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);
    }

    /**
     * Fetch Committees belonging to a specific Conference
     * @param conferenceId ID of the Conference
     * @return ArrayList of Committees
     */
    public static ArrayList<Committee> getConferenceCommittees(String conferenceId) {
        return DbDriver.findAll(Committee.class, c ->
            c.getConferenceId().equals(conferenceId));
    }

    /**
     * Fetch Delegates belonging to a specific Committee
     * @param committeeId ID of the Committee
     * @return ArrayList of Delegates
     */
    public static ArrayList<Delegate> getCommitteeDelegates(String committeeId) {
        return DbDriver.findAll(Delegate.class, d ->
            d.getCommitteeId().equals(committeeId));
    }

    /**
     * Fetch Topics belonging to a specific Committee
     * @param committeeId ID of the Committee
     * @return ArrayList of Topics
     */
    public static ArrayList<Topic> getCommitteeTopics(String committeeId) {
        return DbDriver.findAll(Topic.class, t -> t.getCommitteeId().equals(committeeId));
    }

    /**
     * Fetch Resolutions belonging to a specific Topic
     * @param topicId ID of the Topic
     * @return ArrayList of Resolutions
     */
    public static ArrayList<Resolution> getTopicResolutions(int topicId) {
        return DbDriver.findAll(Resolution.class, r -> r.getTopicId() == topicId);
    }

    /**
     * Deletes a Conference instance from the database,
     * including it’s child Committees and the
     * Delegates in those committees.
     * @param confId ID of the Conference
     */
    public static void deleteConference(String confId) {
        deleteConferenceChildren(confId);
        DbDriver.deleteById(Conference.class, confId);
    }

    /**
     * Deletes a Committee instance form the database,
     * including it’s child Delegates.
     * @param comId ID of the Committee
     */
    public static void deleteCommittee(String comId) {
        deleteCommitteeChildren(comId);
        DbDriver.deleteById(Committee.class, comId);
    }

    /**
     * Deletes a Delegate instance from the database.
     * @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

```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();
    }

    /**
     * Update a persisted entity.
     *
     * @param <T> type of Entity to be updated.
     * @param entity updated entity.
     */
    public static <T> void updateOne(T entity) {
        Session session = openSession();
        session.beginTransaction();
        session.update(entity);
        session.getTransaction().commit();
        session.close();
    }

    /**
     * Update a collection of persisted entities.
     *
     * @param <T> type of Entity to be updated.
     * @param entities collection of updated entities.
     */
    public static <T> void updateAll(Collection<T> entities) {
        Session session = openSession();
        session.beginTransaction();
        for (T entity : entities) session.update(entity);
        session.getTransaction().commit();
        session.close();
    }

    /**
     * Delete an entity from the database.
     *
     * @param <T> type of Entity to be deleted.
     * @param entity entity to be deleted.
     */
    public static <T> void deleteOne(T entity) {
        Session session = openSession();
        session.beginTransaction();
        session.delete(entity);
        session.getTransaction().commit();
        session.close();
    }

    /**
     * Delete an entity from the database
     * using the id
     * @param <T> type of entity to be deleted
     * @param entityClass Class of the entity to be deleted
     * @param id id of the entity to be deleted
     */
    public static <T> void deleteById(Class<T> entityClass, Object id) {
        T entity = DbDriver.fetchOne(entityClass, id);
        deleteOne(entity);
    }

    /**
     * Delete a collection of entities from the database.
     *
     * @param <T> type of Entity to be deleted.
     * @param entities collection of entities to be deleted.
     */
    public static <T> void deleteAll(Collection<T> entities) {
        Session session = openSession();
        session.beginTransaction();
        for (T entity : entities) session.delete(entity);
        session.getTransaction().commit();
        session.close();
    }

    /**
     * Fetch/read a persisted entity from the database.
     *
     * @param <T> type of Entity to be fetched.
     * @param entityClass Class of the entity to be fetched.
     * @param id ID of the entity being fetched.
     * @return the fetched entity, or null if the entity does not exist.
     */
    public static <T> T fetchOne(Class<T> entityClass, Object id) {
        Session session = openSession();
        session.beginTransaction();
        T entity = (T)session.find(entityClass, id);
        session.close();
        return entity;
    }

    /**
     * Fetch/read all persisted entities of a particular
     * type from the database.
     *
     * @param <T> type of entity to be fetched.
     * @param entityClass Class of the entities being fetched.
     * @return ArrayList of fetched entities, or null if none were found.
     */
    public static <T> ArrayList<T> fetchAll(Class<T> entityClass) {
        Session session = openSession();
        session.beginTransaction();
        CriteriaBuilder cb = session.getCriteriaBuilder();
        CriteriaQuery<T> cq = cb.createQuery(entityClass);
        Root<T> rootEntry = cq.from(entityClass);
        CriteriaQuery<T> all = cq.select(rootEntry);

        TypedQuery<T> allQuery = session.createQuery(all);
        ArrayList<T> results = (ArrayList<T>)allQuery.getResultList();
        session.close();
        return results.size() > 0 ? results : null;
    }

    /**
     * Fetches all entities of a particular type and
     * finds the first one satisfying a given predicate.
     * For example, to find a delegate with the name "ABC":
     *
     * @param <T> type of entity to be fetched.
     * @param entityClass Class of the entities being fetched.
     * @param filter predicate for searching through the fetched entities.
     * @return the first entity found to satisfy the predicate, or null if none were found.
     */
    public static <T> T findOne(Class<T> entityClass, Predicate<T> filter) {
        ArrayList<T> entities = fetchAll(entityClass);
        if (entities != null) {
            for (T entity : entities) {
                if (filter.test(entity)) return entity;
            }
        }
        return null;
    }

    /**
     * Fetches all entities of a particular type and
     * finds all that satisfy a given predicate.
     *
     * @param <T> type of entity to be fetched.
     * @param entityClass Class of the entities being fetched.
     * @param filter predicate for searching through the entities.
     * @return ArrayList of entities found to satisfy the predicate, or null if none were
       found.
     */
    public static <T> ArrayList<T> findAll(Class<T> entityClass, Predicate<T> filter) {
        ArrayList<T> found = new ArrayList<>();
        ArrayList<T> entities = fetchAll(entityClass);
        if (entities != null) {
            for (T entity : entities) {
                if (filter.test(entity)) found.add(entity);
            }
        }
        return found.size() > 0 ? found : null;
    }

    /**
     * Destroy database service registry.
     */
    public static void shutdown() {
        if (registry != null) {
            StandardServiceRegistryBuilder.destroy(registry);
        }
    }
}
```

