[Tweety]

[A Twitter Clone]

Tweety, being a Twitter clone, is supposed to act like a social media application that allows multiple users to communicate and interact with each other. (This application, however, is being made purely to demonstrate the features of Angular, Spring MVC, and their ability to integrate. This application is in no way a competitor or replacement for Twitter and doesn't include all the features of such a social media application.) The Tweety application must be publicly accessible via the internet, and registered users must be able to send out tweets. The main feed must be visible without any authentication and the user feed must only be visible after authentication.

Developer

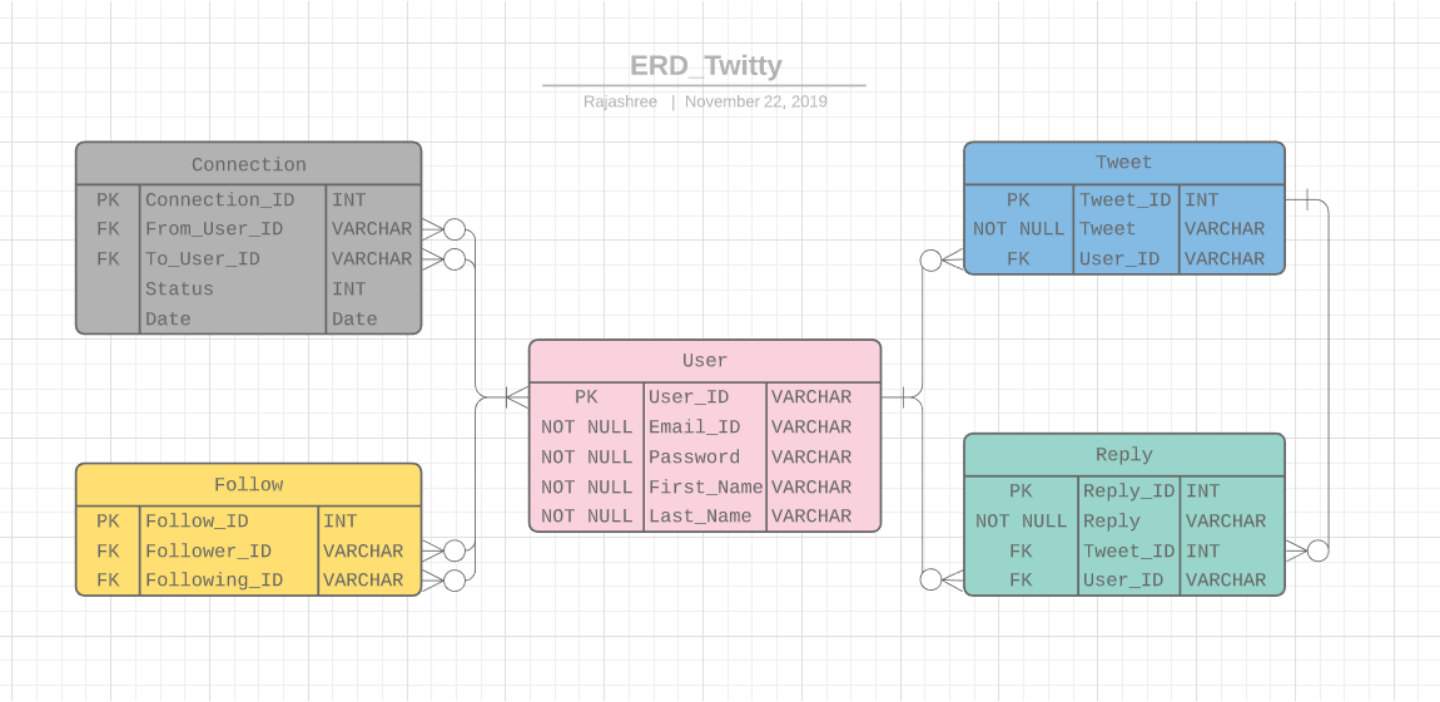
Rajashree Joshi

[Tweety Architecture]

The application architecture for Tweety will have a RESTful API backend and a model-view-viewmodel (MVVM) frontend. MVVM is a design pattern used to enable two-way data binding between the view and the model so that one changes when the other does. Angular is a very famous framework that conforms to the MVVM design pattern, which allows highly responsive frontend applications that efficiently update when data from the server changes or a user interacts with the application. Apart from this, Angular provides routing, dependency injection, components, templates, and so on to enable flexible, modular development.

