Message.ly Solution

Springboard

Download solution

Models

models/user.js

```
/** User class for message.ly */

const db = require("../db");
const bcrypt = require("bcrypt");
const ExpressError = require("../expressError");

const { BCRYPT_WORK_FACTOR } = require("../config");


/** User of the site. */

class User {

  /** register new user -- returns
   *    {username, password, first_name, last_name, phone}
   */

  static async register({username, password, first_name, last_name, phone}) {
    let hashedPassword = await bcrypt.hash(password, BCRYPT_WORK_FACTOR);
    const result = await db.query(
          `INSERT INTO users (
              username,
              password,
              first_name,
              last_name,
              phone,
              join_at,
              last_login_at)
            VALUES ($1, $2, $3, $4, $5, current_timestamp, current_timestamp)
            RETURNING username, password, first_name, last_name, phone`,
        [username, hashedPassword, first_name, last_name, phone]
    );
    return result.rows[0];
  }

  /** Authenticate: is this username/password valid? Returns boolean. */

  static async authenticate(username, password) {
    const result = await db.query(
        `SELECT password FROM users WHERE username = $1`,
        [username]
    );
    let user = result.rows[0];

    return user && await bcrypt.compare(password, user.password);
  }

  /** Update last_login_at for user */

  static async updateLoginTimestamp(username) {
    const result = await db.query(
          `UPDATE users
           SET last_login_at = current_timestamp
           WHERE username = $1
           RETURNING username`,
        [username]);

    if (!result.rows[0]) {
      throw new ExpressError(`No such user: ${username}`, 404);
    }
  }

  /** All: basic info on all users:
   * [{username, first_name, last_name}, ...] */

  static async all() {
    const result = await db.query(
          `SELECT username,
                  first_name,
                  last_name,
                  phone
           FROM users
           ORDER BY username`);

    return result.rows;
  }

  /** Get: get user by username
   *
   * returns {username,
   *          first_name,
   *          last_name,
   *          phone,
   *          join_at,
   *          last_login_at } */

  static async get(username) {
    const result = await db.query(
          `SELECT username,
                  first_name,
                  last_name,
                  phone,
                  join_at,
                  last_login_at
           FROM users
           WHERE username = $1`,
        [username]);

    if (!result.rows[0]) {
      throw new ExpressError(`No such user: ${username}`, 404);
    }

    return result.rows[0];
  }

  /** Return messages from this user.
   *
   * [{id, to_user, body, sent_at, read_at}]
   *
   * where to_user is
   *   {username, first_name, last_name, phone}
   */

  static async messagesFrom(username) {
    const result = await db.query(
          `SELECT m.id,
                  m.to_username,
                  u.first_name,
                  u.last_name,
                  u.phone,
                  m.body,
                  m.sent_at,
                  m.read_at
           FROM messages AS m
             JOIN users AS u ON m.to_username = u.username
           WHERE from_username = $1`,
        [username]);

    return result.rows.map(m => ({
      id: m.id,
      to_user: {
        username: m.to_username,
        first_name: m.first_name,
        last_name: m.last_name,
        phone: m.phone
      },
      body: m.body,
      sent_at: m.sent_at,
      read_at: m.read_at
    }));
  }

  /** Return messages to this user.
   *
   * [{id, from_user, body, sent_at, read_at}]
   *
   * where from_user is
   *   {username, first_name, last_name, phone}
   */

  static async messagesTo(username) {
    const result = await db.query(
          `SELECT m.id,
                  m.from_username,
                  u.first_name,
                  u.last_name,
                  u.phone,
                  m.body,
                  m.sent_at,
                  m.read_at
           FROM messages AS m
             JOIN users AS u ON m.from_username = u.username
           WHERE to_username = $1`,
        [username]);

    return result.rows.map(m => ({
      id: m.id,
      from_user: {
        username: m.from_username,
        first_name: m.first_name,
        last_name: m.last_name,
        phone: m.phone,
      },
      body: m.body,
      sent_at: m.sent_at,
      read_at: m.read_at,
    }));
  }
}

module.exports = User;
```

