WEB APPLICATION DEVELOPMENT

PROJECT PROPOSAL REPORT

Dr: N. V. Sinh

Group 5 – Topic:

Prepared by:

VÕ CÔNG KHA, ITITIU18206
PHẠM ĐĂNG KHOA, ITITIU18276
HÀ MINH CHIẾN, ITITIU18302
PHẠM HÀNG ANH TUẤN, ITITIU18179

CATALOGUE

INTRODUCTION

1. Motivation and Problem Statement:

When running applications on computers, users may encounter various unwanted features, errors that may or may not cause the application to execute thoroughly. Such features or errors like those are known as bugs or glitches.

Although those bugs or glitches might possibly give users some unexpected advantages, they make user's experience go bad overall and generally. In a life cycle of an application, especially in the video games industry, bugs and glitches appearance is inevitable. No matter how careful and accurate a developer or software designer is, they will always encounter them numerous times. Another problem is that, developers and software designers often do not have time to manually search for hidden bugs in their creations. Therefore, they ask users of the application for help. Because user base is vast, every bug which user discovers will save developers lots of time. All they need to do is to collect the complete set of bugs and glitches and resolve them for the next application's update.

That's why the Bugtracker system is crucial to every computer applications and group 5 of the Web Application Development class had started to develop a system used to create reports of bug for the staff team to handle.

2. Learning Outcome:

- Understand the Model – View – Control structure of a Web Application.
- Advance skills in designing databases in application.
- Advance skills in developing Web's frontend and backend.
- Develop skills in programming with various Web Frameworks.

3. Contribution Rates

- Võ Công Kha - ITITIU18206 (25%)
- Phạm Đăng Khoa - ITITIU18276 (25%)
- Hà Minh Chiến - ITITIU18302 (25%)
- Phạm Hàng Anh Tuấn - ITITIU18179 (25%)

➔ More information in the Planning section page 4

LITERATURE REVIEW

Platforms and Tools used:

1. React.js

- React.js is an open – source JavaScript library used for front-end user interface or components development. It was created originally by Jordan Walke and is currently maintained by Facebook and other community developers.
- Based on various online surveys, it indicates that React has been one of the most popular assets to create user interfaces. Mainly because of its high performance, flexibility and the powerful support community.

2. Express.js:

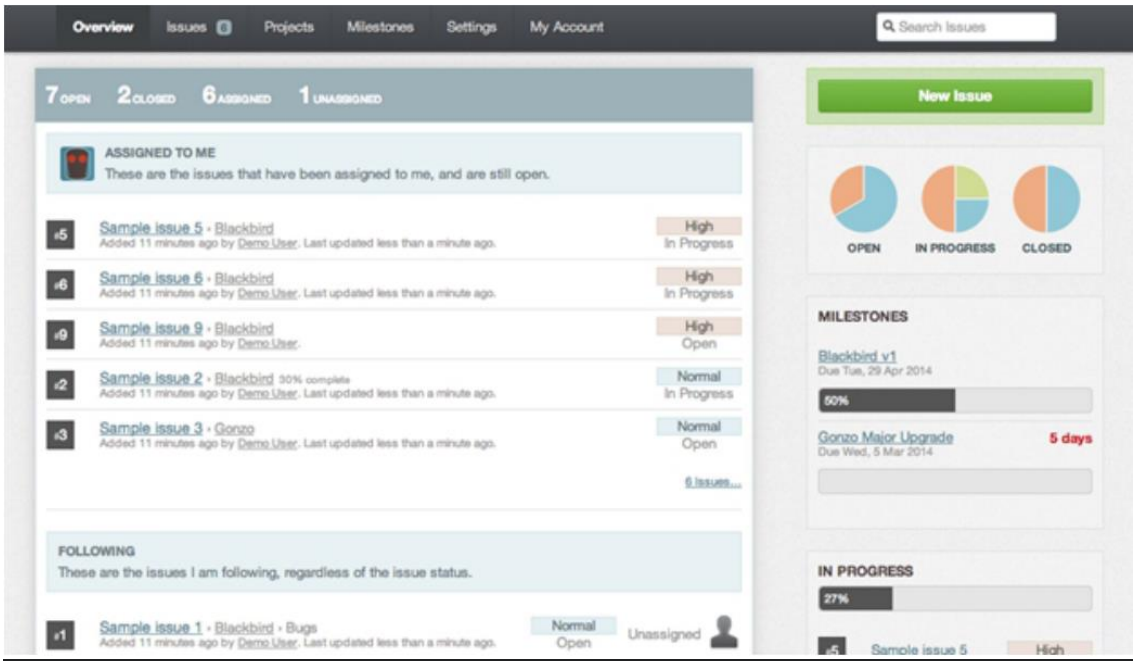
- Express.js is an open – source web application framework for Node.js used for designing and building backend of web applications. Thanks to Express written in JavaScript, it gives programmers and developers easier to build web applications.
- It is lightweight and can help organize one's web application on the backend side into a secure Model – View – Control structure.

3. MySQL:

- MySQL is an open – source relational database management system (RDBMS).
- It works with the host's operating systems to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.
- MySQL is responsible to power storage solution of many popular websites, such as Flickr, MediaWiki, Twitter, YouTube and Facebook.

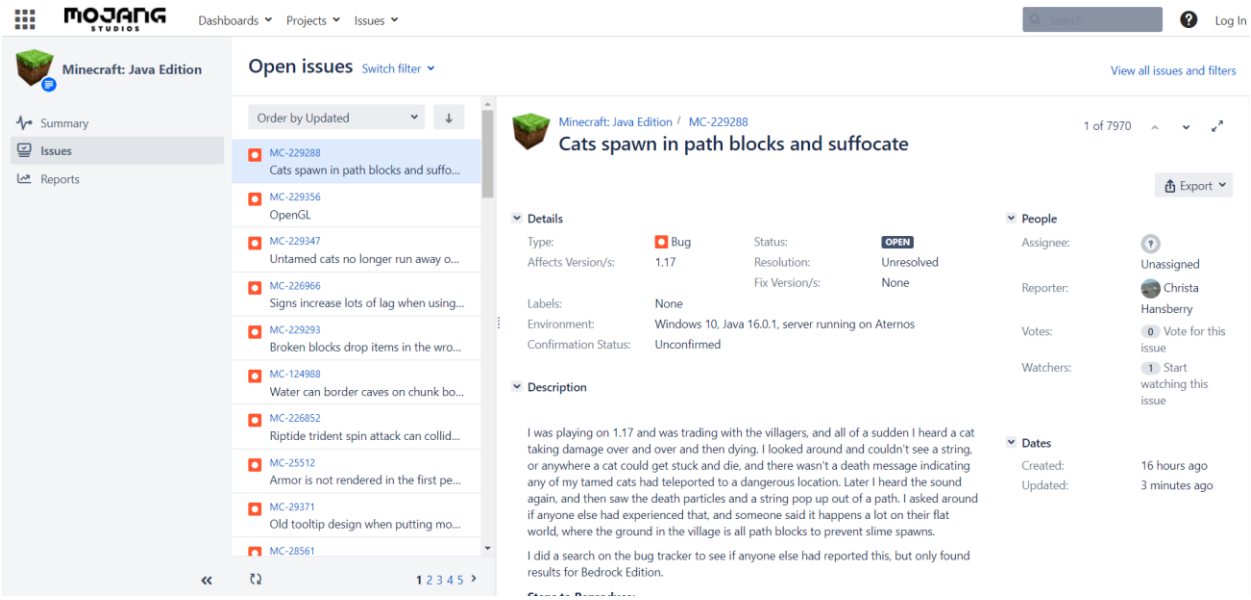
Similar Systems

- Bugify: bug tracking system, free for users to report bugs.