[Class diagram for the domain model]

The following is the simple class diagram for this web service:



There are three main domain models shown in the preceding diagram. Those are as follows:

Tweet: This is the main domain model, which will store the actual tweet content and posted user

User: This is the domain model that will store the username, password, role, and bio of each user on the system

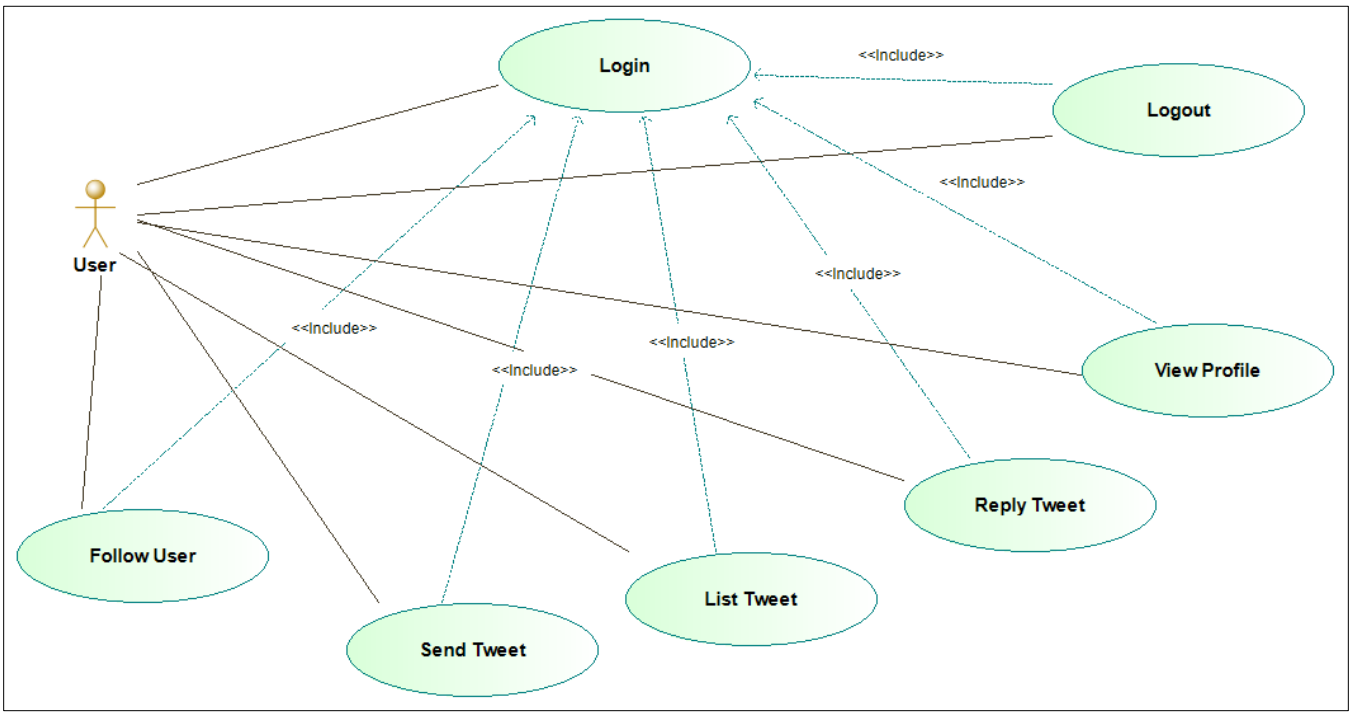
Reply: This is the domain model that will store reply/ comment in the tweet, tweet id and user id of the commenting user.

Follow: This is the domain model that w ill store follower and following user id

Connection: This is the domain model that will store connection of two users by their id, status of their connection and date of the connection.

[The use case diagram]

The following use case diagram shows the requirement for Tweety:



The actor is the User of the Tweety application. It has the following use cases:

Login: This use case is required to authenticate users so that each user can be uniquely identified, to allow only authenticated users to perform actions.

List Tweet: This use case is where a user can list all the tweets available for that user. It requires the user to be authenticated.