ibia/core/Log.java

```java
package ibia.core;

import java.io.File;
import java.nio.file.FileSystems;
import java.util.logging.FileHandler;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

/**
 * Logging utility.
 */
public class Log {
    private static Logger logger;
    private static Handler handler;

    public static Logger getLogger() {
        if (logger == null) {
            logger = Logger.getLogger("");
            logger.setUseParentHandlers(false); // No need to send output to parent loggers

            try {
                // Create the file if it doesn't exist
                String path = getLogFilePath();
                File file = new File(path);
                if (!file.isFile()) {
                    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

```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;
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

```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.
     *
     * @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;

    public Resolution() {}

    public Resolution(String mainSubmitter, int topicId) {
        this.mainSubmitter = mainSubmitter;
        this.topicId = topicId;
        this.passed = false;
    }

    public int getId() {
        return id;
    }

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

    public String getMainSubmitter() {
        return mainSubmitter;
    }

    public void setMainSubmitter(String mainSubmitter) {
        this.mainSubmitter = mainSubmitter;
    }

    public int getTopicId() {
        return topicId;
    }

    public void setTopicId(int topicId) {
        this.topicId = topicId;
    }

    public boolean getPassed() {
        return passed;
    }

    public void setPassed(boolean passed) {
        this.passed = passed;
    }
}
```

ibia/core/utils/Topic.java

```java
package ibia.core.utils;

/**
 * Represents a Committee topic.
 * <br><br>
 * This class is also a Hibernate entity.
 */
public class Topic {
    private int id;
    private String committeeId;
    private String topic;

    public Topic() {}

    public Topic(String committeeId, String topic) {
        this.committeeId = committeeId;
        this.topic = topic;
    }

    public int getId() {
        return id;
    }

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

    public String getCommitteeId() {
        return committeeId;
    }

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

    public String getTopic() {
        return topic;
    }

    public void setTopic(String topic) {
        this.topic = topic;
    }
}
```

ibia/core/entities/Committee.java

ibia/core/entities/Conference.java

```java
package ibia.core.entities;

import java.util.Date;

import ibia.core.utils.Id;

/**
 * Represents an MUN conference.
 */
public class Conference implements Entity {
    private final EntityType type = EntityType.CON;
    private String id;
    private String name;
    private boolean ongoing;
    private Date created;

    /**
     * This constructor is used internally by Hibernate
     * and MUST NOT be used in client-facing code.
     */
    public Conference() {}

    public Conference(String name) {
        this.id = Id.generate(type);
        this.name = name;
        this.ongoing = true;
        this.created = new Date();
    }

    public EntityType getType() {
        return type;
    }

    /**
     * Whether the conference is ongoing (active) or not.
     * The default value is true when a conference is
     * instantiated.
     *
     * @return true if the conference is active, otherwise false.
     */
    public boolean isOngoing() {
        return ongoing;
    }

    /* GETTERS and SETTERS used by hibernate */

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean getOngoing() {
        return ongoing;
    }

    public void setOngoing(boolean ongoing) {
        this.ongoing = ongoing;
    }

    public Date getCreated() {
        return created;
    }

    public void setCreated(Date created) {
        this.created = created;
    }
}
```