Bugify UI (https://bugify.com)

- Mojang's Bugtracker: Bug Tracking System by Mojang, used for reporting bugs and glitches of their products (Minecraft, Cobalt, …)



Mojang's bugtracking system (https://bugs.mojang.com/projects/MC/issues)

SYSTEM DESIGN

1. Planning:

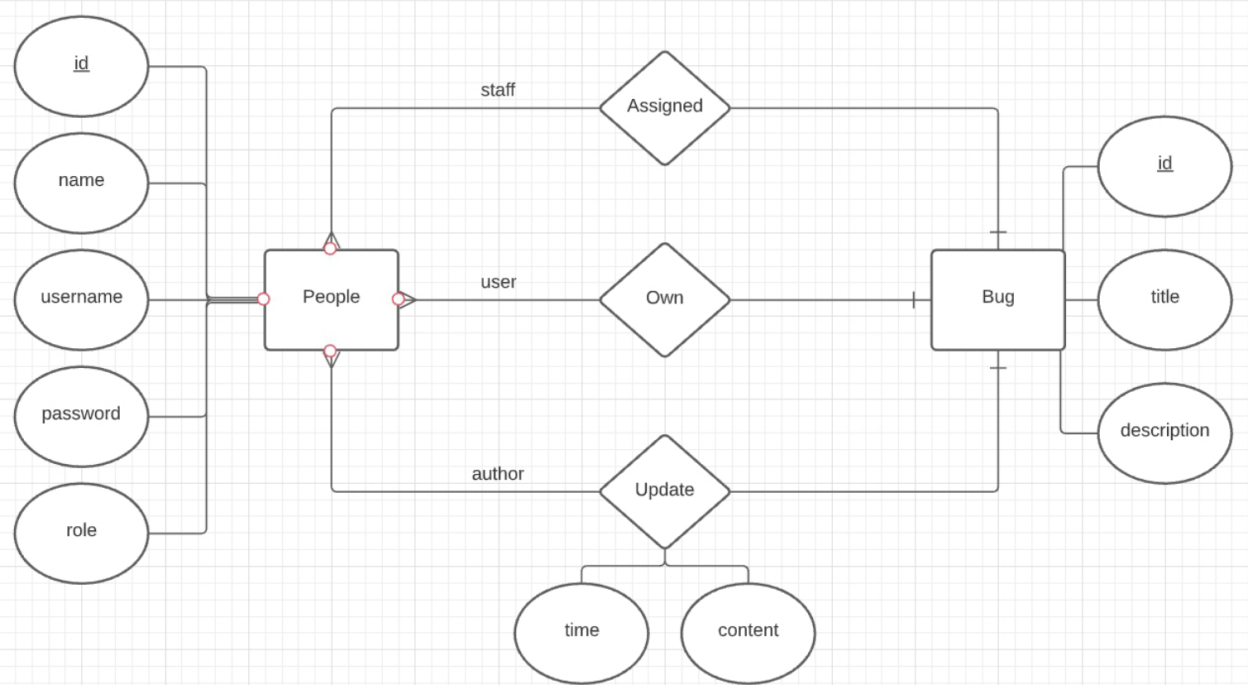
Timestamp and Duration	Activities
Week 1	Research phase for Topic (ALL)
Week 2	Fundamental knowledge preparation (ALL)
Week 3 → 4	Resource Planning (ALL)

Week 5 → 8	Implementation:
	- Backend: Kha, Khoa
	  ○ Create and design databases.
	  ○ Connect backend web server to frontend application
	  ▪ Declare Routes and Controllers.
	  ▪ Declare connection between Express app and DBMS.
	  ▪ Secure connection between React and Express apps.
	  ▪ Considers possible vulnerability.
	- Frontend: Tuan, Chien
	  ○ General Design + Divide UI Components: Tuan, Chien
	  ○ Implements React components:
	  ▪ Login, CustomerRegister, BugView, Message.
	  ▪ AssignBug, StaffView, ProjectView, CreateBug.

Week 9	Testing and Review: - UI: Tuan, Chien - DB and backend: Kha, Khoa
Week 10	General Test (ALL) Issue report (Kha, Others Supporting)

