Appendix C: Source Code

ibia/core/Client.java

```java
package ibia.core;

import java.util.ArrayList;

import ibia.core.entities.Committee;
import ibia.core.entities.Conference;
import ibia.core.entities.Delegate;
import ibia.core.utils.Resolution;
import ibia.core.utils.Topic;

/**
 * Provides a simple interface for interacting
 * with the core backend.
 */
public class Client {
    /**
     * Creates and persists a new Conference instance.
     *
     * @param name name of conference.
     * @return the created Conference
     */
    public static Conference addNewConference(String name) {
        Conference conf = new Conference(name);
        DbDriver.insertOne(conf);
        return conf;
    }

    /**
     * Creates and persists a new Committee instance.
     *
     * @param name name of committee
     * @param conferenceId the parent Conference
     * @return the created Committee
     */
    public static Committee addNewCommittee(String name, String conferenceId) {
        Committee com = new Committee(name, conferenceId);
        DbDriver.insertOne(com);
        return com;
    }

    /**
     * Creates and persists a new Delegate instance.
     *
     * @param name name of delegate
     * @param delegation the country ("delegation") assigned to the delegate
     * @param committeeId the parent Committee
     * @return the created Delegate
     */
    public static Delegate addNewDelegate(String name, String delegation, String
         committeeId) {
        Delegate del = new Delegate(name, delegation, committeeId);
        DbDriver.insertOne(del);
        return del;
    }

    /**
     * @return ArrayList of all persisted Conferences
     */
    public static ArrayList<Conference> getAllConferences() {
        return DbDriver.fetchAll(Conference.class);
    }
```

```
61
62      /**
63       * Fetch Committees belonging to a specific Conference
64       * @param conferenceId ID of the Conference
65       * @return ArrayList of Committees
66       */
67      public static ArrayList<Committee> getConferenceCommittees(String conferenceId) {
68          return DbDriver.findAll(Committee.class, c ->
                c.getConferenceId().equals(conferenceId));
69      }
70
71      /**
72       * Fetch Delegates belonging to a specific Committee
73       * @param committeeId ID of the Committee
74       * @return ArrayList of Delegates
75       */
76      public static ArrayList<Delegate> getCommitteeDelegates(String committeeId) {
77          return DbDriver.findAll(Delegate.class, d ->
                d.getCommitteeId().equals(committeeId));
78      }
79
80      /**
81       * Fetch Topics belonging to a specific Committee
82       * @param committeeId ID of the Committee
83       * @return ArrayList of Topics
84       */
85      public static ArrayList<Topic> getCommitteeTopics(String committeeId) {
86          return DbDriver.findAll(Topic.class, t -> t.getCommitteeId().equals(committeeId));
87      }
88
89      /**
90       * Fetch Resolutions belonging to a specific Topic
91       * @param topicId ID of the Topic
92       * @return ArrayList of Resolutions
93       */
94      public static ArrayList<Resolution> getTopicResolutions(int topicId) {
95          return DbDriver.findAll(Resolution.class, r -> r.getTopicId() == topicId);
96      }
97
98      /**
99       * Deletes a Conference instance from the database,
100      * including it's child Committees and the
101      * Delegates in those committees.
102      * @param confId ID of the Conference
103      */
104     public static void deleteConference(String confId) {
105         deleteConferenceChildren(confId);
106         DbDriver.deleteById(Conference.class, confId);
107     }
108
109     /**
110      * Deletes a Committee instance form the database,
111      * including it's child Delegates.
112      * @param comId ID of the Committee
113      */
114     public static void deleteCommittee(String comId) {
115         deleteCommitteeChildren(comId);
116         DbDriver.deleteById(Committee.class, comId);
117     }
118
119     /**
120      * Deletes a Delegate instance from the database.
121      * @param delId ID of the Delegate
```

```
     */
    public static void deleteDelegate(String delId) {
        DbDriver.deleteById(Delegate.class, delId);
    }

    /**
     * Deletes all child committees belonging to
     * a Conference instance, including the delegates
     * belonging to each of the child committees.
     * @param confId ID of the Conference
     */
    public static void deleteConferenceChildren(String confId) {
        ArrayList<Committee> coms =
            DbDriver.findAll(Committee.class, c -> c.getConferenceId().equals(confId));

        for (Committee com : coms) {
            deleteCommitteeChildren(com.getId());
            DbDriver.deleteOne(com);
        }
    }

    /**
     * Deletes all child delegates belonging to
     * a Committee instance.
     * @param comId ID of the Committee
     */
    public static void deleteCommitteeChildren(String comId) {
        ArrayList<Delegate> dels =
            DbDriver.findAll(Delegate.class, d -> d.getCommitteeId().equals(comId));

        for (Delegate del : dels) {
            DbDriver.deleteOne(del);
        }
    }
}
```

ibia/core/DbDriver.java

```
package ibia.core;

import java.util.ArrayList;
import java.util.Collection;
import java.util.function.Predicate;

import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

/**
 * Handles all database-related operations.
 * All methods are static.
 * Uses an embedded H2 database under the hood,
 * through the Hibernate ORM.
 */
```

```java
public class DbDriver {
    private static StandardServiceRegistry registry;
    private static SessionFactory sessionFactory;

    private static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            try {
                // Create registry
                registry = new StandardServiceRegistryBuilder().configure().build();

                // Create MetadataSources
                MetadataSources sources = new MetadataSources(registry);

                // Create Metadata
                Metadata metadata = sources.getMetadataBuilder().build();

                // Create SessionFactory
                sessionFactory = metadata.getSessionFactoryBuilder().build();

            } catch (Exception e) {
                e.printStackTrace();
                if (registry != null) {
                    StandardServiceRegistryBuilder.destroy(registry);
                }

                throw e;
            }
        }
        return sessionFactory;
    }

    private static Session openSession() {
        return getSessionFactory().openSession();
    }

    /**
     * Persist an entity to the database.
     *
     * @param <T> type of Entity to be persisted.
     * @param entity entity to be persisted.
     */
    public static <T> void insertOne(T entity) {
        Session session = openSession();
        session.beginTransaction();
        session.save(entity);
        session.getTransaction().commit();
        session.close();
    }

    /**
     * Persist a collection of entities
     * to the database.
     *
     * @param <T> type of Entity to be persisted.
     * @param entities collection of entities to be persisted.
     */
    public static <T> void insertAll(Collection<T> entities) {
        Session session = openSession();
        session.beginTransaction();
        for (T entity : entities) session.save(entity);
        session.getTransaction().commit();
        session.close();
    }
```

```
88
89      /**
90       * Update a persisted entity.
91       *
92       * @param <T> type of Entity to be updated.
93       * @param entity updated entity.
94       */
95      public static <T> void updateOne(T entity) {
96          Session session = openSession();
97          session.beginTransaction();
98          session.update(entity);
99          session.getTransaction().commit();
100         session.close();
101     }
102
103     /**
104      * Update a collection of persisted entities.
105      *
106      * @param <T> type of Entity to be updated.
107      * @param entities collection of updated entities.
108      */
109     public static <T> void updateAll(Collection<T> entities) {
110         Session session = openSession();
111         session.beginTransaction();
112         for (T entity : entities) session.update(entity);
113         session.getTransaction().commit();
114         session.close();
115     }
116
117     /**
118      * Delete an entity from the database.
119      *
120      * @param <T> type of Entity to be deleted.
121      * @param entity entity to be deleted.
122      */
123     public static <T> void deleteOne(T entity) {
124         Session session = openSession();
125         session.beginTransaction();
126         session.delete(entity);
127         session.getTransaction().commit();
128         session.close();
129     }
130
131     /**
132      * Delete an entity from the database
133      * using the id
134      * @param <T> type of entity to be deleted
135      * @param entityClass Class of the entity to be deleted
136      * @param id id of the entity to be deleted
137      */
138     public static <T> void deleteById(Class<T> entityClass, Object id) {
139         T entity = DbDriver.fetchOne(entityClass, id);
140         deleteOne(entity);
141     }
142
143     /**
144      * Delete a collection of entities from the database.
145      *
146      * @param <T> type of Entity to be deleted.
147      * @param entities collection of entities to be deleted.
148      */
149     public static <T> void deleteAll(Collection<T> entities) {
150         Session session = openSession();
```

```
151         session.beginTransaction();
152         for (T entity : entities) session.delete(entity);
153         session.getTransaction().commit();
154         session.close();
155     }
156 
157     /**
158      * Fetch/read a persisted entity from the database.
159      *
160      * @param <T> type of Entity to be fetched.
161      * @param entityClass Class of the entity to be fetched.
162      * @param id ID of the entity being fetched.
163      * @return the fetched entity, or null if the entity does not exist.
164      */
165     public static <T> T fetchOne(Class<T> entityClass, Object id) {
166         Session session = openSession();
167         session.beginTransaction();
168         T entity = (T)session.find(entityClass, id);
169         session.close();
170         return entity;
171     }
172 
173     /**
174      * Fetch/read all persisted entities of a particular
175      * type from the database.
176      *
177      * @param <T> type of entity to be fetched.
178      * @param entityClass Class of the entities being fetched.
179      * @return ArrayList of fetched entities, or null if none were found.
180      */
181     public static <T> ArrayList<T> fetchAll(Class<T> entityClass) {
182         Session session = openSession();
183         session.beginTransaction();
184         CriteriaBuilder cb = session.getCriteriaBuilder();
185         CriteriaQuery<T> cq = cb.createQuery(entityClass);
186         Root<T> rootEntry = cq.from(entityClass);
187         CriteriaQuery<T> all = cq.select(rootEntry);
188 
189         TypedQuery<T> allQuery = session.createQuery(all);
190         ArrayList<T> results = (ArrayList<T>)allQuery.getResultList();
191         session.close();
192         return results.size() > 0 ? results : null;
193     }
194 
195     /**
196      * Fetches all entities of a particular type and
197      * finds the first one satisfying a given predicate.
198      * For example, to find a delegate with the name "ABC":
199      *
200      * @param <T> type of entity to be fetched.
201      * @param entityClass Class of the entities being fetched.
202      * @param filter predicate for searching through the fetched entities.
203      * @return the first entity found to satisfy the predicate, or null if none were found.
204      */
205     public static <T> T findOne(Class<T> entityClass, Predicate<T> filter) {
206         ArrayList<T> entities = fetchAll(entityClass);
207         if (entities != null) {
208             for (T entity : entities) {
209                 if (filter.test(entity)) return entity;
210             }
211         }
212         return null;
213     }
```

```
214
215     /**
216      * Fetches all entities of a particular type and
217      * finds all that satisfy a given predicate.
218      *
219      * @param <T> type of entity to be fetched.
220      * @param entityClass Class of the entities being fetched.
221      * @param filter predicate for searching through the entities.
222      * @return ArrayList of entities found to satisfy the predicate, or null if none were
           found.
223      */
224     public static <T> ArrayList<T> findAll(Class<T> entityClass, Predicate<T> filter) {
225         ArrayList<T> found = new ArrayList<>();
226         ArrayList<T> entities = fetchAll(entityClass);
227         if (entities != null) {
228             for (T entity : entities) {
229                 if (filter.test(entity)) found.add(entity);
230             }
231         }
232         return found.size() > 0 ? found : null;
233     }
234
235     /**
236      * Destroy database service registry.
237      */
238     public static void shutdown() {
239         if (registry != null) {
240             StandardServiceRegistryBuilder.destroy(registry);
241         }
242     }
243 }
```