ibia/core/entities/Delegate.java

```java
package ibia.core.entities;

import ibia.core.utils.Country;
import ibia.core.utils.Id;

/**
 * Represents a delegate within an MUN committee.
 */
public class Delegate implements Entity {
    private final EntityType type = EntityType.DEL;
    private String id;
    private String name;
    private String delegation; // An alpha2 country code OR a custom delegation. The
        country code is used to fetch the flag icon.
    private String committeeId;
    private int speeches;
    private int pois;
    private int motions;
    private int time;
    private int amendments;

    /**
     * This constructor is used internally by Hibernate
     * and MUST NOT be used in client-facing code.
     */
    public Delegate() {}

    public Delegate(String name, String delegation, String committeeId) {
        this.id = Id.generate(type);
        this.name = name;
        this.delegation = delegation;
        this.committeeId = committeeId;
        this.speeches = 0;
        this.pois = 0;
        this.motions = 0;
        this.time = 0;
        this.amendments = 0;
    }

    public EntityType getType() {
        return type;
    }

    /**
     * Check whether the delegate has a custom delegation
     * or not. A delegation is considered custom if and
     * only if its value is a valid alpha2 territory code.
     * This check should be used to determine whether or
     * not a flag icon is available for a delegation.
     *
     * @return true if the delegation is custom, otherwise false.
     */
    public boolean hasCustomDelegation() {
        return !Country.listOfCodes().contains(delegation);
    }

    /* GETTERS and SETTERS used by hibernate */

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    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

```java
package ibia.core.entities;

/**
```

```java
 * 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

```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

```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 {
    /**
```

```java
     * Icon for most windows created by ibia.
     */
    public static Image IBIA_ICON = new Image("/images/ibia-icon2.png");

    /**
     * Main application window
     */
    private static Stage window;

    /**
     * The scene being displayed on the main stage
     */
    private static Scene scene;

    /**
     * Indicates the current scene being displayed.
     * The value is always an entity ID or "Home" (for the
     * welcome screen).
     */
    private static String location;

    public static void main(String[] args) {
        Log.info("Initializing JavaFX stage.");
        launch(args);
    }

    /**
     * Loads main stage and starts application.
     *
     * @param stage the main Stage
     */
    @Override
    public void start(Stage stage) throws FileNotFoundException, IOException {
        try {
            // First, set a scene to load initially
            Pane root = new Pane();
            root.setMinWidth(1000);
            root.setMinHeight(600);
            scene = new Scene(root);

            // Load that scene and show the window
            window = stage;
            window.setTitle("ibia");
            window.getIcons().add(IBIA_ICON);
            window.setMinWidth(1000);
            window.setMinHeight(600 + 39); // Accomodate for title bar because JavaFX
                doesn't
            window.setScene(scene);
            window.show();

            // Then navigate to the Home screen, and start the application.
            App.navigate("Home");
        } catch (Exception e) {
            Log.error("Failed to load main window: " + e.getMessage());
            e.printStackTrace();
            window.close();
            System.exit(1);
        }
    }

    /**
     * Updates the scene on the application stage.
     *
     * @param scene the scene to display.
     */
    private static void updateScene(Parent content) {
        scene.setRoot(content);
    }

    /**
     * @return current location of the main application stage
     */
    public static String getLocation() {
        return location;
    }

    /**
     * Sets the internal stage location variable.
     * This is controlled by the App class. To change
     * the application's stage location, use App.navigate()
     *
     * @param newLocation
     */
    private static void setLocation(String newLocation) {
        location = newLocation;
    }