2. Specific details

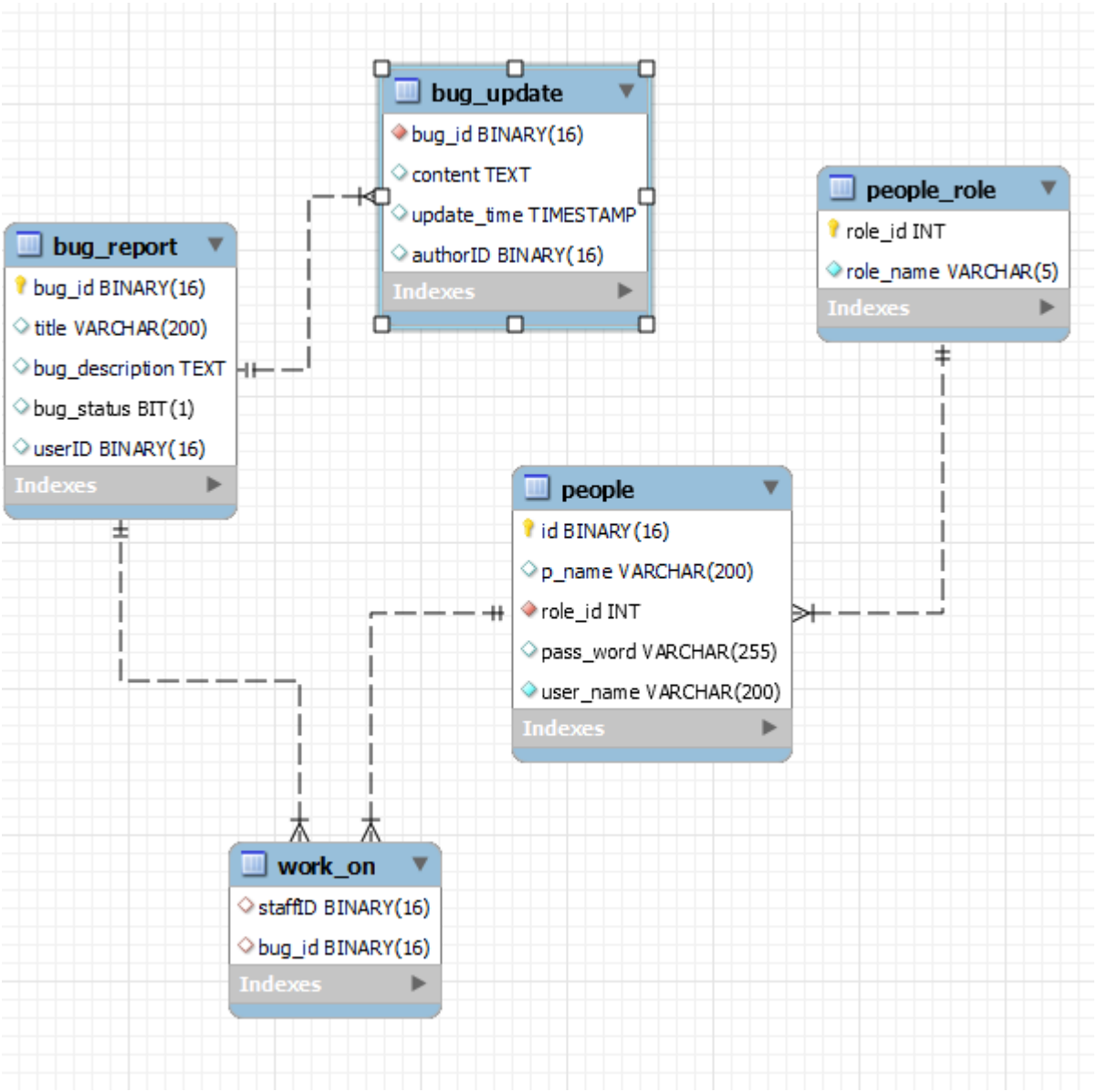
a. Entity Relationship Diagram



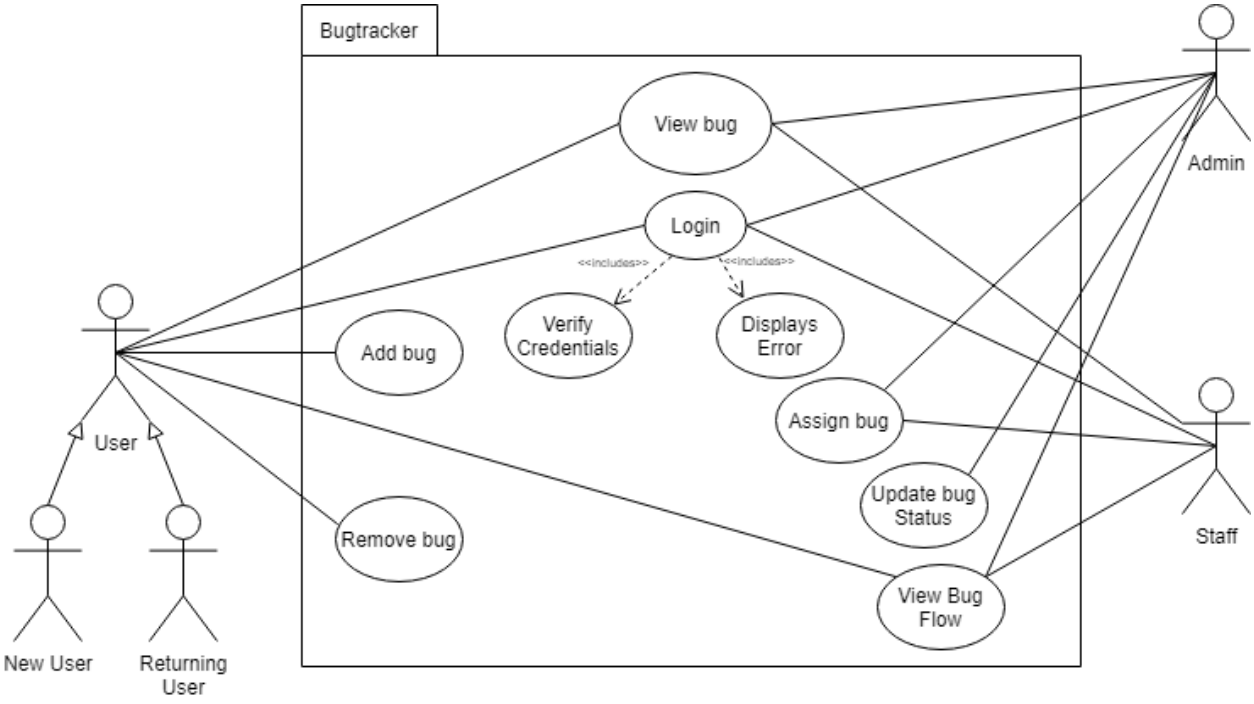
When the database is formatted, it forms this table:

people: (<u>id</u>, name, username, password, role)
bug: (<u>id</u>, title, description, *staffID, userID*)
update: (*<u>authorID, bugID,</u>* time, content)

b. Table Relationship Diagram



c. Use Case Diagram



d. Source code details:

- Backend:

```
const express = require('express')
const cors = require('cors')

const app = express()
app.use(express.json())
app.use(cors())
app.use((req, res, next) => {
  console.log('----------------------------------------')
  console.log(req.method, req.url, req.body)
  next()
})

// Load routes
apiRouter = express.Router()
apiRouter.use('/login', require('./routes/login'))
apiRouter.use('/bugs', require('./routes/bug'))
apiRouter.use('/people', require('./routes/people'))

app.use('/api', apiRouter)

const PORT = process.env.PORT || 5000
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`)
})
```

```
module.exports = {
  MYSQL_CONFIG: {
    // host: 'dbproject.3utilities.com',
    // user: 'wad_guest',
    // password: 'mysqllit1765',
    // database: 'bugtracker',
    connectionLimit: 10,
    // host: 'localhost',
    // user: 'root',
    // password: 'Khoa@35101581',
    // database: 'bugtracker',
    host: 'localhost',
    user: 'root',
    password: 'root',
    database: 'bugtracker',
  },
  secret: 'not-so-secret-anymore',
  tokenExpires: 120, // seconds
  saltRounds: 10
}
```

App.js and config.js at src/

  - The main App.js file for starting the backend server.