ibia/core/Log.java

```
1  package ibia.core;
2
3  import java.io.File;
4  import java.nio.file.FileSystems;
5  import java.util.logging.FileHandler;
6  import java.util.logging.Handler;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import java.util.logging.SimpleFormatter;
10
11 /**
12  * Logging utility.
13  */
14 public class Log {
15     private static Logger logger;
16     private static Handler handler;
17
18     public static Logger getLogger() {
19         if (logger == null) {
20             logger = Logger.getLogger("");
21             logger.setUseParentHandlers(false); // No need to send output to parent loggers
22
23             try {
24                 // Create the file if it doesn't exist
25                 String path = getLogFilePath();
26                 File file = new File(path);
27                 if (!file.isFile()) {
28                     file.getParentFile().mkdirs();
```

```
                    file.createNewFile();
                }

                // Set the formatter and add handler to logger
                SimpleFormatter fmt = new SimpleFormatter();
                handler = new FileHandler(path, true);
                handler.setFormatter(fmt);
                logger.addHandler(handler);

                // Indicate a new log session is starting.
                info("= = = = = SESSION START = = = = =");
            } catch (Exception e) {
                e.printStackTrace();
                System.out.println("WARN: [ibia] Failed to access log file
                    (data/ibia.log).");
            }
        }

        return logger;
    }

    public static void info(String msg) {
        msg = "[ibia] " + msg;
        getLogger().info(msg);
    }

    public static void warn(String msg) {
        msg = "[ibia] " + msg;
        getLogger().log(Level.WARNING, msg);
    }

    public static void error(String msg) {
        msg = "[ibia] " + msg;
        getLogger().log(Level.SEVERE, msg);
    }

    public static Handler getHandler() {
        return logger.getHandlers()[0];
    }

    /**
     * @return the absolute path for the log file.
     */
    public static String getLogFilePath() {
        // Current Working Directory
        String cwd = System.getProperty("user.dir");
        // Join paths to form the absolute path for the Log file.
        // This should be the same as the ibia.db file created by Hibernate.
        String path = FileSystems.getDefault().getPath(cwd, "data", "ibia.log").toString();
        return path;
    }
}
```

ibia/core/utils/Country.java

```
package ibia.core.utils;

import java.io.File;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.nio.file.Files;
```

```java
import java.nio.file.Path;
import java.nio.file.Paths;
import java.net.URI;
import java.net.URL;
import java.util.ArrayList;

import ibia.core.Log;
import com.google.gson.Gson;

/**
 * Utility class for dealing with country names and flags.
 * All methods are static.
 * All data is obtained from the repository at:
     https://github.com/stefangabos/world_countries
 */
public class Country {
    private static CountryData[] data;
    private static ArrayList<String> names = new ArrayList<>();
    private static ArrayList<String> codes = new ArrayList<>();
    private static String dataPath = "world-countries/data/en/world.json";

    /**
     * Data for countries read from a json file
     * is converted to instances of this class.
     */
    public static class CountryData {
        public int id;
        public String name;
        public String alpha2;
        public String alpha3;
    }

    /**
     * On the first call to this method, it
     * reads the raw data from a json file
     * and caches it into an array of
     * CountryData. On subsequent calls, it
     * returns the cached array. If the read
     * operation was unsuccessful, it returns
     * null and tries again on the next call.
     *
     * @return An array containing data for each territory
     */
    public static CountryData[] getData() {
        if (data == null) {
            try {
                ClassLoader classLoader = Country.class.getClassLoader();
                InputStream stream = classLoader.getResourceAsStream(dataPath);

                int b;
                StringBuilder str = new StringBuilder();
                while ((b = stream.read()) != -1) {
                    str.append((char)b);
                }

                stream.close();
                String json = str.toString();
                data = new Gson().fromJson(json, CountryData[].class);
            } catch (Exception e) {
                Log.error(e.getMessage());
                e.printStackTrace();
                data = null; // Set data back to null to try again on the next call.
                return null;
```

```
            }
        }
        return data;
    }

    /**
     * Obtain a list of the full names for all 249 territories
     * that have an officially assigned ISO 3166-1 code.
     *
     * @return A list of names, or null if data is not available
     */
    public static ArrayList<String> listOfNames() {
        if (getData() == null) return null;
        if (names.size() == 0) {
            for (CountryData data : getData()) {
                names.add(data.name);
            }
        }
        return names;
    }

    /**
     * Obtain a list of the alpha-2 codes for all 249 territories
     * that have an officially assigned ISO 3166-1 code.
     *
     * @return A list of alpha2 codes, or null if data is not available
     */
    public static ArrayList<String> listOfCodes() {
        if (getData() == null) return null;
        if (codes.size() == 0) {
            for (CountryData data : getData()) {
                codes.add(data.alpha2);
            }
        }
        return codes;
    }

    /**
     * Returns the alpha-2 code for a territory,
     * given its full name.
     *
     * @param name - The name of the country
     * @return The alpha2 associated with the given country name, or null if none is found.
     */
    public static String codeFromName(String name) {
        if (getData() == null) return null;
        for (CountryData country : getData()) {
            if (country.name.equals(name)) return country.alpha2;
        }
        return null;
    }

    /**
     * Return the full name for a territory,
     * given its alpha-2 code.
     *
     * @param code - An alpha2 code
     * @return The name of the country associated with the given alpha2 code, or null if
         none is found.
     */
    public static String nameFromCode(String code) {
        if (getData() == null) return null;
        for (CountryData country : getData()) {
```

```
            if (country.alpha2.equals(code)) return country.name;
        }
        return null;
    }

    /**
     * Obtain a relative path to the .png file
     * for the country with the given alpha2 code.
     *
     * @param code - An alpha2 code
     * @return The path to the flag, or null if an invalid code is given.
     */
    public static InputStream getFlag(String code) {
        if (listOfCodes().contains(code)) {
            String fileName = code.toLowerCase() + ".png";
            String resource = "world-countries/flags/64x64/" + fileName;
            InputStream flag = Country.class.getClassLoader().getResourceAsStream(resource);

            return flag;
        }
        return null;
    }
}
```

ibia/core/utils/Id.java

```
package ibia.core.utils;

import java.util.Date;

import ibia.core.entities.EntityType;

/**
 * Utility class for dealing with IDs used
 * for entities throughout the application.
 */
public class Id {

    /**
     * Generates an ID based on the provided type,
     * used as a prefix for the ID.
     *
     * @param type - The EntityType for which to generate an ID.
     * @return An ID string
     */
    public static String generate(EntityType type) {
        String ts = Long.toString(System.currentTimeMillis());
        switch (type) {
            case COM:
                return "COM" + ts;
            case CON:
                return "CON" + ts;
            case DEL:
                return "DEL" + ts;
            default:
                return "ENT" + ts;
        }
    }

    /**
     * Obtain the creation Date from a given ID.
     *
```

```java
     * @param id A valid ID string
     * @return The Date when the ID was generated.
     * @throws IllegalArgumentException - if an invalid ID is provided
     */
    public static Date createdAt(String id) throws IllegalArgumentException {
        if (verify(id)) {
            Long ts = Long.parseLong(id.substring(3));
            return new Date(ts);
        } else {
            throw new IllegalArgumentException("Invalid ID provided.");
        }
    }

    /**
     * Verifies IDs based on the following checks:<br><br>
     *
     * - The prefix of the id is one of COM, CON, DEL or ENT.<br><br>
     * - The suffix can be parsed into a valid Date object.<br><br>
     * - The parsed Date is between January 1, 2020 and the current time.<br><br>
     *
     * @param id - The ID string to verify.
     * @return true if all the checks passed, otherwise false.
     */
    public static boolean verify(String id) {
        String prefix = id.substring(0, 3);
        String ts = id.substring(3);

        try {
            // Make sure ID was created between NOW and January 1, 2020
            // otherwise its obviously not a properly generated ID.
            Date created = new Date(Long.parseLong(ts));

            Date current = new Date(System.currentTimeMillis());
            Date epoch = new Date(1577836800000L); // January 1, 2020

            if (current.compareTo(created) < 0) return false;
            if (epoch.compareTo(created) > 0) return false;
        } catch (NumberFormatException e) {
            return false;
        }

        return prefix.equals("COM")
            || prefix.equals("CON")
            || prefix.equals("DEL")
            || prefix.equals("ENT");
    }
}
```

ibia/core/utils/Resolution.java

```java
package ibia.core.utils;

/**
 * Represents a Committee resolution.
 * <br><br>
 * This class is also a Hibernate entity.
 */
public class Resolution {
    private int id;
    private String mainSubmitter;
    private int topicId;
    private boolean passed;
```

```
13
14      public Resolution() {}
15
16      public Resolution(String mainSubmitter, int topicId) {
17          this.mainSubmitter = mainSubmitter;
18          this.topicId = topicId;
19          this.passed = false;
20      }
21
22      public int getId() {
23          return id;
24      }
25
26      public void setId(int id) {
27          this.id = id;
28      }
29
30      public String getMainSubmitter() {
31          return mainSubmitter;
32      }
33
34      public void setMainSubmitter(String mainSubmitter) {
35          this.mainSubmitter = mainSubmitter;
36      }
37
38      public int getTopicId() {
39          return topicId;
40      }
41
42      public void setTopicId(int topicId) {
43          this.topicId = topicId;
44      }
45
46      public boolean getPassed() {
47          return passed;
48      }
49
50      public void setPassed(boolean passed) {
51          this.passed = passed;
52      }
53  }
```

ibia/core/utils/Topic.java

```
1   package ibia.core.utils;
2
3   /**
4    * Represents a Committee topic.
5    * <br><br>
6    * This class is also a Hibernate entity.
7    */
8   public class Topic {
9       private int id;
10      private String committeeId;
11      private String topic;
12
13      public Topic() {}
14
15      public Topic(String committeeId, String topic) {
16          this.committeeId = committeeId;
17          this.topic = topic;
18      }
```

```
19
20      public int getId() {
21          return id;
22      }
23
24      public void setId(int id) {
25          this.id = id;
26      }
27
28      public String getCommitteeId() {
29          return committeeId;
30      }
31
32      public void setCommitteeId(String id) {
33          this.committeeId = id;
34      }
35
36      public String getTopic() {
37          return topic;
38      }
39
40      public void setTopic(String topic) {
41          this.topic = topic;
42      }
43  }
```