models/message.js

```
/** Message class for message.ly */

const db = require("../db");
const ExpressError = require("../expressError");


/** Message on the site. */

class Message {

  /** register new message -- returns
   *    {id, from_username, to_username, body, sent_at}
   */

  static async create({from_username, to_username, body}) {
    const result = await db.query(
          `INSERT INTO messages (
              from_username,
              to_username,
              body,
              sent_at)
            VALUES ($1, $2, $3, current_timestamp)
            RETURNING id, from_username, to_username, body, sent_at`,
        [from_username, to_username, body]);

    return result.rows[0];
  }

  /** Update read_at for message */

  static async markRead(id) {
    const result = await db.query(
          `UPDATE messages
           SET read_at = current_timestamp
           WHERE id = $1
           RETURNING id, read_at`,
        [id]);

    if (!result.rows[0]) {
      throw new ExpressError(`No such message: ${id}`, 404);
    }

    return result.rows[0];
  }

  /** Get: get message by id
   *
   * returns {id, from_user, to_user, body, sent_at, read_at}
   *
   * both to_user and from_user = {username, first_name, last_name, phone}
   *
   */

  static async get(id) {
    const result = await db.query(
          `SELECT m.id,
                  m.from_username,
                  f.first_name AS from_first_name,
                  f.last_name AS from_last_name,
                  f.phone AS from_phone,
                  m.to_username,
                  t.first_name AS to_first_name,
                  t.last_name AS to_last_name,
                  t.phone AS to_phone,
                  m.body,
                  m.sent_at,
                  m.read_at
           FROM messages AS m
             JOIN users AS f ON m.from_username = f.username
             JOIN users AS t ON m.to_username = t.username
           WHERE m.id = $1`,
        [id]);

    let m = result.rows[0];

    if (!m) {
      throw new ExpressError(`No such message: ${id}`, 404);
    }

    return {
      id: m.id,
      from_user: {
        username: m.from_username,
        first_name: m.from_first_name,
        last_name: m.from_last_name,
        phone: m.from_phone,
      },
      to_user: {
        username: m.to_username,
        first_name: m.to_first_name,
        last_name: m.to_last_name,
        phone: m.to_phone,
      },
      body: m.body,
      sent_at: m.sent_at,
      read_at: m.read_at,
    };
  }
}

module.exports = Message;
```

Routes

routes/auth.js

```
const jwt = require("jsonwebtoken");
const Router = require("express").Router;
const router = new Router();

const User = require("../models/user");
const {SECRET_KEY} = require("../config");
const ExpressError = require("../expressError");


/** login: {username, password} => {token} */

router.post("/login", async function (req, res, next) {
  try {
    let {username, password} = req.body;
    if (await User.authenticate(username, password)) {
      let token = jwt.sign({username}, SECRET_KEY);
      User.updateLoginTimestamp(username);
      return res.json({token});
    } else {
      throw new ExpressError("Invalid username/password", 400);
    }
  }

  catch (err) {
    return next(err);
  }
});


/** register user: registers, logs in, and returns token.
 *
 * {username, password, first_name, last_name, phone} => {token}.
 *
 */


router.post("/register", async function (req, res, next) {
  try {
    let {username} = User.register(req.body);
    let token = jwt.sign({username}, SECRET_KEY);
    User.updateLoginTimestamp(username);
    return res.json({token});
  }

  catch (err) {
    return next(err);
  }
});


module.exports = router;
```

routes/users.js

```
const Router = require("express").Router;
const User = require("../models/user");
const {ensureLoggedIn, ensureCorrectUser} = require("../middleware/auth");

const router = new Router();


/** get list of users.
 *
 * => [users: {username, first_name, last_name, phone}, ...]}
 *
 **/


router.get("/", ensureLoggedIn, async function (req, res, next) {
  try {
    let users = await User.all();
    return res.json({users});
  }

  catch (err) {
    return next(err);
  }
});


/** get detail of users.
 *
 * => {user: username, first_name, last_name, phone, join_at, last_login_at}
 *
 **/


router.get("/username", ensureCorrectUser, async function (req, res, next) {
  try {
    let user = await User.get(req.params.username);
    return res.json({user});
  }

  catch (err) {
    return next(err);
  }
});


/** get messages to user
 *
 * => {messages: [{id,
 *                 body,
 *                 sent_at,
 *                 read_at,
 *                 from_user: {username, first_name, last_name, phone}}, ...]}
 *
 **/


router.get("/username/to", ensureCorrectUser, async function (req, res, next) {
  try {
    let messages = await User.messagesTo(req.params.username);
    return res.json({messages});
  }

  catch (err) {
    return next(err);
  }
});


/** get messages from user
 *
 * => {messages: [{id,
 *                 body,
 *                 sent_at,
 *                 read_at,
 *                 to_user: {username, first_name, last_name, phone}}, ...]}
 *
 **/


router.get("/username/from", ensureCorrectUser, async function (req, res, next) {
  try {
    let messages = await User.messagesFrom(req.params.username);
    return res.json({messages});
  }

  catch (err) {
    return next(err);
  }
});


module.exports = router;
```