- On the right are the information that the backend server uses. This includes host, password, database name, IP address of the database. In addition, tokenExpires regulates when will a user's token timeout so that they will have it renewed in order to continue operation on the web application.

```
const { request } = require("express")
const jwt = require("jsonwebtoken")
const config = require("../config")
const connection = require("./database")
const bcrypt = require("bcrypt")


exports.handleLogin = async (req, res) => {
  const { username, password } = req.body

  const rows = await connection.query('SELECT *, BIN_TO_UUID(id) as id FROM people WHERE user_name=?', [username])
  person = rows[0][0]
  if (!person) {
    return res.status(401).json({
      message: "Invalid username"
    })
  }

  const match = await bcrypt.compare(password, person.pass_word);
  if (match) {
    const role = ['admin', 'staff', 'user'][person.role_id - 1]
    const authUser = {
      id: person.id,
      name: person.p_name,
      username,
      role
    }
    const token = jwt.sign(authUser, config.secret, {
      expiresIn: config.tokenExpires
    })
    res.json({
      ...authUser,
      accessToken: token
    })
  } else {
    return res.status(401).json({
      message: "Wrong password"
    })
  }
}

exports.permit = (...permittedRoles) => {
  return (req, res, next) => {
    const token = req.headers['x-access-token']

    if (!token) {
      return res.status(401).json({
        message: 'No token provided'
      })
    }

    jwt.verify(token, config.secret, (err, decodedUser) => {
      if (err) {
        return res.status(401).json({
          message: 'Invalid token'
        })
      }

      req.locals = {
        user: decodedUser
      }

      if (permittedRoles.length === 0) {
        return next()
      }

      if (permittedRoles.includes(decodedUser.role)) {
        next()
      } else {
        res.status(403).json({
          message: 'Unauthorized'
        })
      }
    })
  }
}
```

auth.js file in src/controllers/

- This is the main core authorization of login, package jsonwebtoken is used for replacing sessions. Furthurmore, bcrypt package is also used to hash classified credentials of user, giving them an extra layer of security.

```
const formatTime = (dt) => {
  let current_date = dt.getDate()
  let current_month = dt.getMonth() + 1
  let current_year = dt.getFullYear()
  let current_hrs = dt.getHours()
  let current_mins = dt.getMinutes()
  let current_secs = dt.getSeconds()
  current_date = current_date < 10 ? '0' + current_date : current_date
  current_month = current_month < 10 ? '0' + current_month : current_month
  current_hrs = current_hrs < 10 ? '0' + current_hrs : current_hrs
  current_mins = current_mins < 10 ? '0' + current_mins : current_mins
  current_secs = current_secs < 10 ? '0' + current_secs : current_secs
  return current_year + '-' + current_month + '-' + current_date + ' ' + current_hrs + ':' + current_mins + ':' + current_secs
}

const getUpdatesByBugID = async (id) => {
  const sql =
    `SELECT update_time as time, content, BIN_TO_UUID(authorID) as authorID
    FROM bug_update WHERE bug_id = UUID_TO_BIN(?)`
  return await connection.query(sql, [id])
}

const fixSQLTime = (updates) => updates.map(update => ({
  ...update,
  time: formatTime(update.time)
}))
```

```
const includeBugUpdates = async (bugs, response) => {
  const fullBugs = bugs.map(bug => getUpdatesByBugID(bug.id))
  const fullBug = await Promise.all(fullBugs)
  response.json(bugs.map((bug, index) => {
    return {
      ...bug,
      updates: fixSQLTime(fullBug[index][0])
    }
  }))
}

exports.getBugs = async (req, res) => {
  const sql =
    `SELECT
    BIN_TO_UUID(b.bug_id) AS id,
    title,
    bug_description AS description,
    BIN_TO_UUID(userID) AS userID,
    BIN_TO_UUID(staffID) AS staffID
    FROM bug_report AS b, work_on AS w
    WHERE b.bug_id = w.bug_id AND b.bug_status=0`
  const result = await connection.query(sql)
  await includeBugUpdates(result[0], res)
}
```

```
exports.createBug = async (req, res) => {
  const UUID = uuidv4()
  const { title, description, userID, staffID, updates } = req.body
  const { time, content, authorID } = updates[0]
  sql1 = 'INSERT INTO bug_report(bug_id, title, bug_description, bug_status, userID) VALUES (UUID_TO_BIN(?), ?, ?, 0, UUID_TO_BIN(?))'
  sql2 = 'INSERT INTO work_on(bug_id, staffID) VALUES(UUID_TO_BIN(?), UUID_TO_BIN(?))'
  sql3 = 'INSERT INTO bug_update(bug_id, content, update_time, authorID) VALUES (UUID_TO_BIN(?), ?, ?, UUID_TO_BIN(?))'
  await connection.query(sql1, [UUID, title, description, userID])
  await connection.query(sql2, [UUID, staffID])
  await connection.query(sql3, [UUID, content, time, authorID])
  res.json({
    id: UUID
  })
}
```

```
exports.updateBugByID = async (req, res) => {
  const { updates } = req.body
  const bugID = req.params.bugID
  delete_sql = 'DELETE FROM bug_update WHERE bug_id = UUID_TO_BIN(?)'
  insert_sql = 'INSERT INTO bug_update(bug_id, content, update_time, authorID) VALUES (UUID_TO_BIN(?), ?, ?, UUID_TO_BIN(?))'
  await connection.query(delete_sql, [bugID])
  const promises = updates.map(({ content, time, authorID }) =>
    connection.query(insert_sql, [bugID, content, time, authorID]))
  await Promise.all(promises)
  res.end()
}

exports.forwardBugByID = async (req, res) => {
  const { staffID } = req.body
  const bugID = req.params.bugID
  update_sql = 'UPDATE work_on SET staffID=UUID_TO_BIN(?) WHERE bug_id = UUID_TO_BIN(?)'
  await connection.query(update_sql, [staffID, bugID])
  res.end()
}

exports.deleteBugByID = async (req, res) => {
  const id = req.params.bugID
  await connection.query('DELETE FROM work_on WHERE bug_id = UUID_TO_BIN(?)', [id])
  await connection.query('DELETE FROM bug_report WHERE bug_id = UUID_TO_BIN(?)', [id])
  res.end()
}
```

bug.js file in src/controllers/

- bug.js provides all of the operations needed to interact with bugs, such as: createBug, getBug, updateBug, …
- bug.js contains various queries to the SQL database.

```
const mysql = require('mysql2')
const config = require('../config')
const bcrypt = require('bcrypt')

// Test database connection
const connection = mysql.createConnection(config.MYSQL_CONFIG)
connection.connect((err) => {
  if (err) throw err
  console.log("Database connected")
})
connection.end()

// new password to hash
const password = 'admin'
bcrypt.hash(password, config.saltRounds).then((hashed) => {
  console.log(`${password} -> ${hashed}`)
})

// Use connection pool with Promise support
const pool = mysql.createPool(config.MYSQL_CONFIG).promise()

module.exports = pool
```

database.js at src/controllers

- database.js regulates the constant info of the database used for the application. This file is created for multiple use at different locations in the application directory tree.

```
const connection = require('./database')
const { v4: uuidv4 } = require('uuid')
const bcrypt = require('bcrypt')
const config = require('../config')



exports.getPeople = async (req, res) => {
  const sql =
    `SELECT BIN_TO_UUID(id) as id,
    p_name AS name, role_name AS role, user_name as username
    FROM people as p, people_role as r
    WHERE p.role_id = r.role_id`
  const result = await connection.query(sql)
  res.json(result[0])
}

exports.createPerson = async (req, res) => {
  const { name, role, username, password } = req.body
  const hashedPassword = await bcrypt.hash(password, config.saltRounds)
  const UUID = uuidv4()
  const role_id = ['admin', 'staff', 'user'].indexOf(role) + 1
  const sql = 'INSERT INTO people VALUES(UUID_TO_BIN(?), ?, ?, ?, ?)'
  await connection.query(sql, [UUID, name, role_id, hashedPassword, username])
  res.json({
    id: UUID
  })
}

exports.updatePersonByID = async (req, res) => {
  const id = req.params.id
  const { name, role, username, password } = req.body
  const role_id = ['admin', 'staff', 'user'].indexOf(role) + 1

  if (password && password.length > 0) {
    const hashedPassword = await bcrypt.hash(password, config.saltRounds)
    update_sql = 'UPDATE people SET p_name=?, role_id=?, pass_word=?, user_name=? WHERE id = UUID_TO_BIN(?)'
    await connection.query(update_sql, [name, role_id, hashedPassword, username, id])
  } else {
    update_sql = 'UPDATE people SET p_name=?, role_id=?, user_name=? WHERE id = UUID_TO_BIN(?)'
    await connection.query(update_sql, [name, role_id, username, id])
  }
  res.end()
}

exports.deletePersonByID = async (req, res) => {
  const id = req.params.id
  const sql = 'DELETE FROM people WHERE id=UUID_TO_BIN(?)'
  await connection.query(sql, [id])
  res.end()
}
```