ibia/core/entities/Committee.java

ibia/core/entities/Conference.java

```
1   package ibia.core.entities;
2
3   import java.util.Date;
4
5   import ibia.core.utils.Id;
6
7   /**
8    * Represents an MUN conference.
9    */
10  public class Conference implements Entity {
11      private final EntityType type = EntityType.CON;
12      private String id;
13      private String name;
14      private boolean ongoing;
15      private Date created;
16
17      /**
18       * This constructor is used internally by Hibernate
19       * and MUST NOT be used in client-facing code.
20       */
21      public Conference() {}
22
23      public Conference(String name) {
24          this.id = Id.generate(type);
25          this.name = name;
26          this.ongoing = true;
27          this.created = new Date();
28      }
29
30      public EntityType getType() {
```

```
31          return type;
32      }
33
34      /**
35       * Whether the conference is ongoing (active) or not.
36       * The default value is true when a conference is
37       * instantiated.
38       *
39       * @return true if the conference is active, otherwise false.
40       */
41      public boolean isOngoing() {
42          return ongoing;
43      }
44
45      /* GETTERS and SETTERS used by hibernate */
46
47      public String getId() {
48          return id;
49      }
50
51      public void setId(String id) {
52          this.id = id;
53      }
54
55      public String getName() {
56          return name;
57      }
58
59      public void setName(String name) {
60          this.name = name;
61      }
62
63      public boolean getOngoing() {
64          return ongoing;
65      }
66
67      public void setOngoing(boolean ongoing) {
68          this.ongoing = ongoing;
69      }
70
71      public Date getCreated() {
72          return created;
73      }
74
75      public void setCreated(Date created) {
76          this.created = created;
77      }
78  }
```

ibia/core/entities/Delegate.java

```
1   package ibia.core.entities;
2
3   import ibia.core.utils.Country;
4   import ibia.core.utils.Id;
5
6   /**
7    * Represents a delegate within an MUN committee.
8    */
9   public class Delegate implements Entity {
10      private final EntityType type = EntityType.DEL;
11      private String id;
```

```
12      private String name;
13      private String delegation; // An alpha2 country code OR a custom delegation. The
            country code is used to fetch the flag icon.
14      private String committeeId;
15      private int speeches;
16      private int pois;
17      private int motions;
18      private int time;
19      private int amendments;
20
21      /**
22       * This constructor is used internally by Hibernate
23       * and MUST NOT be used in client-facing code.
24       */
25      public Delegate() {}
26
27      public Delegate(String name, String delegation, String committeeId) {
28          this.id = Id.generate(type);
29          this.name = name;
30          this.delegation = delegation;
31          this.committeeId = committeeId;
32          this.speeches = 0;
33          this.pois = 0;
34          this.motions = 0;
35          this.time = 0;
36          this.amendments = 0;
37      }
38
39      public EntityType getType() {
40          return type;
41      }
42
43      /**
44       * Check whether the delegate has a custom delegation
45       * or not. A delegation is considered custom if and
46       * only if its value is a valid alpha2 territory code.
47       * This check should be used to determine whether or
48       * not a flag icon is available for a delegation.
49       *
50       * @return true if the delegation is custom, otherwise false.
51       */
52      public boolean hasCustomDelegation() {
53          return !Country.listOfCodes().contains(delegation);
54      }
55
56      /* GETTERS and SETTERS used by hibernate */
57
58      public String getId() {
59          return id;
60      }
61
62      public void setId(String id) {
63          this.id = id;
64      }
65
66      public String getName() {
67          return name;
68      }
69
70      public void setName(String name) {
71          this.name = name;
72      }
73
```

```
    public String getDelegation() {
        return delegation;
    }

    public void setDelegation(String delegation) {
        this.delegation = delegation;
    }

    public String getCommitteeId() {
        return committeeId;
    }

    public void setCommitteeId(String committeeId) {
        this.committeeId = committeeId;
    }

    public int getSpeeches() {
        return speeches;
    }

    public void setSpeeches(int speeches) {
        this.speeches = speeches;
    }

    public int getPois() {
        return pois;
    }

    public void setPois(int pois) {
        this.pois = pois;
    }

    public int getMotions() {
        return motions;
    }

    public void setMotions(int motions) {
        this.motions = motions;
    }

    public int getTime() {
        return time;
    }

    public void setTime(int time) {
        this.time = time;
    }

    public int getAmendments() {
        return amendments;
    }

    public void setAmendments(int amendments) {
        this.amendments = amendments;
    }
}
```

ibia/core/entities/Entity.java

```
package ibia.core.entities;

/**
```

```
 * An interface representing any generic entity
 * that posesses a certain type and ID and represents
 * an aspect of a(n) MUN conference. NOTE that these
 * entities do NOT refer to Hibernate entities.
 */
public interface Entity {
    public EntityType getType();

    /* GETTERS and SETTERS used by hibernate */
    public String getId();
    public void setId(String id);

    public String getName();
    public void setName(String name);
}
```

ibia/core/entities/EntityType.java

```
package ibia.core.entities;

/**
 * Enum mapping over the three possible
 * types of entities used throughout
 * ibia:
 * <ul>
 *   <li>COM: Committees,</li>
 *   <li>CON: Conferenes,</li>
 *   <li>DEL: Delegates,</li>
 *   <li>ENT: Any generic entity that implements the Entity interface.</li>
 * </ul>
 */
public enum EntityType {
    COM,
    CON,
    DEL,
    ENT
}
```

ibia/app/App.java

```
package ibia.app;

import javafx.application.Application;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;

import java.io.FileNotFoundException;
import java.io.IOException;

import ibia.core.Log;

/**
 * The main class for ibia.app. Controls
 * the main application stage, and handles
 * navigation logic.
 */
public class App extends Application {
    /**
```

```
22      * Icon for most windows created by ibia.
23      */
24     public static Image IBIA_ICON = new Image("/images/ibia-icon2.png");
25
26     /**
27      * Main application window
28      */
29     private static Stage window;
30
31     /**
32      * The scene being displayed on the main stage
33      */
34     private static Scene scene;
35
36     /**
37      * Indicates the current scene being displayed.
38      * The value is always an entity ID or "Home" (for the
39      * welcome screen).
40      */
41     private static String location;
42
43     public static void main(String[] args) {
44         Log.info("Initializing JavaFX stage.");
45         launch(args);
46     }
47
48     /**
49      * Loads main stage and starts application.
50      *
51      * @param stage the main Stage
52      */
53     @Override
54     public void start(Stage stage) throws FileNotFoundException, IOException {
55         try {
56             // First, set a scene to load initially
57             Pane root = new Pane();
58             root.setMinWidth(1000);
59             root.setMinHeight(600);
60             scene = new Scene(root);
61
62             // Load that scene and show the window
63             window = stage;
64             window.setTitle("ibia");
65             window.getIcons().add(IBIA_ICON);
66             window.setMinWidth(1000);
67             window.setMinHeight(600 + 39); // Accomodate for title bar because JavaFX
                   doesn't
68             window.setScene(scene);
69             window.show();
70
71             // Then navigate to the Home screen, and start the application.
72             App.navigate("Home");
73         } catch (Exception e) {
74             Log.error("Failed to load main window: " + e.getMessage());
75             e.printStackTrace();
76             window.close();
77             System.exit(1);
78         }
79     }
80
81     /**
82      * Updates the scene on the application stage.
83      *
```

```
84          * @param scene the scene to display.
85          */
86         private static void updateScene(Parent content) {
87             scene.setRoot(content);
88         }
89 
90         /**
91          * @return current location of the main application stage
92          */
93         public static String getLocation() {
94             return location;
95         }
96 
97         /**
98          * Sets the internal stage location variable.
99          * This is controlled by the App class. To change
100         * the application's stage location, use App.navigate()
101         *
102         * @param newLocation
103         */
104        private static void setLocation(String newLocation) {
105            location = newLocation;
106        }
107 
108        /**
109         * Changes the scene displayed on the main application
110         * stage based on the id given. The id can
111         * be an entity ID, in which case the ID's corresponding
112         * view will be loaded. The id may also be the string
113         * "Home", in which case the Home scene will be loaded.
114         *
115         * @param id specifies where to navigate
116         * @throws IllegalArgumentException - if an invalid id is provided
117         * @throws IOException - if loading FXML fails
118         */
119        public static void navigate(String id) throws IllegalArgumentException, IOException {
120            // If navigating to the same location, no need to update the scsene.
121            if (location != null && location.equals(id)) return;
122            // Otherwise, update App.location
123            setLocation(id);
124            if (id.equals("Home")) {
125                Parent home = SceneUtil.loadFXML("Home", true);
126                updateScene(home);
127            } else {
128                String entity = id.substring(0, 3);
129                switch (entity) {
130                    case "CON":
131                        Parent conferenceScene = SceneUtil.loadFXML("ConferenceView", true);
132                        updateScene(conferenceScene);
133                        break;
134                    case "COM":
135                        // Do not cache so that the delegate list updates properly
136                        // every time.
137                        Parent committeeScene = SceneUtil.loadFXML("CommitteeView", false);
138                        updateScene(committeeScene);
139                        break;
140                    case "DEL":
141                        Parent delegateScene = SceneUtil.loadFXML("DelegateView", true);
142                        updateScene(delegateScene);
143                        break;
144                    default:
145                        throw new IllegalArgumentException(
146                            "Invalid location provided: " + id +
```

```
                        "\nLocation must be an entity ID or 'Home'."
                        );
            }
        }
    }

    /**
     * Refreshes the current scene displayed by reloading it to the stage.
     * @throws IOException - if loading FXML fails
     */
    public static void refresh() throws IOException {
        String tmp = location;
        setLocation("none");
        App.navigate(tmp);
    }
}
```

