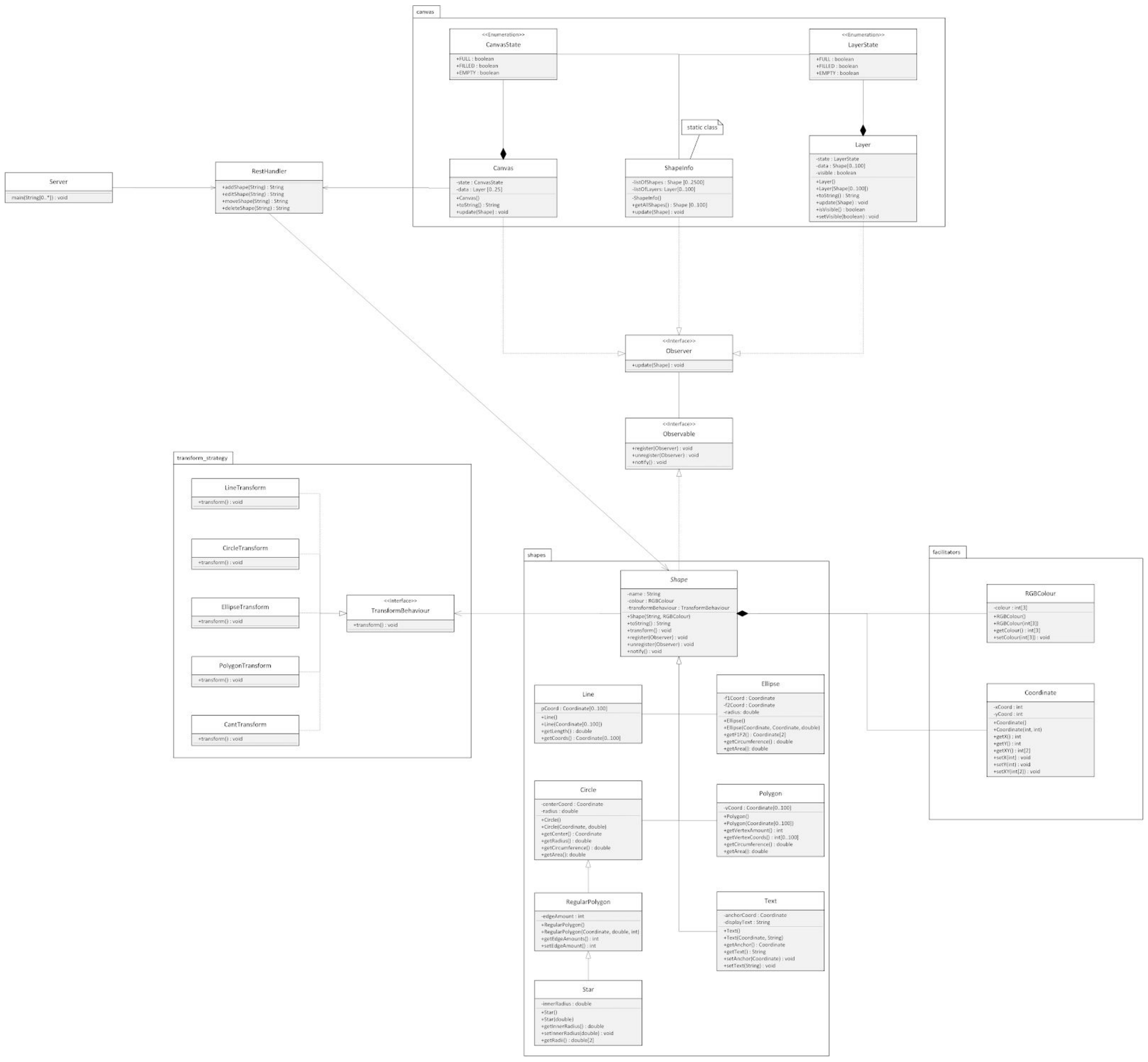
We limited the number of `Layer` instances per `Canvas` to a maximum of 25 and the number of `Shape` instances per `Layer` to a maximum of 100, in order to reduce potential load on the server as a precaution. `Canvas` and `Layer` can either be either empty, filled or full, which is reflected in the enumeration classes `CanvasState` and `LayerState` respectively. We created the static class `ShapeInfo` to gather all the information of the current state in one place.