    /**
     * Changes the scene displayed on the main application
     * stage based on the id given. The id can
     * be an entity ID, in which case the ID's corresponding
     * view will be loaded. The id may also be the string
     * "Home", in which case the Home scene will be loaded.
     *
     * @param id specifies where to navigate
     * @throws IllegalArgumentException - if an invalid id is provided
     * @throws IOException - if loading FXML fails
     */
    public static void navigate(String id) throws IllegalArgumentException, IOException {
        // If navigating to the same location, no need to update the scsene.
        if (location != null && location.equals(id)) return;
        // Otherwise, update App.location
        setLocation(id);
        if (id.equals("Home")) {
            Parent home = SceneUtil.loadFXML("Home", true);
            updateScene(home);
        } else {
            String entity = id.substring(0, 3);
            switch (entity) {
                case "CON":
                    Parent conferenceScene = SceneUtil.loadFXML("ConferenceView", true);
                    updateScene(conferenceScene);
                    break;
                case "COM":
                    // Do not cache so that the delegate list updates properly
                    // every time.
                    Parent committeeScene = SceneUtil.loadFXML("CommitteeView", false);
                    updateScene(committeeScene);
                    break;
                case "DEL":
                    Parent delegateScene = SceneUtil.loadFXML("DelegateView", true);
                    updateScene(delegateScene);
                    break;
                default:
                    throw new IllegalArgumentException(
                        "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

```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
     * resources/fxml
     *
     * @param name the name of the fxml file (without the extension)
     * @param useCache whether to cache to loaded fxml or not. In some cases
     * it may be preferable not to cache (such as when using the same fxml
     * multiple times within the same scene).
     * @return The scene loaded from the fxml file
     * @throws IOException - if loading FXML fails
     */
    public static Parent loadFXML(String name, boolean useCache) throws IOException {
        return instance._loadFXML(name, useCache);
    }

    /**
     * Same as loadFXML, but puts the FXML into a Scene
     * and returns the Scene.
     *
     * @param name name of the FXML file
     * @param useCache whether to cache the loaded FXML or not
     * @return the Scene created from the FXML
     * @throws IOException - if loading FXML fails
     */
    public static Scene loadFXMLScene(String name, boolean useCache) throws IOException {
        return instance._loadFXMLScene(name, useCache);
    }

    /**
     * Useful for quickly loading FXML with default settings
     * to function as a popup window.
     *
     * @param name name of the FXML file
     * @param title title for the popup window
     * @return the Stage created from the FXML
     * @throws IOException - if loading FXML fails
     */
    public static Stage loadPopupStage(String name, String title) throws IOException {
        return instance._loadPopupStage(name, title);
    }

    /**
     * Returns a Stage for displaying errors.
     *
     * @param msg the error msg
     * @return the created Stage
     */
    public static Stage error(String msg) {
        return instance._error(msg);
    }

    /**
     * Returns a stage showing a confirmation message,
     * along with Confirm and Cancel buttons.
     *
     * @param msg the confirmation message
     * @return the created Stage
     */
    public static Stage confirm(String msg) {
        return instance._confirm(msg);
    }

    /**
     * Implementation for Util.loadFXML
     */
```

```java
    private Parent _loadFXML(String name, boolean useCache) throws IOException {
        if (useCache && cache.containsKey(name)) return cache.get(name);

        name = name.endsWith(".fxml") ? name : name + ".fxml";
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource("/fxml/" + name));
        Parent content = loader.load();

        if (useCache) cache.put(name, content);
        return content;
    }

    /**
     * Implementation for Util.loadFXMLScene
     */
    private Scene _loadFXMLScene(String name, boolean useCache) throws IOException {
        Parent root = _loadFXML(name, useCache);
        return new Scene(root);
    }

    /**
     * Implementation for Util.loadPopupStage
     */
    private Stage _loadPopupStage(String name, String title) throws IOException {
        Stage stage = new Stage();
        Scene scene = _loadFXMLScene(name, true);

        stage.setScene(scene);
        stage.setTitle(title);
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.getIcons().add(App.IBIA_ICON);
        stage.setResizable(false);
        return stage;
    }

    /**
     * Implementation for SceneUtil.error
     */
    private Stage _error(String msg) {
        try {
            Stage stage = new Stage();
            // Load fxml file
            Scene scene = _loadFXMLScene("Error", 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("Error:");
            stage.getIcons().add(new Image("/images/red-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;
        }
    }

    /**
     * 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