ibia/app/SceneUtil.java

```
package ibia.app;

import java.io.IOException;
import java.util.HashMap;

import ibia.core.Log;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import javafx.stage.Modality;
import javafx.stage.Stage;

/**
 * Utility class for loading scenes, specifically
 * from fxml files.
 *
 */
public final class SceneUtil {

    /**
     * A private instance so as to provide
     * an easier API with static methods.
     */
    private static SceneUtil instance = new SceneUtil();

    /**
     * Private constructor, to reject instantiation since
     * this class acts as a singleton.
     */
    private SceneUtil() {};

    /**
     * Cache for holding loaded FXML files.
     */
    private HashMap<String, Parent> cache = new HashMap<>();

    /**
     * Loads and returns the content defined by
     * the specified fxml file, located in
```

```
44          * resources/fxml
45          *
46          * @param name the name of the fxml file (without the extension)
47          * @param useCache whether to cache to loaded fxml or not. In some cases
48          * it may be preferable not to cache (such as when using the same fxml
49          * multiple times within the same scene).
50          * @return The scene loaded from the fxml file
51          * @throws IOException - if loading FXML fails
52          */
53         public static Parent loadFXML(String name, boolean useCache) throws IOException {
54             return instance._loadFXML(name, useCache);
55         }
56
57         /**
58          * Same as loadFXML, but puts the FXML into a Scene
59          * and returns the Scene.
60          *
61          * @param name name of the FXML file
62          * @param useCache whether to cache the loaded FXML or not
63          * @return the Scene created from the FXML
64          * @throws IOException - if loading FXML fails
65          */
66         public static Scene loadFXMLScene(String name, boolean useCache) throws IOException {
67             return instance._loadFXMLScene(name, useCache);
68         }
69
70         /**
71          * Useful for quickly loading FXML with default settings
72          * to function as a popup window.
73          *
74          * @param name name of the FXML file
75          * @param title title for the popup window
76          * @return the Stage created from the FXML
77          * @throws IOException - if loading FXML fails
78          */
79         public static Stage loadPopupStage(String name, String title) throws IOException {
80             return instance._loadPopupStage(name, title);
81         }
82
83         /**
84          * Returns a Stage for displaying errors.
85          *
86          * @param msg the error msg
87          * @return the created Stage
88          */
89         public static Stage error(String msg) {
90             return instance._error(msg);
91         }
92
93         /**
94          * Returns a stage showing a confirmation message,
95          * along with Confirm and Cancel buttons.
96          *
97          * @param msg the confirmation message
98          * @return the created Stage
99          */
100        public static Stage confirm(String msg) {
101            return instance._confirm(msg);
102        }
103
104        /**
105         * Implementation for Util.loadFXML
106         */
```

```
107     private Parent _loadFXML(String name, boolean useCache) throws IOException {
108         if (useCache && cache.containsKey(name)) return cache.get(name);
109 
110         name = name.endsWith(".fxml") ? name : name + ".fxml";
111         FXMLLoader loader = new FXMLLoader();
112         loader.setLocation(getClass().getResource("/fxml/" + name));
113         Parent content = loader.load();
114 
115         if (useCache) cache.put(name, content);
116         return content;
117     }
118 
119     /**
120      * Implementation for Util.loadFXMLScene
121      */
122     private Scene _loadFXMLScene(String name, boolean useCache) throws IOException {
123         Parent root = _loadFXML(name, useCache);
124         return new Scene(root);
125     }
126 
127     /**
128      * Implementation for Util.loadPopupStage
129      */
130     private Stage _loadPopupStage(String name, String title) throws IOException {
131         Stage stage = new Stage();
132         Scene scene = _loadFXMLScene(name, true);
133 
134         stage.setScene(scene);
135         stage.setTitle(title);
136         stage.initModality(Modality.APPLICATION_MODAL);
137         stage.getIcons().add(App.IBIA_ICON);
138         stage.setResizable(false);
139         return stage;
140     }
141 
142     /**
143      * Implementation for SceneUtil.error
144      */
145     private Stage _error(String msg) {
146         try {
147             Stage stage = new Stage();
148             // Load fxml file
149             Scene scene = _loadFXMLScene("Error", true);
150             // Cast parent to pane, so that we can access the child node
151             Pane pane = (Pane)scene.getRoot();
152             // get child node and cast it to Vbox
153             VBox vbox = (VBox)(pane.getChildren().get(0));
154             // repeat to get to Text
155             Text text = (Text)(vbox.getChildren().get(0));
156 
157 
158             text.setText(msg);
159             stage.setTitle("Error:");
160             stage.getIcons().add(new Image("/images/red-circle.png"));
161             stage.setScene(scene);
162             stage.setResizable(false);
163 
164             // Makes it so that error popup must be dealt
165             // with before being able to interact with the
166             // rest of the app
167             stage.initModality(Modality.APPLICATION_MODAL);
168 
169             return stage;
```

```
        } catch (Exception e) {
            // If there is an error, log it and stop the application.
            Log.error(e.getMessage());
            System.exit(1);
            return null;
        }
    }

    /**
     * Implementation for SceneUtil.confirm
     */
    private Stage _confirm(String msg) {
        try {
            Stage stage = new Stage();
            // Load fxml file
            Scene scene = _loadFXMLScene("Confirm", true);
            // Cast parent to pane, so that we can access the child node
            Pane pane = (Pane)scene.getRoot();
            // get child node and cast it to Vbox
            VBox vbox = (VBox)(pane.getChildren().get(0));
            // repeat to get to Text
            Text text = (Text)(vbox.getChildren().get(0));

            text.setText(msg);
            stage.setTitle("Confirm:");
            stage.getIcons().add(new Image("/images/yellow-circle.png"));
            stage.setScene(scene);
            stage.setResizable(false);

            // Makes it so that error popup must be dealt
            // with before being able to interact with the
            // rest of the app
            stage.initModality(Modality.APPLICATION_MODAL);

            return stage;
        } catch (Exception e) {
            // If there is an error, log it and stop the application.
            Log.error(e.getMessage());
            System.exit(1);
            return null;
        }
    }
}
```

ibia/app/controllers/CommitteeListItem.java

```
package ibia.app.controllers;

import java.io.IOException;

import ibia.app.App;
import javafx.fxml.FXML;
import javafx.scene.input.MouseEvent;
import javafx.scene.text.Text;

public class CommitteeListItem {
    @FXML protected Text id;

    @FXML
    protected void navigate(MouseEvent event) throws IOException, IllegalArgumentException
        {
        String comId = id.getText().substring(1);
```

```
        App.navigate(comId);
    }

    @FXML
    protected void hoverEffectOn(MouseEvent event) {
        Text text = (Text)event.getTarget();
        text.setUnderline(true);
    }

    @FXML
    protected void hoverEffectOff(MouseEvent event) {
        Text text = (Text)event.getTarget();
        text.setUnderline(false);
    }
}
```

ibia/app/controllers/CommitteeView.java

```
package ibia.app.controllers;

import java.io.IOException;
import java.util.ArrayList;

import ibia.app.App;
import ibia.app.SceneUtil;
import ibia.core.Client;
import ibia.core.DbDriver;
import ibia.core.Log;
import ibia.core.entities.Committee;
import ibia.core.entities.Conference;
import ibia.core.entities.Delegate;

import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class CommitteeView {
    @FXML protected Text conferenceCrumb;
    @FXML protected Text committeeCrumb;
    @FXML protected Text name;
    @FXML protected Text id;
    @FXML protected VBox delegatesList;

    private Committee instance;

    @FXML
    public void initialize() throws IOException {
        this.instance = DbDriver.fetchOne(Committee.class, App.getLocation());

        fillBreadcrumbs();
        fillName();
        fillId();
        fillDelegatesList();
    }
```

```
45
46      /*********************/
47      /*** FXML Controls ***/
48      /*********************/
49
50      /**
51       * @param event the MouseEvent instance
52       */
53      @FXML
54      protected void handleDeleteAction(MouseEvent event) {
55          Stage stage = SceneUtil.confirm("Are you sure you wish to delete this committee?
                 This action cannot be reversed.");
56          Scene root = stage.getScene();
57          Button cancel = (Button)root.lookup("#cancel");
58          Button confirm = (Button)root.lookup("#confirm");
59
60          cancel.setOnMouseClicked(evt -> {
61              stage.close();
62          });
63
64          confirm.setOnMouseClicked(evt -> {
65              Client.deleteCommittee(instance.getId());
66              try {
67                  App.navigate(instance.getConferenceId());
68                  stage.close();
69              } catch (Exception e) {
70                  // If Home failed to load, this is a fatal error
71                  // and the application is exited.
72                  Log.error(e.getMessage());
73                  System.exit(1);
74              }
75          });
76
77          stage.show();
78      }
79
80      @FXML
81      protected void handleEditAction(MouseEvent event) throws IOException {
82          Stage stage = SceneUtil.loadPopupStage("EditCommittee", "Edit Committee");
83          stage.show();
84      }
85
86      @FXML
87      protected void openTimer(MouseEvent event) throws IOException {
88          Stage stage = SceneUtil.loadPopupStage("SpeechTimer", "Timer");
89          stage.show();
90      }
91
92      @FXML
93      protected void addNewDelegate(MouseEvent event) throws IOException {
94          Stage stage = SceneUtil.loadPopupStage("NewDelegate", "Create new Delegate");
95          stage.show();
96      }
97
98      @FXML
99      protected void openStats(MouseEvent event) {
100         SceneUtil.error("Unimplemented!").show();
101     }
102
103     @FXML
104     protected void openTopics(MouseEvent event) throws IOException {
105         Stage stage = SceneUtil.loadPopupStage("Topics", "Topics");
106         stage.show();
```

```
107     }
108
109     @FXML
110     protected void openResolutions(MouseEvent event) throws IOException {
111         Stage stage = SceneUtil.loadPopupStage("Resolutions", "Resolutions");
112         stage.show();
113     }
114
115     @FXML
116     protected void navigateHome(MouseEvent event) throws IOException,
            IllegalArgumentException {
117         App.navigate("Home");
118     }
119
120     @FXML
121     protected void navigateConference(MouseEvent event) throws IOException,
            IllegalArgumentException {
122         String id = instance.getConferenceId();
123         App.navigate(id);
124     }
125
126     @FXML
127     protected void crumbHoverEffectOn(MouseEvent event) {
128         Text text = (Text)event.getTarget();
129         text.setUnderline(true);
130     }
131
132     @FXML
133     protected void crumbHoverEffectOff(MouseEvent event) {
134         Text text = (Text)event.getTarget();
135         text.setUnderline(false);
136     }
137
138     /******************/
139     /*** Templating ***/
140     /******************/
141
142     private void fillBreadcrumbs() {
143         Conference conf = DbDriver.fetchOne(Conference.class, instance.getConferenceId());
144         conferenceCrumb.setText(conf.getName());
145         committeeCrumb.setText(instance.getName());
146     }
147
148     private void fillName() {
149         name.setText(instance.getName());
150     }
151
152     private void fillId() {
153         id.setText(instance.getId());
154     }
155
156     private void fillDelegatesList() throws IOException {
157         String comId = instance.getId();
158         ArrayList<Delegate> dels = DbDriver.findAll(Delegate.class, d ->
                d.getCommitteeId().equals(comId));
159         if (dels == null) return;
160
161         ObservableList<Node> list = delegatesList.getChildren();
162         for (Delegate del : dels) {
163             FXMLLoader loader = new
                    FXMLLoader(getClass().getResource("/fxml/DelegateListItem.fxml"));
164             Parent root = loader.load();
165             DelegateListItem controller = loader.getController();
```

```
            controller.setInstanceId(del.getId());

            list.add(root);
        }
    }
}
```

ibia/app/controllers/ConferenceListItem.java

```
package ibia.app.controllers;

import java.io.IOException;

import ibia.app.App;
import javafx.fxml.FXML;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundFill;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Paint;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class ConferenceListItem {
    @FXML protected HBox container;
    @FXML protected Text id;

    @FXML
    protected void hoverItemEffectOn(MouseEvent event) {
        BackgroundFill bgFill = new BackgroundFill(Paint.valueOf("#363648"), null, null);
        Background bg = new Background(bgFill);
        container.setBackground(bg);
    }

    @FXML
    protected void hoverItemEffectOff(MouseEvent event) {
        BackgroundFill bgFill = new BackgroundFill(null, null, null);
        Background bg = new Background(bgFill);
        container.setBackground(bg);
    }

    @FXML
    protected void navigate(MouseEvent event) throws IOException, IllegalArgumentException
         {
        String confId = id.getText().substring(1);
        App.navigate(confId);
        Stage stage = (Stage)container.getScene().getWindow();
        stage.close();
    }
}
```