Every shape has at least one variable of type `Coordinate` (similar to `Point` in Design 1) and `RGBColour`. The different shapes extend the abstract `Shape` class directly, unless they extend another (concrete) shape. Like in Design 1, the program's main class is `Server`, with `RestHandler` responsible for answering every API call made by the client.

Concerning the observer pattern, we thought it would make the most sense to let `Canvas` and `Layer` observe each of their shapes individually. For example, if the `RestHandler` class modifies a shape, which is meant to be done directly in this design, the `Layer` and `Canvas` classes get notified. After reflecting for some time, we decided that the application of a strategy pattern would best fit into the transform operation, which applies the `transform` HTML attribute (includes `translate`, `scale`, `rotate` and `skew`). `TransformStrategy` defines how to apply the attribute on a given shape.

UML Class Diagram

Note: PNG file of this diagram can be found at /designs/DesignStateLess/DesignStateLess.png



Class, Interface and Method Signatures

Note: Javadoc HTML file with the descriptions of the classes/interfaces and methods can be found at designs/DesignStateLess/javadoc/index.html

```
public interface Observer {
  void update(Shape shape);
}

public interface Observable {
  void register(Observer observer);
  void unregister(Observer observer);
  void notifyObservers();
}

public class Canvas implements Observer {
  public Canvas();
  public String toString();
  public void update(Shape shape);
}

public enum CanvasState {
  FULL,
  FILLED,
  EMPTY
}

public class Layer implements Observer {
  public Layer(Shape[] shape);
  public Layer();
  public String toString();
  public void update(Shape shape);
}

public enum LayerState {
  FULL,
  FILLED,
  EMPTY
}

public class ShapeInfo implements Observer {
  private ShapeInfo();
  public Shape[] getAllShapes();
  public void update(Shape shape);
}
```

```
public class Coordinate {
  public Coordinate();
  public Coordinate(int xCoord, int yCoord);
  public int getxCoord();
  public int getyCoord();
  private void setxCoord(int xCoord);
  private void setyCoord(int yCoord);
  public void setXY(int[] xy);
}

public class RGBColour {
  public RGBColour(int[] colour);
  public int[] getColour();
  public void setColour(int[] colour);
}

public abstract class Shape implements Observable {
  public Shape(String name, RGBColour colour);
  public String toString();
  public void move();
  public void register(Observer observer);
  public void unregister(Observer observer);
  public void notifyObservers();
}

public class Circle extends Shape {
  public Circle();
  public Circle(String name, RGBColour colour, Coordinate centerCoord,
      double radius);
  public Coordinate getCenterCoord();
  public double getRadius();
  public double getCircumference();
  public double getArea();
}

public class RegularPolygon extends Circle {
  public RegularPolygon();
  public RegularPolygon(String name, RGBColour colour, Coordinate centerCoord,
      double radius, int edgeAmount);
  public int getEdgeAmount();
  public void setEdgeAmount(int edgeAmount);
}

public class Star extends RegularPolygon {
  public Star();
  public Star(String name, RGBColour colour, Coordinate centerCoord, double radius,
      int edgeAmount, double innerRadius);
  public double getInnerRadius();
  public double[] getRadii();
  public void setInnerRadius(double innerRadius);
}
```

```
public class Ellipse extends Shape {
  public Ellipse(String name, RGBColour colour, Coordinate f1Coord,
      Coordinate f2Coord, double radius);
  public Ellipse();
  public Coordinate getF1F2();
  public double getCircumference();
  public double getArea();
}

public class Line extends Shape {
  public Line(String name, RGBColour colour, Coordinate[] pCoord);
  public Line();
  public double getLength();
  public Coordinate[] getCoords();
}

public class Polygon extends Shape {
  public Polygon(String name, RGBColour colour, Coordinate[] coordinates);
  public Polygon();
  public int getVertexAmount();
  public int[] getVertexCoords();
  public double getCircumference();
  public double getArea();
}

public class Text extends Shape {
  public Text();
  public Text(String name, RGBColour colour, Coordinate anchorCoordinate,
      String displayText);
  public Coordinate getAnchorCoordinate();
  public String getDisplayText();
  public void setAnchorCoordinate(Coordinate anchorCoordinate);
  public void setDisplayText(String displayText);
}

public interface TransformBehaviour {
  void transform();
}

public class CantTransform implements TransformBehaviour {
  public void transform();
}

public class CircleTransform implements TransformBehaviour {
  public void transform();
}

public class EllipseTransform implements TransformBehaviour {
  public void transform();
}

public class LineTransform implements TransformBehaviour {
  public void transform();
}
```

```
public class PolygonTransform implements TransformBehaviour {
  public void transform();
}

public class RestHandler {
  public String addShape(String json);
  public String editShape(String json);
  public String moveShape(String json);
  public String deleteShape(String json);
}

public class Server {
  public static void main(String[] args);
}
```