```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

```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();
    }

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

    /**
     * @param event the MouseEvent instance
     */
    @FXML
    protected void handleDeleteAction(MouseEvent event) {
        Stage stage = SceneUtil.confirm("Are you sure you wish to delete this committee?
             This action cannot be reversed.");
        Scene root = stage.getScene();
        Button cancel = (Button)root.lookup("#cancel");
        Button confirm = (Button)root.lookup("#confirm");

        cancel.setOnMouseClicked(evt -> {
            stage.close();
        });

        confirm.setOnMouseClicked(evt -> {
            Client.deleteCommittee(instance.getId());
            try {
                App.navigate(instance.getConferenceId());
                stage.close();
            } catch (Exception e) {
                // If Home failed to load, this is a fatal error
                // and the application is exited.
                Log.error(e.getMessage());
                System.exit(1);
            }
        });

        stage.show();
    }

    @FXML
    protected void handleEditAction(MouseEvent event) throws IOException {
        Stage stage = SceneUtil.loadPopupStage("EditCommittee", "Edit Committee");
        stage.show();
    }

    @FXML
    protected void openTimer(MouseEvent event) throws IOException {
        Stage stage = SceneUtil.loadPopupStage("SpeechTimer", "Timer");
        stage.show();
    }

    @FXML
    protected void addNewDelegate(MouseEvent event) throws IOException {
        Stage stage = SceneUtil.loadPopupStage("NewDelegate", "Create new Delegate");
        stage.show();
    }

    @FXML
    protected void openStats(MouseEvent event) {
        SceneUtil.error("Unimplemented!").show();
    }

    @FXML
    protected void openTopics(MouseEvent event) throws IOException {
        Stage stage = SceneUtil.loadPopupStage("Topics", "Topics");
        stage.show();
    }

    @FXML
    protected void openResolutions(MouseEvent event) throws IOException {
        Stage stage = SceneUtil.loadPopupStage("Resolutions", "Resolutions");
        stage.show();
    }

    @FXML
    protected void navigateHome(MouseEvent event) throws IOException,
         IllegalArgumentException {
        App.navigate("Home");
    }

    @FXML
    protected void navigateConference(MouseEvent event) throws IOException,
         IllegalArgumentException {
        String id = instance.getConferenceId();
        App.navigate(id);
    }

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

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

    /******************/
    /*** Templating ***/
    /******************/

    private void fillBreadcrumbs() {
        Conference conf = DbDriver.fetchOne(Conference.class, instance.getConferenceId());
        conferenceCrumb.setText(conf.getName());
        committeeCrumb.setText(instance.getName());
    }

    private void fillName() {
        name.setText(instance.getName());
    }

    private void fillId() {
        id.setText(instance.getId());
    }

    private void fillDelegatesList() throws IOException {
        String comId = instance.getId();
        ArrayList<Delegate> dels = DbDriver.findAll(Delegate.class, d ->
             d.getCommitteeId().equals(comId));
        if (dels == null) return;

        ObservableList<Node> list = delegatesList.getChildren();
        for (Delegate del : dels) {
            FXMLLoader loader = new
                FXMLLoader(getClass().getResource("/fxml/DelegateListItem.fxml"));
            Parent root = loader.load();
            DelegateListItem controller = loader.getController();
            controller.setInstanceId(del.getId());

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

ibia/app/controllers/ConferenceListItem.java

```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

```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;
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();
        }
    }

    @FXML
    protected void handleStatusButtonAction(MouseEvent event) {
        String confId = App.getLocation();
        Conference instance = DbDriver.fetchOne(Conference.class, confId);
        if (instance.isOngoing()) {
            instance.setOngoing(false);
            status.setText("FINISHED");
            statusButton.setText("Re-open Conference");
        } else {
            instance.setOngoing(true);
            status.setText("ONGOING");
            statusButton.setText("Finish Conference");
        }
        DbDriver.updateOne(instance);
    }

    @FXML
    protected void handleEditAction(MouseEvent event) throws IOException {
        Stage stage = SceneUtil.loadPopupStage("EditConference", "Edit Conference");
        stage.show();
    }

    @FXML
    protected void handleDeleteAction(MouseEvent event) {
        Stage stage = SceneUtil.confirm("Are you sure you wish to delete this conference?
             This action cannot be reversed.");
        Scene root = stage.getScene();
        Button cancel = (Button)root.lookup("#cancel");
        Button confirm = (Button)root.lookup("#confirm");

        cancel.setOnMouseClicked(evt -> {
            stage.close();
        });

        confirm.setOnMouseClicked(evt -> {
            Client.deleteConference(instance.getId());
            try {
                App.navigate("Home");
                stage.close();
            } catch (Exception e) {
                // If Home failed to load, this is a fatal error
                // and the application is exited.
                Log.error(e.getMessage());
                System.exit(1);
            }
        });

        stage.show();
    }

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

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

    @FXML
    protected void navigateHome(MouseEvent event) throws IOException,
        IllegalArgumentException {
        App.navigate("Home");
    }

    /******************/
    /*** Templating ***/
    /******************/

    private void fillBreadcrumbs() {
        conferenceCrumb.setText(instance.getName());
    }

    private void fillName() {
        name.setText(instance.getName());
    }

    private void fillId() {
        id.setText(instance.getId());
    }

    private void fillDetails() {
        // Sets OPENED date
        Date date = instance.getCreated();
        SimpleDateFormat fmt = new SimpleDateFormat("dd/MM/yyyy");
        created.setText(fmt.format(date));

        // Sets status to ONGOING or FINISHED
        String currentStatus = instance.isOngoing() ? "ONGOING" : "FINISHED";
        status.setText(currentStatus);

        // Sets number of committees
        ArrayList<Committee> coms = Client.getConferenceCommittees(instance.getId());
        int i = coms != null ? coms.size() : 0;
        committees.setText(Integer.toString(i));

        // Sets number of delegates
        ArrayList<Delegate> dels = new ArrayList<>();
        if (coms != null) {
            for (Committee com : coms) {
                ArrayList<Delegate> fetched = Client.getCommitteeDelegates(com.getId());
                if (fetched != null) {
                    dels.addAll(fetched);
                }
            }
        }
        delegates.setText(Integer.toString(dels.size()));

        // Sets number of resolutions
        ArrayList<Resolution> resos = DbDriver.findAll(Resolution.class, r ->
            r.getPassed());
        int totalResos = 0;
        if (resos != null) {
            for (Resolution reso : resos) {
                for (Delegate del : dels) {
                    if (del.getId().equals(reso.getMainSubmitter())) {
                        totalResos += 1;
                    }
                }
            }
        }
        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