ibia/app/controllers/ConferenceView.java

```
package ibia.app.controllers;

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

import ibia.app.App;
```

```java
import ibia.app.SceneUtil;
import ibia.core.Client;
import ibia.core.DbDriver;
import ibia.core.Log;
import ibia.core.entities.Committee;
import ibia.core.entities.Conference;
import ibia.core.entities.Delegate;
import ibia.core.utils.Resolution;

import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class ConferenceView {
    @FXML protected Button statusButton;
    @FXML protected Text id;
    @FXML protected Text status;
    @FXML protected Text conferenceCrumb;
    @FXML protected Text name;
    @FXML protected Text created;
    @FXML protected Text committees;
    @FXML protected Text delegates;
    @FXML protected Text resolutions;
    @FXML protected VBox committeesList;

    private Conference instance;

    @FXML
    public void initialize() throws IOException {
        this.instance = DbDriver.fetchOne(Conference.class, App.getLocation());

        fillBreadcrumbs();
        fillName();
        fillId();
        fillDetails();
        fillStatusButton();
        fillCommitteesList();
    }

    /*********************/
    /*** FXML Controls ***/
    /*********************/

    /**
     * @param event the MouseEvent instance
     * @throws IOException - if loading FXML fails
     */
    @FXML
    protected void handleNewCommitteeAction(MouseEvent event) throws IOException {
        try {
            Stage stage = SceneUtil.loadPopupStage("NewCommittee", "Create new Committee");
            stage.show();
        } catch (Exception e) {
            SceneUtil.error("Failed to load window!").show();
        }
    }
```

```
72
73      @FXML
74      protected void handleStatusButtonAction(MouseEvent event) {
75          String confId = App.getLocation();
76          Conference instance = DbDriver.fetchOne(Conference.class, confId);
77          if (instance.isOngoing()) {
78              instance.setOngoing(false);
79              status.setText("FINISHED");
80              statusButton.setText("Re-open Conference");
81          } else {
82              instance.setOngoing(true);
83              status.setText("ONGOING");
84              statusButton.setText("Finish Conference");
85          }
86          DbDriver.updateOne(instance);
87      }
88
89      @FXML
90      protected void handleEditAction(MouseEvent event) throws IOException {
91          Stage stage = SceneUtil.loadPopupStage("EditConference", "Edit Conference");
92          stage.show();
93      }
94
95      @FXML
96      protected void handleDeleteAction(MouseEvent event) {
97          Stage stage = SceneUtil.confirm("Are you sure you wish to delete this conference?
                This action cannot be reversed.");
98          Scene root = stage.getScene();
99          Button cancel = (Button)root.lookup("#cancel");
100         Button confirm = (Button)root.lookup("#confirm");
101
102         cancel.setOnMouseClicked(evt -> {
103             stage.close();
104         });
105
106         confirm.setOnMouseClicked(evt -> {
107             Client.deleteConference(instance.getId());
108             try {
109                 App.navigate("Home");
110                 stage.close();
111             } catch (Exception e) {
112                 // If Home failed to load, this is a fatal error
113                 // and the application is exited.
114                 Log.error(e.getMessage());
115                 System.exit(1);
116             }
117         });
118
119         stage.show();
120     }
121
122     @FXML
123     protected void crumbHoverEffectOn(MouseEvent event) {
124         Text text = (Text)event.getTarget();
125         text.setUnderline(true);
126     }
127
128     @FXML
129     protected void crumbHoverEffectOff(MouseEvent event) {
130         Text text = (Text)event.getTarget();
131         text.setUnderline(false);
132     }
133
```

```
134     @FXML
135     protected void navigateHome(MouseEvent event) throws IOException,
            IllegalArgumentException {
136         App.navigate("Home");
137     }
138 
139     /******************/
140     /*** Templating ***/
141     /******************/
142 
143     private void fillBreadcrumbs() {
144         conferenceCrumb.setText(instance.getName());
145     }
146 
147     private void fillName() {
148         name.setText(instance.getName());
149     }
150 
151     private void fillId() {
152         id.setText(instance.getId());
153     }
154 
155     private void fillDetails() {
156         // Sets OPENED date
157         Date date = instance.getCreated();
158         SimpleDateFormat fmt = new SimpleDateFormat("dd/MM/yyyy");
159         created.setText(fmt.format(date));
160 
161         // Sets status to ONGOING or FINISHED
162         String currentStatus = instance.isOngoing() ? "ONGOING" : "FINISHED";
163         status.setText(currentStatus);
164 
165         // Sets number of committees
166         ArrayList<Committee> coms = Client.getConferenceCommittees(instance.getId());
167         int i = coms != null ? coms.size() : 0;
168         committees.setText(Integer.toString(i));
169 
170         // Sets number of delegates
171         ArrayList<Delegate> dels = new ArrayList<>();
172         if (coms != null) {
173             for (Committee com : coms) {
174                 ArrayList<Delegate> fetched = Client.getCommitteeDelegates(com.getId());
175                 if (fetched != null) {
176                     dels.addAll(fetched);
177                 }
178             }
179         }
180         delegates.setText(Integer.toString(dels.size()));
181 
182         // Sets number of resolutions
183         ArrayList<Resolution> resos = DbDriver.findAll(Resolution.class, r ->
                r.getPassed());
184         int totalResos = 0;
185         if (resos != null) {
186             for (Resolution reso : resos) {
187                 for (Delegate del : dels) {
188                     if (del.getId().equals(reso.getMainSubmitter())) {
189                         totalResos += 1;
190                     }
191                 }
192             }
193         }
194         resolutions.setText(Integer.toString(totalResos));
```

```
    }

    private void fillStatusButton() {
        if (instance.isOngoing()) {
            statusButton.setText("Finish Conference");
        } else {
            statusButton.setText("Re-open Conference");
        }
    }

    private void fillCommitteesList() throws IOException {
        ArrayList<Committee> coms = DbDriver.findAll(Committee.class, c ->
            c.getConferenceId().equals(instance.getId()));
        if (coms == null) return;
        ObservableList<Node> list = committeesList.getChildren();
        for (Committee com : coms) {
            HBox hbox = (HBox)SceneUtil.loadFXML("CommitteeListItem", false);
            Text name = (Text)hbox.getChildren().get(0);
            name.setText(com.getName());
            Text comId = (Text)hbox.getChildren().get(1);
            comId.setText("#" + com.getId());
            list.add(hbox);
        }
    }
}
```

ibia/app/controllers/Confirm.java

```
package ibia.app.controllers;

public class Confirm {
    // This controller does nothing, but is required
    // for JavaFX to load scenes properly.
}
```

ibia/app/controllers/DelegateListItem.java

```
package ibia.app.controllers;

import java.io.IOException;
import java.io.InputStream;

import ibia.app.App;
import ibia.core.DbDriver;
import ibia.core.entities.Delegate;
import ibia.core.utils.Country;
import javafx.application.Platform;
import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.text.Text;

public class DelegateListItem {
    @FXML protected ImageView flag;
    @FXML protected TextField speeches;
    @FXML protected TextField pois;
    @FXML protected TextField amendments;
    @FXML protected TextField motions;
    @FXML protected Text id;
```

```
25      @FXML protected Text delegation;
26
27      private Delegate instance;
28      private String instanceId;
29
30      @FXML
31      public void initialize() {
32          // runLater must be used to ensure
33          // that the instance id has been initialized
34          // in CommitteeView#fillDelegatesList
35          Platform.runLater(() -> {
36              this.instance = DbDriver.fetchOne(Delegate.class, instanceId);
37
38              delegation.setText(instance.getDelegation());
39              id.setText("#" + instance.getId());
40
41              // Update cells with delegates' data
42              String a = Integer.toString(instance.getSpeeches());
43              speeches.setText(a);
44              String b = Integer.toString(instance.getPois());
45              pois.setText(b);
46              String c = Integer.toString(instance.getAmendments());
47              amendments.setText(c);
48              String d = Integer.toString(instance.getMotions());
49              motions.setText(d);
50
51              // Set listeners for updating the db when cell
52              // values are changed.
53              attachListeners();
54
55              String code = Country.codeFromName(instance.getDelegation());
56              if (code != null) {
57                  InputStream stream = Country.getFlag(code);
58                  Image img = new Image(stream);
59                  flag.setImage(img);
60              }
61          });
62      }
63
64      // Attach listeners to update the
65      // db with the value of the cells
66      // whenever the TextField goes out
67      // of focus.
68      private void attachListeners() {
69          speeches.focusedProperty().addListener((observable, oldFocus, newFocus) -> {
70              // Run code when node is out of focus
71              if (!newFocus) {
72                  int n;
73                  try {
74                      String value = speeches.getText();
75                      String nonEmpty = value.isEmpty() ? "0" : value;
76                      n = Integer.parseInt(nonEmpty);
77                  } catch (NumberFormatException e) {
78                      n = instance.getSpeeches();
79                      speeches.setText(Integer.toString(n));
80                  }
81                  instance.setSpeeches(n);
82                  DbDriver.updateOne(instance);
83              }
84          });
85
86          pois.focusedProperty().addListener((observable, oldFocus, newFocus) -> {
87              // Run code when node is out of focus
```

```
88              if (!newFocus) {
89                  int n;
90                  try {
91                      String value = pois.getText();
92                      String nonEmpty = value.isEmpty() ? "0" : value;
93                      n = Integer.parseInt(nonEmpty);
94                  } catch (NumberFormatException e) {
95                      // If a non-integer is entered,
96                      // revert back to the previous value.
97                      n = instance.getPois();
98                      pois.setText(Integer.toString(n));
99                  }
100                 instance.setPois(n);
101                 DbDriver.updateOne(instance);
102             }
103         });
104
105         amendments.focusedProperty().addListener((observabe, oldFocus, newFocus) -> {
106             // Run code when node is out of focus
107             if (!newFocus) {
108                 int n;
109                 try {
110                     String value = amendments.getText();
111                     String nonEmpty = value.isEmpty() ? "0" : value;
112                     n = Integer.parseInt(nonEmpty);
113                 } catch (NumberFormatException e) {
114                     n = instance.getAmendments();
115                     amendments.setText(Integer.toString(n));
116                 }
117                 instance.setAmendments(n);
118                 DbDriver.updateOne(instance);
119             }
120         });
121
122         motions.focusedProperty().addListener((observable, oldFocus, newFocus) -> {
123             // Run code when node is out of focus
124             if (!newFocus) {
125                 int n;
126                 try {
127                     String value = motions.getText();
128                     String nonEmpty = value.isEmpty() ? "0" : value;
129                     n = Integer.parseInt(nonEmpty);
130                 } catch (NumberFormatException e) {
131                     n = instance.getMotions();
132                     motions.setText(Integer.toString(n));
133                 }
134                 instance.setMotions(n);
135                 DbDriver.updateOne(instance);
136             }
137         });
138     }
139
140     @FXML
141     protected void navigate(MouseEvent event) throws IOException {
142         App.navigate(instance.getId());
143     }
144
145     @FXML
146     protected void hoverEffectOn(MouseEvent event) {
147         Text text = (Text)event.getTarget();
148         text.setUnderline(true);
149     }
150
```

```
    @FXML
    protected void hoverEffectOff(MouseEvent event) {
        Text text = (Text)event.getTarget();
        text.setUnderline(false);
    }

    public void setInstanceId(String delId) {
        this.instanceId = delId;
    }
}
```