Differences Between Design 1 & 2

The main difference between Design 1 and Design 2 is that Design 1 is state-based, while Design 2 is stateless. In our case, this implies that the server stores projects (modeled by the `Canvas` class in both cases) in Design 1, whereas in Design 2 projects are only created temporarily to process requests, and deleted again afterwards. This also means that Design 1 puts more strain on the server's physical memory, while Design 2 puts puts a bigger load on the network connection.

Furthermore, since Design 1 assigns an ID to each each project, it allows for multiple users to be working on the same project, which is something we did not want to allow in Design 2. The main drawback of this option and reason to exclude it from Design 2 is that it naturally incurs modification conflicts, which have to be dealt with (the `RevisionData` class coupled with the `ModifyStrategy` interface in Design 1 are a very simplistic approach to solving this problem).

There is a small difference in how a `Layer` works. The number of shapes a `Layer` can hold was not clearly specified in the requirements, so we approached it differently in the two designs. Design 1 has a `Layer` defined to hold exactly one `GraphicElement` instance. Design 2 allows for a `Layer` to hold any amount of `Shape` instances (other than the bound specified in the UML diagram due to server load capacity).

As for the `Shape` and `GraphicElement` classes (which are mostly synonymous), Design 1 adds an additional layer of abstraction (in `PointBasedElement`, `RectangularElement` and `RadialElement`) between the `GraphicElement` class and its concrete subclasses, while Design 2 has the concrete subclasses inherit directly from the `Shape` class. The concrete subclasses themselves are fairly similar (excluding `Ellipse`, which uses a bounding rectangle in Design 1 and a radius and two foci in Design 2).

Direct Comparison:

	Design 1	**Design 2**
Communication basis	State-based	Stateless
Project access	Multiple users	Single user
Number of shapes in a `Layer`	One	Any
Relation between `Shape`/`GraphicElement` and concrete subclasses	Inheritance through abstract class	Direct inheritance

Design Choice and Changes

After thoroughly thinking through and discussing both designs and their potential uses, we chose Design 2 as the basis. The reasons for our choice was that we thought a stateless design would be easier to deal with. Especially the multiple user access problem was something we wanted to avoid if possible. We also deemed the additional layer of abstraction for the shapes unnecessary. This assumption remains true at of the time of writing this document.

However, we did not disregard the approach we chose for Design 1 at all. In fact, once we started developing, we soon decided that going state-based would eliminate a lot of potential problems and take some of the responsibility for guaranteeing a working communication from the client to make the API take care of it instead.

We also decided to remove a few classes as we transitioned into the state-based approach, namely the `CanvasState`, `LayerState` and `ShapeInfo` classes.

Regardless of design choice, a big change we had to make was the structure of the client-server communication itself. We realised that for our project to integrate the Spring Boot framework well, we would need classes for the different kinds of messages passed between client and server, something we had not planned for in our original designs.

As for the implementation of the observer and strategy patterns, we ended up dropping all of our previous ideas in favour of our current implementations (described later in this report).

In retrospect, even though we decided on implementing Design 2, it is difficult to say which of the two designs really had more influence on our current project structure. In any case, we think that we learned a lot through either of the two designs and that it was crucial that we allotted a sufficient amount of time to designing our application, because it allowed our minds to adapt to the task and for the ideas for what the project should look like on a system level to grow steadily in our heads.

Technology Stack

- **Spring Boot Framework v2.0.5** (Web-/Networking capabilities)

- **Batik Transcoder v1.6.1** (file format conversion for download)

- **Vue.js Framework** (more flexible frontend / UI design compared to vanilla JS)

API Specification

Title	**Create New Project**
URL	/create
Method	GET
URL Params	None
Data Params	None

Success Response

A successful response will include an alphanumeric projectID as well as an empty canvas SVG object of size 200x200 px.

```
body :
{
  projectID : [string],
  canvas : {
    width : [double],
    height : [double],
    layers : [Array[Layer]],
    html : [string]
  }
}
```

The structure for a prototype response will be identical for most of the following requests and will therefore be omitted from the affected API calls.