people.js at src/controllers/

- people.js provides all information about the user and interactions.
- people.js contains various queries to the SQL database.

```
const bugController = require('../controllers/bug')
const express = require('express')
const { permit } = require('../controllers/auth')
const router = express.Router()


router.get('/', permit(), async (req, res) => {
  const { role } = req.locals.user
  if (role === 'staff') {
    await bugController.getBugsByStaffID(req, res)
  } else if (role === 'user') {
    await bugController.getBugsByUserID(req, res)
  } else {
    await bugController.getBugs(req, res)
  }
})

router.post('/', permit('admin'), bugController.createBug)

router.patch('/:bugID', permit('admin', 'staff'), async (req, res) => {
  const { staffID } = req.body
  if (staffID) {
    await bugController.forwardBugByID(req, res)
  } else {
    await bugController.updateBugByID(req, res)
  }
})

router.delete('/:bugID', permit('admin', 'user'), bugController.deleteBugByID)


module.exports = router
```

```
const peopleController = require('../controllers/people')
const express = require('express')
const { permit } = require('../controllers/auth')
const router = express.Router()


router.get('/', permit(), peopleController.getPeople)

router.post('/', permit('admin'), peopleController.createPerson)

router.patch('/:id', permit('admin'), peopleController.updatePersonByID)

router.delete('/:id', permit('admin'), peopleController.deletePersonByID)


module.exports = router
```

```
const authController = require('../controllers/auth')
const express = require('express')
const router = express.Router()


router.post('/', authController.handleLogin)


module.exports = router
```

Routing .js files at src/routes/ (bug.js, people.js, login.js) respectively

- These files regulates the routing between React app and Express server, allowing crossing over data transfer.

- Frontend: (click here to view the source codes)

e. MySQL schema generation code:

```
CREATE TABLE `bug_report`
(
  `bug_id` binary(16) NOT NULL,
  `title` varchar(200) DEFAULT NULL,
  `bug_description` text,
  `bug_status` bit(1) DEFAULT NULL,
  `userID` binary(16) DEFAULT NULL,
  PRIMARY KEY (`bug_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

➔ Initialize 'bug_report' table

```
CREATE TABLE `bug_update`
(
  `bug_id` binary(16) NOT NULL,
  `content` text,
  `update_time` timestamp NULL DEFAULT NULL,
  `authorID` binary(16) DEFAULT NULL,
  KEY `update_report_idx` (`bug_id`),
  CONSTRAINT `update_report` FOREIGN KEY (`bug_id`) REFERENCES `bug_report` (`bug_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

➔ Initialize 'bug_update' table

```
CREATE TABLE `people`
(
  `id` binary(16) NOT NULL,
  `p_name` varchar(200) DEFAULT NULL,
```

```
  `role_id` int NOT NULL,
  `pass_word` varchar(255) DEFAULT NULL,
  `user_name` varchar(200) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `user_name` (`user_name`),
  KEY `people_role_idx` (`role_id`),
  CONSTRAINT `ppl_role` FOREIGN KEY (`role_id`) REFERENCES `people_role` (`role_id`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

➔ Initialize the 'people' table

```
CREATE TABLE `people_role`
(
  `role_id` int NOT NULL,
  `role_name` varchar(5) NOT NULL,
  PRIMARY KEY (`role_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