```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

```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;
    @FXML protected Text delegation;

    private Delegate instance;
    private String instanceId;

    @FXML
    public void initialize() {
        // runLater must be used to ensure
        // that the instance id has been initialized
        // in CommitteeView#fillDelegatesList
        Platform.runLater(() -> {
            this.instance = DbDriver.fetchOne(Delegate.class, instanceId);

            delegation.setText(instance.getDelegation());
            id.setText("#" + instance.getId());

            // Update cells with delegates' data
            String a = Integer.toString(instance.getSpeeches());
            speeches.setText(a);
            String b = Integer.toString(instance.getPois());
            pois.setText(b);
            String c = Integer.toString(instance.getAmendments());
            amendments.setText(c);
            String d = Integer.toString(instance.getMotions());
            motions.setText(d);

            // Set listeners for updating the db when cell
            // values are changed.
            attachListeners();

            String code = Country.codeFromName(instance.getDelegation());
            if (code != null) {
                InputStream stream = Country.getFlag(code);
                Image img = new Image(stream);
                flag.setImage(img);
            }
        });
    }

    // Attach listeners to update the
    // db with the value of the cells
    // whenever the TextField goes out
    // of focus.
    private void attachListeners() {
        speeches.focusedProperty().addListener((observable, oldFocus, newFocus) -> {
            // Run code when node is out of focus
            if (!newFocus) {
                int n;
                try {
                    String value = speeches.getText();
                    String nonEmpty = value.isEmpty() ? "0" : value;
                    n = Integer.parseInt(nonEmpty);
                } catch (NumberFormatException e) {
                    n = instance.getSpeeches();
                    speeches.setText(Integer.toString(n));
                }
                instance.setSpeeches(n);
                DbDriver.updateOne(instance);
            }
        });

        pois.focusedProperty().addListener((observable, oldFocus, newFocus) -> {
            // Run code when node is out of focus
            if (!newFocus) {
                int n;
                try {
                    String value = pois.getText();
                    String nonEmpty = value.isEmpty() ? "0" : value;
                    n = Integer.parseInt(nonEmpty);
                } catch (NumberFormatException e) {
                    // If a non-integer is entered,
                    // revert back to the previous value.
                    n = instance.getPois();
                    pois.setText(Integer.toString(n));
                }
                instance.setPois(n);
                DbDriver.updateOne(instance);
            }
        });

        amendments.focusedProperty().addListener((observabe, oldFocus, newFocus) -> {
            // Run code when node is out of focus
            if (!newFocus) {
                int n;
                try {
                    String value = amendments.getText();
                    String nonEmpty = value.isEmpty() ? "0" : value;
                    n = Integer.parseInt(nonEmpty);
                } catch (NumberFormatException e) {
                    n = instance.getAmendments();
                    amendments.setText(Integer.toString(n));
                }
                instance.setAmendments(n);
                DbDriver.updateOne(instance);
            }
        });

        motions.focusedProperty().addListener((observable, oldFocus, newFocus) -> {
            // Run code when node is out of focus
            if (!newFocus) {
                int n;
                try {
                    String value = motions.getText();
                    String nonEmpty = value.isEmpty() ? "0" : value;
                    n = Integer.parseInt(nonEmpty);
                } catch (NumberFormatException e) {
                    n = instance.getMotions();
                    motions.setText(Integer.toString(n));
                }
                instance.setMotions(n);
                DbDriver.updateOne(instance);
            }
        });
    }

    @FXML
    protected void navigate(MouseEvent event) throws IOException {
        App.navigate(instance.getId());
    }

    @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);
    }

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

ibia/app/controllers/DelegateView.java

```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();
        fillDetails();
        fillFlag();
    }

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

    /**
     * @param event the MouseEvent instance
     * @throws IOException - if loading FXML fails
     */
    @FXML
    protected void handleEditAction(MouseEvent event) throws IOException {
        Stage stage = SceneUtil.loadPopupStage("EditDelegate", "Edit Delegate");
        stage.show();
    }

    @FXML
    protected void handleDeleteAction(MouseEvent event) {
        Stage stage = SceneUtil.confirm("Are you sure you wish to delete this delegate?
            This action cannot be reversed.");
        Scene root = stage.getScene();
        Button cancel = (Button)root.lookup("#cancel");
        Button confirm = (Button)root.lookup("#confirm");

        cancel.setOnMouseClicked(evt -> {
            stage.close();
        });

        confirm.setOnMouseClicked(evt -> {
            Client.deleteDelegate(instance.getId());
            try {
                App.navigate(instance.getCommitteeId());
                stage.close();
            } catch (Exception e) {
                // If Home failed to load, this is a fatal error
                // and the application is exited.
                Log.error(e.getMessage());
                System.exit(1);
            }
        });

        stage.show();
    }

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

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

    @FXML
    protected void navigateHome(MouseEvent event) throws IOException {
        App.navigate("Home");
    }