Reply Tweet: This use case is where a user can reply to a Tweet. It requires the user to be authenticated.

Send Tweet: This use case is where a user can send a Tweet. It requires the user to be authenticated.

Follow User: This use case is where a user can follow another user. It requires the user to be authenticated.

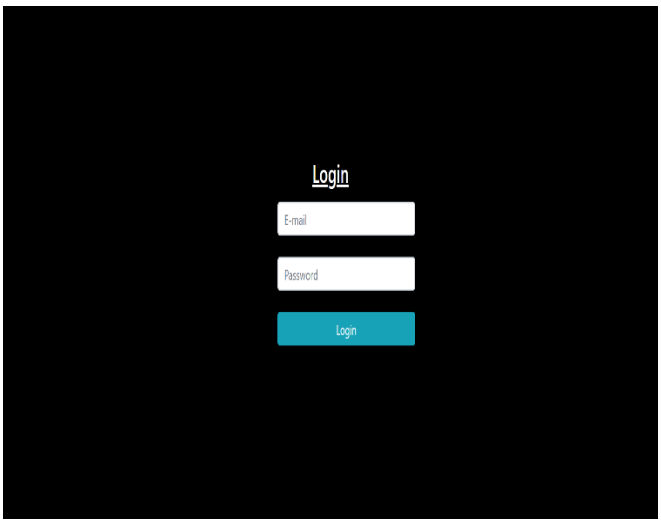
View Profile: This use case is where a user's profile can be viewed by another. It requires the user to be authenticated.

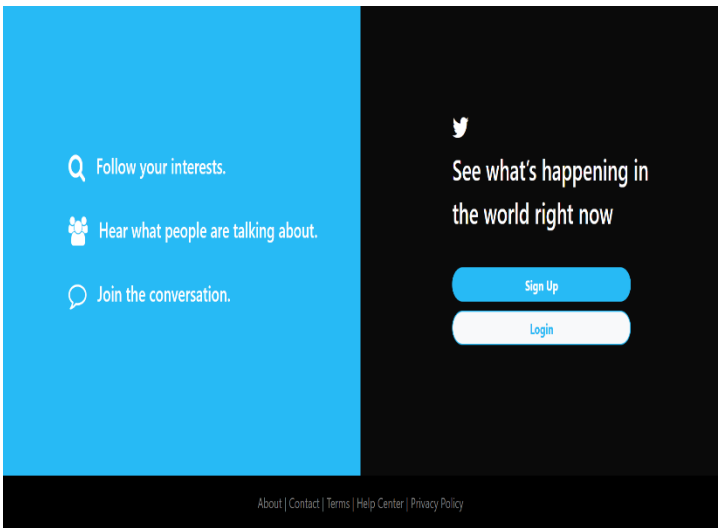
Logout: This use case is where a logged in user can log out.

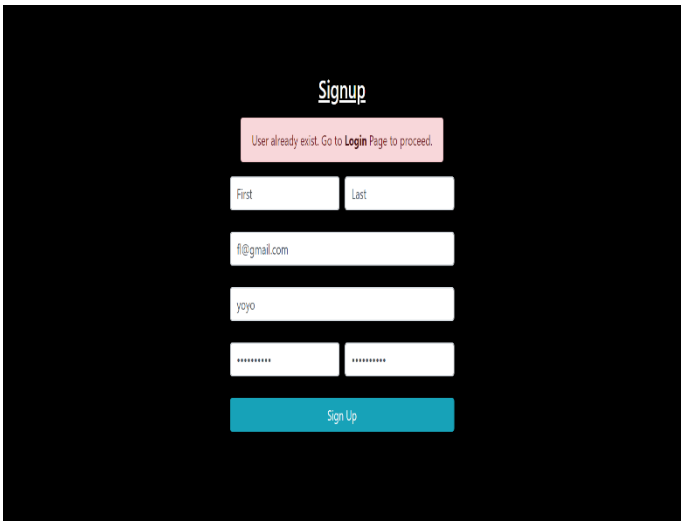
Connection: This use case is where a user can send connection request to another user.