Example:
```
body:
{
  projectID : "exampleID123",
  canvas : {
    width : 200.0,
    height : 200.0,
    layers : [],
    html : "<svg width="200.0" height="200.0" baseProfile="full"
             xmlns="http://www.w3.org/2000/svg"></svg>"
  }
}
```

Error Response

An invalid request will yield a standardized error response message (time zone GMT+1).

```
{
  timestamp : [yyyy-mm-dd, hh:mm:ss.ms],
  status : [HTTPErrorCode],
  error : HTTPErrorMessage,
  message : [string],
  path : [URI]
}
```

The prototype structure for an error response will be identical for all of the subsequent requests and will therefore be omitted from the following API calls.

Example:
```
{
  timestamp : "2018-12-31 23:59:59.999",
  status : 405,
  error : "Method Not Allowed",
  message : "Request method 'POST' not supported",
  path : "/create"
}
```

Sample Call	`$.ajax({`
	`  url : "/create",`
	`  type : "GET",`
	`  success : function(r) {`
	`    console.log(r);`
	`  }`
	`});`

Title	**Edit Canvas**
URL	/editCanvas/:projectID
Method	POST
URL Params	**Required:** `projectID : [string]` **Example:** `projectID: "ExampleID123"`
Data Params	`{ width : [double], height : [double] }` **Example:** `{ width : 400, height : 400 }`
Success Response	A successful response sets the attributes of the Canvas as specified. **Example:** `body : { projectID : "exampleID123", canvas : { width : 400.0, height : 400.0, layers : [], html : "<svg width="400.0" height="400.0" baseProfile="full" xmlns="http://www.w3.org/2000/svg"></svg>" } }`
Error Response	**Example:** `{ timestamp : "2018-12-31 23:59:59.999", status : 500, error : "Internal Server Error", message : "Height and width must both be positive!", path : "/editCanvas/ExampleID123" }`
Sample Call	`$.ajax({ url : "/editCanvas/ExampleID123", type : "POST", data : { width : 400, height : 400 }, success : function(r) { console.log(r); } });`

Title	**Add New Layer To Canvas**
URL	/addLayer/:projectID
Method	POST

URL Params	**Required:**
	`projectID : [string]`
	
	Example:
	`projectID: "ExampleID123"`

Data Params	`{}`

Success Response	A succesful response adds a Layer to the Layer list (compare object with call for /create) in Canvas.
	
	Example:
	`body :`
	`{`
	  `projectID : "exampleID123",`
	  `canvas : {`
	    `width : 200.0,`
	    `height : 200.0,`
	    `layers : [{`
	      `shapes : [],`
	      `visible : true`
	    `}],`
	    `html : "<svg width="200.0" height="200.0" baseProfile="full"`
	            `xmlns="http://www.w3.org/2000/svg"></svg>"`
	  `}`
	`}`

Error Response	**Example:**
	`{`
	  `timestamp : "2018-12-31 23:59:59.999",`
	  `status : 405,`
	  `error : "Method Not Allowed",`
	  `message : "Request method 'GET' not supported",`
	  `path : "/addLayer/ExampleID123"`
	`}`

Sample Call	`$.ajax({`
	  `url : "/addLayer/ExampleID123",`
	  `type : "POST",`
	  `data : {},`
	  `success : function(r) {`
	    `console.log(r);`
	  `}`
	`});`

Title	**Edit Layer**
URL	/editLayer/:projectID
Method	POST

URL Params	**Required:**
	`projectID : [string]`
	
	Example:
	`projectID: "ExampleID123"`

Data Params	`{`
	`  layerIndex : [integer],`
	`  visible : [boolean]`
	`}`
	
	Example:
	`{`
	`  layerIndex : 0,`
	`  visible : false`
	`}`

Success Response	A succesful response sets the attributes of the Layer as specified.
	