ibia/app/controllers/DelegateView.java

```
package ibia.app.controllers;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;

import ibia.app.App;
import ibia.app.SceneUtil;
import ibia.core.Client;
import ibia.core.DbDriver;
import ibia.core.Log;
import ibia.core.entities.Committee;
import ibia.core.entities.Conference;
import ibia.core.entities.Delegate;
import ibia.core.utils.Country;
import ibia.core.utils.Resolution;
import javafx.fxml.FXML;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class DelegateView {
    @FXML protected Text delegation;
    @FXML protected Text id;
    @FXML protected Text delName;
    @FXML protected Text comName;
    @FXML protected Text speeches;
    @FXML protected Text pois;
    @FXML protected Text amendments;
    @FXML protected Text motions;
    @FXML protected Text resos;
    @FXML protected Text committeeCrumb;
    @FXML protected Text delegateCrumb;
    @FXML protected Text conferenceCrumb;
    @FXML protected ImageView flag;

    private Delegate instance;

    @FXML
    public void initialize() {
        this.instance = DbDriver.fetchOne(Delegate.class, App.getLocation());

        fillBreadcrumbs();
        fillDelegation();
        fillId();
```

```
50          fillDetails();
51          fillFlag();
52      }
53
54      /*********************/
55      /*** FXML Controls ***/
56      /*********************/
57
58      /**
59       * @param event the MouseEvent instance
60       * @throws IOException - if loading FXML fails
61       */
62      @FXML
63      protected void handleEditAction(MouseEvent event) throws IOException {
64          Stage stage = SceneUtil.loadPopupStage("EditDelegate", "Edit Delegate");
65          stage.show();
66      }
67
68      @FXML
69      protected void handleDeleteAction(MouseEvent event) {
70          Stage stage = SceneUtil.confirm("Are you sure you wish to delete this delegate?
                This action cannot be reversed.");
71          Scene root = stage.getScene();
72          Button cancel = (Button)root.lookup("#cancel");
73          Button confirm = (Button)root.lookup("#confirm");
74
75          cancel.setOnMouseClicked(evt -> {
76              stage.close();
77          });
78
79          confirm.setOnMouseClicked(evt -> {
80              Client.deleteDelegate(instance.getId());
81              try {
82                  App.navigate(instance.getCommitteeId());
83                  stage.close();
84              } catch (Exception e) {
85                  // If Home failed to load, this is a fatal error
86                  // and the application is exited.
87                  Log.error(e.getMessage());
88                  System.exit(1);
89              }
90          });
91
92          stage.show();
93      }
94
95      @FXML
96      protected void crumbHoverEffectOn(MouseEvent event) {
97          Text text = (Text)event.getTarget();
98          text.setUnderline(true);
99      }
100
101     @FXML
102     protected void crumbHoverEffectOff(MouseEvent event) {
103         Text text = (Text)event.getTarget();
104         text.setUnderline(false);
105     }
106
107     @FXML
108     protected void navigateHome(MouseEvent event) throws IOException {
109         App.navigate("Home");
110     }
111
```

```
112     @FXML
113     protected void navigateCommittee(MouseEvent event) throws IOException {
114         String id = instance.getCommitteeId();
115         App.navigate(id);
116     }
117 
118     @FXML
119     protected void navigateConference(MouseEvent event) throws IOException {
120         String comId = instance.getCommitteeId();
121         Committee com = DbDriver.fetchOne(Committee.class, comId);
122         String conId = com.getConferenceId();
123         App.navigate(conId);
124     }
125 
126     /******************/
127     /*** Templating ***/
128     /******************/
129 
130     private void fillBreadcrumbs() {
131         delegateCrumb.setText(instance.getDelegation());
132 
133         String comId = instance.getCommitteeId();
134         Committee com = DbDriver.fetchOne(Committee.class, comId);
135         committeeCrumb.setText(com.getName());
136         Conference conf = DbDriver.fetchOne(Conference.class, com.getConferenceId());
137         conferenceCrumb.setText(conf.getName());
138     }
139 
140     private void fillDelegation() {
141         delegation.setText(instance.getDelegation());
142     }
143 
144     private void fillId() {
145         id.setText(instance.getId());
146     }
147 
148     private void fillDetails() {
149         delName.setText(instance.getName());
150         comName.setText(committeeCrumb.getText());
151         String a = Integer.toString(instance.getSpeeches());
152         speeches.setText(a);
153         String b = Integer.toString(instance.getPois());
154         pois.setText(b);
155         String c = Integer.toString(instance.getAmendments());
156         amendments.setText(c);
157         String d = Integer.toString(instance.getMotions());
158         motions.setText(d);
159         ArrayList<Resolution> submitted = DbDriver.findAll(Resolution.class, r ->
                r.getMainSubmitter().equals(instance.getId()));
160         int n = submitted != null ? submitted.size() : 0;
161         resos.setText(Integer.toString(n));
162     }
163 
164     private void fillFlag() {
165         String code = Country.codeFromName(instance.getDelegation());
166         InputStream stream = Country.getFlag(code);
167         if (stream != null) {
168             Image img = new Image(stream);
169             flag.setImage(img);
170         }
171     }
172 
173 }
```

ibia/app/controllers/EditCommittee.java

```java
package ibia.app.controllers;

import ibia.app.App;
import ibia.app.SceneUtil;
import ibia.core.DbDriver;
import ibia.core.Log;
import ibia.core.entities.Committee;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

public class EditCommittee {
    @FXML protected TextField name;

    @FXML
    protected void update(MouseEvent event) {
        String comName = name.getText();

        // validate data
        if (comName.isEmpty()) {
            SceneUtil.error("The committee name is required!").show();
        }
        else if (comName.length() > 30) {
            SceneUtil.error("The committee name must be between 1 and 30 characters!\nTry
                using an abbreviation.").show();
        }
        else {
            try {
                String comId = App.getLocation();
                Committee com = DbDriver.fetchOne(Committee.class, comId);
                com.setName(comName);
                DbDriver.updateOne(com);
                App.refresh();
                closeStage(event);
            } catch (Exception e) {
                Log.error(e.getMessage());
                e.printStackTrace();
                SceneUtil.error(e.getMessage()).show();
            }
        }
    }

    @FXML protected void cancel(MouseEvent event) {
        closeStage(event);
    }

    private void closeStage(MouseEvent event) {
        // cast source to Button so we can access window
        Button source = (Button)(event.getSource());
        // get window, cast it to Stage so we can close it
        Stage stage = (Stage)(source.getScene().getWindow());
        stage.close();
    }
}
```

ibia/app/controllers/EditConference.java

```java
package ibia.app.controllers;
```

```java

import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;

import ibia.core.DbDriver;
import ibia.core.Log;
import ibia.core.entities.Conference;

import ibia.app.App;
import ibia.app.SceneUtil;

import javafx.fxml.FXML;
import javafx.stage.Stage;

public class EditConference {
    @FXML TextField name;

    @FXML
    protected void update(MouseEvent event) {
        String confName = name.getText();

        // validate form data
        if (confName.isEmpty()) {
            SceneUtil.error("The conference name is required!").show();
        }
        else if (confName.length() > 30) {
            SceneUtil.error("The conference name must be between 1 and 30 characters!\nTry
                using an abbreviation.").show();
        }
        else {
            try {
                String confId = App.getLocation();
                Conference conf = DbDriver.fetchOne(Conference.class, confId);
                conf.setName(confName);
                DbDriver.updateOne(conf);
                App.refresh();
                closeStage(event);
            } catch (Exception e) {
                Log.error(e.getMessage());
                e.printStackTrace();
                SceneUtil.error(e.getMessage()).show();
            }
        }
    }

    @FXML protected void cancel(MouseEvent event) {
        closeStage(event);
    }

    private void closeStage(MouseEvent event) {
        // cast source to Button so we can access window
        Button source = (Button)(event.getSource());
        // get window, cast it to Stage so we can close it
        Stage stage = (Stage)(source.getScene().getWindow());
        stage.close();
    }
}
```

ibia/app/controllers/EditDelegate.java

```java
package ibia.app.controllers;
```

```java

import java.util.ArrayList;

import ibia.app.App;
import ibia.app.SceneUtil;
import ibia.core.DbDriver;
import ibia.core.Log;
import ibia.core.entities.Delegate;
import ibia.core.utils.Country;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.MenuButton;
import javafx.scene.control.MenuItem;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

public class EditDelegate {
    @FXML protected TextField name;
    @FXML protected TextField delegation;
    @FXML protected MenuButton choose;

    @FXML
    public void initialize() {
        ArrayList<String> countries = Country.listOfNames();
        if (countries == null) return;

        for (String country : countries) {
            MenuItem choice = new MenuItem();
            choice.setText(country);
            choice.setOnAction((ActionEvent event) -> {
                delegation.setText(country);
            });

            choose.getItems().add(choice);
        }
    }

    @FXML
    protected void update(MouseEvent event) {
        String delName = name.getText();
        // validate form data
        if (delName.isEmpty()) {
            SceneUtil.error("The delegate name is required!").show();
            return;
        }
        else if (delName.length() > 120) {
            SceneUtil.error("The delegate name must be between 1 and 120
                characters!").show();
            return;
        }

        String delegationStr = delegation.getText();
        if (delegationStr.isEmpty()) {
            SceneUtil.error("The delegation name is required!").show();
            return;
        }
        else if (delegationStr.length() > 120) {
            SceneUtil.error("The delegation name must be between 1 and 120
                characters!").show();
            return;
        }
```

```
63
64          try {
65              String delId = App.getLocation();
66              Delegate del = DbDriver.fetchOne(Delegate.class, delId);
67              del.setName(delName);
68              del.setDelegation(delegationStr);
69              DbDriver.updateOne(del);
70              App.refresh();
71              closeStage(event);
72          } catch (Exception e) {
73              Log.error(e.getMessage());
74              e.printStackTrace();
75              SceneUtil.error(e.getMessage()).show();
76          }
77      }
78
79      @FXML
80      protected void cancel(MouseEvent event) {
81          closeStage(event);
82      }
83
84      private void closeStage(MouseEvent event) {
85          // cast source to Button so we can access window
86          Button source = (Button)(event.getSource());
87          // get window, cast it to Stage so we can close it
88          Stage stage = (Stage)(source.getScene().getWindow());
89          stage.close();
90      }
91  }
```