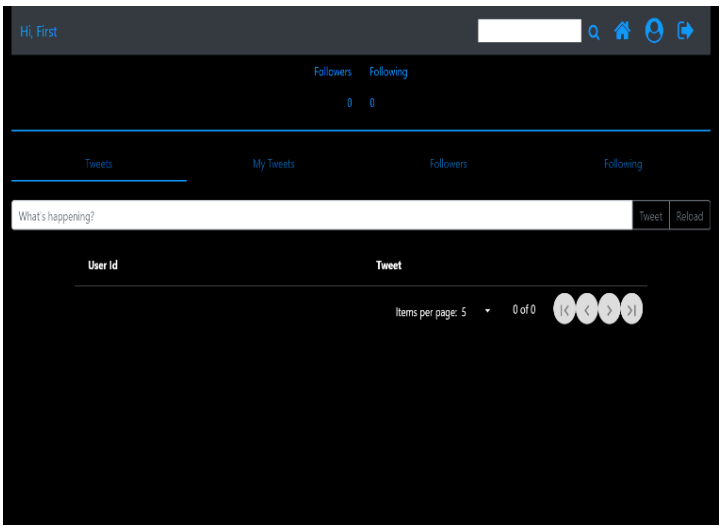
[Screenshots]

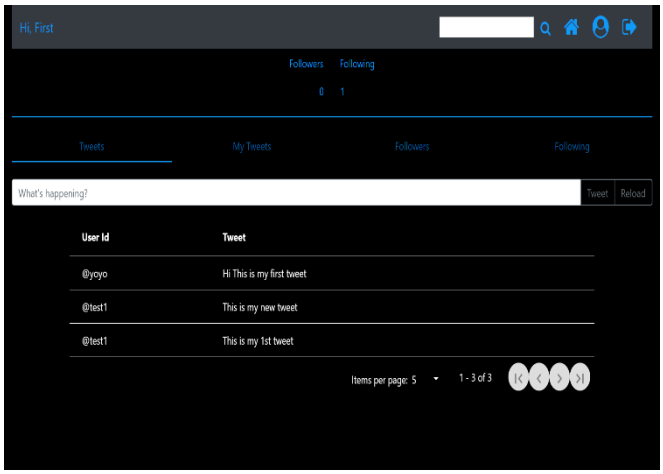
Below are some screenshots of the major screens:



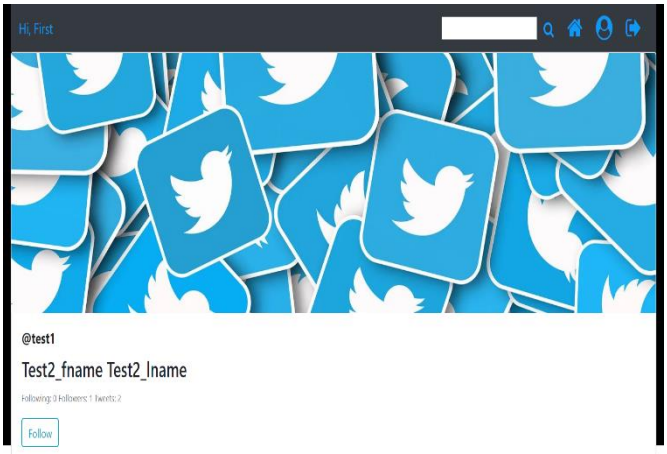


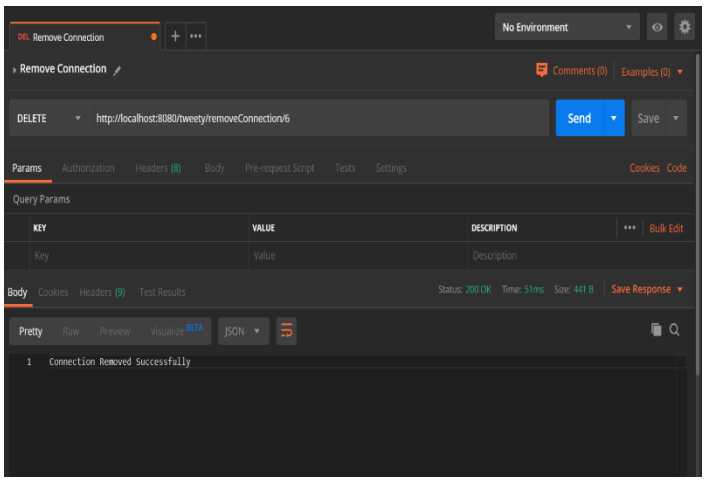


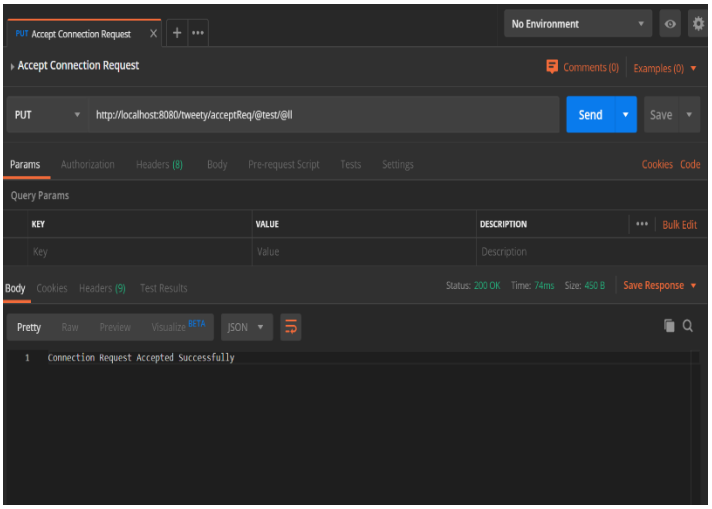


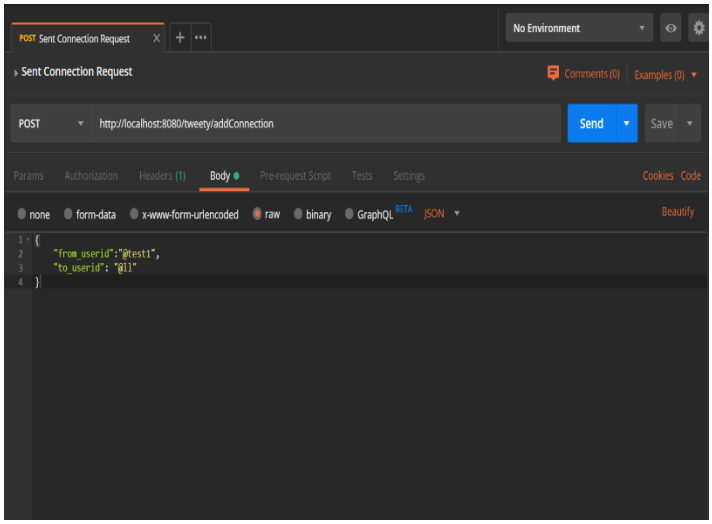

Hi, First
Followers Following
0 1
Tweets My Tweets Followers Following
What's happening?
Tweet Reload
User Id Tweet
@yoyo Hi This is my first tweet
@test1 This is my new tweet
@test1 This is my 1st tweet
Items per page: 5
1 - 3 of 3

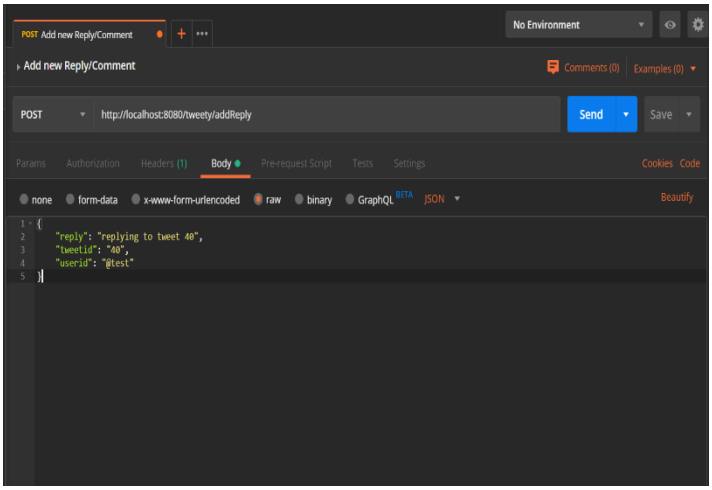

Hi, First
@te
@test|test_first|test_last
@test1|Test2_fname|Test2_lname
Followers Following
0 0
Tweets My Tweets Followers Following
Reload
User Id Tweet Action
@yoyo Hi This is my first tweet Delete