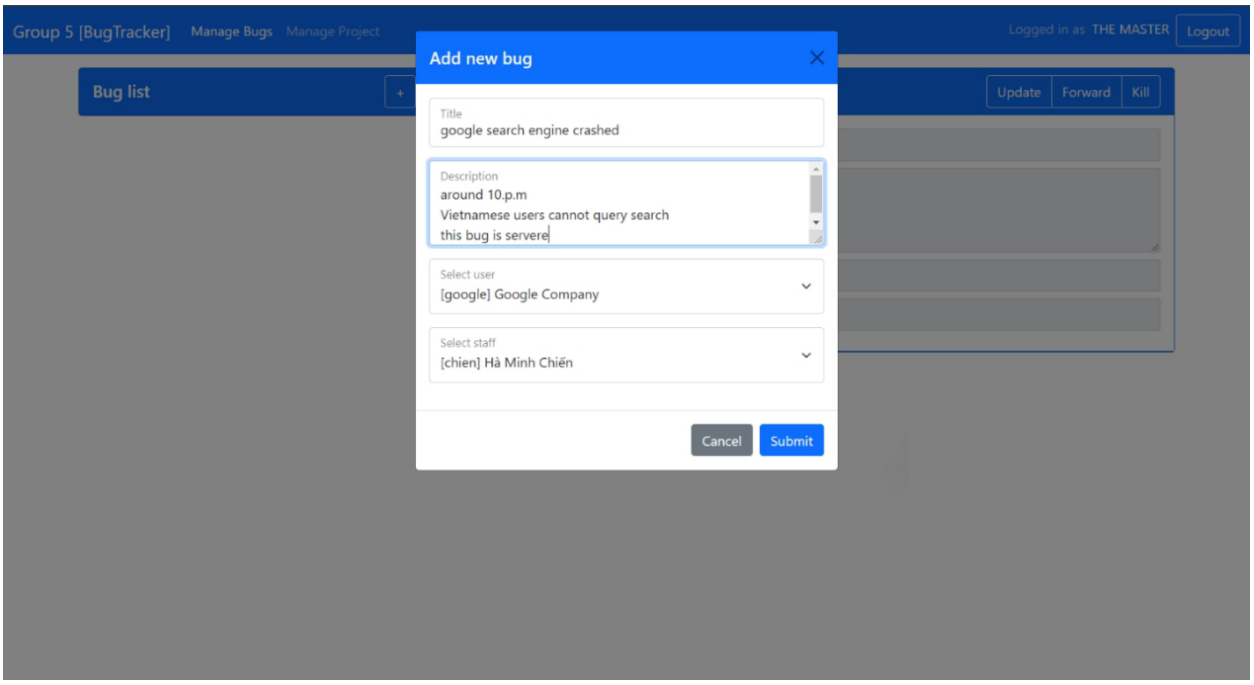
➔ Initialize the 'people_role' table

```
CREATE TABLE `work_on` (
  `staffID` binary(16) DEFAULT NULL,
  `bug_id` binary(16) DEFAULT NULL,
  KEY `staffID` (`staffID`),
  KEY `bug_id` (`bug_id`),
  CONSTRAINT `work_on_ibfk_1` FOREIGN KEY (`staffID`) REFERENCES `people` (`id`),
  CONSTRAINT `work_on_ibfk_2` FOREIGN KEY (`bug_id`) REFERENCES `bug_report`
(`bug_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

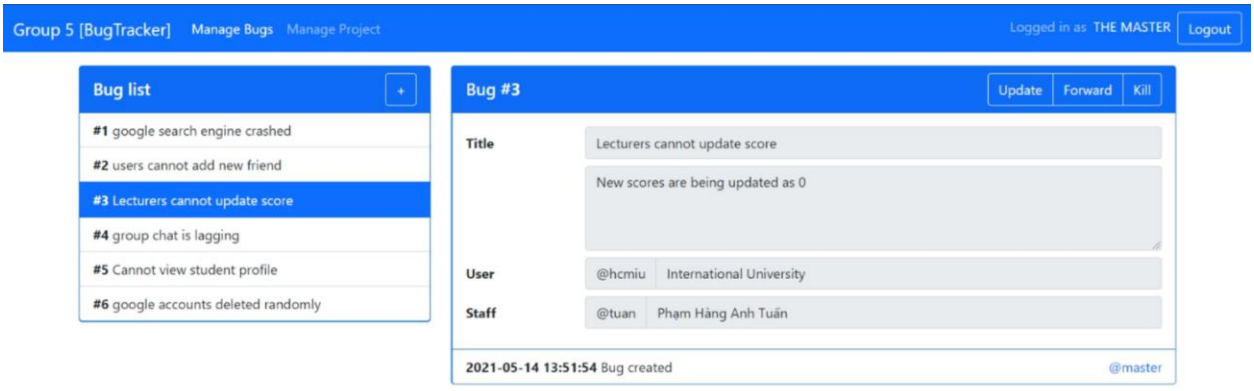
➔ Initialize the 'work_on' table

IN-DEV IMAGES

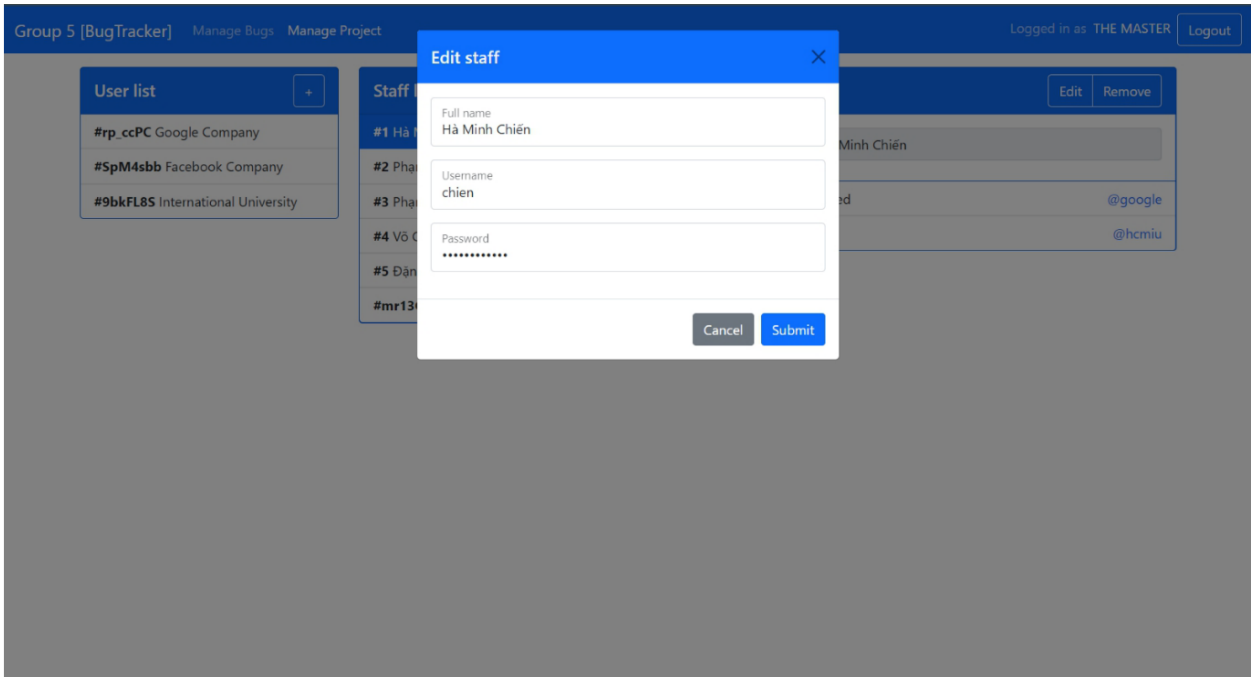
Images of the In – Development application:



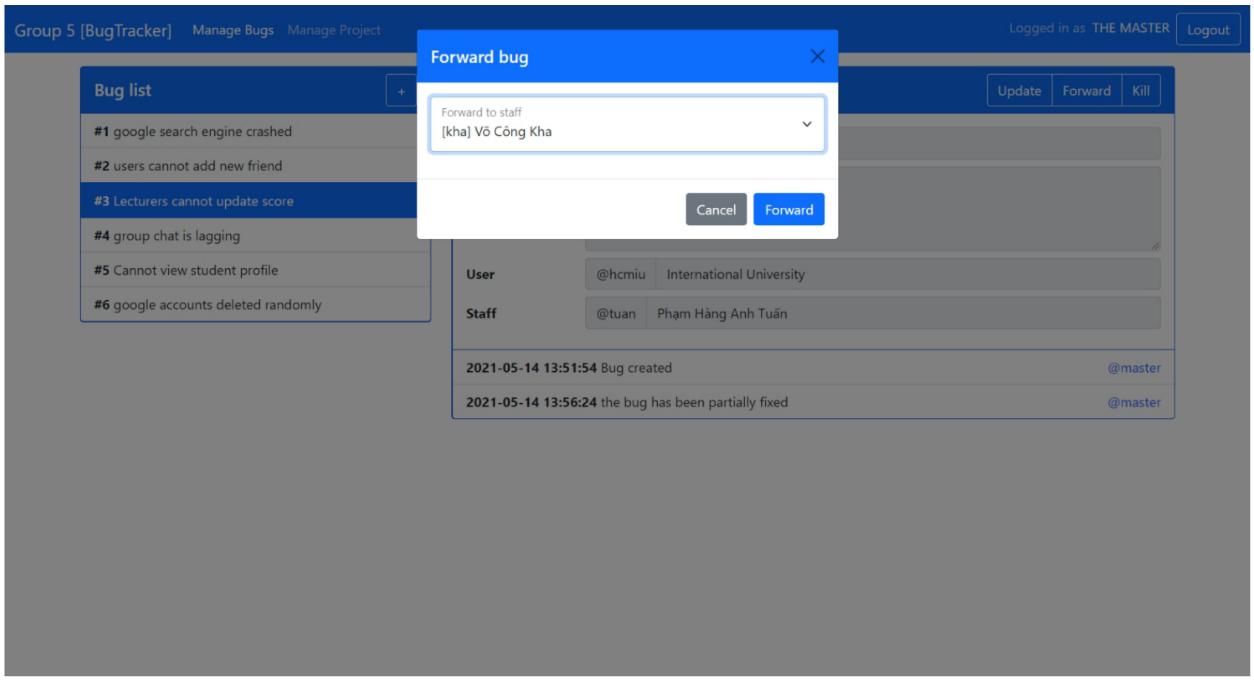
Adding new bugs



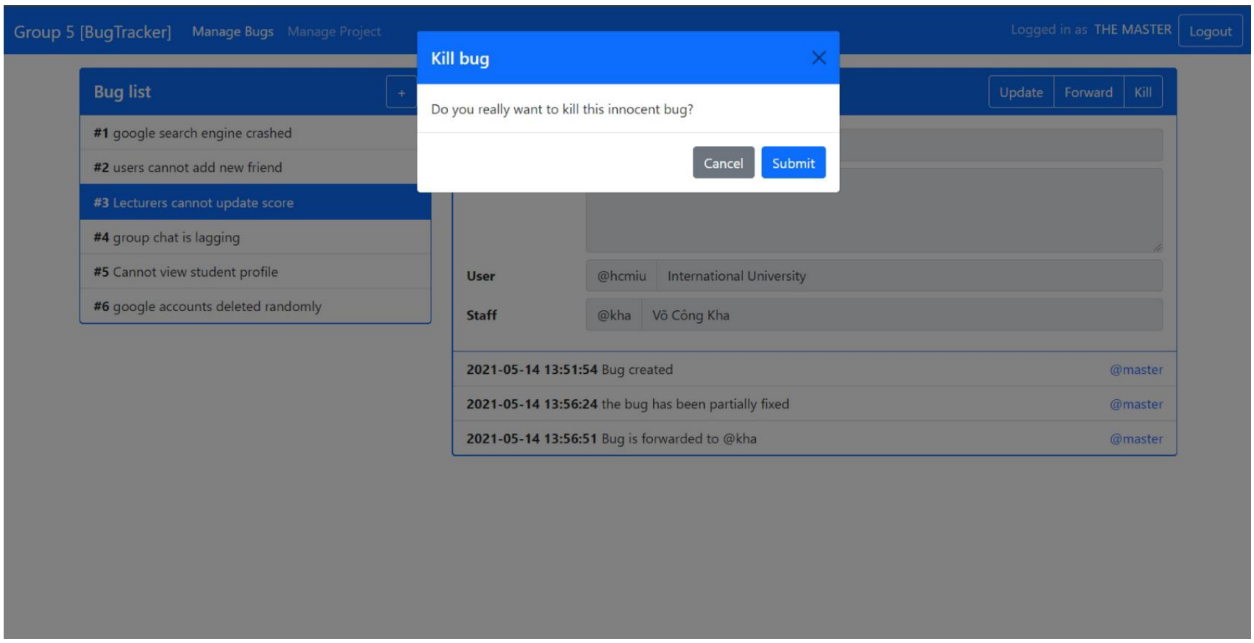
Bug view



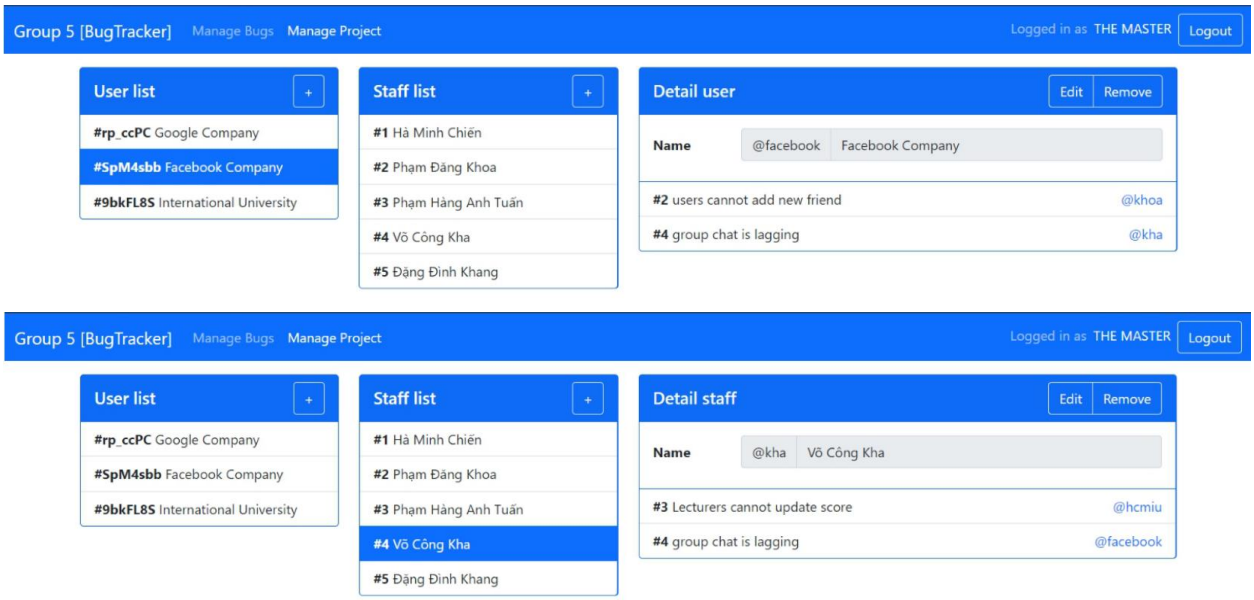
Editing Staff information



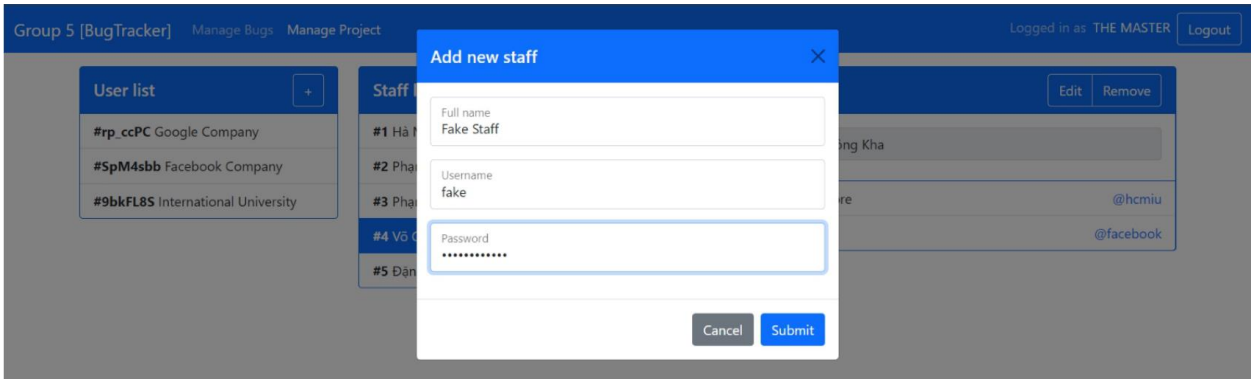
Forwarding bugs



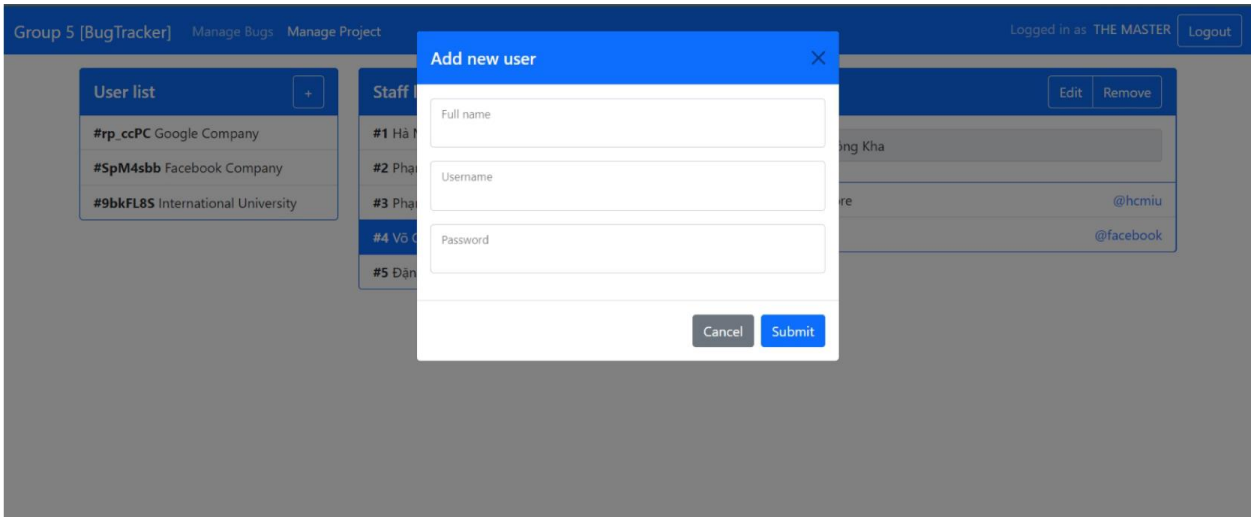
Killing bug



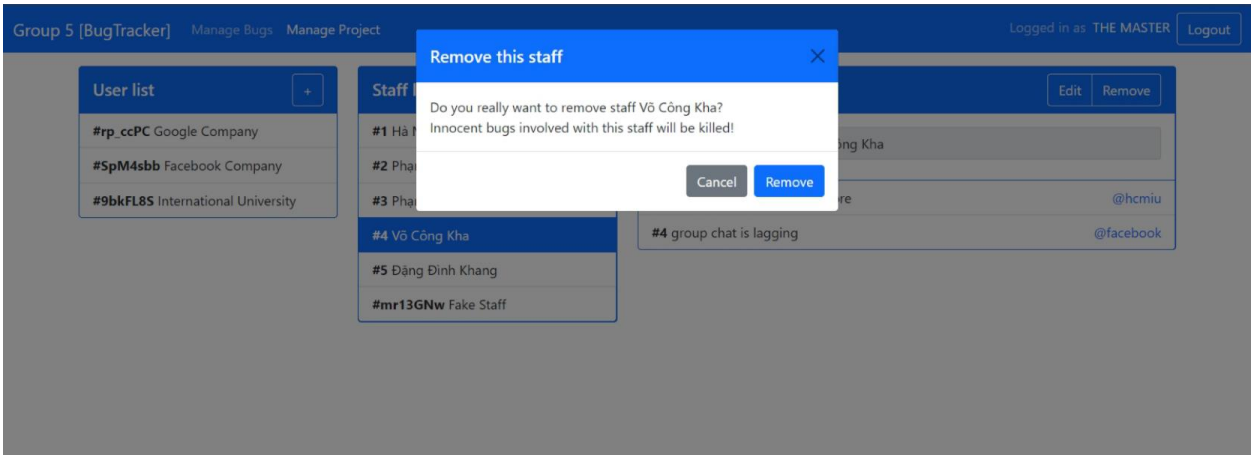
Project Management UI



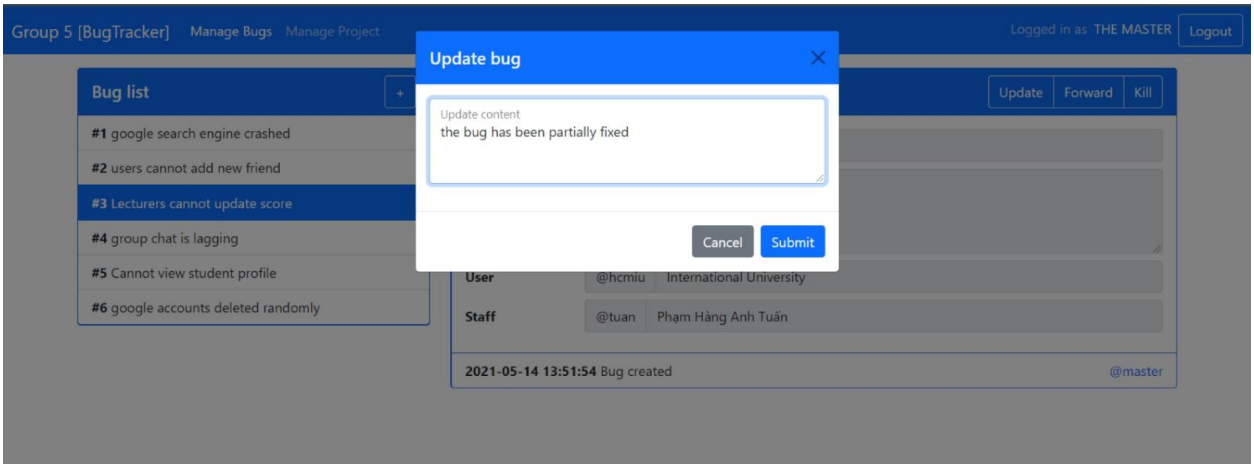
Adding new Staff to team



Adding new User



Removing / Demoting Staff



Update Bug Status

CONCLUSION

1. List of completed works

- Designed the UI.
- Succesful implementation of backend Express server.
- Testing phase shows pleasant and expected results. All errors had been handled.

2. Pros and Cons

- Pros:
  - Information strictly secured by using bcrypt library.
  - Lightweight, High Performance Application.
  - Async, Await implemented to avoid the Pyramid of Doom.
  - Try/Catch the Tower of Terror.

- Cons:
  - Media such as images or videos are not supported.
  - UI design is still decent, not very eye – catching.

3. Future Works

- Extend the bug creation system to include media.
- More development on the web user interface.
- Implementation of search bug function.
- Integrate into mobile app with React Native.