ibia/app/controllers/Home.java

```
1   package ibia.app.controllers;
2
3   import java.awt.Desktop;
4   import java.io.IOException;
5   import java.net.URI;
6   import java.util.ArrayList;
7
8   import javafx.fxml.FXML;
9   import javafx.stage.Stage;
10  import javafx.scene.Group;
11  import javafx.scene.input.MouseEvent;
12  import javafx.scene.paint.Color;
13  import javafx.scene.shape.Rectangle;
14  import javafx.scene.text.Text;
15
16  import ibia.app.SceneUtil;
17  import ibia.core.DbDriver;
18  import ibia.core.entities.Conference;
19
20  public class Home {
21      @FXML protected Text resumeMsg;
22
23      @FXML
24      public void initialize() throws IOException {
25          ArrayList<Conference> confs = DbDriver.findAll(Conference.class, c ->
                c.isOngoing());
26          if (confs != null && confs.size() > 0) {
27              resumeMsg.setText("Click to view ongoing conferences");
28          }
29      }
```

```
30
31      @FXML
32      protected void handleResumeConfAction() throws IOException {
33          Stage stage = SceneUtil.loadPopupStage("OngoingConferences", "Choose an ongoing
                Conference");
34          stage.show();
35      }
36
37      @FXML
38      protected void handleNewConfAction() throws IOException {
39          Stage stage = SceneUtil.loadPopupStage("NewConference", "Create new Conference");
40          stage.show();
41      }
42
43      @FXML
44      protected void handlePastConfAction() throws IOException {
45          Stage stage = SceneUtil.loadPopupStage("PastConferences", "Choose a finished
                Conference");
46          stage.show();
47      }
48
49      @FXML
50      protected void handleGuidesAction() {
51          String url = "https://github.com/quantomistro/ibia-app";
52
53          try {
54              openURL(url);
55          } catch (Exception e) {
56              SceneUtil.error("Failed to open URL!").show();
57          }
58      }
59
60      @FXML
61      protected void handleAboutAction() throws IOException {
62          Stage stage = SceneUtil.loadPopupStage("About", "About ibia");
63          stage.show();
64      }
65
66      @FXML
67      protected void handleFeedbackAction() {
68          String url = "https://github.com/quantomistro/ibia-app/issues/";
69
70          try {
71              openURL(url);
72          } catch (Exception e) {
73              SceneUtil.error("Failed to open URL!").show();
74          }
75      }
76
77      @FXML
78      protected void handleViewLogsAction() throws IOException {
79          Stage stage = SceneUtil.loadPopupStage("Logs", "Logs");
80          stage.show();
81      }
82
83      @FXML
84      protected void hoverEffectOn(MouseEvent event) {
85          Group btn = (Group)event.getTarget();
86
87          Rectangle rect = (Rectangle)btn.getChildren().get(0);
88          rect.setStrokeWidth(2);
89          rect.setStroke(Color.WHITE);
90      }
```

```
91
92      @FXML
93      protected void hoverEffectOff(MouseEvent event) {
94          Group btn = (Group)event.getTarget();
95
96          Rectangle rect = (Rectangle)btn.getChildren().get(0);
97          rect.setStrokeWidth(0);
98          rect.setStroke(null);
99      }
100
101     @FXML
102     protected void crumbHoverEffectOn(MouseEvent event) {
103         Text text = (Text)event.getTarget();
104         text.setUnderline(true);
105     }
106
107     @FXML
108     protected void crumbHoverEffectOff(MouseEvent event) {
109         Text text = (Text)event.getTarget();
110         text.setUnderline(false);
111     }
112
113     private void openURL(String url) throws Exception {
114         if (Desktop.isDesktopSupported() &&
                Desktop.getDesktop().isSupported(Desktop.Action.BROWSE)) {
115             Desktop.getDesktop().browse(new URI(url));
116         }
117     }
118 }
```

ibia/app/controllers/Logs.java

```
1  package ibia.app.controllers;
2
3  import java.io.BufferedReader;
4  import java.io.FileNotFoundException;
5  import java.io.FileReader;
6  import java.io.IOException;
7
8  import javafx.fxml.FXML;
9  import javafx.scene.control.TextArea;
10
11 public class Logs {
12     @FXML protected TextArea textArea;
13
14     @FXML
15     public void initialize() throws FileNotFoundException, IOException {
16         FileReader fr = new FileReader("data/ibia.log");
17         // Use BufferedReader for fast reading
18         BufferedReader br = new BufferedReader(fr);
19         String contents = "";
20
21         while (true) {
22             String line = br.readLine();
23             if (line == null) break;
24             contents += line + "\n";
25         }
26
27         br.close();
28         textArea.setText(contents);
29     }
30 }
```

ibia/app/controllers/NewCommittee.java

```java
package ibia.app.controllers;

import ibia.app.App;
import ibia.app.SceneUtil;
import ibia.core.Client;
import ibia.core.Log;
import ibia.core.entities.Committee;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

public class NewCommittee {
    @FXML protected TextField name;

    @FXML
    protected void handleCreateAction(MouseEvent event) {
        String comName = name.getText();

        // validate data
        if (comName.isEmpty()) {
            SceneUtil.error("The committee name is required!").show();
        }
        else if (comName.length() > 30) {
            SceneUtil.error("The committee name must be between 1 and 30 characters!\nTry
                using an abbreviation.").show();
        }
        else {
            try {
                Committee com = Client.addNewCommittee(comName, App.getLocation());
                App.navigate(com.getId());
                closeStage(event);
            } catch (Exception e) {
                Log.error(e.getMessage());
                e.printStackTrace();
                SceneUtil.error(e.getMessage()).show();
            }
        }
    }

    @FXML protected void handleCancelAction(MouseEvent event) {
        closeStage(event);
    }

    private void closeStage(MouseEvent event) {
        // cast source to Button so we can access window
        Button source = (Button)(event.getSource());
        // get window, cast it to Stage so we can close it
        Stage stage = (Stage)(source.getScene().getWindow());
        stage.close();
    }
}
```

ibia/app/controllers/NewConference.java

```java
package ibia.app.controllers;

import javafx.scene.control.Button;
import javafx.scene.control.TextField;
```

```java
import javafx.scene.input.MouseEvent;

import ibia.core.Client;
import ibia.core.Log;
import ibia.core.entities.Conference;

import ibia.app.App;
import ibia.app.SceneUtil;

import javafx.fxml.FXML;
import javafx.stage.Stage;

public class NewConference {
    @FXML TextField name;

    @FXML
    protected void handleCreateAction(MouseEvent event) {
        String confName = name.getText();

        // validate form data
        if (confName.isEmpty()) {
            SceneUtil.error("The conference name is required!").show();
        }
        else if (confName.length() > 30) {
            SceneUtil.error("The conference name must be between 1 and 30 characters!\nTry
                using an abbreviation.").show();
        }
        else {
            try {
                Conference conf = Client.addNewConference(confName);
                App.navigate(conf.getId());
                closeStage(event);
            } catch (Exception e) {
                Log.error(e.getMessage());
                e.printStackTrace();
                SceneUtil.error(e.getMessage()).show();
            }
        }
    }

    @FXML protected void handleCancelAction(MouseEvent event) {
        closeStage(event);
    }

    private void closeStage(MouseEvent event) {
        // cast source to Button so we can access window
        Button source = (Button)(event.getSource());
        // get window, cast it to Stage so we can close it
        Stage stage = (Stage)(source.getScene().getWindow());
        stage.close();
    }
}
```

ibia/app/controllers/NewDelegate.java

```java
package ibia.app.controllers;

import java.util.ArrayList;

import ibia.app.App;
import ibia.app.SceneUtil;
import ibia.core.Client;
```

```java
import ibia.core.Log;
import ibia.core.entities.Delegate;
import ibia.core.utils.Country;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.MenuButton;
import javafx.scene.control.MenuItem;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

public class NewDelegate {
    @FXML protected TextField name;
    @FXML protected TextField delegation;
    @FXML protected MenuButton choose;

    @FXML
    public void initialize() {
        ArrayList<String> countries = Country.listOfNames();
        if (countries == null) return;

        for (String country : countries) {
            MenuItem choice = new MenuItem();
            choice.setText(country);
            choice.setOnAction((ActionEvent event) -> {
                delegation.setText(country);
            });

            choose.getItems().add(choice);
        }
    }

    @FXML
    protected void create(MouseEvent event) {
        String delName = name.getText();
        // validate form data
        if (delName.isEmpty()) {
            SceneUtil.error("The delegate name is required!").show();
        }
        else if (delName.length() > 120) {
            SceneUtil.error("The delegate name must be between 1 and 120
                characters!").show();
        }
        else {
            try {
                String comId = App.getLocation();
                String delegationName = delegation.getText();
                Delegate del = Client.addNewDelegate(delName, delegationName, comId);
                App.navigate(del.getId());
                closeStage(event);
            } catch (Exception e) {
                Log.error(e.getMessage());
                e.printStackTrace();
                SceneUtil.error(e.getMessage()).show();
            }
        }
    }

    @FXML
    protected void cancel(MouseEvent event) {
        closeStage(event);
    }
```

```
70
71      private void closeStage(MouseEvent event) {
72          // cast source to Button so we can access window
73          Button source = (Button)(event.getSource());
74          // get window, cast it to Stage so we can close it
75          Stage stage = (Stage)(source.getScene().getWindow());
76          stage.close();
77      }
78  }
```