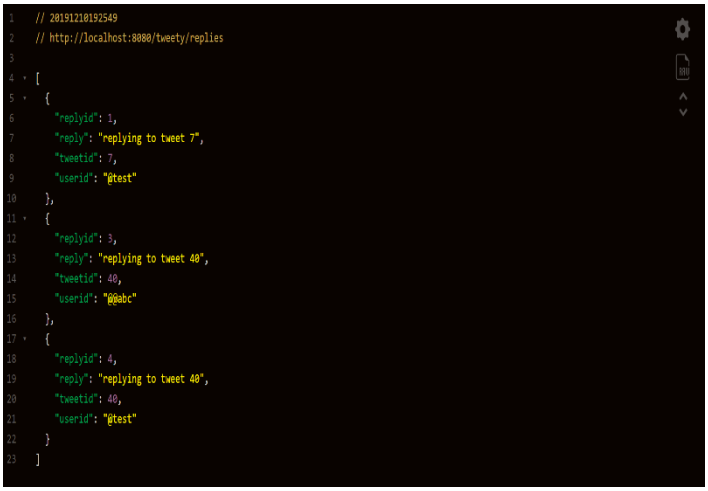

Hi, First
@test1
Test2_fname Test2_lname
Following:3 Followers:1 Tweets:2
Follow


Remove Connection
No Environment
Remove Connection
Comments (0) Examples (0)
DELETE http://localhost:8080/tweety/removeConnection/6
Send Save
Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Code
Query Params
KEY VALUE DESCRIPTION Bulk Edit
Key Value Description
Body Cookies Headers (9) Test Results
Status: 200 OK Time: 51ms Size: 441 B Save Response
Pretty Raw Preview Visualize BETA JSON
1 Connection Removed Successfully









[Controllers' Code Snippet]

User Controller

```
@Controller
public class UserController {
	private static final Logger Logger = LoggerFactory.getLogger(UserController.class);

	@Autowired
	TweetyDBService dbService;

	@RequestMapping(value = "/users", method = RequestMethod.GET, produces="application/json")
	public @ResponseBody ArrayList<User> getUsers(Model model) {
		ArrayList<User> list = (ArrayList<User>)dbService.getUsers();
		model.addAttribute("usersList", list);
		return list;
	}

	@RequestMapping(value="/register", method=RequestMethod.POST)
	@ResponseBody
```

```
	public User addUser(@RequestBody User user) {
		System.out.println("User: "+ user.getEmailid());
		return dbService.addNewUser(user);
	}

	@RequestMapping(value="/remove/{id}", method=RequestMethod.DELETE)
	@ResponseBody
	public ResponseEntity<String> removeUser(@PathVariable String id) {
		System.out.println("User id to be removed: "+ id);
		if(dbService.removeUser(id))
			return new ResponseEntity<String>("success", HttpStatus.ACCEPTED);
		else
			return new ResponseEntity<String>("invalid input or no such user exist",
HttpStatus.OK);
	}

	@RequestMapping(value="/update/{id}", method=RequestMethod.PUT)
	@ResponseBody
	public ResponseEntity<String> updateUser(@PathVariable String id, @RequestBody User user){
		System.out.println("User id to be updated: "+ id);
		if(dbService.updateUser(id, user))
			return new ResponseEntity<String>("success", HttpStatus.ACCEPTED);
		else
			return new ResponseEntity<String>("invalid input or no such user exist",
HttpStatus.OK);
	}

	@RequestMapping(value = "/user/{searchkey}/{type}", method = RequestMethod.GET,
produces="application/json")
	public @ResponseBody ResponseEntity<User> getUser(@PathVariable String searchkey,
@PathVariable String type) {
		System.out.println("Search Key: "+ searchkey + " Type: "+ type);
		User user = (User)dbService.getUser(searchkey, type);
		if(user == null) {
			return new ResponseEntity<User>(user, HttpStatus.OK);
		}
		return new ResponseEntity<User>(user, HttpStatus.OK);
	}

	@RequestMapping(value = "/search/{searchkey}", method = RequestMethod.GET,
produces="application/json")
	public @ResponseBody ResponseEntity<List<Search>> searchUser(@PathVariable String
searchkey) {
		System.out.println("Search Key: "+ searchkey);
		List<Search> user = dbService.searchUser(searchkey);
		if(user == null) {
			return new ResponseEntity<List<Search>>(user, HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<List<Search>>(user, HttpStatus.OK);
	}
}
```