    @FXML
    protected void navigateCommittee(MouseEvent event) throws IOException {
        String id = instance.getCommitteeId();
        App.navigate(id);
    }

    @FXML
    protected void navigateConference(MouseEvent event) throws IOException {
        String comId = instance.getCommitteeId();
        Committee com = DbDriver.fetchOne(Committee.class, comId);
        String conId = com.getConferenceId();
        App.navigate(conId);
    }

    /******************/
    /*** Templating ***/
    /******************/

    private void fillBreadcrumbs() {
        delegateCrumb.setText(instance.getDelegation());

        String comId = instance.getCommitteeId();
        Committee com = DbDriver.fetchOne(Committee.class, comId);
        committeeCrumb.setText(com.getName());
        Conference conf = DbDriver.fetchOne(Conference.class, com.getConferenceId());
        conferenceCrumb.setText(conf.getName());
    }

    private void fillDelegation() {
        delegation.setText(instance.getDelegation());
    }

    private void fillId() {
        id.setText(instance.getId());
    }

    private void fillDetails() {
        delName.setText(instance.getName());
        comName.setText(committeeCrumb.getText());
        String a = Integer.toString(instance.getSpeeches());
        speeches.setText(a);
        String b = Integer.toString(instance.getPois());
        pois.setText(b);
        String c = Integer.toString(instance.getAmendments());
        amendments.setText(c);
        String d = Integer.toString(instance.getMotions());
        motions.setText(d);
        ArrayList<Resolution> submitted = DbDriver.findAll(Resolution.class, r ->
            r.getMainSubmitter().equals(instance.getId()));
        int n = submitted != null ? submitted.size() : 0;
        resos.setText(Integer.toString(n));
    }

    private void fillFlag() {
        String code = Country.codeFromName(instance.getDelegation());
        InputStream stream = Country.getFlag(code);
        if (stream != null) {
            Image img = new Image(stream);
            flag.setImage(img);
        }
    }

}
```

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;
        }

        try {
            String delId = App.getLocation();
            Delegate del = DbDriver.fetchOne(Delegate.class, delId);
            del.setName(delName);
            del.setDelegation(delegationStr);
            DbDriver.updateOne(del);
            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/Home.java

```java
package ibia.app.controllers;

import java.awt.Desktop;
import java.io.IOException;
import java.net.URI;
import java.util.ArrayList;

import javafx.fxml.FXML;
import javafx.stage.Stage;
import javafx.scene.Group;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;

import ibia.app.SceneUtil;
import ibia.core.DbDriver;
import ibia.core.entities.Conference;

public class Home {
    @FXML protected Text resumeMsg;

    @FXML
    public void initialize() throws IOException {
        ArrayList<Conference> confs = DbDriver.findAll(Conference.class, c ->
             c.isOngoing());
        if (confs != null && confs.size() > 0) {
            resumeMsg.setText("Click to view ongoing conferences");
        }
    }

    @FXML
    protected void handleResumeConfAction() throws IOException {
        Stage stage = SceneUtil.loadPopupStage("OngoingConferences", "Choose an ongoing
             Conference");
        stage.show();
    }

    @FXML
    protected void handleNewConfAction() throws IOException {
        Stage stage = SceneUtil.loadPopupStage("NewConference", "Create new Conference");
        stage.show();
    }

    @FXML
    protected void handlePastConfAction() throws IOException {
        Stage stage = SceneUtil.loadPopupStage("PastConferences", "Choose a finished
             Conference");
        stage.show();
    }

    @FXML
    protected void handleGuidesAction() {
        String url = "https://github.com/quantomistro/ibia-app";

        try {
            openURL(url);
        } catch (Exception e) {
            SceneUtil.error("Failed to open URL!").show();
        }
    }

    @FXML
    protected void handleAboutAction() throws IOException {
        Stage stage = SceneUtil.loadPopupStage("About", "About ibia");
        stage.show();
    }

    @FXML
    protected void handleFeedbackAction() {
        String url = "https://github.com/quantomistro/ibia-app/issues/";

        try {
            openURL(url);
        } catch (Exception e) {
            SceneUtil.error("Failed to open URL!").show();
        }
    }

    @FXML
    protected void handleViewLogsAction() throws IOException {
        Stage stage = SceneUtil.loadPopupStage("Logs", "Logs");
        stage.show();
    }

    @FXML
    protected void hoverEffectOn(MouseEvent event) {
        Group btn = (Group)event.getTarget();