ibia/app/controllers/NewResolution.java

```
1   package ibia.app.controllers;
2
3   import java.io.IOException;
4   import java.util.ArrayList;
5
6   import ibia.app.App;
7   import ibia.app.SceneUtil;
8   import ibia.core.DbDriver;
9   import ibia.core.entities.Delegate;
10  import ibia.core.utils.Resolution;
11  import ibia.core.utils.Topic;
12  import javafx.event.ActionEvent;
13  import javafx.fxml.FXML;
14  import javafx.scene.control.MenuButton;
15  import javafx.scene.control.MenuItem;
16  import javafx.scene.control.TextField;
17  import javafx.scene.input.MouseEvent;
18  import javafx.stage.Stage;
19
20  public class NewResolution {
21      @FXML protected TextField delegate;
22      @FXML protected MenuButton delegateDropdown;
23      @FXML protected TextField topic;
24      @FXML protected MenuButton topicDropdown;
25
26      @FXML
27      protected void initialize() {
28          String comId = App.getLocation();
29
30          ArrayList<Delegate> dels = DbDriver.findAll(Delegate.class, d ->
                d.getCommitteeId().equals(comId));
31          if (dels != null) {
32              for (Delegate del : dels) {
33                  MenuItem item = new MenuItem(del.getDelegation());
34                  item.setText(del.getDelegation());
35                  item.setId(del.getId());
36                  item.setOnAction((ActionEvent event) -> {
37                      delegate.setText(del.getDelegation());
38                      delegate.setId(del.getId());
39                  });
40                  delegateDropdown.getItems().add(item);
41              }
42          }
43
44          ArrayList<Topic> topics = DbDriver.findAll(Topic.class, t ->
                t.getCommitteeId().equals(comId));
45          if (topics != null) {
46              for (Topic t : topics) {
47                  MenuItem item = new MenuItem(t.getTopic());
48                  item.setText(t.getTopic());
```

```
                item.setId(Integer.toString(t.getId()));
                item.setOnAction((ActionEvent event) -> {
                    topic.setText(t.getTopic());
                    topic.setId(Integer.toString(t.getId()));
                });
                topicDropdown.getItems().add(item);
            }
        }


    }

    @FXML
    protected void create(MouseEvent event) throws IOException {
        SceneUtil.error("Unimplemented!");
        if (delegate.getText().isEmpty()) {
            SceneUtil.error("Please choose a delegate as the main submitter for this
                resolution!").show();
            return;
        }
        if (topic.getText().isEmpty()) {
            SceneUtil.error("Please choose a topic for this resolution!").show();
            return;
        }

        Resolution reso = new Resolution(delegate.getId(), Integer.parseInt(topic.getId()));
        DbDriver.insertOne(reso);
        // do this to update the resolutions list
        Stage stage = SceneUtil.loadPopupStage("Resolutions", "Resolutions");
        stage.show();
        close();
    }

    @FXML
    protected void cancel(MouseEvent event) {
        Stage stage = (Stage)delegate.getScene().getWindow();
        stage.close();
    }

    private void close() {
        Stage stage = (Stage)delegate.getScene().getWindow();
        stage.close();
    }
}
```

ibia/app/controllers/OngoingConferences.java

```
package ibia.app.controllers;

import java.io.IOException;
import java.util.ArrayList;

import ibia.app.SceneUtil;
import ibia.core.DbDriver;
import ibia.core.entities.Conference;
import javafx.fxml.FXML;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;

public class OngoingConferences {
    @FXML protected VBox ongoingList;
```

```
16
17      @FXML
18      public void initialize() throws IOException {
19          ArrayList<Conference> confs = DbDriver.findAll(Conference.class, c ->
                c.isOngoing());
20          if (confs != null) {
21              for (Conference conf : confs) {
22                  HBox item = (HBox)SceneUtil.loadFXML("ConferenceListItem", false);
23                  Text name = (Text)item.lookup("#name");
24                  name.setText(conf.getName());
25                  Text id = (Text)item.lookup("#id");
26                  id.setText("#" + conf.getId());
27                  ongoingList.getChildren().add(item);
28              }
29          }
30      }
31  }
```

ibia/app/controllers/PastConferences.java

```
1   package ibia.app.controllers;
2
3   import java.io.IOException;
4   import java.util.ArrayList;
5
6   import ibia.app.SceneUtil;
7   import ibia.core.DbDriver;
8   import ibia.core.entities.Conference;
9   import javafx.fxml.FXML;
10  import javafx.scene.layout.HBox;
11  import javafx.scene.layout.VBox;
12  import javafx.scene.text.Text;
13
14  public class PastConferences {
15      @FXML protected VBox pastList;
16
17      @FXML
18      public void initialize() throws IOException {
19          ArrayList<Conference> confs = DbDriver.findAll(Conference.class, c ->
                !c.isOngoing());
20          if (confs != null) {
21              for (Conference conf : confs) {
22                  HBox item = (HBox)SceneUtil.loadFXML("ConferenceListItem", false);
23                  Text name = (Text)item.lookup("#name");
24                  name.setText(conf.getName());
25                  Text id = (Text)item.lookup("#id");
26                  id.setText("#" + conf.getId());
27                  pastList.getChildren().add(item);
28              }
29          }
30      }
31  }
```

ibia/app/controllers/Resolutions.java

```
1   package ibia.app.controllers;
2
3   import java.io.IOException;
4   import java.util.ArrayList;
5
6   import ibia.app.App;
```

```java
import ibia.app.SceneUtil;
import ibia.core.DbDriver;
import ibia.core.utils.Resolution;
import ibia.core.utils.Topic;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class Resolutions {
    @FXML protected VBox list;

    @FXML
    protected void initialize() throws IOException {
        String comId = App.getLocation();

        ArrayList<Topic> topics = DbDriver.findAll(Topic.class, t ->
            t.getCommitteeId().equals(comId));
        if (topics == null) return;
        ArrayList<Integer> topicIds = new ArrayList<>();
        for (Topic t : topics) topicIds.add(t.getId());

        ArrayList<Resolution> resos = DbDriver.findAll(Resolution.class, r ->
            topicIds.contains(r.getTopicId()));
        if (resos == null) return;

        for (Resolution reso : resos) {
            FXMLLoader loader = new
                FXMLLoader(getClass().getResource("/fxml/ResolutionsListItem.fxml"));
            HBox root = loader.load();
            ResolutionsListItem controller = loader.getController();
            controller.setInstanceId(reso.getId());
            controller.setRefs(list, root);

            list.getChildren().add(root);
        }
    }

    @FXML
    protected void newResolution() throws IOException {
        Stage stage = SceneUtil.loadPopupStage("NewResolution", "Create a new resolution");
        stage.show();
        close();
    }

    private void close() {
        Stage stage = (Stage)list.getScene().getWindow();
        stage.close();
    }
}
```

ibia/app/controllers/ResolutionsListItem.java

```java
package ibia.app.controllers;

import ibia.core.DbDriver;
import ibia.core.entities.Delegate;
import ibia.core.utils.Resolution;
import ibia.core.utils.Topic;
import javafx.application.Platform;
import javafx.fxml.FXML;
```

```java
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;

public class ResolutionsListItem {
    @FXML protected Text delegate;
    @FXML protected Text topic;

    private Resolution instance;
    private int instanceId;
    private VBox listRef;
    private HBox itemRef;

    @FXML
    protected void initialize() {
        Platform.runLater(() -> {
            this.instance = DbDriver.fetchOne(Resolution.class, instanceId);
            Delegate del = DbDriver.fetchOne(Delegate.class, instance.getMainSubmitter());
            delegate.setText(del.getDelegation());

            Topic t = DbDriver.fetchOne(Topic.class, instance.getTopicId());
            topic.setText(t.getTopic());
        });
    }

    @FXML
    protected void deleteReso(MouseEvent event) {
        DbDriver.deleteById(Resolution.class, instanceId);
        listRef.getChildren().remove(itemRef);
    }

    public void setInstanceId(int id) {
        this.instanceId = id;
    }

    // Get reference to the parent popup's VBox list
    // and this item's node, so that it can be
    // deleted directly from here when the Delete
    // button is pressed
    public void setRefs(VBox list, HBox item) {
        this.listRef = list;
        this.itemRef = item;
    }
}
```

ibia/app/controllers/SpeechTimer.java

```java
package ibia.app.controllers;

import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.text.Text;
import javafx.util.Duration;

public class SpeechTimer {
    @FXML protected Text minutes;
    @FXML protected Text seconds;
```

```
    @FXML protected Button toggle;

    private boolean on = false;
    private IntegerProperty mins = new SimpleIntegerProperty(0);
    private IntegerProperty secs = new SimpleIntegerProperty(0);

    // JavaFX Timelines are mainly for animations,
    // but can also be used for scheduling tasks
    // on the same thread that handles the scenes
    // and displays.
    private Timeline timeline;

    @FXML
    public void initialize() {
        // Bind mins and secs to the Nodes' text properties
        // so that they update automatically.
        minutes.textProperty().bind(mins.asString());
        seconds.textProperty().bind(secs.asString());
    }

    @FXML
    protected void toggleTimer() {
        if (on) {
            stopTimer();
            toggle.setText("Start");
            on = false;
        } else {
            startTimer();
            toggle.setText("Stop");
            on = true;
        }
    }

    protected void startTimer() {
        KeyFrame keyframe = new KeyFrame(Duration.seconds(1), event -> {
            if (secs.greaterThanOrEqualTo(59).get()) {
                mins.set(mins.get() + 1);
                secs.set(0);
            } else {
                secs.set(secs.get() + 1);
            }
        });
        timeline = new Timeline(keyframe);
        timeline.setCycleCount(Timeline.INDEFINITE);
        timeline.play();
    }

    protected void stopTimer() {
        timeline.stop();
    }
}
```

ibia/app/controllers/Topics.java

```
package ibia.app.controllers;

import java.io.IOException;
import java.util.ArrayList;

import ibia.app.App;
import ibia.core.DbDriver;
import ibia.core.utils.Topic;
```

```java

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.VBox;

public class Topics {
    @FXML protected VBox list;

    @FXML
    protected void initialize() throws IOException {
        String comId = App.getLocation();
        ArrayList<Topic> topics = DbDriver.findAll(Topic.class, t ->
            t.getCommitteeId().equals(comId));
        if (topics == null) return;

        for (Topic topic : topics) {
            FXMLLoader loader = new
                FXMLLoader(getClass().getResource("/fxml/TopicsListItem.fxml"));
            Parent root = loader.load();
            TopicsListItem controller = loader.getController();
            controller.setInstanceId(topic.getId());
            list.getChildren().add(root);
        }
    }

    @FXML
    protected void newTopic(MouseEvent event) throws IOException {
        Topic topic = new Topic(App.getLocation(), "");
        DbDriver.insertOne(topic);

        FXMLLoader loader = new
            FXMLLoader(getClass().getResource("/fxml/TopicsListItem.fxml"));
        Parent root = loader.load();
        TopicsListItem controller = loader.getController();
        controller.setInstanceId(topic.getId());
        list.getChildren().add(root);
    }
}
```

ibia/app/controllers/TopicsListItem.java

```java
package ibia.app.controllers;

import ibia.core.DbDriver;
import ibia.core.utils.Topic;
import javafx.application.Platform;
import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;

public class TopicsListItem {
    @FXML protected TextField topic;

    private Topic instance;
    private int instanceId;

    @FXML
    protected void initialize() {
        // runLater must be used to ensure
```

```
20          // that the instance id has been initialized
21          // from the Topics class.
22          Platform.runLater(() -> {
23              this.instance = DbDriver.fetchOne(Topic.class, instanceId);
24              topic.setText(instance.getTopic());
25
26              attachListener();
27          });
28      }
29
30      private void attachListener() {
31          topic.focusedProperty().addListener((observable, oldFocus, newFocus) -> {
32              // Run code when node is out of focus
33              if (!newFocus) {
34                  String value = topic.getText();
35
36                  if (value.isEmpty()) {
37                      DbDriver.deleteById(Topic.class, instance.getId());
38                      HBox container = (HBox)topic.getParent();
39                      VBox list = (VBox)container.getParent();
40                      list.getChildren().remove(container);
41                  } else {
42                      instance.setTopic(value);
43                      DbDriver.updateOne(instance);
44                  }
45              }
46          });
47      }
48
49      public void setInstanceId(int topicId) {
50          this.instanceId = topicId;
51      }
52  }
```