Connection Controller

```
@Controller
public class ConnectionController {
	private static final Logger Logger = LoggerFactory.getLogger(ConnectionController.class);

	@Autowired
	TweetyDBService dbService;

	@RequestMapping(value = "/addConnection", method = RequestMethod.POST, produces =
"application/json")
	public @ResponseBody ResponseEntity<ConRequest> addConnection(@RequestBody Connection con)
{
		int flag = dbService.isConnectionExists(con);
		if (flag == 1) {
			ConRequest cr = new ConRequest();
			cr.setRequestConfirmation("You are already connected!");
			cr.setUser(new User());
			return new ResponseEntity<ConRequest>(cr, HttpStatus.CONFLICT);
		} else if (flag == 0) {
			ConRequest cr = new ConRequest();
			cr.setRequestConfirmation("Connection Request is pending!");
			cr.setUser(new User());
			return new ResponseEntity<ConRequest>(cr, HttpStatus.CONFLICT);
		} else {
			User user = dbService.addConnection(con);
			if(null == user.getUserid()) {
				ConRequest cr = new ConRequest();
				cr.setRequestConfirmation("Error while adding connection into the
databse!");
				cr.setUser(new User());
				return new ResponseEntity<ConRequest>(cr, HttpStatus.BAD_REQUEST);
			}
			ConRequest cr = new ConRequest();
			cr.setRequestConfirmation("Connection Request Sent Successfully!");
			cr.setUser(user);
			return new ResponseEntity<ConRequest>(cr, HttpStatus.ACCEPTED);
		}
	}

	@RequestMapping(value = "/connections", method = RequestMethod.GET, produces =
"application/json")
	public @ResponseBody ArrayList<Connection> getAllConnections() {
		ArrayList<Connection> list = (ArrayList<Connection>) dbService.getConnections();
		return list;
	}

	@RequestMapping(value = "/connections/{id}", method = RequestMethod.GET, produces =
"application/json")
	public @ResponseBody ArrayList<Connection> myConnections(@PathVariable String id) {
		ArrayList<Connection> list = (ArrayList<Connection>) dbService.getMyConnections(id);
		return list;
	}

	@RequestMapping(value = "/removeConnection/{id}", method = RequestMethod.DELETE, produces =
"application/json")
	public @ResponseBody ResponseEntity<String> removeConnection(@PathVariable String id) {
		if (dbService.removeConnection(Integer.parseInt(id)))
			return new ResponseEntity<String>("Connection Removed Successfully",
HttpStatus.OK);
```

```
		else
			return new ResponseEntity<String>("Error while removing Connection",
HttpStatus.BAD_GATEWAY);
	}

	@RequestMapping(value = "/acceptReq/{fid}/{tid}", method = RequestMethod.PUT, produces =
"application/json")
	public @ResponseBody ResponseEntity<String> acceptConnectionRequest(@PathVariable String
fid,
			@PathVariable String tid) {
		if (dbService.acceptRequest(fid, tid))
			return new ResponseEntity<String>("Connection Request Accepted Successfully",
HttpStatus.OK);
		else
			return new ResponseEntity<String>("Error while accepting the Connection
Request", HttpStatus.BAD_GATEWAY);
	}
}
```