        Rectangle rect = (Rectangle)btn.getChildren().get(0);
        rect.setStrokeWidth(2);
        rect.setStroke(Color.WHITE);
    }

    @FXML
    protected void hoverEffectOff(MouseEvent event) {
        Group btn = (Group)event.getTarget();

        Rectangle rect = (Rectangle)btn.getChildren().get(0);
        rect.setStrokeWidth(0);
        rect.setStroke(null);
    }

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

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

    private void openURL(String url) throws Exception {
        if (Desktop.isDesktopSupported() &&
            Desktop.getDesktop().isSupported(Desktop.Action.BROWSE)) {
            Desktop.getDesktop().browse(new URI(url));
        }
    }
}
```

ibia/app/controllers/Logs.java

```java
package ibia.app.controllers;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

import javafx.fxml.FXML;
import javafx.scene.control.TextArea;

public class Logs {
    @FXML protected TextArea textArea;

    @FXML
    public void initialize() throws FileNotFoundException, IOException {
        FileReader fr = new FileReader("data/ibia.log");
        // Use BufferedReader for fast reading
        BufferedReader br = new BufferedReader(fr);
        String contents = "";

        while (true) {
            String line = br.readLine();
            if (line == null) break;
            contents += line + "\n";
        }

        br.close();
        textArea.setText(contents);
    }
}
```

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;
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;
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);
    }

    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/NewResolution.java

```java
package ibia.app.controllers;

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

import ibia.app.App;
import ibia.app.SceneUtil;
import ibia.core.DbDriver;
import ibia.core.entities.Delegate;
import ibia.core.utils.Resolution;
import ibia.core.utils.Topic;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
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 NewResolution {
    @FXML protected TextField delegate;
    @FXML protected MenuButton delegateDropdown;
    @FXML protected TextField topic;
    @FXML protected MenuButton topicDropdown;

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

        ArrayList<Delegate> dels = DbDriver.findAll(Delegate.class, d ->
            d.getCommitteeId().equals(comId));
        if (dels != null) {
            for (Delegate del : dels) {
                MenuItem item = new MenuItem(del.getDelegation());
                item.setText(del.getDelegation());
                item.setId(del.getId());
                item.setOnAction((ActionEvent event) -> {
                    delegate.setText(del.getDelegation());
                    delegate.setId(del.getId());
                });
                delegateDropdown.getItems().add(item);
            }
        }

        ArrayList<Topic> topics = DbDriver.findAll(Topic.class, t ->
            t.getCommitteeId().equals(comId));
        if (topics != null) {
            for (Topic t : topics) {
                MenuItem item = new MenuItem(t.getTopic());
                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

```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;

   @FXML
   public void initialize() throws IOException {
       ArrayList<Conference> confs = DbDriver.findAll(Conference.class, c ->
            c.isOngoing());
       if (confs != null) {
           for (Conference conf : confs) {
               HBox item = (HBox)SceneUtil.loadFXML("ConferenceListItem", false);
               Text name = (Text)item.lookup("#name");
               name.setText(conf.getName());
               Text id = (Text)item.lookup("#id");
               id.setText("#" + conf.getId());
               ongoingList.getChildren().add(item);
           }
       }
   }
}
```

ibia/app/controllers/PastConferences.java

```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 PastConferences {
    @FXML protected VBox pastList;

    @FXML
    public void initialize() throws IOException {
        ArrayList<Conference> confs = DbDriver.findAll(Conference.class, c ->
             !c.isOngoing());
        if (confs != null) {
            for (Conference conf : confs) {
                HBox item = (HBox)SceneUtil.loadFXML("ConferenceListItem", false);
                Text name = (Text)item.lookup("#name");
                name.setText(conf.getName());
                Text id = (Text)item.lookup("#id");
                id.setText("#" + conf.getId());
                pastList.getChildren().add(item);
            }
        }
    }
}
```

ibia/app/controllers/Resolutions.java

```java
package ibia.app.controllers;

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

import ibia.app.App;
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;
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

```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;

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
        // that the instance id has been initialized
        // from the Topics class.
        Platform.runLater(() -> {
            this.instance = DbDriver.fetchOne(Topic.class, instanceId);
            topic.setText(instance.getTopic());

            attachListener();
        });
    }

    private void attachListener() {
        topic.focusedProperty().addListener((observable, oldFocus, newFocus) -> {
            // Run code when node is out of focus
            if (!newFocus) {
                String value = topic.getText();

                if (value.isEmpty()) {
                    DbDriver.deleteById(Topic.class, instance.getId());
                    HBox container = (HBox)topic.getParent();
                    VBox list = (VBox)container.getParent();
                    list.getChildren().remove(container);
                } else {
                    instance.setTopic(value);
                    DbDriver.updateOne(instance);
                }
            }
        });
    }

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