	Example:
	`body :`
	`{`
	`  projectID : "exampleID123",`
	`  canvas : {`
	`    width : 200.0,`
	`    height : 200.0,`
	`    layers : [{`
	`      shapes : [],`
	`      visible : false`
	`    }],`
	`    html : "<svg width="200.0" height="200.0" baseProfile="full"`
	`            xmlns="http://www.w3.org/2000/svg"></svg>"`
	`  }`
	`}`

Error Response	**Example:**
	`{`
	`  timestamp : "2018-12-31 23:59:59.999",`
	`  status : 500,`
	`  error : "Internal Server Error",`
	`  message : "Layer index out of bounds!",`
	`  path : "/editLayer/ExampleID123"`
	`}`

Sample Call	`$.ajax({`
	`  url : "/editLayer/ExampleID123",`
	`  type : "POST",`
	`  data : {`
	`    layerIndex : 0,`
	`    visible : false`
	`  },`
	`  success : function(r) {`
	`    console.log(r);`
	`  }`
	`});`

Title	**Delete Layer**
URL	/deleteLayer/:projectID
Method	POST
URL Params	**Required:**
	`projectID : [string]`
	Example:
	`projectID: "ExampleID123"`
Data Params	`{`
	`  layerIndex : [integer]`
	`}`
	Example:
	`{`
	`  layerIndex : 0`
	`}`
Success Response	A succesful response deletes the Layer.
	Example:
	`body :`
	`{`
	`  projectID : "exampleID123",`
	`  canvas : {`
	`    width : 200.0,`
	`    height : 200.0,`
	`    layers : [],`
	`    html : "<svg width="200.0" height="200.0" baseProfile="full"`
	`            xmlns="http://www.w3.org/2000/svg"></svg>"`
	`  }`
	`}`
Error Response	**Example:**
	`{`
	`  timestamp : "2018-12-31 23:59:59.999",`
	`  status : 500,`
	`  error : "Internal Server Error",`
	`  message : "Layer index out of bounds!",`
	`  path : "/deleteLayer/ExampleID123"`
	`}`
Sample Call	`$.ajax({`
	`  url : "/deleteLayer/ExampleID123",`
	`  type : "POST",`
	`  data : {`
	`    layerIndex : 0`
	`  },`
	`  success : function(r) {`
	`    console.log(r);`
	`  }`
	`});`

Title	**Add New Shape To Layer**
URL	/addShape/:projectID
Method	POST

URL Params	**Required:**
	`projectID : [string]`
	
	Example:
	`projectID: "ExampleID123"`

Data Params	`{`
	`  layerIndex : [integer],`
	`  shapeClass : [string]`
	`}`
	
	Example:
	`{`
	`  layerIndex : 0,`
	`  shapeClass : "shapes.Circle"`
	`}`

Success Response	A successful response adds a Shape to the specified Layer on the Canvas.
	
	Example:
	`body : {`
	`  projectID : "exampleID123",`
	`  canvas : {`
	`    width : 200.0,`
	`    height : 200.0,`
	`    layers : [{`
	`      shapes : [{`
	`        shapeClass : "shapes.Circle",`
	`        fillColour : {`
	`          rgbColour : "#FFFFFF",`
	`          opacity : 0.0`
	`        },`
	`        strokeColour : {`
	`          rgbColour : "#000000",`
	`          opacity : 1.0`
	`        },`
	`        strokeWidth : 1.0,`
	`        opacity : 1.0,`
	`        center : {`
	`          x : 50.0,`
	`          y : 50.0`
	`        },`
	`        radius : 50.0`
	`      }],`
	`      visible : true`
	`    }],`
	`    html : "<svg width="200.0" height="200.0" baseProfile="full"`
	`            xmlns="http://www.w3.org/2000/svg"><circle fill="#FFFFFF"`
	`            fill-opacity="0.0" stroke="#000000" stroke-opacity="1.0"`
	`            stroke-width="1.0" opacity="1.0" cx="50.0" cy="50.0"`
	`            r="50.0"></circle></svg>"`
	`  }`
	`}`

Error Response	**Example:**
	`{`
	`  timestamp : "2018-12-31 23:59:59.999",`
	`  status : 500,`
	`  error : "Internal Server Error",`
	`  message : "Class Trapezoid does not exist",`
	`  path : "/addShape/ExampleID123"`

	`}`
Sample Call	`$.ajax({ url : "/addShape/ExampleID123", type : "POST", data : { layerIndex : 0, shapeClass : shapes.Circle }, success : function(r) { console.log(r); } });`

Title	**Edit Shape**
URL	/editShape/:projectID
Method	POST

URL Params

Required:
```
projectID : [string]
```

Example:
```
projectID: "ExampleID123"
```

Data Params

```
{
  layerIndex : [integer],
  shapeIndex : [integer],
  shape : [Shape]
}
```

Example:
```
{
  layerIndex : 0,
  shapeIndex : 0,
  shape : {
    layerIndex : 0,
    shapeIndex : 0,
    shape : {
      shapeClass : "shapes.Circle",
      fillColour : {
        rgbColour : "#AAAAAA",
        opacity : 1.0
      },
      strokeColour : {
        rgbColour : "#AAAAAA",
        opacity : 0.5
      },
      strokeWidth : 2.0,
      opacity : 0.85,
      center : {
        x : 40.0,
        y : 80.0
      },
      radius : 9.5
    }
  }
}
```

Success Response

A successful response sets the attributes of the Shape as specified.