Follow Controller

```
@Controller
public class FollowController {
private static final Logger Logger = LoggerFactory.getLogger(FollowController.class);

	@Autowired
	TweetyDBService dbService;

	@RequestMapping(value = "/allFollowRecord", method = RequestMethod.GET, produces =
"application/json")
	public @ResponseBody ArrayList<Follow> getAllFollowRecords() {
		ArrayList<Follow> list = (ArrayList<Follow>) dbService.getAllFollowRecords();
		return list;
	}

	@RequestMapping(value = "/followers/{userid}", method = RequestMethod.GET, produces =
"application/json")
	public @ResponseBody ArrayList<String> getFollowers(@PathVariable String userid) {
		if(null == dbService.getFollowers(userid)) {
			return null;
		}
		ArrayList<String> list = (ArrayList<String>) dbService.getFollowers(userid);
		return list;
	}

	@RequestMapping(value = "/following/{userid}", method = RequestMethod.GET, produces =
"application/json")
	public @ResponseBody ArrayList<String> getFollowing(@PathVariable String userid) {
		if(null == dbService.getFollowing(userid)) {
			return null;
		}
		ArrayList<String> list = (ArrayList<String>) dbService.getFollowing(userid);
		return list;
	}

	@RequestMapping(value = "/unfollow/{id}", method = RequestMethod.DELETE, produces =
"application/json")
	public @ResponseBody String unfollow(@PathVariable String id) {
		if (dbService.unfollow((Integer.parseInt(id))))
```

```
                return "Unfollowed Successfully";
            else
                return null;
        }

        @RequestMapping(value = "/follow", method = RequestMethod.POST,
produces="application/json")
        public @ResponseBody User follow(@RequestBody Follow follow) {
            return dbService.followUser(follow);
        }

        @RequestMapping(value = "/followId/{follower}/{following}", method = RequestMethod.GET,
produces="application/json")
        public @ResponseBody Integer getFollowId(@PathVariable String follower, @PathVariable
String following) {
            return dbService.getFollowId(follower, following);
        }
}
```

Reply Connection

```
@Controller
public class ReplyController {
        private static final Logger Logger = LoggerFactory.getLogger(ReplyController.class);

        @Autowired
        TweetyDBService dbService;

        @RequestMapping(value="/addReply", method=RequestMethod.POST)
        @ResponseBody
        public User addUser(@RequestBody Reply reply) {
            return dbService.addReply(reply);
        }

        @RequestMapping(value="/removeReply/{id}", method=RequestMethod.DELETE)
        @ResponseBody
        public ResponseEntity<String> removeReply(@PathVariable String id ) {
            if(dbService.removeReply(Integer.parseInt(id)))
                return new ResponseEntity<String>("success", HttpStatus.ACCEPTED);
            else
                return new ResponseEntity<String>("invalid input or no such reply exist",
HttpStatus.BAD_REQUEST);
        }

        @RequestMapping(value = "/replies", method = RequestMethod.GET,
produces="application/json")
        public @ResponseBody ArrayList<Reply> getUsers() {
            ArrayList<Reply> list = (ArrayList<Reply>)dbService.getReplies();
            return list;
        }
}
```

Tweet Controller

```
@Controller
public class TweetController {
private static final Logger Logger = LoggerFactory.getLogger(TweetController.class);

	@Autowired
	TweetyDBService dbService;

	@RequestMapping(value = "/addTweet/{id}", method = RequestMethod.PUT,
produces="application/json")
	public @ResponseBody User getUsers(@PathVariable String id, @RequestBody Tweet tweet) {
		return dbService.addTweet(tweet, id);
	}

	@RequestMapping(value = "/removeTweet/{id}", method = RequestMethod.DELETE,
produces="application/json")
	public @ResponseBody ResponseEntity<String> removeTweet(@PathVariable String id) {
		if(dbService.removeTweet(Integer.parseInt(id)))
			return new ResponseEntity<String>("Tweet Removed Successfully",HttpStatus.OK);
		else
			return new ResponseEntity<String>("Error while removing
Tweet",HttpStatus.BAD_REQUEST);
	}

	@RequestMapping(value = "/tweets", method = RequestMethod.GET, produces="application/json")
	public @ResponseBody ArrayList<Tweet> getAllTweets() {
		ArrayList<Tweet> list = (ArrayList<Tweet>)dbService.getTweets();
		return list;
	}

	@RequestMapping(value = "/followingTweets/{id}", method = RequestMethod.GET,
produces="application/json")
	public @ResponseBody ArrayList<Integer> getFollowersTweet(@PathVariable String id) {
		ArrayList<Integer> list = (ArrayList<Integer>)dbService.getFollowerTweets(id);
		return list;
	}

	@RequestMapping(value = "/connectionTweets/{id}", method = RequestMethod.GET,
produces="application/json")
	public @ResponseBody ArrayList<Integer> getConnectionsTweet(@PathVariable String id) {
		ArrayList<Integer> list = (ArrayList<Integer>)dbService.getConnectionTweets(id);
		return list;
	}
}
```