Example:
```
body : {
  projectID : "exampleID123",
  canvas : {
    width : 200.0,
    height : 200.0,
    layers : [{
      shapes : [{
        shapeClass : "shapes.Circle",
        fillColour : {
          rgbColour : "#AAAAAA",
          opacity : 1.0
        },
        strokeColour : {
          rgbColour : "#AAAAAA",
          opacity : 0.5
        },
        strokeWidth : 2.0,
        opacity : 0.85,
        center : {
          x : 40.0,
          y : 80.0
        },
        radius : 9.5
      }],
      visible : true
    }],
    html : "<svg width="200.0" height="200.0" baseProfile="full"
            xmlns="http://www.w3.org/2000/svg"><circle fill="#AAAAAA"
            fill-opacity="1.0" stroke="#AAAAAA" stroke-opacity="0.5"
            stroke-width="2.0" opacity="0.85" cx="40.0" cy="80.0"
            r="9.5"></circle></svg>"
  }
}
```

Error Response

Example:
```
{
  timestamp : "2018-12-31 23:59:59.999",
  status : 500,
  error : "Internal Server Error",
  message : "Shape index out of bounds!",
  path : "/editShape/ExampleID123"
}
```

Sample Call

```
$.ajax({
  url : "/editShape/ExampleID123",
  type : "POST",
  data : {
    layerIndex : 0,
    shapeIndex : 0,
    shape : {
      layerIndex : 0,
      shapeIndex : 0,
      shape : {
        shapeClass : "shapes.Circle",
        fillColour : {
          rgbColour : "#AAAAAA",
          opacity : 1.0
        },
        strokeColour : {
          rgbColour : "#AAAAAA",
          opacity : 0.5
        },
        strokeWidth : 2.0,
        opacity : 0.85,
        center : {
          x : 40.0,
          y : 80.0
        },
        radius : 9.5
      }
    }
  },
  success : function(r) {
    console.log(r);
  }
});
```

Title	Delete Shape
URL	/deleteShape/:projectID
Method	POST

URL Params

Required:
```
projectID : [string]
```

Example:
```
projectID: "ExampleID123"
```

Data Params

```
{
  layerIndex : [integer],
  shapeIndex : [integer]
}
```

Example:
```
{
  layerIndex : 0,
  shapeIndex : 0
}
```

Success Response

A successful response deletes the Shape.

Example:
```
body : {
  projectID : "exampleID123",
  canvas : {
    width : 200.0,
    height : 200.0,
    layers : [{
      shapes : [],
      visible : true
    }],
    html : "<svg width="200.0" height="200.0" baseProfile="full"
            xmlns="http://www.w3.org/2000/svg"></svg>"
  }
}
```

Error Response

Example:
```
{
  timestamp : "2018-12-31 23:59:59.999",
  status : 500,
  error : "Internal Server Error",
  message : "Shape index out of bounds!",
  path : "/deleteShape/ExampleID123"
}
```

Sample Call

```
$.ajax({
  url : "/deleteShape/ExampleID123",
  type : "POST",
  data : {
    layerIndex : 0,
    shapeIndex : 0
  },
  success : function(r) {
    console.log(r);
  }
});
```

Title	**Download As Image**
URL	/download/:projectID/:format
Method	GET

URL Params	**Required:**
	`projectID : [string],`
	`format : "svg" | "png" | "jpg"`
	
	Example:
	`projectID: "ExampleID123",`
	`format : "svg"`

Data Params	None

Success Response	A successful response attaches the image in the desired format.
	
	Example:
	`body : "<svg width="200.0" height="200.0" baseProfile="full"`
	`            xmlns="http://www.w3.org/2000/svg"></svg>"`

Error Response	**Example:**
	`{`
	`  timestamp : "2018-12-31 23:59:59.999",`
	`  status : 500,`
	`  error : "Internal Server Error",`
	`  message : "Unknown format!",`
	`  path : "/download/ExampleID123/gif"`
	`}`

Sample Call	`window.open("/download/ExampleID123/svg");`

Design Patterns

Observer Pattern

We have created an interface `Subject` and an interface `Observer` in the `observer` package. These define the basic functionalities needed by our observers and subjects. In our case, a `Subject` should be able to register and unregister `Observers`, as well as notify them. An `Observer` only needs a method to update itself after being notified.

The concrete classes implementing `Subject` and `Observer` are `main.input.ProjectService` and `persistence.FileManager` respectively.

`ProjectService` is the class that implements all the functionality the API offers, and is thus very closely linked to `RESTHandler`. It stores the projectIDs and the `Canvas` instances linked to them in a `Map` and is able to modify a `Canvas` given a projectID with its various functions. We have decided to make it observable so we could monitor changes to the `Map` without having to worry too much about which classes actually need to know about said changes (other than having to register them as observers). It also manages a seed counter for generating unique projectIDs.

The only concrete `Observer` (as of yet) is `LocalFileManager`, a class that implements the interface `FileManager` (which extends `Observer`) and that stores the `Map` and seed counter stored in `ProjectService` persistently on the local file system of the server (in the concrete case of our implementation, in the `projects` directory). Naturally, the persistent data needs to be updated whenever the `Map` or seed counter managed in `ProjectService` change, which is why we have decided to use the observer pattern for it.

Strategy Pattern

One of the functional requirements of the implementation, being able to download the canvas in SVG format, proved to be extensible by using the strategy pattern.

We created a functional interface `DownloadStrategy` with the function `java.net.URI download(Canvas canvas, String projectID)` and three classes implementing the interface: `DownloadSVG`, `DownloadPNG` and `DownloadJPG`.

Each of the classes does mostly what its name implies. It creates an SVG file from the `Canvas` and stores it in the `projects/<projectID>` directory. In the case of `DownloadPNG` or `DownloadJPG`, it then uses the Batik API to convert the SVG file to a PNG or JPG file respectively. It then returns the URI of the converted file to be processed for download.

The strategy is determined dynamically, depending on the format the API receives in the download URL (see API specification).

Code Metrics

Facts

	Packages/Directories	Classes/Files	Total Lines	Source Code Lines	Comment Lines	Blank Lines
Java	9	35	2737	1346	1020	371
vue	3	12	1084	793	103	188
Sum	**12**	**47**	**3821**	**2139**	**1123**	**559**

Used: “Statistics” - Plugin for Intellij

Bugs 1

Warning	**Nr of occurences**	**Package/Class/Method**
Field injection is not recommended (now fixed)	1	RESTHandler (@Autowired)
Unchecked cast	2	LocalFileManager (java.lang.Object to java.util.Map<java.lang.String,T>)
Loop statement does not loop (now fixed)	1	main.Server
Declaration access can be weaker	14	shape and facilitators package
Declaration can have final modifier	8	Main, canvas, facilitators
Unused declaration	90	All packages
Performance issue: redundant String.toString() (now fixed)	1	Hasher (hashValue.toString())

Used: “Analyze” - Tool of Intellij

Bugs 2

Bug Category	**Bug**	**Nr of occurences**	**Occurence**
Bad Practice	Class is Serializable, but doesn't define serialVersionUID	12	Canvas, facilitators, shapes
Bad Practice	Method may fail to close stream	5	Persistence, download
Internationalization	Reliance on default encoding	3	download
Performance	Method invokes toString() on a String (now fixed)	1	Hasher
Dodgy Code	Dead store to dirsCreated	4	Download, persistence
Dodgy Code	Write to static field from instance method	2	ProjectService (seedCounter)
Dodgy Code	Exception is caught when Exception is not thrown	2	Persistence (LocalFileManager)

Used: "SpotBugs" - Plugin for Maven

Discussion

After using analyzing tools such as the integrated IntelliJ tool or SpotBugs. Some of the Bugs which the IntelliJ tool showed us were already marked in the code as we programmed, but we didn't recognize them or forgot to resolve them.

SpotBugs gave us a more in-depth look at our project. It reminded us of the issues, which we had forgotten about and highlighted new problems, which were hidden from us up until that point. It also provided us with a certain peace of mind to see a definitive list of bugs and issues that were prevalent within our project, which we then would be able to work on and tick off one item after the other from the list.